# Intelligent Optical Methods in Image Analysis for Human Detection

A thesis submitted for the degree of Doctor of Philosophy

by
Jean-Marc Graumann

School of Engineering and Design
Brunel University
June 2005

# Acknowledgments

I would like to use this opportunity to extend my thanks to the various people who helped me along the way:

Dr. Chris Kirkham

Dr. Stan Swallow

Neil Brown (Weyrad Electronics Ltd.)

Paul Gammans

Adrian Long

My father, for proofreading

And last but not least, my wife, for putting up with me and helping me to finish !

*Jean-Marc Graumann, June 2005*

# Abstract

This thesis introduces the concept of a person recognition system for use on an integrated autonomous surveillance camera.

Developed to enable generic surveillance tasks without the need for complex setup procedures nor operator assistance, this is achieved through the novel use of a simple dynamic noise reduction and object detection algorithm requiring no previous knowledge of the installation environment and without any need to train the system to its installation.

The combination of this initial processing stage with a novel hybrid neural network structure composed of a SOM mapper and an MLP classifier using a combination of common and individual input data lines has enabled the development of a reliable detection process, capable of dealing with both noisy environments and partial occlusion of valid targets.

With a final correct classification rate of 94% on a single image analysis, this provides a huge step forwards as compared to the reported 97% failure rate of standard camera surveillance systems.

# Table of Contents

# Index of Images

# Referenced Publications

[01] - B. Aird, A. Brown, "No Fire without Smoke, CCTV Breakthrough in Fire Detection" - Journal of Applied Fire Science, Vol. 11, No. 2, 2002-2003, pp. 183-193.

[02] - Alarm Tech, "AlarmTech - How your Infrared sensors work", http://www.alarmtech.org/security/pir.html

[03] - K. Arakawa, "Fuzzy Rule Based Image Processing with Optimisation", International Journal of Imaging Systems and Technology, Vol 8, 1997, pp. 222-247.

[04] - H. Bandemer, "Specifying Fuzzy Data from Grey-Tone Images for Pattern Recognition", Pattern Recognition Letters, Vol. 17, No. 6, May 1996, pp. 585-592.

[05] - V.Becanovic, M.Kermit, A.J. Eide, "Feature Extraction from Photographic Images using a Hybrid Neural Network", Proc. SPIE Vol. 3728, 1999, pp. 351-361.

[06] - J. Bigun, "Pattern Recognition in Images by Symmetries and Coordinate Transformations", Computer Vision and Understanding, Vol 68, No. 3, Dec 1997, pp. 290-307.

[07] - A. Blum, "Neural Networks in C++ an object oriented framework for building connectionist systems", John Wiley and Sons, 1992.

[08] - T. Caelli, C. Dillon, E. Osman, G. Krieger, "IPRS Image Processing and Pattern Recognition System", Spatial Vision, Vol 11, No. 1, 1997, pp. 107-122.

[09] - M.F. Campos, R. A. Mini, "Visual Tracking of Multiple Objects using Wavelet Transforms", Proceedings of SPIE, Vol 3723, 1999, pp. 341-349.

[10] - A. Caplier, C. Dumontier, F. Luthon, P.Y. Coulon, "Algorithme de Detection de Mouvement par Modelisation Markovienne, Mise en Oeuvre sur DSP", Traitement du Signal, Vol 13, Issue 2, 1996, pp.177-190.

[11] - CCTV Today, "Integrated Package System", - CCTV Today, Vol 5, number 1, http://www.cctvmags.com/.

[12] - L. M. Chang, Y. A. Razig, D.M. Abraham, M.Chae, "Hybrid Computerised Decision Support System for Infrastructure Assessment", June 2000.

[13] - C.H. Chen, "Neural Networks in Pattern Recognition and their Applications", University of Massachusetts, 1992, 168pp.

[14] - H.D. Cheng. M. Miyojim, "Novel System for Automatic Pavement Distress Detection", Journal of Computing in Civil Engineering, Vol. 12, No. 3, July 1998, pp. 145-152.

[15] - **C**hloride, Chloride Product catalog, VM320 Integrated Camera – www.chloridegroup.com.

[16] - K. J. Cios, I. Shin, "Image Recognition Neural Network : IRNN", Neurocomputing, Vol 7, No. 2, 1995, pp. 159-185.

[17] - Clecom, "Continuous Wavelet Transform", http://www.clecom.co.uk/science/autosignal/help/Continuous_Wavelet_Transfor.htm.

[18] - R.T. Collins, A.L. Lipton, T. Kanade, "A System for Video Surveillance and Monitoring", Report CMU-RI-TR-00-12, Robotics Institue, Carnegie Mellon University, May 2000.

[19] - R.T. Collins, Y. Tsin, "Calibration of an Outdoor Active Camera System", IEEE Computer Vision and Pattern Recognition, June 1999, pp. 528-534.

[20] - J. Daugman, "Neural Image Processing Strategies Applied in Real-Time Pattern Recognition", Real-Time Imaging, Vol. 3, No. 3, June 1997, pp. 157-171.

[21] - F.C.C. DeCastro, J.N. Amaral, P.R.G. Franco, "Invariant Pattern Recognition of 2D Images usign Neural Networks and Frequency Domain Representation", , IEEE International Conference on Neural Networks - Conference Proceedins, Vol. 3, 1997, pp. 1644-1649.

[22] - E.R. Dougherty, "Digital Image Processing Methods", Published by Dekker, New York, 1994.

[23] - N. Drakos, "Kohonen SOM 2D Neural Network", http://www.ese-metz.fr/~dedu/docs/kohPaper/node3.html.

[24] - ERA, "ERA Technology", http://www.era.co.uk.

[25] - ERA, "Neural Computing and Complementary Technologies", http://www.era.co.uk/div80/bc45/comptec.htm.

[26] - ERA, "Study into Intelligent Alarm Systems", http://www.era.co.uk/div80/bc85/alarms.htm.

[27] - R. Fageth, W. Allen, U. Jager, "Fuzzy Logic Classification in Image Processing", Fuzzy Sets and Systems, Vol 82, Issue 3, Sept. 1996, pp. 265-278.

[28] - D.A Fay, A.M. Waxman, "Neurodynamics of Real-Time Image Velocity Extraction", Neural Networks for Vision and Image, MIT Press, 1992, pp. 220-246.

[29] - H. Fonga, "Pattern Recognition in Gray-Level Images by Fourier Analysis", Pattern Recognition Letters, Vol. 17, No. 14, Dec 1996, pp. 1477-1489.

[30] - G. Franceschetti, A. Iodice, M. Tesauro, "From Image Processing to Feature Processing", Signal Processing, Vol 60, No. 1, 1997, pp. 51-63.

[31] - D.M. Gavrila, "The analysis of Human Motion and its Application for Visual Surveillance", Proc. of the 2nd IEEE International Workshop on Visual Surveillance, 1999, pp. 3-5.

[32] - D.M. Gavrila, "The Chamfer System", http://www.gavrila.net/Computer_Vision/Chamfer_System/chamfer_system.html.

[33] - D.M. Gavrila, "The Visual Analysis of Human Movement, A Survey", Computer Vision and Image Understanding, vol.73, No. 1, 1999, pp. 82-98.

[34] - [22]- K. Gay, "Ergonomics-Making Products and Places fit People", New York: Enslow Publishers, 1986.

[35] - E. Gelenbe, H. Bakircioglu, T. Kocak, "Image Processing with the Random Neural Network", Proc. of the SPIE Conf. on Electronic Imaging, vol. 3307, 1998, pp. 38-49.

[36] - J. Ghoshal, K.S Ray, "Neuro Fuzzy Approach to Pattern Recognition", Neural Networks, Vol.10, No. 1 Jan 1997, pp. 161-182.

[37] - A. Graps, "An Introduction to Wavelets", IEEE Computaional Science & Engineering, Volume 2, Issue 2, 1995, pp.50-61.

[38] - Greenseal, "Choose Green Report - Occupancy Sensors", Green Seal, www.greenseal.org, Feb 1997.

[39] - T. Hengl, "Neural Network Fundamentals: A Neural Computing Primer", PC AI, Volume 16, Issue 3, 2002, pp.32.

[40] - F. Hoffman, "An Introduction to Fourier Theory", http://aurora.phys.utk.edu/~forrest/papers/fourier/index.html.

[41] - Home Security Store, "DSC Bravo 3 Infrared", http://www.homesecuritystore.com/detail_pages/bravo3.htm.

[42] - P. Johansen, "Adaptive Pattern Recognition", Johansen P., Journal of Mathematical Imaging and Vision, Vol. 7, Oct. 1997, pp.325-339.

[43] - A.J. Katz, P.R. Thrift, "Generating Image Filters for Target Recognition by Genetic Learning", IEEE Transactions on Pattern Analysis and Machine Intelligence, Vol 16, No. 9, Sept. 1994, pp. 906-910.

[44] - M. I Khalil, M.M. Bayoumi, "Invariant 2D Object Recognition using Wavelet Transform and Structured Neural Networks", Proceedings of SPIE, Vol 3723, No. 37, 1999.

[45] - KJB Security, "Motion Sensor Camera", http://www.kjbsecurity.com/products/Motion Sensor.asp.

[46] - V. Kober, V. Lashin, L. Moreno, J. Campos, L. Yaroslavsky, M.J. Yzuel, "Color Component Transformations for Optical Pattern Recognition", Journal of the Optical Society of America : Optics and Image Science and Vision, Vol 14, Number 10, Oct. 1997, pp.2656-2669.

[47] - H. Konik, V. Lozano, B. Laget, "Color Pyramids for Image Processing", The Journal of Imaging Science and Technology, Vol 40, Number 6, 1996, pp. 535-542.

[48] - A. Kuehnle, W. Burghout, "Image Based Winter Road Condition Recognition", Applications of Advanced Technologies in Transportation, Sept. 1998, pp. 225-232.

[49] - A.D. Kulkarni, "Artificial Neural Networks for Image Understanding", Van Nostrand Reinhold, 1994.

[50] - C.J. Kuo, C.H. Lin, C.H. Yeh, "Noise Reduction of VQ encoded images through Anti-Grey Coding",IEEE Transactons on Image Processing, Vol.8, Jan. 1993, pp. 30-40.

[51] - D. Lake, "Getting the Picture", Advanced Imaging, Jan 2000.

[52] - C.S. Lee, Y.H. Kuo, P.T. Yau, "Weighted Fuzzy Mean Filters for Image Processing", Fuzzy Sets and Systems, Vol 89, No. 2, 1997, pp. 157-180.

[53] - G. Lin, B. Shi, "A Current-Mode Fuzzy Processor for Pattern Recognition", Journal of Circuits and Systems, Vol.4, No.3, 1999.

[54] - D. Liu, Y. Yamashita, H. Ogawa, "Pattern Recognition in the Presence of Noise", Pattern Recognition, Vo. 28, No. 7, July 1995, pp. 989-995.

[55] - D. Marr, "Representing Shapes for Recognition", W.H. Freeman & Co Publishers, 1982.

[56] - McMaster University, "Nonlinear Associative Memory Models", http://www.psychology.mcmaster.ca/3W03/nlam.html.

[57] - Micro Actuators, sensors and systems group, MASS cameras technical sheets, http://mass.micro.uiuc.edu/index.html.

[58] - L. Monostori, "From Pattern Recognition Techniques through Artificial Neural Networks to Hybrid AI Solutions in Manufacturing", Proceedings of the Japan/USA Symposium on Flexible Automation, Vol. II, July 1996, pp. 1453-1460.

[59] - I.S. Moreno, V. Kober, V. Lashin, J. Campos, L.P. Yaroslavsky, M.J. Yzuel, "Whitening Preprocessing of Color Components for Pattern Recognition", Proceedings of SPIE, 1996, Vol 2730.

[60] - C.Nakajima, M.Pontil, B.Heiele, T.Poggio, "Full Body Person Recognition System", Pattern Recognition, Vol. 36, Jan. 2003, pp.1997-2003.

[61] - Neurodynamics, "Witness Security, Affordable & Flexible Image Surveillance" , http://www.neurodynamics.com/Vision/WitnessSecurity.htm.

[62] - J.K. Paik, J.C. Brailean, A.K. Katsaggelos, "An Edge Detection Algorithm using Multi-State Adalines", Pattern Recognition, Vol 25, Number 12, 1992, pp.1495-1504.

[63] - S. Pal, A. Ghosh, "Neuro Fuzzy Computing for Image Processing and Pattern Recognition", International Journal of Systems Science, Vol. 27, No.12, 1996, pp.1179-1193.

[64] - S.C. Pei, C.N. Lin, "Image Normalisation for Pattern Recognition", Image and Vision Computing, Vol 13, No. 10, Dec. 1995, pp. 711-723.

[65] - Photonics Spectra, "Surveillance Booms", Photonics Spectra, June 2000, http://www.photonics.com/spectra/business/XQ/ASP/businessid.488/QX/read.htm.

[66] - Primary Image, "Primary Image Video Tracker", http://www.primary-image.com.

[67] - Privacy International, "Security's New Image : New Technologies in Image Processing are paving the Way for the Future Development of Surveillance", Communicate, Issue 3, 1997, http://www.privacy.org/pi/issues/cctv/.

[68] - D. de Ridder, "Adaptive Methods of Image Processing", PhD Thesis, Delft University, 2001, pp.1-288.

[69] - J.F. Rivest, R. Fortin, "Detection of Dim Targets in Digital Infrared Imagery by Morphological Image Processing", Optical Engineering, Vol 35, No. 7, July 1996, pp. 1886-1893.

[70] - I. Russel, C.M. Colebourn, P- Vitiello, "A Comparison of Backpropagation and ART via Pattern Recognition", Journal of Intelligent Systems, Vol. 7, Number 4, 1997.

[71] - J.C. Russel, "The Image Processing Handbook", CRC Press, 1999.

[72] - W.S. Sarle, "Neural Nets FAQ", ftp://ftp.sas.com/pub/neural/FAQ.html.

[73] - G. Sebastiani, S. Stramaglia, "A Bayesian Approach for the Median Filter in Image Processing", Signal Processing, Vol. 62, Number 3, 1997, pp. 303-309.

[74] - N.T. Siebel, "People Tracking for Visual Surveillance", University of Reading, http://www.siebel-research.de/people_tracking/.

[75] - R. Smits, L. Ten Bosch, "The Single Layer Perceptron as a Model of Human Categorisation Behaviour", http://www.phon.ucl.ac.uk/home/shl9/roel2a/smits2a.htm.

[76] - J. Smokelin, "Wavelet Feature Extraction for Image Pattern Recognition", Proceedings of SPIE, Vol 2751, 1996, pp. 110-121.

[77] - V. Srinivasan, P. Bhatia, S.H. Ong, "Edge Detection using a Neural Network", Pattern Recognition Vol 27, No. 12, 1994, pp. 1653-1662.

[78] - A.D. Stoyenko, P.A. Laplante, "Real-Time Imaging, Theory, Techniques and Applications", IEEE Press, Piscataway, 1996.

[79] - T.Sziranyi, "Video Understanding and Indexing for Surveillance: Image Perception, Quality and Understanding", ERCIM News, No. 55, Oct. 2003.

[80] - L. Tarassenko, "A Guide to Neural Computing Applications", Arnold Press, 1998.

[81] - S.Tate, Y.Takefuji, "Video Based Human Shape Detection by Deformable Templates and Neural Networks", Sixth International Conference on Knowledge-Based Intelligent Information Engineering Systems and Allied Technologies, Sept. 2002, pp. 280-285.

[82] - G. Tsai, A. Chiang, T. Yang, C. Lai, W. Wang, C. Liu, "Video Tracking and Recognition System", http://www.iii.org.tw/special/article/VideoTracking.htm, 2000.

[83] - M. Van Buren, "Metrics for Architects, Designers and Builders", Publisher: van Nostrand Reinhold, 1983.

[84] - P.Viola, M.Jones, D.Snow, "Detecting Pedestrians using Patterns of Motion and Appearance", Ninth IEEE International Conference on Computer Vision, Vol. 2, 2003, pp. 734-741.

[85] - VSAM, "Video Surveillance and Monitoring", 2000 Carnegie Mellon University, Robotics Institute , http://www-2.cs.cmu.edu/~vsam/vsamhome.html.

[86] - B. Walczak, B. Van den Bogaert, D.L. Massart, "Application of Wavelet Packet Transform in Pattern Recognition of Near-IR Data", Analytical Chemistry, Vol 68, May 1996, pp.1742-1747.

[87] - O. Weissmann, Z. Pollack, "The Perceptron", http://www.cs.bgu.ac.il/~omri/Perceptron/.

[88] - L. Weygang, N.C. Dasilva, "Implementation of Parallel Self Organising Map to the Classification of Image", Proceedings of SPIE - The International Society for OpticalEngineering, Vol 3722., 1999.

[89] - R. Winn Harding, "Neural Networks Target Security and Surveillance", Image Processing Europe, May/June 2000.

[90] - T. Wogelsong, T. Zarnowski, J. Zarnowski, "Inexpensive Image Sensors Challenge CCD Supremacy", Photonics Spectra, May 2000, pp. 188-192.

[91] – V. Bockaert, "The 123 of digital imaging Interactive Learning Suite", www.123di.com.

[92] - L.Zhao, C.Thorpe, "Stereo and Neural Network-based Pedestrian Detection", IEEE Transactions on Intelligent Transportation Systems, Vol. 1, No. 3, Sept. 2000, pp. 148-154.

[93] - E-Frontier – www.e-frontier.com.

[94] – Adobe – www.adobe.com.

[95] – Creative Labs – www.creative.com.

[96] – Autodesk – www.autodesk.com.

[97] – G.A. Miller, "The Psychological Review", Vol.63, 1956, pp.81-97.

[98] – National Communications System Technology and Standards Division – FED-STD-1037C, August 1996.

[99] – T. Kopert, "CCTV Surveillance System, Technology in Transition", White Paper, Array Microsystems Inc., 1997.

# List of Abbreviations

| | |
|---|---|
| **AI** | **A**rtificial **I**ntelligence |
| **ANN** | **A**rtificial **N**eural **N**etwork |
| **CCD** | **C**harge **C**oupled **D**evice, a particular form of sensor used in most digital cameras |
| **CMOS** | **C**harged **M**etal **O**xide **S**emiconductor. A newer form of light sensor, cheaper to produce than CCDs but more susceptible to noise in moving images |
| **LED** | **L**ight **E**mitting **D**iode |
| **MLP** | **M**ulti **L**ayer **P**erceptron. A popular form of neural network |
| **LVQ** | **L**earned **V**ector **Q**uantisation |
| **OCR** | **O**ptical **C**haracter **R**ecognition |
| **PIR** | **P**assive **I**nfra **R**ed. PIR sensors are used to detect the emission or reflectancy of Infra Red wavelengths within a certain off off and object or person |
| **PTZ** | **P**an **T**ilt **Z**oom. This refers mostly to cameras whose motion is remotely controlled. |
| **RBF** | **R**adial **B**asis **F**unction |
| **SOM** | **S**elf **O**rganising **M**ap |

# Used Terms

| | |
|---|---|
| **Cycle** | When training a network, one cycle is taken as a pass of one data line through the network. |
| **Epoch** | When training a neural network, one epoch is taken as one pass of the entire data set through the network. |
| **False Negative** | When a classifier defines a valid object as being non-valid. |
| **False Positive** | When a classifier defines a non-valid object as being valid. |
| **Noise** | Unwanted data in a data set which may lead to corruption of wanted data by its presence. |
| **Target** | An object which is being looked for. |

# 1 - Introduction and hypothesis

*First say to yourself what you would be; and then do what you have to do. - Epictetus*

Spatial detection and perimeter monitoring have progressed tremendously over the past two decades. Vibration sensors have been dramatically improved in their sensitivity and targeted applications. The PIR (Passive Infra Red) sensor has also matured and become a cheap yet reliable component of most motion detection systems.

However, as security and surveillance systems have dropped in price and become available to the individual home owner, the shortcomings of these sensors have also been highlighted. Whilst being rugged items needing little or no maintenance, they can be highly sensitive as to their mounting location, and specialist knowledge is still required to provide optimal performance.. False alarms triggered by PIR's viewing moving objects behind windows or radiator heat emissions have become commonplace. Conversely, due to their construction in vertical sections, a PIR can be fooled by staying within one of its detection bands whilst approaching the system. Intruders have also developed methods to prevent detection such as wearing heat absorbent materials, or simply wearing plastic bin bags over their heads.

Now, with increasingly accurate manufacturing technology, digital cameras are starting to take hold of the security market. Surveillance cameras are being installed on every street corner (quite literally in certain towns), and the public is becoming used to this trend as being an inevitable, if not entirely acceptable part, of modern urban life. These cameras are not however, being used to their full capabilities, whether this is due to the limitations of the systems they are replacing, or simply to their relative newness. Most systems indeed still rely on a human operator, using the camera purely as a remote eye without any further processing of any kind. Certain features such as night visibility through limited infrared illumination are present, but these were already standard features of PIR sensors.

The intention of this study is to develop a camera based surveillance system which could be fully autonomous, i.e. no human operator would be required without limiting the system capabilities. The camera unit itself must be able to replace not only the eye but also the brain of the human operator, and should be able to process the information which it is gathering, raising an alert when a human being enters the image frame. This would provide the platform for an advanced surveillance system which could be used in a number of situations without having to rely on the less than adequate human concentration spans which plague all current camera based systems.

Considering the implementation of modern surveillance systems, their requirements are tending towards an ever increasing resolution, linked to such functional specifications such as installation tolerant operation, automatic evaluation of the observed environment and a minimum error output.

These requirements raise a number of questions:

1. Is it possible to develop a system capable of processing the output from an industry-standard camera in order to identify the presence of a person or multiple persons within the image ?
2. Can such a process be developed to allow for real-time or close to real-time (i.e.: a few frames per second) analysis ?

3. Can such a system be developed in such a manner as to be independent of the camera installation location and method ?
4. Can such a system be independent of the final system operator ?

These could be combined into the single question:

***Can a machine autonomously recognise a person, rapidly, reliably, regardless of the system environment and image noise, using the lowest resources possible whilst maintaining the highest level of accuracy possible ?***

The following chapters are dedicated to providing the framework for a solution to this question, by employing a combination of analysis methods.

This thesis shall deal with concept of introducing dynamic elements into the image analysis process in order to allow for a close to real-time evaluation of the images streamed from an standard industrial camera.

It shall examine the merits of traditional image analysis processes as compared to the use of "intelligent" neural network-based solutions, in an effort to identify the best possible solution to the question posed.

This study shall be structured accordingly:

- Overview of environmental sensors, concentrating on the use of

cameras in detection.

- Summary of image analysis techniques including the use of various neural architectures applied to image processing problems.
- Definition of the problem at hand, resulting in a mathematical desription of the basic requirements.
- Initial proposal, based on simplistic methods and evaluating the potential of the chosen approach.
- In depth development of the analysis framework, leading up to a final system proposal.
- Final comments on further developments.

This thesis shall expand on the target of developing an accurate detection of unidentified persons (i.e.: not taken from a previously collected database) using a single camera with minimal resources and running in close to real-time.

This shall be achieved through the use of a dynamic pre-processing stage which shall be able to adapt itself to each incoming image without any need for previous knowledge of either the environment nor the type of the implementation.

A further stage, based on a neural network or combination of neural networks shall be employed to output a final decision probablility of the image observed.

# 2 - Literature Review

*An undefined problem has an infinite number of solutions - Robert Humphrey*

A study conducted by the ERA has determined that for passive IR sensors up to 90% of all the alarms are erroneous and half of these are due to some kind of operator error, whilst the rest are due to environmental factors such as 'stray' rays of sun or heaters in a room switching on [38, *41, 45, 85*].

Various methods have been suggested to decrease the number of false alarms but these have mostly been deemed prohibitively expensive (These normally involve coupling the PIR with a multitude of other sensors or arranging for a pure IR environment where spurious noise sources are blocked or filtered) [*02, 41, 24*].

The following is a presentation and discussion of current methods and industrial trends.

## 2.1 - Why do we need sensors ?

Sensors are used in so many different applications, it would be impossible to classify them all. We are, however, primarily interested in motion sensors and heat sensors, i.e., systems which can detect the presence of a human being.

These types of sensors are used in many applications to provide purely monitoring information, early warning signals and anti-theft or anti-entry protection.

The use of artificial sensors allows the introduction of remote surveillance where the manpower required to monitor a site would be too expensive or might in itself represent a security risk (bank vaults, jewel display areas...)[*26*]. There is now also the paradox of incompatible surveillance systems. For example, if a certain area is monitored by PIR's, a human operator could no longer patrol that area without setting off the alarm, without first having access to some bypass switch for the sensor concerned. This in itself then represents a security risk, as any reasonably well equipped intruder could then also bypass the system. This then negates the entire point of a non-human monitoring setup, leading to an increasingly complex layout of overlapping sensors, each protecting its neighbours from being tampered with, which in turn leads to spiralling installation costs and also a reduction in the overall

effectiveness of the system. Placement of individual sensors then becomes increasingly critical to the correct operation of the entire system and the sheer level of complexity makes errors and malfunctions not only more likely but also more disastrous in their symptoms [*02, 24*].

In general the currently available commercial intruder detection systems rely on three main types of sensors each with its own advantages and disadvantages given the current implementations.

## 2.2 - *Types of Sensors*

The three main sensor types are

# PIR, sensors (Passive Infra-Red)

# Vibration sensors

# Cameras

### 2.2.1 - PIR Sensors

Up to present, most intruder detection has been carried out using a single PIR sensor or an array of sensors, using defined sectors and zones to achieve reliable detection of possible intruders and to minimize false alarm occurrences.

A PIR sensor is most commonly a single infrared sensitive receiver with a vertically segmented lens mounted to the front of the unit to provide a number of vertically defined zones over the entire field of view of the sensor, which is generally quite wide (approx. 120 degrees)[*02, 41*]. The overall sensitivity of the sensor is determined by the number of these vertical segments, as the sensor depends on identifying a change of state from one segment to another in order to trigger a successful detection. PIR's are therefore intrinsically motion detectors, but this motion must occur within the low infrared range of the sensor used. This is both a strength and a weakness for this type of sensor, as they can operate both during the day and during the night without need for extra illumination, but will be very sensitive to heat changes such as radiators activating or gas heating vents [*02, 45*].

The limitations of such a system are apparent mostly in cold conditions, where a car might not be detected, but a person will be. As such systems are often used as courtesy lights in house entrances, this can lead to much waving and jumping around, as a person vainly tries to activate the sensor.

Due to the segmented lens design, these sensors can also be completely oblivious to a person moving straight towards them [59].As long as the movement is limited to a single one of the lens segments no motion will be detected, which introduces the tricky problem of sensor placement.

Due to the low production cost of these sensors, they are often offered as home assembly kits for a number of applications ranging from house alarms to simple light activators . They do however require a certain amount of knowledge on the placement techniques, and many false alarms are triggered by sensors being allowed to 'look' through windows and picking up legitimate movement outside of the intended surveillance zone, sensors pointing straight at heating elements which obviously create quite marked infrared signatures when in operation, or sensors being activated by house pets wandering around [*38, 59*].

Although modern PIR sensors are becoming more selective and more adjustable, their intrinsic features make them useful only when coupled with other types of sensors, or when used in conjunction with a human operator to actually determine the cause of activation, and distinguish between real and false alarm situations. They are also highly effective when used indoors in areas normally void of any movement and where discrimination as to the source of motion is not required.

The trend in current practice has been to use PIR sensors as a primary alarm. This would then activate a camera, whose image is transmitted back to an operator for further analysis [*93, 38, 15, 94*]. The advantage of this type of setup is manifold: PIR sensors are inexpensive, fairly rugged and require minimal maintenance.

Once a sensor is triggered, a PTZ (Pan Tilt Zoom) camera can be activated by a controller to determine the exact nature of the alarm. This helps to minimize costs and reduce operator boredom as the concentration time required is limited to the PIR activation period only.

Lately, PIR sensors are being assembled into the same housing as a small camera [*15, 11*], whereby the PIR sensor, using its wider field of view acts as a pre-alarm for the camera, switching on the camera before anything actually enters the resultant image. This type of assembly obviously suffers from all the typical PIR faults and can be tricked in many ways. Intruders have been known to wear heat absorbent clothing to fool PIR driven systems, or more simply, to wear black plastic bags over their bodies, which minimise body heat dissipation and can thus present too low an infrared signature for successful detection by the PIR sensor [*41, 75, 85*].

## 2.2.2 - Vibration Sensors

Vibration sensors are similar to PIR's in that most of them do not incorporate any form of intelligent data processing. The units are manufactured to be mounted in a number of locations, from underground to mounted on fence tension wires or on doors and windows, and have normally a number of adjustable settings such

as vibration intensity sensitivity and maybe a small time delay circuit. Once a certain level of vibration above the user determined threshold is encountered, the unit activates a switch which could trigger an alarm or floodlighting.

Obviously, the difficulty with this type of approach is in actually determining what could represent a 'valid' vibration (i.e., burglar climbing over the fence) or what might simply be a spurious noise effect (a cat climbing over the fence). Most systems are adjusted to give a higher incidence of false alarms so as not to miss any marginal cases. What the psychological effects of this policy on the human operator are, who will slowly become less responsive as more and more alerts are deemed non-valid, is not really the topic of this study, but is worth taking into account when weighing up the pros and cons of each approach [*38*].

Due to their very non-discriminatory nature (vibration sensors will report motion, not the type or intensity of the motion and its possible cause), vibration sensors are rarely used on their own in industrial applications, normally forming the first line of warning in on overall surveillance system. They have however been used recently in home security applications as primary sensors on doors and principally on windows to report illicit entry. Any further processing of the signals received from such a unit will be dependent on a high level of adjustability to accommodate for different mounting positions: an intruder coming in through a door

and one coming in through a window are likely to give rise to very different vibration signals, although basically the same target in each case has been detected.

The consumer intended systems described above are fairly small and simple to install, minimising the initial costs as well as further maintenance. The price for this versatility is however a fairly high risk of false alarms and a very low alarm resolution when one does occur. This in turn then becomes counter-productive, as emergency services start putting limits on the acceptable number of alarms from households, before limiting their response or imposing financial penalties on the home concerned. Vibration sensors perform excellently in the context they were created for, but cannot be relied on to provide accurate information on the presence of intruders in a property.

### 2.2.3 - Cameras

In recent years, a sharp increase in demand coupled with improvements in manufacturing technology have led to a dramatic drop in prices of conventional CCD (Charged Coupled Device) cameras, and their implementations now range across an entire spectrum from high quality medical inspection applications through consumer camcorders, door entry systems and, of course,

surveillance systems. CCD cameras are fairly small (average board size including lens and processing electronics is about 50x40mm) [*15, 57*], can take a number of different lenses from fish-eye to telephoto (normally with the now industry standard C-mount) and require little or no extra support hardware. Apart from a power supply, most cameras can plug straight into a conventional video recorder or any television set with a video input.
This trend started around the late 1980's. Prior to this date, due to the very high initial cost of hardware, any camera surveillance systems were mostly restricted to larger companies or government projects, where constant surveillance is required. In these cases, the cameras were operating as a simple backup for the security personnel already employed.

By their very nature, cameras require higher levels of maintenance than other, more conventional sensors: lenses need to be kept clean, dry and free from obstructions. Due to the magnifying property of most lenses, cameras are fairly sensitive to vibrations, which dictate where and how they should be installed. An operator forced to watch a constantly vibrating image will rapidly develop eye strain and cease to monitor the camera in question. Latterly, vibration compensators have been developed, and although these can assist tremendously in this aspect, they are currently, both in their mechanical (generally gyroscopic) and electronic versions, too expensive to be incorporated onto every camera in an entire system. As opposed to PIR's or vibration sensors, a camera cannot,

on its own, actually raise an alarm when an intruder appears, but can provide the human operator with very precise information as to the nature of the situation and the potential level of threat present. This does however still preserve the requirement for a human operator [*85*].

Due to the falling price of hardware, it does become easier to monitor a large site, as a single operator can manage a number of cameras without the need for constant patrols, therefore the system manager could benefit through lower personnel costs. A smaller site will probably not have this advantage though, and this has partially lead to a switch from active surveillance to passive surveillance.

A further disadvantage linked to the use of cameras is that of sheer information overload. Whilst PIRs or vibration sensors have a purely digital mode of operation (either on or off)[*41*] a camera is more complex in the data which it feeds back. The operator must be constantly watching the image to evaluate any potential threats. The simple fact of having to closely observe an unchanging image over long periods of time lead to boredom and a loss of concentration. A number of studies have revealed the maximum observation span of a single image to be about 20 minutes, after which time the operator will not even notice a person walking through the camera's field of view [*26, 61*].

This is obviously not an ideal solution, as the expensive camera

system becomes in effect counter productive. This can become particularly acute on larger systems involving a number of cameras and display units, especially when the cameras used are static. A number of solutions have been devised to counter this human weakness. The simplest is to introduce another operator. A slightly more advanced method is the introduction of the multiplexer, which will allow the displayed image to cycle through all the cameras on site, with an adjustable dwell time on each. Whilst this might seem attractive, it can present the risk of an operator missing vital information, as not all camera images are visible simultaneously. This can be countered through the use of a split screen display, for generally up to 4 separate input sources, which can also be set to rotate in a number of ways [*99*].

These methods certainly present ever changing information, but can also have the undesired effect of actually breaking the operator's concentration span as the image switches from one camera to another. This leaves the operator unable to correctly process the information presented as the source of the new image might remain unclear for a few seconds, which could easily be the entire dwell time for that particular camera [*45*]. Although the actual psychological effects of these systems have not received much attention, they are a factor which we cannot afford to omit when considering their suitability for each location, given the degree of surveillance required in each case.

A slightly more suitable solution is the use of operator controlled PTZ units (Pan Tilt Zoom). Whilst simultaneously reducing the amount of cameras required for a given site, they also allow the operator full control over the received image. Thus, the operator maintains his or her situational awareness while also keeping a higher and more effective level of concentration as the image presented is not static . This obviously also allows for more detailed examination of features through the camera's zoom feature. The danger with such a system is however similar to static cameras with a cycling display, that the operator might miss valuable information whilst examining a different site location. A PTZ unit also has the disadvantage of clearly indicating the actual direction of observation of the camera, a feature which has been reduced through the use of dome housings or panoramic cameras [*15, 57*].

These are all purely camera based systems.

What is actually required in all of these situations is for the operator to be notified only when an incident occurs, and to then have the camera available as a purely observational tool. Such a system was initially devised through a somewhat inelegant but effective attachment intended to be mounted on the image display unit . This array of light sensors would monitor the image and sound an alert when the preset light threshold had been exceeded, indicating that some type of movement had occurred in the observed image.

The most recent incarnations of this system consist of a combination of PIR and camera within a single housing [*15, 11, 57, 94*]. The PIR, generally having a wider field of view than the camera can be used to warn of movement, whereupon it will activate the camera to enable the operator to determine more accurately the cause of the alarm. Conventional PIR sensors are, however, notoriously susceptible to false alarms, which can lead to the operator growing bored of examining invalid alarms and ignoring valid threats when they do occur - a definite case of cry-wolf. It has been estimated that the proportion of false alarms for PIR sensors is often over 90%, of which half can be attributed to operator error and environmental factors [*67, 65, 24, 26*]. Obviously, the larger the system, the more acute the problem becomes.

The unreliability of these systems, linked to the need for cheap surveillance for small businesses and home owners has led to an increase in passive surveillance. This is a corrective solution, as it cannot help prevent a crime (except perhaps through the psychological deterrent of warning signs) but can help analyse a crime once it has happened. Generally, this involves a camera linked to a time lapse recorder. Such a system will typically operate 24 hours a day, fitting either a full day or half a day onto a single regular 4 hour VHS cassette. Whilst this allows the system to be financially available to most people, the unfortunate side effect is that the resultant images are near to unusable from the amount of time compression involved, and also through operator laziness in

not replacing worn tapes [*95*]. A slightly more advanced system is to use the PIR driven camera and recorder to only capture actual incidents, removing the need for time compression at night time when premises are normally closed to public access. Recent developments are linking such systems to digital recording methods allowing for higher resolution images to be stored, in addition to reduced wear on the actual recording medium.

In addition to the conventional CCD type cameras, recent developments are also making CMOS (Charged Metal Oxide Semiconductor) cameras available to the small business and home market [*65*] . CMOS type cameras are nearly fully integrated into a single chip, allowing for yet further reductions in size and power consumption, whilst retaining image quality. Having a nearly fully integrated construction, with the actual image sensor itself delivering the digital image data (a CCD type relies on extra circuitry to convert its analogue output), this really is at the point of an entire camera on a single chip. It also means that the signal which is obtained from such a camera is also a lot cleaner as the post-processing phase has been minimized, reducing the risk of signal corruption.

Of more interest however, is the nature of the signal which is obtained from a CMOS type camera. Whilst a CCD type will scan the entire image area before transmitting the data as a single group, CMOS cameras use a line scan principle. Each image line is scanned and transmitted immediately, resulting in a faster data throughput .

This does however also have the undesirable effect of somewhat blurring any motion, and thus presenting a distorted image which can be somewhat misleading if any fine analysis is being carried out on the images obtained from the sensor [*90*]. On a more positive side, due to the more integrated nature of the entire camera, they are less likely to be susceptible to the effect of blooming. A regular CCD camera pointing straight at a relatively bright light source such as a car headlight will output an image with a marked halo effect around the light source, effectively resulting in what can be a quite important loss of detail in the area concerned . The only preventative measure for this is the auto-iris feature now present on most cameras, which effectively simply lowers the light level in the entire image. A CMOS camera can automatically reduce or enhance the response of individual sensor cells, resulting in a much clearer and sharper image.

Both types of cameras do however, also rely on a number of common features.

Black and white cameras (which generally output at least 256 shades of grey) offer a very useful low infrared response. Pure IR, as used in thermal imaging sensors (which are financially beyond the scope of this study), is located around 800 nanometres. This type of response would be ideal, as any living object would be drastically identified in any image [*Appendix G*]. The IR response of these small cameras lies between 700nm and 750nm, where 720nm is approximately the range limit for the human eye . This

response can be slightly enhanced through the use of IR illumination, and many cameras are now shipped with limited IR in the form of a few IR LED's mounted around the lens, providing the camera with a blind range of a few meters. Whilst these have no benefit whatsoever in a well lit area, they can be very useful at night. Obviously, this limited IR capacity could be enhanced through the use of regular IR spotlights, which are now commonly mounted on the same post as the camera itself. Depending on their power, these can increase the blind range of the camera to about 10m.

This IR response is however, limited to monochrome cameras. A colour camera works on the principle of having a number of layers of sensors, each sensitive to a certain light bandwidth in the ranges of blue, green and red. To obtain a satisfactory resultant image, a red filter has to be used (similar to black and white photography where a red filter has a sharpening effect, by cutting out a large amount of the IR haze) [*Appendix G*] and this obviously is also most effective at filtering out any IR element in the received image. We therefore have to decide between a good night response (lying at minimum values of approx. 1.1lux) but slightly less definition through the loss of the colour information, or an image very rich in information but with a very poor night response.

The second common and limiting feature of all cameras is the choice of lens used. These range from simple pinhole constructions to full telephoto or fish-eye lenses. The only difference with

conventional photography being in the absence of zoom lenses, this operation generally relying on the fact that the displayed image has a lower resolution than the actual image sensor, allowing a virtual zoom to be carried out by simply extracting the image information only from a limited area of the total sensor. A pinhole lens, admittedly the cheapest type of lens, is simply a glass covered hole mounted over the sensor. This might be the smallest type of lens available, lending itself to use in spy applications, but has also a fairly limited useful range lying between 1m and 3m approx., with image quality dropping off severely outside of this, and severe distortions appearing at the image edges.

Regular optical lenses are also available, with f-stop values ranging between about f2 and f8. The f-stop value directly affects the overall sharpness of an image, dictating how much light will enter the lens in a given time period. High f-stops such as f22 or f30 guarantee an extremely deep sharp focus region, but also severely limit the amount of light reaching the actual image sensor. A lower f-stop value such as f3 will let in more light but will also cut down the actual focus area to a few metres, with the actual focal point lying about 1/3 of the way into this range. These values however also have to be considered in conjunction with the actual focal length of the lens concerned. The focal length represents the distance travelled by the light internally in the lens from the front element to the rear element, or to the surface of the actual image sensor. The more complex the internal lenses in a single lens

become, the more light they will tend to absorb and dissipate, thus, a telephoto lens with a focal length of 300mm will be absorbing about 4 or 5 times the amount of light that a panoramic lens with a focal length of 35mm would. The net result is that longer lenses generally tend to be " slower" lenses, i.e., they require a longer time period to achieve the same exposure as a shorter lens would, meaning that their minimum f-stop value is effectively raised [*91*].

Whilst a conventional still camera will compensate for this by simply taking a longer exposure, CCD and CMOS cameras generally do not have this luxury, as the user will be requiring a frame rate from anywhere between 10 and 30 frames per second. The only actual mechanism available to these is to adjust the sensitivity of the actual sensor, which obviously has its own minimum boundary which is governed by the material and the methods of manufacture. Whilst telephoto lenses thus absorb more light, they also have the tendency to distort shapes, most noticeable in parallel lines which will be viewed as strongly converging or bowed [*51*]. The closest lens to matching the way we view objects through the human eye is one with a focal length between 35mm and 50mm, although for practical purposes, a value nearer 100mm is normally preferred to enhance the actual surveillance range. Wide area surveillance can also be carried out with a wide angle lens or a fish-eye lens (from 15mm to 24mm) which can have a field of view of nearly 180 degrees, albeit again at the expense of severe distortions for

anything at the edges of the image or too close to the actual lens .

## 2.3 - Modes of Failure

It is first necessary to define what is going to be understood as a failure in these conditions. In this study, when a failure is referred to, it is taken to be indicating either a false alarm (alarm status given although no intruder is present) or a false negative (no alarm status raised, although an intruder is present). We will never be considering actual material or hardware failure in the conventional sense of something being mechanically 'broken'.

Listing all possible failure modes for the various types of sensors would be a lengthy and quite useless exercise. What must be realised is that each of the above described types of sensors has been designed for a specific range of applications, in which they perform very well. Thus, vibration sensors are primarily employed to be mounted on fences or windows to indicate some type of illicit entry.

As the demand for security and surveillance methods is growing and expanding from the sphere of large companies to smaller businesses and private individuals [*65*], these conventional sensors are being pushed to their operational limits, and integrated into

systems where the scope for incorrect adjustment and misinterpretation of the return signals is quite large. Thus, a wrongly adjusted window-mounted vibration sensor could easily be set off by a bird flying into the window. In a private home system, this might be the one and only type of sensor used, which could easily lead to a general alarm being sounded (where in an industrial context it would simply have lead to an extra patrol by the already present system operator), leading to inconvenience for the emergency services and a severe penalty to the concerned home owner. PIR sensors are, for example, notoriously sensitive to their mounting location. Most home alarm kits are nowadays based on some type of PIR system, and whilst a professional company can be called upon to carry out the installation process, many home owners will carry out this process themselves out of financial considerations [*38, 15*]. Many sensors are thus inconveniently placed, resulting in quite a high number of false alarms. Camera based systems are generally still relying on the presence of an alarm operator, and are therefore maybe less prone to false alarms, given their more passive nature as compared to other conventional sensor types.

It is therefore quite clear that most system failures have two main sources:

- System designers making use of inadequate sensor types to achieve more results at a cheaper overall cost.
- System installers not being aware of the limitations and operational parameters of the type of sensor they are

dealing with, resulting in inadequate sensor placement and use.

These two main errors result in two main types of failures, false positives and false negatives, a false positive being an indicated alarm state with no apparent reason, and a false negative being an actual alarm state being ignored and missed. For PIR systems, the first case amount to over 90% of all alarm states throughout the UK [26], a staggering value ! No figure is available as to the number of false negatives, as this is a rather difficult value to assess accurately.

In human operated systems, (either through patrolling or remote surveillance with the help of cameras) we see an inverted condition. Here, false positives are quite rare, but false negatives can be quite common, less so where an actual patrol system is in place. The reason for these occurring now falls back onto the human element and the type of surveillance system in place. Whilst the human element is very useful at filtering out any false positives, false negatives can occur through such simple factors as boredom and limited concentration spans. As mentioned in the previous section, studies have determined the maximum effective concentration time in front of an unchanging image to be approximately 20 minutes [95]. Not a great amount when we take into consideration the entire shift period of six to eight hours.

## 2.4 - Possible Solutions

The problem which is presented to the modern surveillance system designer is based on a number of factors:

- Final system cost.
- Ease of installation and operation.
- Overall system performance.

### 2.4.1 - Final System Cost

This depends very much on the targeted market for the system in question. This will be less critical for large industrial installations, where more focus is put on system performance. It is however an important factor where the end user is intended to be small businesses or home owners.

### 2.4.2 - Ease of Installation and Operation

Again, very dependent on the target market. Professional systems will be relying on trained personnel as far as the system installation is concerned, less so in the day to day operational duties. Home systems will be rarely installed by trained personnel, and will practically never be operated by anyone other than the property owner, who will have little or no training in the system. This lays

out the prerequisite for a system with simple and straightforward installation and running phases. Ease of operation can be quite easily achieved through satisfactory ergonomical considerations at the design stage of the product. Features such as visual displays and organised menu systems can help even a novice to successfully operate what might initially seem quite a complex setup. The point where a system's actual performance can easily be compromised is during actual installation. Actual sensor placement, cable routing, control panel placement, all these are highly critical and can easily negate any technological advances in sensors and system integration if not carried out properly [*38*].

### 2.4.3 - Overall System Performance

System performance will be dictated by a number of criteria, such as correct system installation (see 4.4.2) and correct choice and combination of sensors for the given situation. For example, a PIR based system would be quite inadequate in a glass walled house, where the PIR would constantly be detecting motion occurring outside the actual intended surveillance area. Sensor combinations can also be utilised to enhance the overall situational awareness provided by a system. For example, a PIR sensor mounted at a front door would be able to signal the arrival of something moving within its surveillance range. A camera on its own would also

provide the same information, but would require the user to be constantly monitoring the resultant image to actually determine when the person or object arrived. However, a PIR might be linked up to only activate the camera when it detects movement, thus providing not only temporal information, but also allowing the user to examine the resultant image and determine exactly what was the cause of the alarm.

This approach of multiple sensor combination has been adopted by a number of manufacturers [*93, 15, 57*], and although it might not directly reduce the number of false positive alarm situations, it can help in this context: many home alarms are directly linked to the emergency services, either through a dedicated line or via conventional telephone connection, to allow a faster response time. When a camera is linked into the system, we now have an extra layer of information which can also be forwarded in conjunction with the alarm signal. This has been implemented using either direct digital image transfer or by relying on facsimile signals. This then presents the emergency services with enough information to determine whether they are dealing with a valid alarm situation or not [*94, 95*]. This is certainly an improvement to actually having to investigate every alarm state on site.

## 2.5 - Sensor Summary

The security industry is currently using technologies which were partially developed over 2 decades ago, with small improvements in response quality. New sensors in the form of advanced cameras are for the most part not being used to their full potential, and the industry as a whole is still struggling with the enormous problem of false alarm conditions. Whilst electronic sensors can be adjusted to pick up most potential threats, they do not possess the intrinsic discriminatory judgement which a human operator has.

To minimise costs and make systems available to as large a market as possible, sacrifices are being made in the area of alarm resolution. Systems are plagued by installation and operator errors, and the only really reliable processes involve the expense and potential security threat of constant operator presence, as a complement and backup to the various electronic sensors.

## *2.6 - Analysis Techniques*

Any successful signal interpretation relies on having the correct analysis methods and techniques available, otherwise the data obtained might as well be ignored. A successful analysis also depends on having achieved a correct understanding of the data itself, and the elements of which it is comprised. Many analysis tools have been developed, which are now regarded as somewhat of a standard approach, given a certain type of data.

Before even starting with a certain type of analysis, we also have to be confident as to which features we will be attempting to study.

The various techniques which are available to effectively dissect and analyse a data stream could be broadly grouped into two categories, which we will designate as Conventional and Smart.

### 2.6.1 - Conventional Techniques

Conventional analysis techniques involve statistical methods such as PCA(Principle Component Analysis) and regression analysis. These tend to be based on mathematical transformations which will ultimately lead to attempting to match the data to as simple as possible a linear function. Non-linear functions, due to their complexity and high level of magnitude, are not normally

considered, especially as these go beyond the scope of what is readily understandable or what can be possibly visualised by the human brain. Although adept at identifying trends and patterns, the human brain will rarely cope when more than 5 or 6 interdependent dimensions are involved[*97*]. Whilst solutions obtained using conventional techniques might lead to a certain amount of success, they do tend to break down as soon as the data presented to them starts to deviate in any way or form from the development data set. One of the main drawbacks of conventional analysis however could be their sheer complexity. These tasks will generally be carried out by trained statisticians, who will probably not have any concept of the source of the data and what it really represents. This effectively splits the analysis phase between the engineer or end user who will be providing the data and the person who will be analysing it, creating the possibility for comprehension or communication errors.

## 2.6.2 - Smart Techniques

The concept of Smart Computing started in the early 1940's with the advent of the first digital computer systems and the first attempts at modelling the abstraction layer (or cognitive abilities) of the human brain. These were trying to develop generalised models of the biological synapses and of the overall reasoning methodology [*72*].

McCulloch and Pitts (1943) developed the formal concept of MLP Neurons, which form the basis of most networks today. These initial networks were quite limited, and in 1949, D.O. Hebb developed the first learning rule, which increases the adaptability of the network.

Following this quite successful start, many different attempts were made in different directions and levels of attempted simulation. The main breakthrough came with Frank Rosenblatt and his proposed model of the Perceptron for use in classification problems (1958) [*87*]. Widrow and Hoff developed the Adaline (Adaptive Linear Element) model (1960) in an attempt to introduce a level of error feedback into the system, this was then improved in their later developed Madaline Complex (Multiple Adalines), which effectively overcame the weaknesses of the simple Perceptron.

Development continued until the publication of "Perceptrons" by Minsky and Pappert, which raised awareness to the inability of the Perceptron to resolve non-linear problems (i.e., the XOR problem). This effectively stopped public development of neural models until the early 1980's, although some notable models did still appear in that time, notably Teuvo Kohonen's Self Organising Maps[*23*] and Stephen Grossberg's competitive learning models and Adaptive Resonance Theory, which was widely used and modified in later systems.

Neural computing then experienced a sharp revival, mainly due to John Hopfield's contribution in the form of an auto-associative system for pattern compression and reconstruction. In 1985, David Rumelhardt and Geoffrey Hinton introduced the concept of the Generalising Delta-Rule, which overcame the weaknesses of the Perceptron pointed out by Minsky and Pappert, and boosted renewed interest and development[*80*].

A number of factors now contribute to the growing success of neural computing. Firstly, the available processing power has dramatically increased in the last decade, allowing more complex models to be developed. Additionally, the basic ground rules for neural computing have been laid out and are now recognised worldwide. This allows for faster development as successful methods are improved on. As neural systems prove their commercial success, the amount of funding in the public domain has also increased, leading to the development of even more systems.

Artificial Intelligence has now become a commercial buzzword which is used, correctly or not, for a multitude of applications.

Whilst theoretically not so distant from standard statistical analysis, they rely mostly on providing non-linear solutions to a data field, thus providing a closer match to real world conditions. This issue of non-linearity is directly related to the problem data complexity: when considering a classification involving more than two different parameters, the mapping of the data to the possible classes is,

most probably, going to follow a non-linear separation, as each class might share given parameter ranges of other classes (i.e.:When classifying animals, all birds have wings, but a hummingbird has got a smaller wing range than a sparrow). The level of non-linearity will change according to the problem at hand and the correct data selection.

Neural Computing has now experienced a shift from an expected fully intelligent solution for autonomous machines and systems to practical tools and methods which can assist and complement existing technologies. These are appearing in everyday life, mostly as system assistants. One of the most widespread applications probably being OCR software (Optical Character Recognition) [*64, 13*]. Other applications are Facial or Iris recognition systems, stock exchange prediction assistants, machine condition monitoring or even the intelligent microwave by Sharp.

Modern network systems have a number of common features in their operational characteristics, which effectively define them. These are, amongst others:

- Robustness: Most networks function on a principle of common responsibility. Each neuron is contributing only a tiny part to the overall performance and output. If one neuron fails (through incorrect data for example), the system should be able to generalise the approximately expected value and still provide a

reasonable output. As more and more neurons fail, the system performance will gradually worsen without a sudden drop in output. This is referred to as graceful degradation[*98*].

- Parallel Structuring: Each neuron in the network could, theoretically, be a separate processor for computing processes. Whilst this is practically impossible due to the sheer number of interconnections required for such a system, it remains a basis for rapid and efficient data processing.

- Ability to learn from experience of past data. Due to the training methods, a network will adapt itself and "recognise" past data combinations or trends, which could be difficult or impossible for a human to visually recognise in a multidimensional data set.

- Generalisation: A networks capacity for generalisation depends on the training phase of the network and the type of data which it was presented with. As long as the training data is composed of an unbiased dataset comprising examples from all known data conditions which might be encountered in actual use, and as long as the network training process is stopped to prevent over-fitting, the network will should be able to classify similar data tendencies by fitting them to the nearest known similar data pattern encountered during training. Over-fitting is the direct opposite of a networks generalisation potential, and occurs when the training dataset is either too small or when the network is trained for too

many cycles on a single dataset. In either of these cases, the network will become over-optimised by attempting to match itself perfectly to the training data used. The network will then fail to classify previously unseen data.

- Relative computational efficiency once trained. Whilst requiring a fair amount of computing power and time during the training or learning phase, once established, data can be processed by a network quite rapidly, making them ideal for time-critical applications.

- Non-Linearity. Most networks are trained in a manner which allows for non-linear data mapping, closely reflecting the actual state of real data. This becomes increasingly critical as the number of data dimensions increases.

- Ease of use. Given appropriate training and correct output transformation, little or no knowledge of the actual field of analysis is required from the potential user.

In general, a network requires a learning phase before data analysis can actually occur. The term learning or training can be easily misunderstood. Contrary to human learning, where specific concepts and memories are stored, as well as general techniques, a neural network depends on processing as large as possible a data

set with a given number of dimensions to achieve some type of classification. This learning process can be split into two main categories:

- Supervised learning: In this case, the network is presented not only with the original input data, but for every data line is also given an expected result or output function. By adjusting internal values, the network will attempt, for each line of data, to recreate this given output value. Excellent in the case where the data parameters are well defined. This process does require a sufficient amount of data to prevent the network from simply learning to replicate the data set with which it has been trained.

- Unsupervised learning: The network is simply presented with the original data, and will attempt to recognise trends and groupings within the data, and to present these as definable areas within the data map. These areas will generally maintain the same dimensionality as the original input data. Once training is complete, these areas can then be manually labelled to differentiate the various data subgroupings.

In both cases, the choice of the training data and also the number of training cycles or epochs (number of times the entire data set will be passed through the network) are critical values. Insufficient data or over training will cause the network to effectively replicate

the training data set, losing the ability to generalise when presented with new data.

Typically of such a data driven system, networks are very good examples of "Rubbish In - Rubbish Out". A certain knowledge of the input data fields and their origins will contribute to more efficiently developed systems. Correct analysis of a finalised network can also help prevent quite embarrassing moments (US army tank recognition - trained to find sunshine, BR rail crossing doesn't like snow). Certain types of networks are also more resilient to non-relevant data than others [*51, 35, 20, 89*].

On the following pages, a number of network architectures which can be applied to pattern recognition and image processing shall be discussed, including sample applications based on these architectures and the advantages and disadvantages of each approach.

## 2.6.3 - Types of Networks and Intelligent Processes

The term Intelligent Process can be used to define any system capable of reaching a decision, given adequate data and a reasonable training phase. The nearest analogy which can be found is that of an expert in a given field. Early networks attempted to

mimic human intelligence by copying the logical thought processes which assist in the decision making process.

This introduces the concept of network architectures. Although most neural computing per se can be taken as adaptations of the work resulting in the first perceptrons, this has resulted in a number of varying architectures and internal processes which are more or less suited to different types of applications. Of interest in this case are those systems which could be used for image recognition or analysis.

Due to the large number of such architectures, only those most pertinent to this study shall be reviewed below. This will include architectures already successfully in use, as well as potential candidates for the task at hand.

This will not expand on systems used for face recognition or for database comparison two very common detection approaches which rely either on the location of a face, or on matching the currently found object with a previously recorded object placed in a database to which the detection system has access. These are much more limited applications, working on very strictly defined parameters with few degrees of change in their detection processes.

**1. Knowledge-based systems, or expert systems:**

Although not strictly a neural -based technique, this approach uses a stored knowledge base to analyse any data with which it is presented. The actual analysis is similar to a logical progression through the data, using defined rules and filters which form the

knowledge base of the system. This is an overall attempt to mimic the thought processes and patterns of a human expert facing a given problem. The knowledge base can be expanded on, within the parameters of its current subject without extensive reworking of the entire network. Additionally, the actual decision making processes are fully transparent, available to the end user, and not subject to empirical decisions but purely logically and rule led reasoning.

An expert system's weakness lies in the actual knowledge base itself. This must be provided with all possible conditions and rules to facilitate actual decision making. This is often done by simply referring to an actual human expert. The problem which presents itself is in the actual encoding of the thus obtained rules. The resulting system will also be highly domain specific, able to deal with its specific data but not able to extrapolate the knowledge to related fields, or generalise for missing data inputs or noisy data. The knowledge stored in the system will often be treated in a purely binary manner, providing a form of yes/no response but without making any allowance for any response with a more analogue dimension. This makes expert systems ideal where a pure access to knowledge is required, but does make them less appropriate for problem solving applications. Such systems have been successfully employed on surveillance networks which require a polling from a number of cameras. The drawback remains in the actual generation of the rules to be used – these must be sufficiently detailed to accurately describe the problem, whilst simultaneously remaining flexible enough to deal with any unexpected artefacts. This makes

this type of architecture ideal for a strictly controlled environment, where all parameters are known.

## 2. Perceptrons:

Following the discovery of the limitations of the simple perceptron, efforts were made to develop an architecture which would be capable of solving multi-dimensional, non-linear problems.

The Perceptron itself, developed in the 1960's by Frank Rosenblatt, was an attempt to model the structure of the human brain, where simple stimulations to a given number of nodes result in an output dependant on the internal state of each node.

The perceptron model is based on a learning process which adapts the system parameters according to the difference between the expected and the actual system output. This is repeated until the lowest possible energy state is reached, at which point the network may be presented with new data, and can be expected to output correct results. This state is reached when no further approximation of the network weights to match the expected output value(s) for given input data is possible.

Due to the actual architecture, the basic perceptron was rapidly found to be limited to solving fairly linear problems ("Perceptrons", Minsky and Pappert, 1969), which led many people to discard neural approaches entirely.

Due to the process of learning by reducing the output error against

the expected error, the Perceptron is classified as a supervised learning architecture.

The actual learning function can be summarised as shown in *fig.1.*

$$\Delta w_i = \eta x_i^p t^p$$

Where $\Delta W_i$ is the change to weight $w_i$,
$\eta$ is the learning rate,
$x_i^p$ is the network input for a given pattern $p$
and $t^p$ is the target value for an input data pattern $p$

*Fig.1:Perceptron Learning Function*

If the calculated output is correct (i.e.: $t=y$), no weight adjustment is made. If the output is incorrect ($t \neq y$), the weights $w_{old}$ are adjusted such that the output taking the new weights values into consideration is closer to the expected output *t.*

The actual network output is calculated as shown in *fig.2.*

$$y = f_h\left(\sum_{i=1}^{n} w_i x_i + w_0\right)$$

*Where* $y$ is the neuron output,
$x_i$ is a particular input pattern,
applied to weight $w_i$ of n weights,
$w_0$ is a bias weight
and $f_h$ is a limiting function.

*Fig.2:Perceptron Network output*

Where the limiting function $f_n$ serves to force the oputput to either

+1 or -1 and the bias is an additional network input which is initially set to some random value. This simply serves to add a measure of flexibility to the entire system.

Where the change of weight can also be expressed as in *fig.3*.

$$\Delta w_i = -\eta \frac{\delta E}{\delta w_i}$$

*Where E is the network output error*

*Fig.3:Output Error Calculation*

The algorithm shown in *fig.2* will eventually converge to the correct (expected!) classification if the problem is linearly separable and the initial learning rate is not too large. It must be noted that the solution achieved is not unique, and will depend on parameters such as the selected learning rate value and gradient, the initial network weight values and the order in which the training data is presented to the network.

## 3. Multilayer Perceptrons:

The concept of creating layers of perceptrons which would be treated serially for training was rapidly presented, but the difficulty resided in generating a training algorithm which would be capable of distributing the network error throughout the entire structure.

In 1986, Rummelhart, Hinton and Williams presented the backpropagation training algorithm, which effectively solved this

problem.

The network error i now calculated using a sum of squares error function as shown in *fig.4*.

$$E = \frac{1}{2}\sum_{p=1}^{p}(y^p - t^p)^2$$

Where $p$ is a given data pattern presented to the network,
$E$ is the calculated error,
$y^p$ is the network output
and $t^p$ is the expected output

*Fig.4:MLP Error Function*

A minimisation of this error will lead to a set of network weights such that the delta of the expected to the actual output for the entire training data set is as low as possible.

In order to allow this optimisation to be carried out, with regard to every weight in the network, it is necessary to introduce a non-linear factor to the calculations, the most popular of which is the sigmoid function. This effectively behaves as a linear adjustment for small inputs but saturates larger values, whether these be positive or negative.

For the hidden layer to output layer weights, the weight delta is calculated as before (*fig.5*).

$$\Delta W_{jk} = -\eta \frac{\delta E}{\delta W_{jk}} = -\eta \delta_k y_j$$

$$Where\ \delta_k = \frac{\delta E}{\delta a_k} = (y_k - t_k) y_k (1 - y_k)$$

Where $W_{jk}$ refers to the hidden to output layer weights

*Fig.5:MLP Hidden to output layer weight update*

For the hidden layer(s), the error is propagated somewhat differently, as an expected output is now not directly available. This value will instead be extrapolated from the error values of the previous node(s) (In the backpropagation process) and the weight value(s) to the given node(s)

The weight update is expressed as in *fig.6*.

$$\Delta W_{ij} = -\eta \frac{\delta E}{\delta W_{ij}} = -\eta \delta_j y_i$$

$$Where\ \delta_j = \frac{\delta E}{\delta a_j} = \sum_k \delta_k W_{jk} y_j (1 - y_j)$$

Where $W_{ij}$ is the weight between two neurons i and j,
$\eta$ is the learning rate
and $E$ is the output error

*Fig.6:MLP hidden Weight Update*

The difference to the hidden to output layer weights delta is that the $\delta_j$ for the hidden units depends on the $\delta_k$ of all the output units to which it is connected through its $w_{jk}$ weights.

The convergence (how rapidly the network will settle to a global

minima) of a network can be improved by introducing a momentum term ($\alpha$). The weights change can then be reconsidered as in *fig.7*.

*For output Weights :*

$$\Delta W_{jk} = w_{jk}(\tau+1) - w_{jk}(\tau) = -\eta \delta_k y_j + \alpha(w_{jk}(\tau) - w_{jk}(\tau-1))$$

*For hidden weights :*

$$\Delta W_{ij} = w_{ij}(\tau+1) - w_{ij}(\tau) = -\eta \delta_j y_i + \alpha(w_{ij}(\tau) - w_{ij}(\tau-1))$$

*Where $\alpha$ is a momentum value between $0 \wedge 1$*
*¿ $\tau$ is the iteration number.*

*Fig.7:MLP Weights delta with momentum*

The main advantages of multilayer perceptrons lie in the rapid training process, and the massive memorising capacity, where the number of "memorised" cases is much larger then the number of internal neurons. From this, the system obtains its excellent generalisation capabilities: the ability to classify data is has not previously encountered based on its similarity to data classes encountered in the training phase.

Its main disadvantage is a direct result of its training process, requiring it to be presented with a full range of data covering all possible input conditions in roughly equal amounts. Any distortion in the data will also be reflected in the performance of the final network, where data classes encountered in large numbers during training will be more readily recognised at the expense of those classes less well represented in the training data set. This mandates a careful selection of the training data set, in order to cover all

known data classes which will be encountered in use.

MLPs also tend to have long training phases (can be a few hundred thousand iterations) which are a direct consequence of the network structure itself. Variable parameters such as the learning rate (which can be dynamic during training, i.e. Can gradually reduce to a near zero value) and the momentum (which can and normally is dynamic during training) will also influence the speed and accuracy of the training phase: large initial learning rate and momentum values will lead to a very rapid but badly optimised convergence of the network. An ideal selection for the initial values of both of these parameters is largely an empirical decision, based on the results obtained from previous training sequences.

It must be pointed out that the solution achieved (in the form of a final network) is not a single solution to a given problem, as the network configuration will mostly offer a number of global minima (a point where the weight values are optimised to give the lowest possible error rate throughout the network for a given data set). Non optimal training can be the result of two other criteria: local minima, and overtraining.

Local minima occur throughout the network topology, and are points where no further improvement in the network error rate can be achieved by the application of the network learning rule, although a general error rate reduction would be possible had the weights been adapted in a different pattern. The momentum parameter helps to reduce the likelyhood of become "trapped" in such a local minimum, by occasionally adjusting the weights in the

opposite direction to the normal error reduction and thus allowing different weight optimisation paths to be found.

Overtraining [*70*] occurs when a given network is trained for too long on a given data set. The network weights will then be adjusted in such a way that they match the training data set as closely as possible (due to the normal error reduction process). In itself, this is what we want to achieve, but too large an optimisation in training will also lead to a reduction in the networks potential for generalisation: data not previously encountered in the training set cannot be correctly classified, as the data class boundaries will have been too tightly defined. This will occur when the network is trained for too many epochs, or when the training data set is too small.

Overtraining is however relatively easy to avoid, initially through sufficiently large training data set selection, and during the training process by a close observation of the network error value for the training set, as compared to the errror value for a validation data set.

A validation data set is a data set similar to the training data which should ideally represent a full data spread. After a set number of training epochs, the network performance (i.e. Output error) can then be evaluated using this validation set. It is important that the network weights are not modified during validation, otherwise this data set will simply become an extension of the training data.

Once the validation error starts to increase, it can be assumed that no futher network optimisation is possible without leading to overtraining, although the phenomenon of local minima must be

taken into account as these can lead to temporary but small reductions in the validation performance.

One last drawback of the MLP architecture is that these types of networks can also have difficulties when analysing data containing radically different dimensions to be classified, often requiring a small network to be used as a condition filter before the actual decision making process.

**4. Self-Organising Maps:**

Originally introduced in 1982 by Tuevo Kohonen, a self organising map using a network structure which is allowed to adapt during the training phase in such a way as to represent the various energy levels (effectively classes) of the input data. This is an entirely free mathematical approach, as no expected outputs are presented as with the perceptron model.

The actual training process is dependant on a neighbourhood size, which determines exactly how rapidly the network will adjust its internal connections.

The first step involves finding the euclidean distance from each neuron in the network to the input data pattern, where the euclidean distance is defined as in *fig.8*.

$$D_{out} = \sum_{n} (W_{io} - X_i)^2$$

Where $D$ is the Euclidean Distance
$W_{io}$ are the connection weights and
$X_i$ is the input pattern

*Fig.8:SOM Euclidean Distance*

The neuron with the smallest distance is considered to be the winner. The winning neuron and its surrounding neurons are then adjusted, to bring them closer to the input data, using the method shown in *fig.9*.

$$W_{io} = W_{io} + \eta (X_i - W_{io})$$

Where $\eta$ is the network learning rate

*Fig.9:SOM Weight Adjustment*

The learning rate thus regulates the network convergence speed. The larger the value, the faster the network will reach its stable condition, although the risk then exists that a more valid optimisation path be effectively skipped over. Generally, the learning rate is dynamic, and decreases according to the number of training cycles carried out.

This is in parallel to the topological neighbourhood, initially large to allow for large changes in the network structure, this should gradually decrease to zero so that at the end of the training sequence, only the winning neuron is being modified.

A trained Self Organising Map thus effectively represents a point mapping of the input data, convoluted to the number of dimensions defined not by the number of inputs, but by the network architecture itself.

As such an architecture does not provide any clearly defined output groups, it is then often necessary to label the resulting network. This can be carried out visually, by observing the winning neurons in different cases and marking the winning node to the class of the input given.

Due to their structure, Self Organising Maps are suited to solving problems such as classification and data mapping.
Their particularity is that they can be used to map an n-dimensional data space into a two dimensional vectorial representation commonly known as the feature space. This high level of data compression and vectorialisation provide excellent noise immunity. Their ability to cope with multidimensional data means that they are often used as a front end to a backpropagation or RBF type network. Weygang and Dasilva *[88]* forwarded procedures to overcome the mapping classification aspect by using multiple parallel approaches, which, whilst providing a more accurate description of the data classes involved, also lead to much higher processing requirements, depending on the complexity of the data at hand.

**5. Hybrid Networks:**

Although not an architecture per se, combining a number of "classical" network architectures together in specific manners, so as to utilise the salient features of each individual network type is gradually gaining acceptance. Certain aspects of utilising such an approach are discussed by Monostori [*58*], although this is not focused on image analysis applications.

The actual term "hybrid network", is mostly used to describe any combination of classical data preparation coupled to a neural network stage of any architecture, tasked with the final sorting of the produced data sets.

Throughout this study, the process of data preparation prior to presentation to a network is considered to be a basic requirement of successful analysis: unprepared data fed directly into a neural network of any format is going to result in haphazard output patterns, especially when the original data ranges are widely varying. Some type of data preparation is to be considered an absolute requirement to any successful output.

Considering past and current work in the field of image analysis, the closest approach to hybrid networks are massively parallel architectures of similar networks, effectively used either as a polling set, where each network gives a probability as to the final output (Zhao and Thorpe [*92*]), or where each network is contributing a sub part by searching for a single particular feature, as used by Chang, Razig, Abraham and Chae [*12*].

## 2.6.4 - Summary of analysis techniques

Whilst a large number of analysis techniques are available, it must be stressed that the ultimate success of any data analysis is going to be dependent on the understanding of the data by the analyser and the integrity of the data to be analysed.

A poor understanding of the data dimensions will probably lead to important data aspects being omitted, and an incomplete data set for analysis will simply not offer certain dimensionalities of the data which might be encountered in later use.

Whether a manual or autonomous data analysis method is selected, the integrity of the data throughout the process is paramount: any transformations on the original data can lead to data loss for later processes, and such operations must therefore be carefully considered and selected before being applied.

One further important point in any data analysis lies in the the scale of the data. Whether using a traditional technique or a smart technique, data sets mostly require some form of scaling in order to allow the examination of inter-relationships in the data stream – this affects not only the mathematical scale of the data but also the order in which a particular data stream is observed.

Smart techniques in data analysis do seem to offer more flexibility, especially when considering non-linearities, which can then be mapped into hyperspace and evaluated using various visualisation methods, but do require more in depth data preparation than a

more manual approach. The actual process chosen must be selected to balance the type of data, the integrity of the data(will interpolation be required in the analysis phase), the availability of set results for the data (a pre-requisite for any type of perceptron structure!) and the type of analysis: must this be on online process, must the analysis methods be flexible at a later date or not.

The advantages of hybrid approaches allow various processes to be implemented at the most suitable stage of the analysis and on all or only part of the data. These structures seem to be best suited to real life data, which will be inherently noisy and inconsistent.

# 3 - Study Definitions

*Recognising the need is the primary condition for design - Charles Eames*

In order to carry out a thorough study of the case presented, a certain number of variables must first be defined, in order to refine the field of study.

## *3.1 - Study Objective*

The aim of this study is to enable a simple fixed lens security camera as used in common CCTV applications, to detect and react to the presence of a person within the surveillance area which the camera has been assigned to.

This detection process must not be impaired by environmental factors nor by the presence of other mobile or immobile objects within the camera's field of view.

Environmental factors include such variables as overall lighting conditions, surface reflectancies, temperature variations, as well as camera setting changes, although the latter are not to occur whilst the system is in operation.
The detection and reaction processes are to function in realtime, allowing the system to operate either as a stand-alone unit or using

operator backup. In either case, false alarm conditions are to be minimised.

When functioning with operator backup, irrelevant data is to be filtered out, whilst retaining sufficient information for operator decision making processes.

The overall system must be simple to install and operate, requiring little or no knowledge in the field of intruder surveillance.

## *3.2 - Definition of a Person*

Probably the most important item to define is the actual target, what the system is trying to identify, in this case, a person.
To allow data parameters to be defined, it is necessary to quantify the term person into mathematical parameters.
These parameters will also largely depend on the type of camera/lens combination which is to be used.

According to Anthropomorphic data , and considering the population groups from the 5th to the 95th percentile (Which covers 90% of the population spread)[*34, 83*] , the following values can be taken:

| Measure | Adult Male | Adult Female |
|---|---|---|
| Height | 161-185 cm | 150-170 cm |
| Elbow/Elbow breadth | 35-50 cm | 31-49 cm |
| Mean Shoulder Width | 45 cm | 40 cm |

This represents, however, only data for adult humans, and this system will also have to be taking into count threats posed by children.

It can be assumed that a child under two years need not be considered as capable of posing a threat, which sets a height limit of approximately 100cm, when the person is standing.

Target proportions can be derived from the adult proportions as defined above.

This results in a minimum target size of 100x24 cm, which would represent an average child standing. It cannot however be assumed that the target will always be in an upright stance, behaviours such as crouching or crawling must also be taken into account. This would effectively halve the height of the target, resulting in an object around 50x24 cm.

These given target dimensions are independent of the type of camera/lens combination to be used in actual system development and operation. It must also be noted that these values represent preliminary guidelines, which may well be adapted to cope with system changes or detection problems which might be encountered

in further stages. These values thus represent an ideal minimal object size for detection and analysis within the parameters of the system to be developed.

Depending on the image sensor and lens combination selected, and considering the minimum and maximum required detection distances, the values given above shall be translated into camera dependent dimensions for defining the minimum sized object which should be considered for target processing. This has an impact on the amount of processing each image will undergo, as objects smaller than the minimum defined can be immediately discarded, thus speeding up the entire process.

# 4 - Data Extraction

*We need above all to know about changes; no one wants or needs to be reminded 16 hours a day that his shoes are on - David Hubel*

In all analysis systems, a correct and reliable source of raw data is of utmost importance to the correct functioning of the system.
It is therefore important to develop rigorous data extraction techniques, if any type of reliable system is to be developed.

Data extraction considers not only which type of data is to be extracted, but also which preparation steps are required, and which actual extraction methods will be employed in order to retain uncorrupted values.
The inconsiderate application of masks and filters over an image may well enhance certain aspects, may can also lead to irreversible loss of what may be important information.

In order to establish a reliable process, it is important to initially define what type of information is to be extracted and in which form this information should be presented at the outcome of the extraction process.

In the system which is being considered, the final details of the

image information required have not yet been finalised. These can actually only take on a final aspect once an initial testing phase has been completed.

In order not to unnecessarily limit this initial testing and experimentation phase, it would be advisable to provide as large a spread of data as possible. Whether the entire data range will then be utilised is not of any concern, but this approach will provide as wide a starting base as possible, allowing for a structured approach to the data analysis, which in turn will enable the original data set to be narrowed down to hold only those required elements.

## *4.1 - Initial Extraction*

For the purpose of initial experimentation, a camera has been provided in the form of a Creative Labs WebBlaster Webcam[*95*].

This is a small CCD based device with a fixed focal length and a relatively cheap plastic lens.

The reason for this choice lies in the setup rapidity, relative cost and rapid access to initial data which it will provide, as no external video capture hardware or software is required.

The specifications of this camera are as follow:

(These have been derived from the manufacturer as well as practical tests)

**Maximum Effective Range**:

10m, for visual object recognition in the size limitations which are being considered for this study.

**Lens:**

Horizontal angle: 52 degrees.

Minimum lighting: Could not be obtained.

A feature of this camera which has been noted in low light conditions is severe image corruption resulting in a marked elliptical shadow blooming around the borders of the image. This defect is presumed to be due to the cheap nature of the lens used in the camera, and effectively means that very low light testing will either have to be abandoned or will have to take into account the varying nature of this phenomenon.

As the type of camera for which this system is to be developed has normally higher quality components, the elimination of this effect will not be incorporated into the actual analysis algorithms.

The webcam can output images to the harddrive in a number of format/colour/size combinations.

In order to minimise the loss of data during this initial transformation stage, it has been decided to save these images in a bitmap format which provides for easy later processing in 24 bit colour (16 Million colours) and saved to a 320x240 pixels size. This

provides an acceptable compromise between image dimensions and level of detail, and also approaches the standard output for surveillance equipment, thus providing a close match to the equipment to be using the finalised system.

Using the target definitions provided earlier, it is now possible to calculate the minimum image size object to be detected, given the above camera specifications.

Given the Field of view of 52 degrees and a maximum range of 10m, the maximum horizontal spread can be extrapolated as shown in *fig.10.*

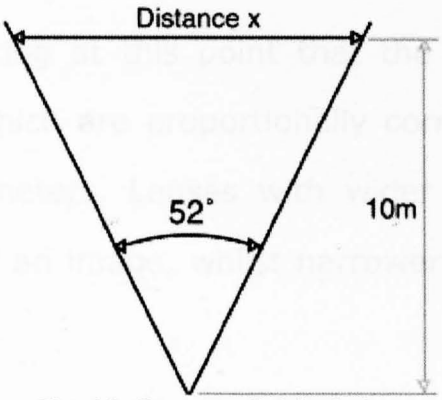



*Fig.10:Camera field of view*

Distance x = 20tan 26

(the range is composed of two identical right angled triangles with an initial angle of 26°)

Distance x = 9.75m

Taking, as specified, a 1m wide object and using a final image of 320x240 pixels, the equivalent image size for his object will be:

Equivalent size = 1 x 320 / 9.75

= 33 pixels

Therefore, a 1x1m object placed 10 m away from the camera lens will be represented in the resulting image as a 33x33 pixel sized square. Obviously, as the object approaches the camera lens, the relative screen size will be increasing.

Using these values once again for the minimum target size of 100x24cm, this would result in on screen dimensions of 33x8 pixels. This value thus represents the minimum image object size to be considered as a potential threat and thus to be correctly analysed.

It is worthwhile noting at this point that the lens currently used results in images which are proportionally correct as compared to human vision parameters. Lenses with wider angles will tend to stretch the edges of an image, whilst narrower angles will result in slight compression.

## 4.2 - Methods of Analysis

There are a number of analysis methods available when considering the extraction of image data. Generally, these can be split into two main categories:

. Searching for known data.

. Searching for unknown data.

Each method has its advantages and disadvantages, which will be explained in depth.

### 4.2.1 - Searching for known Data

Searching for known data implicitly implies that all situations of interest are known to the user and system developer prior to the product development. The data itself need not be restricted to a single type but can take the shape of patterns, image trends, light levels, known positions of certain items, etc [*67*]. The type of data is dictated by the intended application. This is corroborated by the highly targeted nature of such processes, mostly aimed at recognising a single object type [*42*].

Due to the nature of such a search, the entire system can be fairly small and the detection process might be fairly rapid, as only clearly defined parameters are used in the data search. This aspect of rapid recognition comes to light in the work of Francesschetti, Iodice and

Tesauro [*30*].

The drawback of such a system is however its inflexibility, in a number of contexts:

Initially, all possible image situations must be known, which implies a very limited or specialised implementation field. An ideal application for such a process is, for example, object sorting on a factory feed belt. In such a condition, the camera is never moved, lighting remains constant, the area of use is predefined and restricted with no external noise influences and the objects appearing in the image have also known profiles and dimensions [*55*]. The task of such a system is normally fairly limited, normally object orientation detection or fault detection by profile matching, as highlighted by Kuehnle and Burghout [*48*]. Anything not matching the predefined object parameters is automatically discarded and does not undergo any further analysis.

The main drawback of such an approach is directly due to its high specialisation. The process has been adapted to very strict implementation rules and generally cannot be adapted to a different environment without modifying important process parameters[*14*]. This results in a very reliable but inflexible system.

### 4.2.2 - Searching for unknown Data

Whilst more complex, this is a much more flexible approach to image analysis.
Instead of attempting to carry out pattern matching or detect other fixed features within the image, this approach relies on comparing the actual image with a previously obtained datum image. Changes within the two sets can then be determined and analysed in any suitable manner, depending on the type of data and the desired output.

Whilst this provides much more scope for development, it also depends on the obtention of a reliable datum image, from which all further analysis will be derived. If this initial image is in any way corrupted, any later processing will reflect this corruption and most likely result in nonsense output.

Such an approach generally offers more system flexibility, as far as the system environment is concerned. Depending on the data extraction and analysis algorithms used, modifications such as light level changes and even emplacement changes can be catered for, without having to review the analysis processes.

The sheer flexibility of such an approach also calls for a much more rigorous data extraction and analysis phase, as changing contexts

need to be taken into account without corrupting or affecting the output data obtained [*89*]. It is also very easy to misinterpret fluctuating values due to environmental factors as targeted data. Data must not only be correctly extracted but also correctly interpreted, which in turn will lengthen the process development phase.

Due to the flexible nature of the targeted data, or rather, the varying way in which this data will be presented within the image, the entire image analysis/data extraction phase will tend to be more complex, and thus also more time intensive, than when using the known data extraction approach[*16*].

### 4.2.3 - Extraction Mode Balance

In the situation which is being considered for this study, it would be impossible to predict the total number of variables which might at any stage come into play and would then have to be analysed or at least filtered by the system. Indeed, a condition of the hypothesis is that no prior knowledge of the surveillance environment be needed prior to running the detection process.

Possible environmental effects can additionally be split into two groups:

1. Effects on the overall image.
2. Effects on the target data within the image.

Of course, the two groups are closely linked, as, for example, shadows within the image will most probably cause shadows on or around the target as well, but for processing purposes, the two can be considered separately.

Effects on the overall image will be affecting the target acquisition and extraction phase, whilst effects on the target will be affecting the target analysis stage.

Separating these two processes has a number of advantages.

Initially, overall processing is speedier: If the target detection stage is negative (i.e., no detected target), the target analysis phase can be discarded, thus reducing redundant processing.

Secondly, this results in a modular structure where each module has a dedicated task. This is quite an elegant solution, as the processes from each module can be called up as and when they are required.

The modular approach also facilitates the actual system development stage, as each task can be developed separately. Modifying or adjusting a given process is both easier and quicker, and the various modules can easily be fine tuned to one another's input/output requirements.

The processing methods for each stage must thus be considered as totally separate sequences operating under different conditions.

## 4.2.4 - How Many Stages ?

Before the actual number of stages required for the processing/preprocessing functions can be accurately defined, it is important to understand the operational conditions of the entire system.

Intended as a security system, it is likely to be used all year round in both night-time and daytime conditions. As yet, the system has not been restricted to purely indoor or outdoor use, which has implications on the lighting conditions which will have to be taken into consideration.

Indeed, whilst indoor conditions can be described as fairly stable, with little or no change to overall environmental parameters such as lighting levels or spacial occupation (distribution of objects around the area to be surveyed), an outdoor environment is much more dynamic, with changes occurring on all levels.

**1. Overall Light Changes:**

These will occur as a result of the natural day/night cycle. It cannot be assumed that the system will only be in used at night time or only in bright sunlight. Light changes will also occur as a

result of weather influences, clouds, mist and rain will all affect overall light levels.

**2. Point Light Changes:**

These are light changes due to direct or indirect shadows.

Thus, as the sun crosses the sky, the shadows cast by objects will seem to be rotating around their source. Shadows will also vary in size and intensity.

**3. Camera Movement:**

Camera movement will occur as the result of improperly mounted cameras, or due to wind vibration. In an extreme case, this can also occur through human or other interference with the camera itself, i.e., an attempt to destroy or rotate the camera mounting. As camera vibration, even on a small scale, is nearly impossible to prevent, some form of image stabiliser would be highly recommendable.

**4. Object Movement:**

As opposed to target movement, other objects might also move within the surveyed scene. This might be the result of wind (Moving clouds, trees, floating plastic bags or papers) or through some other mechanical influence (ventilation systems opening, cars passing by...) as shown in *fig.11* and *fig.12*.

*Fig.11:Carpark Datum*

*Fig.12:Carpark with changes*

These figures actually illustrate a complex situation with both an additive and a subtractive change: *Fig.12* features the blue car appearing in the scene as well as a red car having vacated its parking spot behind the former.

Examining the image difference analysis more closely also shows motion changes in the trees in the background (highlighted in yellow, *fig.13.*)

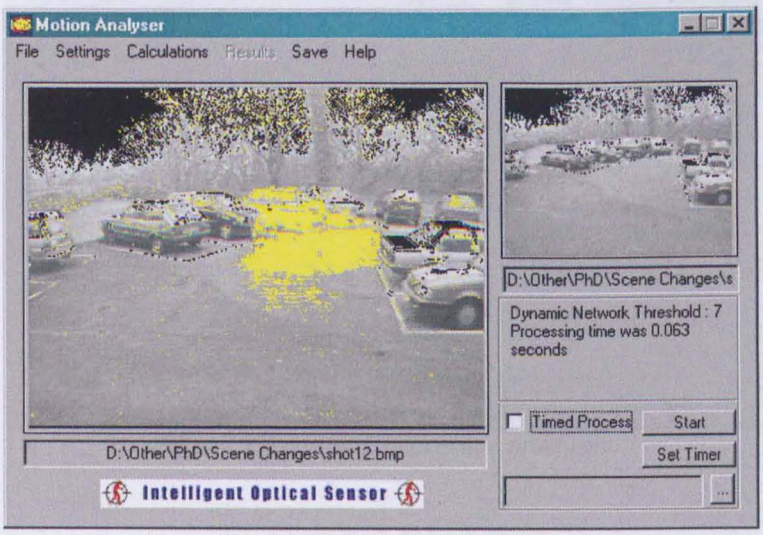



*Fig.13:Full scene change*

Whereas the retained areas of change for further analysis are

shown in fig.*14* (highlighted in magenta).

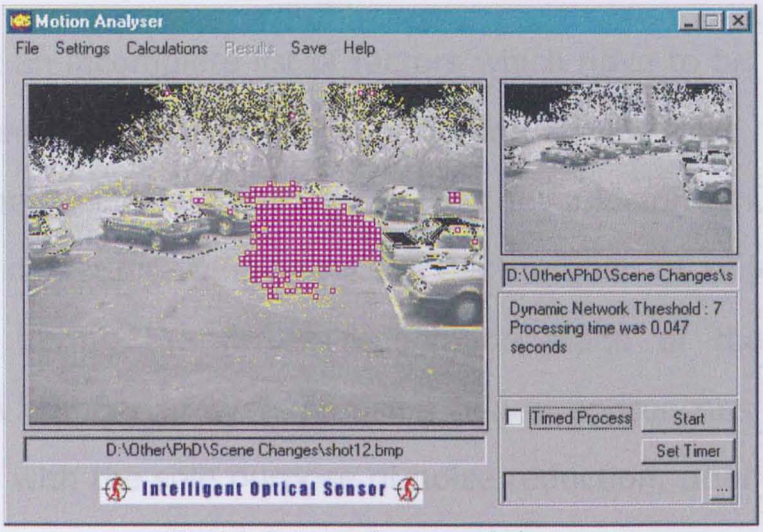



*Fig.14:Retained areas of change*

The resultant object recognition, correctly identifying both changes, is shown in *fig.15*.

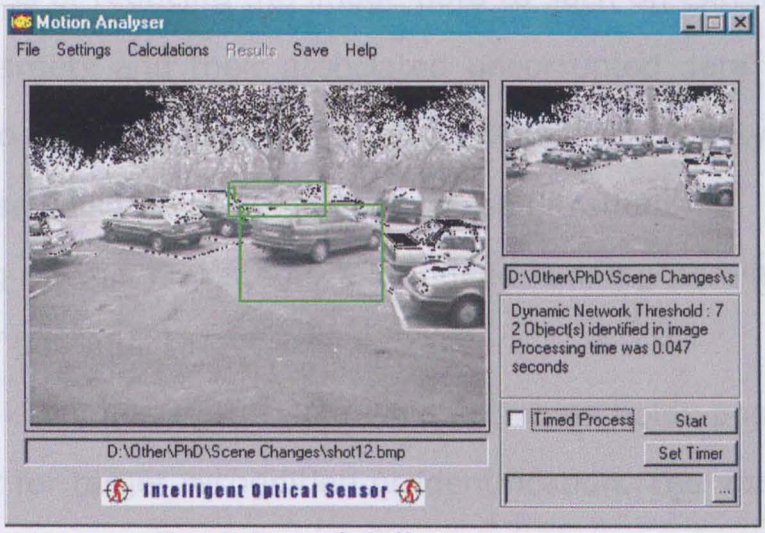



*Fig.15:Carpark Difference analysis*

## 4.2.5 - Target Area Recognition

This is but an incomplete list of factors which have to be considered for a system with such a wide application range. Most of the thus resulting noise should ideally be filtered out prior or simultaneously to target detection, thus reducing the number of objects to be examined during the target recognition stage of the detection process. A number of systems using neural approaches have been developed with the sole purpose of noise reduction, prior to analysis [*50, 05*], although these tend to be complex systems which often require a certain knowledge of the operating environment.

The entire aim of the preprocessing stage thus can be summed up as a process of reducing the image data to such an extent that only possible targets and their associated uncorrupted data remain, to be fed into the next processing level of the detection system. This initial phase could be designated as noise filtration.
Depending on this first stage, and indeed running parallel to it is a process of potential target area extraction.
Once this is complete, the data which remains holds possible areas of interest for target recognition or identification. The various target areas do however first have to be recognised within this overall presentation.

Although this might initially seem to be a doubling of the target

extraction process, the difference is illustrated with the images below (*Fig.16, Fig.17, Fig.18, Fig.19*):

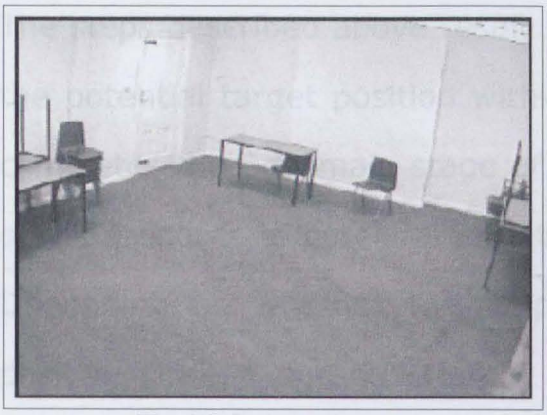


*Fig.16:Datum Image*

*Fig.17:Camera Image*

*Fig.18:Difference image, showing potential targets*

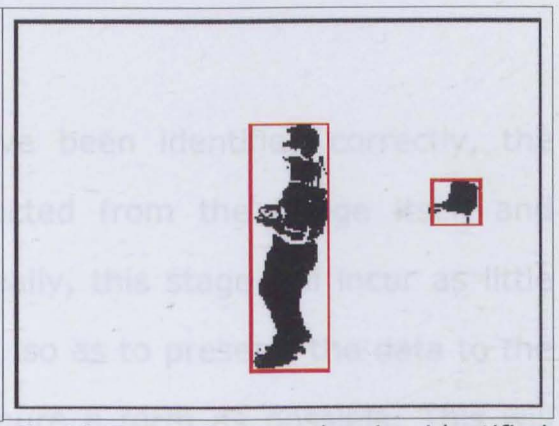


*Fig.19:Target Image, showing identified potential target areas*

NB: The above image sequence has been obtained by the use of custom filters within Adobe Photoshop, purely as an illustrative measure showing possible analysis processes.

As can be quite clearly seen, the potential target detection and potential target area detection produce two quite different results, the output resolution being refined during each stage of the

detection process.

The steps described above result purely in an accurate definition of the potential target position within the given camera image, which completes the first main stage of the detection process: Evaluation of the presence of potential targets.
Depending on whether the output from the first main stage is positive (i.e., a potential target has been identified), the second main phase can be initialised: Potential Target data extraction and preparation.

Given that the target areas have been identified correctly, the image data must now be extracted from the image itself and prepared for further analysis. Ideally, this stage will incur as little data distortion or loss as possible, so as to present the data to the Target classification stage in as pure a form as possible. This will help to ensure a high level of positive (correct) target classification.

## 4.2.6 - Analysis Sequence

It is initially impossible to specify the amount and type of data which the classifier will be requiring. This will not only depend on the architecture of the classifying stage itself, but also on the desired output from this stage of the overall system. The actual type of data required , will however not have much effect on the

overall process distribution as such, but will only affect which types of filters and algorithms are employed for the actual potential target data extraction phase. The overall system sequence can already at this early stage be roughly defined. Minor changes to this arrangement are bound to occur as the system development progresses, but this will assist in laying out an experimental development order.

An initial summary of possible processing modules, as defined above, results in the following breakdown:

1 - Image comparator.

2 - Image Noise Filter.

3 - Image Potential Target Detection.

4 - Potential Target Separation.

5 - Potential Target Data Extraction.

6 - Potential Target Data Preparation.

7 - Potential Target Analysis.

8 - Target Recognition and Classification.

9 - System Output Decision.

## 1 Image Comparator

Due to the intended dynamic nature of the system, its ability to be used in varying contexts without the need for a system rewrite, this stage is a necessity to the correct functioning of the target detector.

As we are bound to one of the primary definitions of the problem (point 3: *Can such a system be developed in such a manner as to be independent of the camera installation location and method?*), it is difficult to employ a method of scene mapping or prediction, such as that described by Collins and Tsin [*19*], as this calls for multiple differing views of the environment in order to generate an accurate mapping of the surroundings.

As the environment cannot be predicted, and would be too complex to be mathematically defined by the operator every time a change occurred, a process which would anyway compromise the aspect of autonomous operation as laid down in the basic requirements, it is faster and more reliable to rely on the system itself to define its normal operating conditions.
The obtention of a datum image would in such a case be highly appropriate. The role of the datum image is to store the state of the surveyed area in a non-alarm situation. To deal with changing environments, this datum image would ideally be obtained immediately when the surveillance system is activated. As the

system then operates, each new incoming image will be compared to this datum to determine whether any changes occurred within the surveyed area.

Although an elegant solution, such an approach does also have its potential problem areas:

**I. Accidental Environmental Changes:**

These could be the result of doors or windows being opened without realising that a surveillance system was in operation. For example, the person enabling the system might have entered the room and left the door open whilst arming the system, but then closed the door as they went out, resulting in a constant discrepancy between the datum image and the incoming camera images, even though no valid target was present. Such setup errors are a major source of malfunctions in basic surveillance systems using any types of sensors [*95,26*].

**II.Lighting Changes:**

Especially relevant if the system is operational overnight, where a brightly illuminated datum image might have to be compared to camera images in low light conditions. Although essentially identical, the comparison algorithm could easily be fooled by the

absence of shadows in the camera image, resulting in negative potential targets.

These problems can however be overcome by the introduction of a historical component. This might enable the automatic update of the datum images as overall conditions change without ensuing positive alarm states.

The role of the image comparator is critical in providing the ensuing processes with the correct balance of information. Overestimated tolerances will result in a surplus of redundant information being passed on for analysis, whilst underestimated tolerances would result in actual targets being overseen.

In order to provide an ideal flow of information, this process should have a certain level of dynamic response towards the system's environment.

## 2 Image Noise Filter

This stage follows on the image comparator. Due to the very nature of the image comparison process, the resulting data will not only be representing potential target areas but also various types of image noise, whether these be due to camera vibration, local light changes

or background variations.

In order not to overload the target identification process with unnecessary data, resulting in longer and unnecessary process times and the need for more system resources, it would be advisable to employ this stage as a means of filtering out as much obvious noise as possible from the image.

By the term "obvious noise", is meant data which cannot in any way be representing a potential target within the detection parameters which have been set for this system. The most direct example of such data would be areas of noise which are smaller than the minimum specified identifiable target.

Whilst such areas may well be representing real targets which are beyond the actual search or surveillance range, as long as they do not enter into the strictly defined search parameters they are of no interest to the system in general. As soon as such a target enters the actual defined range, the system will change its classification from that of unwanted noise to that of a potential target, and the target identification process will be carried out as intended.

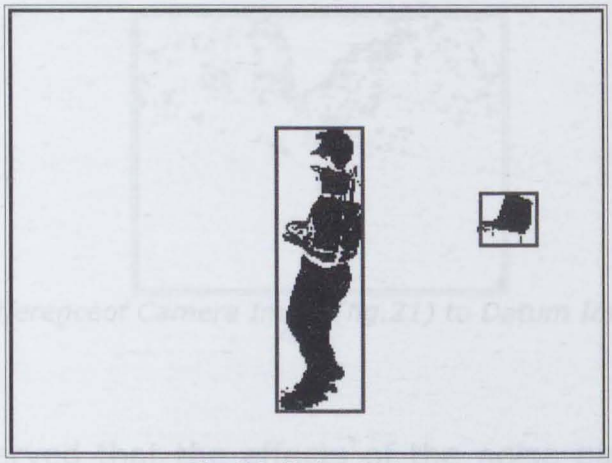


*Fig.20:Point Noise*

The example shown in *Fig.20* clearly illustrates a condition of point noise, i.e., noise which is restricted to a certain area of the image.

The second type of noise which must be considered is distributed noise, a typical result of camera vibration, where the noise is purely a result of image origin shift in any direction. A similar effect can be obtained from tree branch movement, as illustrated below in *Fig.21, Fig.22 and Fig.23*:

*Fig.21:Datum Image*

*Fig.22:Camera Image*

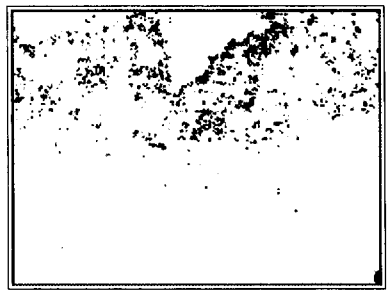


*Fig.23:Differenceof Camera Image(fig.21) to Datum Image(fig.22)*

It can be observed that the effects of the noise patterns are fairly similar, varying only in their spacial distribution.

Whereas in the first case (point noise), noise elimination is possibly carried out by masking or ignoring the affected area until the displayed noise gains sufficient significance to be considered a potential target, the second condition (distributed noise) cannot be treated in the same way, as we could not afford to even momentarily mask the entire image without even considering the source of the noise in the first place.

As mentioned earlier, this type of noise is most commonly due to wind-induced camera vibration or background vibration/motion. This generally represents a fairly regular type of displacement, either in the nature of the motion or in the area affected by the motion. Although such motion is difficult, even impossible, to predict in any accurate way without detailed knowledge of the operating environment, there is a proven method of compensating such noise, as can be seen in high-end camcorders. The solution, commonly known as "jitter-correction", is occasionally based on

mechanical stabilising devices based on gyroscopic platforms [*Appendix E*], but is mainly carried out by full electronic counterparts which function on the principle of image frame comparison:

Consecutive images are analysed for generic motion patterns. Large motion levels, such as those caused by a person walking across the image, are immediately discarded or ignored, whilst minimal image-wide motion is compensated for by shifting the entire image in the appropriate direction.

Such a process could be refined by applying the filter not only to the entire image, but to individual pixels within the images, thus resulting in a point by point level of jitter-correction. This can be further fine tuned by specifying a correction range, implying that pixel-wide motion below a given threshold could be automatically compensated for.

The following images illustrate a few cases displaying various forms of distributed noise and the effect such occurrences have on a motion detection process.

## 3 Image Potential Target Detection

Following the process outlined previously, this phase represents a crucial stage for the successful operation of the entire system.
This phase for detecting potential targets within the presented image will ideally be relying on obtaining highly optimised data with as low a loss of resolution from the original image as possible, in other words, the original image must be optimised but in no way corrupted to be losing potentially important data to this stage of the detection process.

Such data requirements stress the need for entirely optimised data preparation, as any modification to the original data may well lead to a drop in performance of the potential target identification stage.

The actual mode of detection is as yet open. A number of different techniques exist which may be considered for this purpose, but all rely on some way or other on some form of template matching, whether this be in a purely mathematical or purely graphical (nearly empirical) manner.

**Template Matching:**

A stylised template of a perceived threat is created and modelled in a mathematical manner, in such a way that it may be applied as a

comparative filter to the incoming image. This template may have a number of deformation axes which allow for a certain degree of flexibility whilst retaining the general attributes of the template. If a match occurs, this is then first analysed for the degree of deformation necessary to obtain the overlay with the template, and can then be given a relative degree of importance for further processing, see target identification.

The main problem linked with such an approach is the requirement for well defined and separated potential targets. If the image has a high noise level, or if a potential target is partially hidden or overlapping with another target, this can cause the template matching function to fail, as the deformations required to retain actual matching are then too large to fall within the limited deformation constraints of the given template. To allow for such situations by increasing the template's degree of deformation will only lead to a higher number of negative targets being identified by the template matching process, thus reducing the system efficiency. This difficulty introduces the need for a stage of quite rough potential target evaluation, which might be able to distinguish such noisy target images. This has been implemented in a number of studies by using much more generalised templates which only when combined can be evaluated to be presenting a target match or not, as shown in the work of Viola, Jones and Snow [*84*] amongst others [*42, 30, 43, 64, 54*]. The problem which this raises is a direct result

of the multitude of filters employed: when is the filter combination defined as valid or invalid ? This again returns to the basic assumption that not all but most object features are already defined and recorded in a template format.

Another quite straightforward way of countering detection failure through target corruption is to introduce a historical feature into the detection process. Once a target has been positively identified, the system will then track this object, biasing the detection process by reverting to historical data on the object. Thus, the longer a target is present within the image, the more certain the system becomes of its validity and the less likely the system is of loosing the object if the latter becomes corrupted in any way during a few surveillance cycles. [*89, 78, 31, 09*]

Introducing a historical feature however also calls for increased system storage, as data on the target must be stored and updated, frame by frame, until the examined target definitely exits the surveillance area, a feature which must be treated carefully, depending on the parameters which are to be stored, especially in the system at hand, intended to be integrated into a portable autonomous and yet still discrete unit.

This approach has been used extensively in systems relying on multiple camera arrays to overcome environment clutter and thus

enhance tracking efficiency (Collins, Lipton and Kanade[*18*] - Collins and Tsin[*19*]). These do however rely heavily on accurate calibration of all cameras involved and a fixed installation location, which does not fall within the specifications of this study.

Such a feature also calls for a certain degree of **motion tracking and prediction,** which is not entirely necessary in the system being considered. This might however become a necessary addition to the processing system if difficulties do arise in target detection and identification.

See *figures 19 and 20* for examples of borderline cases, where the (valid) target is in some way corrupted out of normal template deformation parameters.

*Fig.24:Target features partially cut off by other objects*

*Fig.25:Target only partially in the image frame*

All these methods are describing some type of target separation

## 4 Potential Target Separation

What this process describes is the act of separating the identified potential target from not only the image background, but also from other intrusive effects such as shadows or objects in front of the identified target.

It is also the act of identifying not only the fact that the image has changed, but also of identifying exactly which and exactly how many separately identifiable regions of the image have undergone this change.

In actual processing, this step cannot be separated from that of potential target identification, as one relies on the other, and indeed the processes are not carried out in a purely linear manner, but are called in as and when necessary depending on the image observed. The theory of the process however can be explained in a separate step.

Target area identification for simple cases (one target in image) is fairly straightforward. The boundaries of the image change will also be defining the boundaries of the potential target, meaning that the affected image area can be rapidly extracted and analysed. This is shown in *Fig.26*.

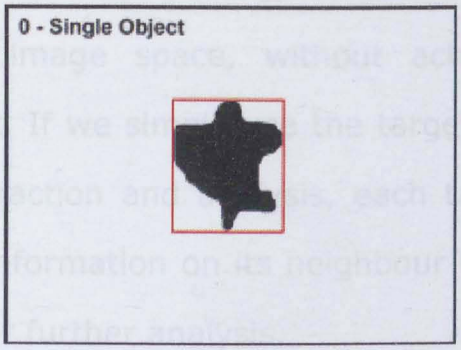



*Fig.26:Single Object*

We cannot however, rely on always encountering such a simple ideal case. Whether due to noise interference or any other inputs, a more complex situation should be expected.

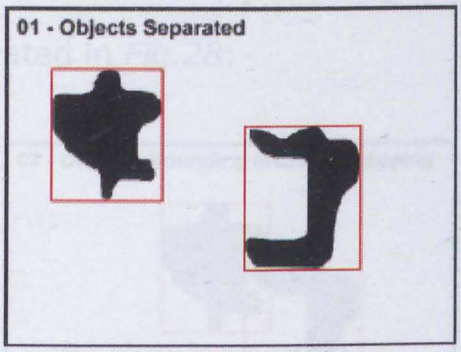



*Fig.27:Separated Objects*

Two separate potential targets, as seen in *Fig.27*, present a step in this direction. Here the total area of change within the image is much larger and some slightly more advanced processing is required to separate the two target components and extract their data for analysis. This approach can be used for increasing the number of potential targets in a single image.

A more complex situation will occur when potential targets encroach on one another's image space, without achieving apparent 2 dimensional contact. If we simply use the target extent boundaries for image data extraction and analysis, each target will contain a certain amount of information on its neighbour target, thus leading to corrupted data for further analysis.

This can be rectified by refining the identification process in such a way that a potential target can be checked for entity continuity. Thus only a group of pixels which are joined together on the image plane will be considered as forming a single object. In this case, a certain leeway can be incorporated, giving a possible gap range of a few pixels in order to allow for possible lighting effects within the image. This is illustrated in *Fig.28*:

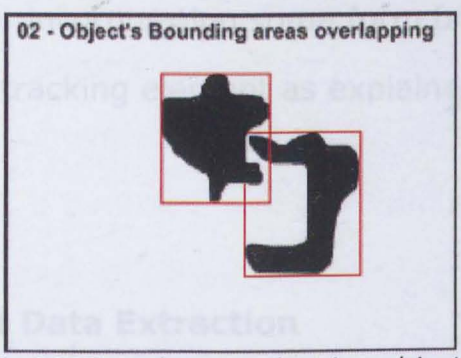



*Fig.28:Boundary overlapping objects*

The most complex situation however, will occur when potential targets are either partially or entirely overlapping in their image areas. Here, as far as the image change area is concerned, the multiple overlapping potential targets will simply appear as a single fluctuating shape, which might or might not meet the parameters

for a valid target identification. This is illustrated in *Fig.29*.

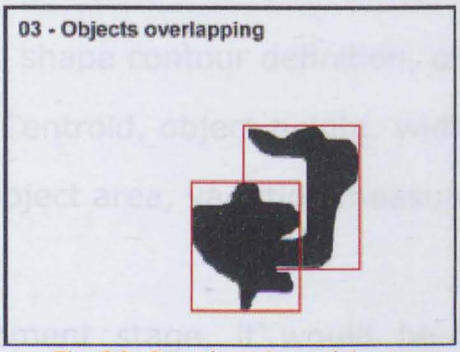



*Fig.29:Overlapping objects*

Such a situation is difficult to interpret without having recourse either to historical data, or to a second image from a different angle.

As the system considered is intended solely for a single camera setup, the second option is not available, and the only solution would be to consider incorporating some form of historical data, and thus a given object tracking element as explained earlier.

## 5 Potential Target Data Extraction

Up to this point, we have been concerned with aspects of overall image analysis. Once the actual potential target areas have been tightly identified and defined, the system can move onto the stage of analysing each potential target for possible positive target match. This requires a certain number of parameters to be extracted from

the identified potential target regions.

Such information might take the form of vectorial measurements leading to an overall shape contour definition, or might contain such elements as object Centroid, object height, width, placement within the overall image, object area, variation measurements etc.

During the development stage, it would be highly advisable to ensure that as many different parameters as possible are extracted from the image for further analysis. Whilst this might lead to more intensive data processing algorithms, it is the only way to ensure that important parameters are not omitted. As further investigations are carried out, these important parameters will become known, and less useful measurements can then be omitted entirely from the data extraction process. Care must be taken to ensure that the parameters extracted are also universal to every image in a surveillance sequence. Obviously, if no potential target is present, this is not a problem, but we cannot rely on empirical values such as "distance of target 1 from target 2", as there is no way of guaranteeing that two potential targets will always be present in the image, thus leading to a lack of data for the following analysis stage.

Care must also be taken not to provide data which might be locally affected. This will include features of physical location (terrain layout, angle of inclination, lighting source variations) and weather phenomena.

## 6 Potential Target Data Preparation

Once the data has been extracted from the image, it will have to be prepared before being presented to the detection algorithms.
This is a necessary step, mainly to ensure that the various data streams are using standardised ranges. Whilst data from varying sources can be simultaneously analysed, this process becomes easier if the said data is restricted to common minima and maxima.
For use with a neural network, it is customary to use a data range of either -1/+1 or 0/+1. Such ranges not only provide a certain clarity in the data set, but are also optimal for much of the range checking within certain networks. It is also well suited to being processed by algorithms utilising angular relationships (many neural networks are based on sigmoid functions at some stage of their processing, thus providing a matching 0/+1 output range corresponding to the input).
The actual data transformation processes required to achieve this standardised range must be determined as best matching or representing the available data. Such processes might contain linear or logarithmical scaling, empirical stepping, sigmoid based functions, square root based transforms and many more. [*68, 22, 30*]
Selecting the process which will provide the most accurate representation of the data required is important, as many functions,

such as logarithmic transforms can seriously distort the original data, enhancing certain ranges and reducing others. It is therefore important to maintain an accurate knowledge of the original data, the data which is required to be fed into the analysis network and the associated data transform required to achieve this without corrupting possibly important ranges of the original data stream.

Once the selected data streams have been optimised, using appropriate transform processes, the actual detection/classification network can be applied on these to achieve at least an initial stage of target classification.

One of the main difficulties in the entire data preparation phase, is that of unintentional data distortion or corruption.
Depending on the application, certain data windows of a given input's data range might need to be enhanced, whether to accentuate the given range , or due to sensor responses leading to the need for input amplification. Whilst carrying out this range amplification process, outlying data which may initially have not been considered as crucial can be severely distorted, which in turn may lead to unexpected network behaviour.
It is thus important, before carrying out a data transformation, to evaluate the entire data range and assess the importance of the entire data range using statistical or heuristic methods, whichever suit the application under development. One of the main statistical

analysis methods available is PCA, Principal Component Analysis, which can give a useful insight into the structure of the data stream and its variations.

Whilst considering each data stream for its individual characteristics is important, it must not be forgotten that, within a functioning network, each input stream (where a stream consists of a data set for a single object) is considered as a function of its accompanying inputs. Thus, the network is attempting to define a relationship between the as yet totally separate inputs, which will then allow a spatial separation to be defined. This is an intrinsic quality of a network, as it is composed of a defined number of highly interlinked nodes defined by linking functions which are data variable, i.e., whose values are defined by the incoming data streams, at least during the dynamic learning phase of the network.

Obtaining a good understanding of possible data relationships prior to any network development work is therefore crucial to the development of an optimised system. If redundant or repetitive data can be filtered out of the network input streams, this will ultimately lead to a more robust system, as the network will be able to establish simpler internal nodal correlations which will be optimised to representing the wanted output patterns, and not wasting processing resources in carrying out internal data noise filtering.

The more optimised a network is in this aspect (dealing with pure data instead of declassifying noisy or unnecessary data), the higher the expected network's performance can be. We must also take into account the networks potential for data generalisation and operational noise filtering. If the network has been able to train with optimised data, less of the internal resources will have been "wasted" in data cleaning, and the network should thus be able to use these resources for operational noise filtering.

## 7 Target Recognition and Classification

The phase of target recognition, whilst less crucial than the actual data preparation, is much more spectacular in its results, as this is the stage where a single network, or a conglomeration of networks, will be utilised to analyse the prepared data and output a target directed decision based on this data.

The simplest method would be to feed the previously prepared data straight into a classification network which would then output one of two possible results: " Target" or " No Target".

Due to the probable high complexity of the data to be analysed, as well as the sometimes fairly ambiguous separation lines between

valid and invalid targets, we might well have to refine this structure somewhat.

Consider for example the case where a person crawling on all fours is detected by the camera:
The system might be indicating a borderline condition between a person and a dog. Here, a simple ON/OFF decision type might not be able to provide sufficient definition to enable a satisfactory output.

In such a condition, it might be preferable to first utilise a general sorter which then feeds its results into a final classifier. Such an approach would provide a much increased response certainty, or simply a better system reliability, as each network type will be used in its optimal area.
The actual linking method and internal network types depend very much on the type of data to be analysed, as well as the type of response expected from the system.

## 8 System Output Decision

In this study, we are expecting the system to output a decisive YES or NO when a data stream to be analysed is provided.
When an uncertain condition occurs, it is debatable in a security

application whether a bias should then be applied to the output. Using such a bias can easily lead to a large increase in false alarms, which in turns reduces the effectiveness of the entire system as operator trust decreases. It might be more appropriate to give a "Probable" warning, maybe accompanied by a percentage probability. Such an approach would then leave the ultimate decision to the system operator's own discretion.

In the case of a fully autonomous system, it can be left up to the system installer to induce a positive or negative bias up to a predetermined maximum. Thus the system can be fine tuned to its given operation location.
The actual type of expected output is going to have an effect on the final classification method to be used.

# 5 - Proof of Concept

*A computer cannot turn bad data into good data - John R. Pierce*

## *5.1 - Introduction*

This chapter serves not to develop processes or theories, but rather to prove the viability of a certain level of processing, thus ensuring that the entire hypothesis regarding the extraction and validation of image data has further perspective. It also serves to evaluate various existing processing techniques against the type of data likely to be encountered, were the system considered to be developed further.

## *5.2 - Feasibility Framework*

The main aim of this study is to create a system which will be able to correctly determine whether a shape in an image is a human or not. For the purpose of feasibility evaluation, it should therefore be sufficient to develop a framework system capable of proving certain features:

1. Emulation of camera image input.
2. Ability to apply image filters in such a manner as to preserve intended analysis contours whilst eliminating image aberrations

due to changing light conditions between image frames.

3. Ability to filter images in order to eliminate image differences due to minor effects such as camera vibration and image background motion.
4. Ability to distinguish one or multiple target areas within a processed image.
5. Ability to extract relevant data from the aforementioned target area(s).
6. Correct analysis of resulting data.

This represents a rough breakdown of the entire detection process, and not all of these steps need to be emulated to obtain a confirmation of the studys feasibility. The main concerns are initially covered within the first 3 stages, as these represent the actual data gathering stage. If this initial task is not possible, it would be useless to develop any of the later analysis stages.

For the purposes of this initial feasibility study, it is not necessary to obtain real data, a rough evaluation or simulation of possible conditions is sufficient, as the aim is not to evaluate the success ratio, but purely the probability of success.

Considering the system parameters, there are two main aspects which have to be considered:

- Changing light levels in an image.
- Moving objects within an image.

These parameters can be combined in a number of ways, and for the purpose of these initial tests, the following combinations will be used:

**1 - Constant light levels, changing scene**

**2 - Changing light levels, constant scene**

If these prove to be solvable cases, it can be assumed that more complex combinations of the conditions should also be solvable using adapted algorithms.

## *5.3 - Methods of Testing*

The initial system proposal calls for the object detection process to be functioning by using an image comparison: When the system is initially enabled, a datum image will be taken. This might or might not be updated during the surveillance period, depending on final system architecture and whether the need for this arises or not.

When the system is in surveillance mode, each incoming image (the exact image refresh rate has yet to be determined) will then be compared to the initial datum image.

To simulate this operating condition, we have then to generate an initial datum image as well as a series of surveillance images. The advantage of this process is the amount of control which is available over the images. Parameters may be modified in a supervised manner to observe the system reactions to these changes.

The initial testing process will be limited to observing how well the image comparison process can function, without going into the actual target identification and analysis. It is therefore not of extreme importance if these initial testing images are quite noisy, as long as they fulfil the test parameters:

**1 - Constant scene, changing light levels**

**2 - Changing scene, constant light levels.**

The ideal result from this initial test would be the development of a comparison algorithm which could, with little adaptation, cope equally well with both conditions outlined above, that is, a process which could cancel out overall image light changes whilst being able to identify point changes within the image.

Considering condition 1:

### 5.3.1 - Constant Scene with Lighting Changes

The aim of this test is to develop a comparison process which will ideally report a "no change" condition between the datum image and the camera image, thereby cancelling out all changes purely due to fluctuating light levels.

It is important to consider one point: The light levels in question must be overall light levels. It must be noted that a fluctuating spot light within an image actually represents a change in scene and not a change in light levels.

Whereas the human eye is able to distinguish between these two seemingly obvious condition, this effect may be best explained by using an image threshold example.

The following two images (*Fig.30* and *Fig.31*)have exactly the same scenic elements, apart from the fact that in the left hand image, the small desk lamp is switched on:

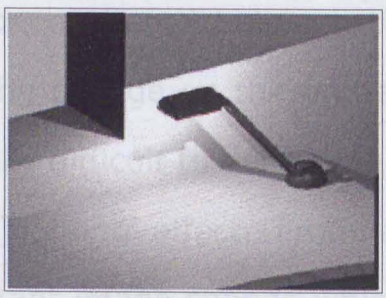


*Fig.30:Lamp on*

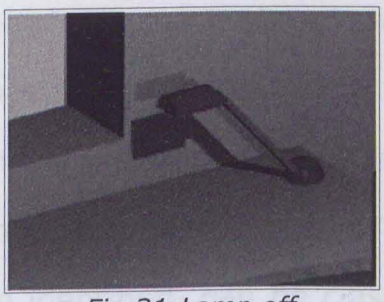


*Fig.31:Lamp off*

To enhance this comparison, both images have been grayscaled and then thresholded to the same value.

Carrying out a direct difference comparison of the two images yields the result seen in *Fig.32*:

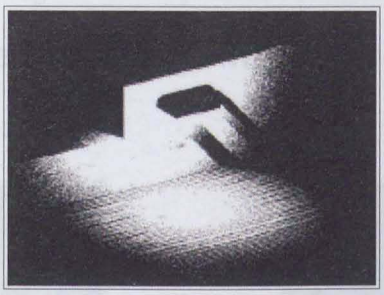


*Fig.32:Lamp Difference Image*

Which would then be interpreted mathematically as the object shown in *Fig.33*.

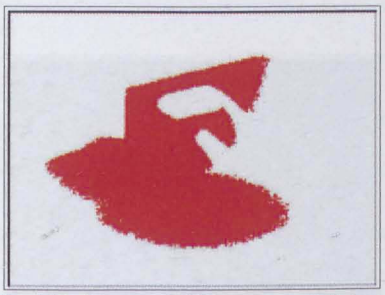


*Fig.33:Lamp Difference Object*

As can be seen, once the reference image of the lamp (which has not been altered between the two images, apart from an overall lighting change on the entire object) has been removed, it is even visually difficult to define the change in the image as being purely a local lighting effect. From a purely objective point of view, the resultant image change which has been identified can be classified as being due to a scene change and not a lighting change. Obviously this type of condition will have to be considered at some stage during the development of the detection algorithms.

For the purpose of the lighting change test, a short sequence of images was taken using the WebBlaster Webcam[*95*]. These are images of a desk under varying room lighting conditions, as seen in *Fig.34*, *Fig.35* and *Fig.36.*

*Fig.34:Light Room*

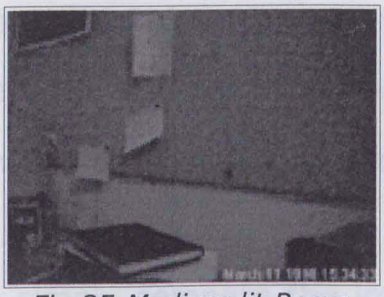


*Fig.35:Medium lit Room*

*Fig.36:Dark Room*

It is important to note that the actual scene in this image sequence has not been altered, only the overall light level is varying.

*Fig.36* however highlights one of the main problems which occur in low-light conditions: image graininess, effectively a drop in image resolution, resulting in an apparently noisier image. This effect and its severity is very much dependant upon the camera/lens combination used. In this case, this low-light aberration is quite

severe and can be tracked back to the low quality plastic lens used on the Webcam. If IR illumination was provided for such low-light conditions, these effects would also be greatly reduced. [*69*]

Considering the sequence of three images, the aim of this process is to try and obtain a zero or near to zero result from a subtraction between two images. By zero must be understood a blank output, thus showing no detectable change between the images compared.

Theoretically, it should be sufficient to simply compare the images using a direct comparison, however, this will not be able to compensate for the overall light change, and objects which are simply less illuminated will be marked as representing a change in scene. In practice, the unit might well be in use over long periods of time, where overall light-level changes would be commonplace.
When considering image light level, or, as represented in a captured image, image colour levels, as general light conditions are reduced, the actual colour range in the image itself is also reduced, i.e., the entire colour distribution gets shifted into darker tones. Accompanying this, the mean light value also dramatically reduces, as can be seen in the following histograms (*Fig.37, Fig.38, Fig.39*). These represent the unmodified colour level distributions for the previous three images.

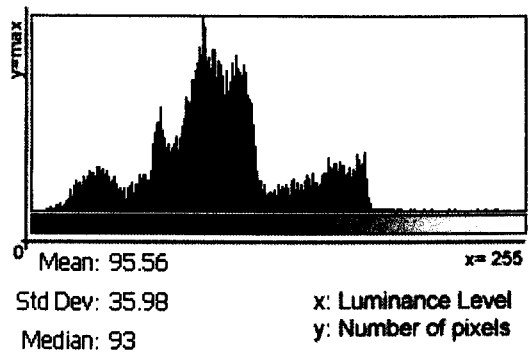



*Fig.37:Light Room Histogram*

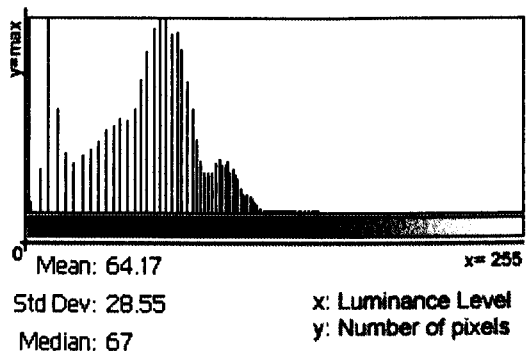



*Fig.38:Medium lit room histogram*

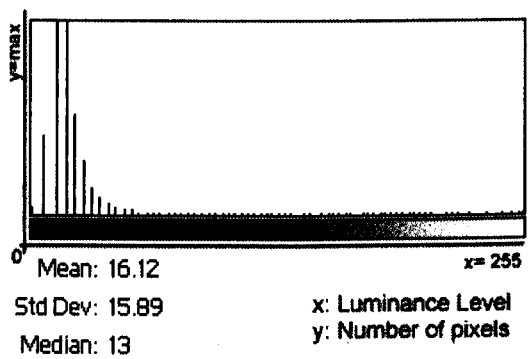



*Fig.39:Dark room histogram*

The median light level value drops from 93 in the first well illuminated image (*Fig.37*) to 13 in the last darker image (*Fig.39*), from a possible maximum of 255.

If we are in any way to compare these images, the overall light

level change must in some way be compensated for. A fairly rapid way of achieving this would be to equalise the mean lighting values between the different images.

Considering the first two images *Fig.34* and *Fig.35*. If the first image (*Fig.34*) represents the camera's datum image, and the second image (*Fig.35*) represents the currently captured surveillance snapshot, this equalisation can be carried out in a number of manners:

1. Equalise both images to a given fixed value.
2. Equalise to datum: the snapshot image will be modified.
3. Equalise to Snapshot: The datum image will be modified.
4. Equalise to brightest/darkest: Depending on current light conditions, the lightest (or darkest) image will be modified to match the other.
5. Equalise both images to a mean value determined by both the datum and the snapshot image values.

There are a number of points which must be considered when selecting the appropriate process:

When the overall light level of an image is reduced, there follows with it a proportionate loss in image detail. Depending on the actual level of correction, this could severely impede the object detection

process.

The use of a fixed threshold can limit the system effectiveness in extreme level changes, such as are bound to occur over a longer surveillance period. The act of artificially attempting to standardise the current light levels can lead to image corruption with subsequent loss of data for further processing.

What is required for maximum flexibility is an adaptation level which will be dependant on each image to be processed.

Comparing *fig.38* and *fig.41*, representing the same image, *fig.38* shows the light level distribution severely biased towards the lower levels (left hand side of graph), with the top third luminance levels effectively missing or only poorly represented. This is an inefficient use of the available luminance bandwidth, leading to a potential loss or restriction of data in certain regions of the image (contrast between different objects is too low to enable efficient detection). Enhancing the right hand side of the histogram, as is the case in this example, has the net effect of lightening the image but simultaneously preserving previously dark areas and actually enhancing the contrast between image components, hence the term luminance stretch, as each component is handled separately.

As shown in *fig.41*, the histogram contour is largely unmodified, but in comparison to *fig.38* makes much better use of the available range.

As is clearly shown in *fig.43*, this is however not a magical method which can somehow fill in missing image detail. Although the full luminance range is now used, the actual data density (i.e. image detail) within the image remains constant, and is actually reduced within a given luminance range (due to the scaling effect). This can also be a source of potential image aberrations, as small errors are also enhanced, and become more significant in a sparser data population then they might previously have been. This is however not critical as an overall detail enhancement has been carried out, it is simply the contrast of data feature to data error which has been increased.

Considering the new useful luminance range, the detail density is actually calculated as shown in *Fig.40*:

$$D_f = D_p \frac{R_p}{R_f}$$

where
$D_f$ :density over the full range
$D_p$ :density over the partial range
$R_f$ : full range
$R_p$ : partial range

*Fig.40:Luminance Range*

*Fig.42* shows the same image, but with the luminance levels now normalised, i.e. stretched to occupy the full available range. The actual transform is fairly simple, and involves identifying the near zero luminance level range within the histograms (i.e. finding which brightness levels within the image are missing), then scaling the

remaining values by a dependent factor. Enhancing the right hand side of the histogram.

The following histograms represent the same three images, albeit now processed in such a way that absent light levels have been chopped off, and the entire image then re-stretched to cover the entire available range, effectively a level stretch.

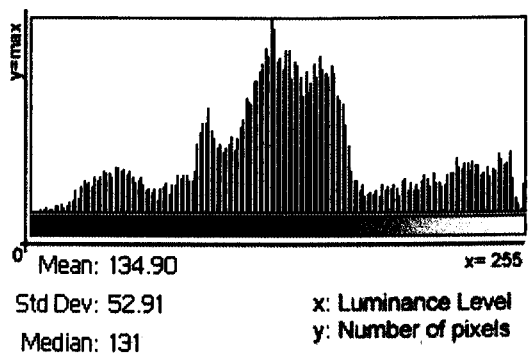



*Fig.41:Medium Histogram Stretched*

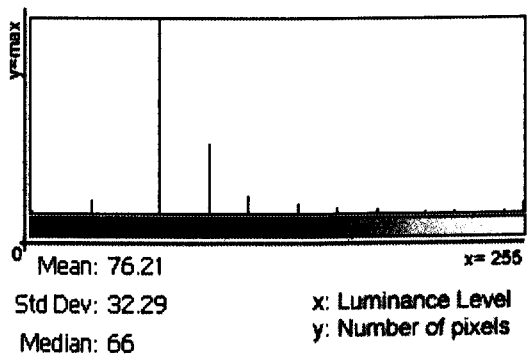



*Fig.42:Dark Histogram Stretched*

The most interesting result can be observed between *figures 37* and *38*. Apart from a few spikes in the lower ranges, the resulting distribution trends over the histograms are very similar, thus

showing that the overall images must also be very similar. *Figure 28* displays the obvious signs of a highly corrected image, with very little density over the entire histogram.

The actual images in their corrected form are shown in *Figures 43, 44* and *45*:

*Fig.43:Corrected Room Light*

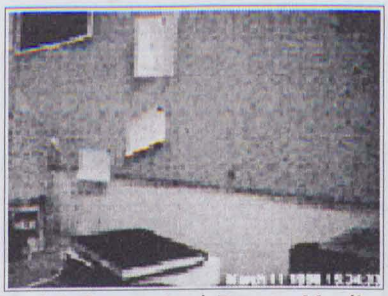


*Fig.44:Corrected Room Medium*

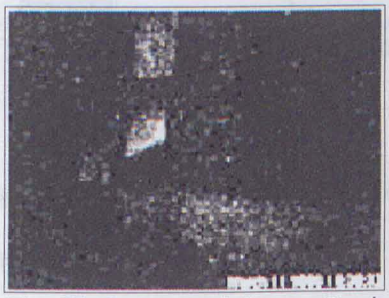


*Fig.45:Corrected Room Dark*

Comparing this sequence to the original sequence, the resulting images are now much easier to compare both visually and mathematically. Image 3 (*fig.45*) is still quite dark, due to the extreme example which was used, a situation unlikely to ever occur if the final system is operating with IR illumination in any form.

One effect which can be noted from the above images is the quite strong haloing effect (in this case a darker circular border to the

image), specifically in image 2 (*fig.44*). This dark image edging is largely due to the poor quality lens used for these sequences which has a relatively high light loss, and would be dramatically reduced were a decent lens used.

These resulting images can now be thresholded to their local median values, resulting in the following sequence (*Fig.46* and *Fig.47*):

*Fig.46:Light Room Threshold*

*Fig.47:Medium Room Threshold*

These may now easily be compared or subtracted from each other to identify the resulting image changes:

*Fig.48:Difference of Fi.44 to Fig.43*

*Fig.49:Difference of Fig.45 to Fig.43*

*Fig.48* shows a simple subtraction of *fig.44* to *fig.43*.

*Fig.49* shows a simple subtraction of *fig.45* to *fig.43*

These reveal the differences in the images once the luminance level corrections have been carried out. As can be noted, the lens aberration causes quite a striking effect in the final comparison images.

Although the results could be used, they are not very refined, with lots of extraneous noise still present.

Another solution, which leads to less interference in the final difference images, is to compare the grayscale images instead of the thresholded images. Doing this would result in more information being compared, thus supposedly giving a more accurate resultant image.

Carrying out such a process provides the results shown in *Fig.50* and *Fig.51*:

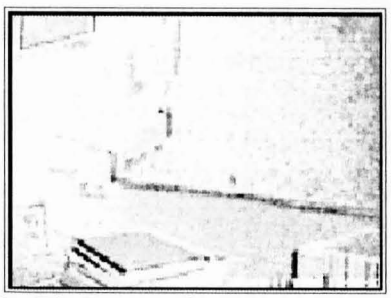


*Fig.50:Grayscale Fig.44-Fig-.43*

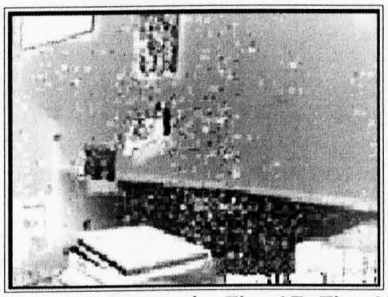


*Fig.51:Grayscale Fig.45-Fig.43*

These images provide more information by giving the actual difference value between the compared images, instead of simply a state of change.

A more accurate result can now be obtained by thresholding the entire image. The threshold value is once again dynamic, depending

this time on the values of both initial images.

Where in *figs.48* & *49* the thresholds for the images were taken prior to the subtraction process, thus providing results based on images with different reference levels, performing a grayscale subtraction results in a more accurate and balanced outcome, as the threshold value is now calculated on the final subtraction image and not on the two subtraction components, allowing the subtraction process itself to be taken into account and subtraction errors to be partially compensated. This calculation is shown in *Fig.52*:

$$T = \frac{\sum_{i=0}^{n} (V_1 - V_2)}{n}$$

$$if\ (V_1 - V_2) > T\ then\ V_3 = 1\ else\ V_3 = 0$$

where
$V_1$: image 1 value
$V_2$: image 2 value
$V_3$: resultant value
$T$: threshold
$n$: number of points considered

*Fig.52:Threshold Value Calculation*

Thus , for the (2-1) (*fig.53*)comparison, the original medians where 131 and 141, giving a mean of both means of 136. Applying this to the comparison image results in the following, the same for comparison (3-1(*fig.54*)):

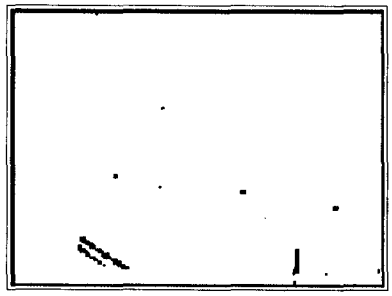


*Fig.53:Threshold of img2-img1*

*Fig.54:Threshold of img3-img1*

This approach displays a much improved response, at least in the field of overall light change compensation, which was the aim of this initial experiment. It might be advisable to shift the final thresholding function to represent the maximum available value, in this case 141, so as to not risk losing too much detail in the final resultant image.

The resultant from this operation may be seen in *Fig.55* below:

*Fig.55:Maximum Threshold*

Given the small difference between the two images, using even the maximum threshold value does dot affect the result very much.

## 5.3.2 - Constant Lighting, Changing Scene

For the purpose of this test, the following two images *Fig.56* and *Fig.57* were used:

*Fig.56:Datum Image*

*Fig.57:Camera Image*

By providing a situation with mainly back lighting, the problem of object cast shadows has been largely eliminated in the above sequence. Whilst this is an ideal condition which we assume to encounter very much in actual live situations, it is ideal to carry out this change of scene test, as some of the possibly disturbing parameters have been cancelled.

As for the first test condition, exactly the same calculation process will be applied to the above images. If the results are satisfactory, this will represent an ideal condition: a single preparation algorithm able to deal simultaneously with general light level changes whilst correctly highlighting scene changes.

The first stage, image levels correction using clipping and stretching, outputs the following two images (*Fig.58* and *Fig.59*):

*Fig.58:Datum Stretched*

*Fig.59:Camera Stretched*

As can be seen, the images have not changed very much, simply a general lightening effect.

The second stage, grayscale image subtraction, gives the result shown in *Fig.60*:

*Fig.60:Differenceof Camera to Datum image*

This shows nicely how the profile of the person has been correctly identified, as well as a few object edges, which is probably attributable to both shadow effects and camera wobble. Generally these other image artefacts are light enough to be easily filtered out at some stage. Note the vertical light stripe running through the person's profile, due to the already shadowed area in the datum image from the space between door and doorpost.

The final step, image thresholding to the current maximum value, results in the output shown in *Fig.61*:

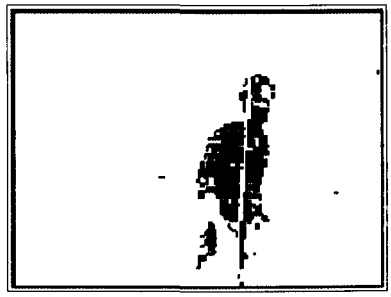


*Fig.61:Thresholded Difference Image of Camera to Datum*

As predicted, the surrounding noise has been cleanly eliminated. It is however interesting to note that the person's legs have also been eliminated.

Whilst this indicates that the comparison algorithm will need to be fine tuned, it must be noted that exactly the same process as for the light level correction has been applied, with quite satisfactory results when the outputs of the process are visually evaluated against expected outcomes [*Appendix A*].

## 5.3.3 - Conclusion

Through simple experimentation using Adobe Photoshop, an image comparison process has been developed which, whilst still requiring refining in a number of stages, provides a satisfactory level of performance.

These results show that the same algorithm can be used to deal with both equalising general lighting changes and detecting local image changes, whilst retaining a maximum amount of information for later processing.

The basic steps of the developed algorithm are detailed here:

1. Image level correction. Unused levels, or levels only present below a certain value (this value must still be defined somehow, whether as a static value or as a dynamic image dependant value) are cancelled out, and the entire image histogram is then stretched to cover the entire available range (0-255 for grayscale), resulting in a general light level correction.
2. The histogram median values of light level for each image are recorded for further reference. The maximum median alue is stored.
3. The two images (camera datum and camera snapshot) are then subtracted one from another to obtain a grayscale difference image.

4. The resulting grayscale difference image is then thresholded to the previously recorded maximum histogram value, resulting in a monotone difference image which can then be used for further processing stages.

It must be noted that the entire process considers each image separately, thus providing a system with a very dynamic response to varying environmental conditions and which retains maximum image information throughout.

The fact that only grayscale images are being used is due to the fact that many surveillance systems currently on the market rely on grayscale (Black and White) cameras for reasons of cost, but also due to the fact that a camera with any degree of IR sensitivity, thus ideal for low light level use, will provide colour images with a strong red component. As explained in the product study, colour cameras come equipped with a red filter to provide more natural images, which would however cancel out any IR sensitivity.

Using grayscale images also allows for a considerable reduction in image size, thus requiring both less storage and less processing power to correctly process. These advantages result in more rapid processing, which must remain a major consideration in an online detection system.

## *5.4 - Application*

Now that a rough outline of a possible functioning preprocessing procedure has been proposed and found to be performing to a suitable degree, the entire process must now be refined and fine tuned to the application at hand.

It is important to always consider the fact that this system is intended for real time application with a limited hardware resource, so any algorithms should be kept as simple as possible, whilst still retaining a correct volume of data for correct image analysis.
Additionally, as the system is intended to be embedded onto standard hardware, any coding must take into account the fact that many of the advanced graphics handling routines currently available within the PC environment, whether due to programming API's or hardware advances in graphics handling will most probably not be available in the final system.

## 5.4.1 - VosDemo

Before any further experimentation can be carried out in the area of initial image comparisons and processing, the processing stages which have previously been roughly defined using Adobe Photoshop must be converted into custom code, which will provide more flexibility with regards to fine tuning the system and applying it to better customised analysis processes.

The result of this initial coding is **VosDemo**, a program which is designed to interface with the Creative Labs WebBlaster.
This allows the user to first select a datum or reference image, then either manually select an incoming camera image, or set the incoming image update onto a preset timing sequence, thus allowing "hands-free" operation.
When the camera image has been selected, both images are processed and compared, to obtain a resulting difference image, which is then overlaid over the currently selected camera image.
The full sequence is as shown in *Fig.62*:

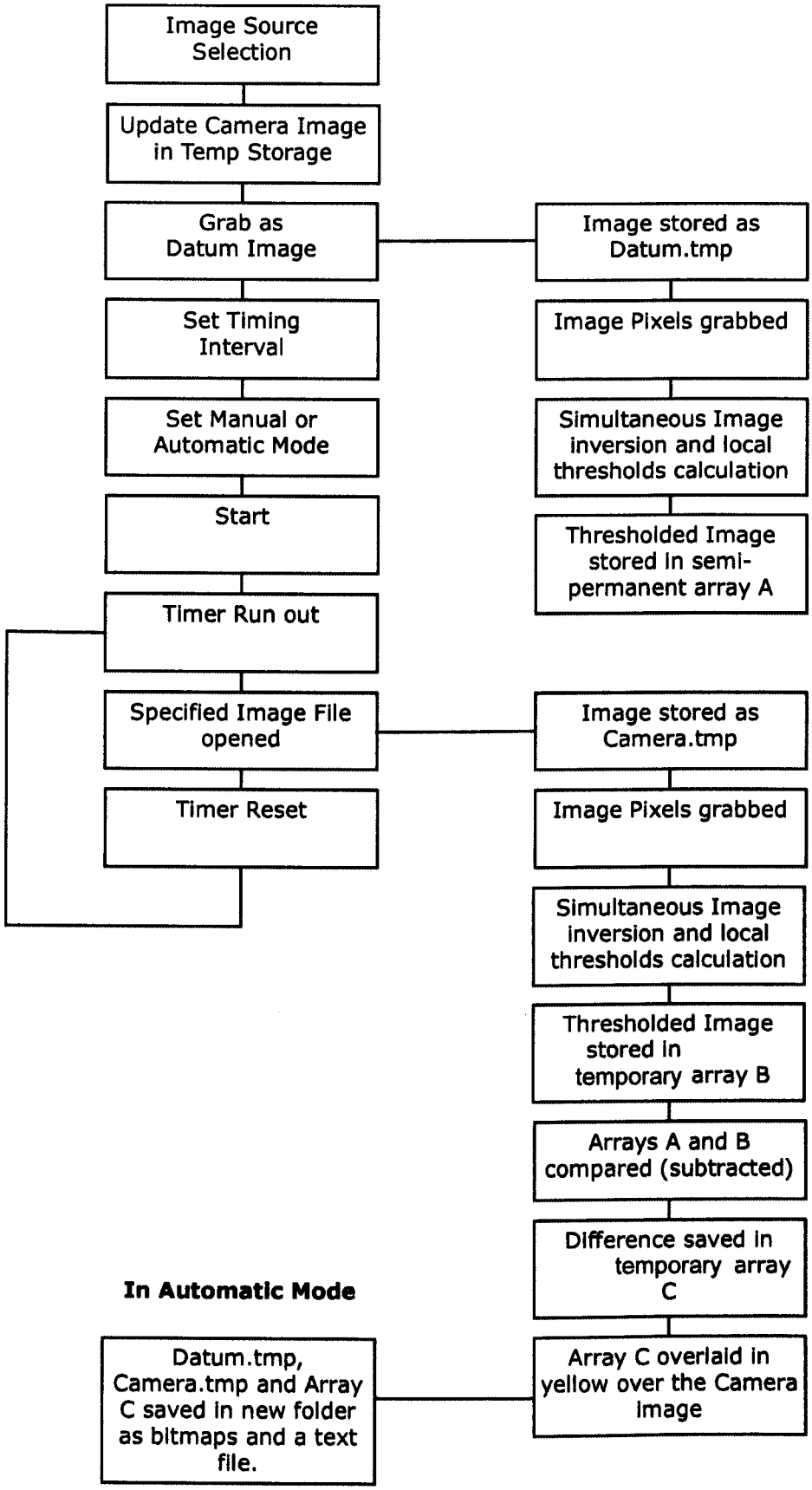



*Fig.62:VosDemo Operation Sequence*

As can be seen on the flow diagram, no image correction processes are currently included into the VosDemo sequences. The only process which is as yet adaptive to the image is the dynamic threshold calculation, which is in any case necessary for the correct colour to grayscale transformation.

Whilst Vosdemo can import full colour images, these are immediately converted to Grayscale, the reason being an enormous savings in required storage space as well as subsequent processing time.

As this initial image comparison process is only using the thresholded Datum and Camera images, with neither of these stages being affected by the other image, this is simply a matter of reducing the actual processing code.

Should it be determined that the colour or grayscale images themselves need to be dynamically adapted to one another, this approach will obviously have to be modified to allow for a full grayscale or colour image storage area within the detection system.

The output of this version of VosDemo is a simple text formatted file which stores the values of the set pixels resulting from the image comparison process. This can then be easily analysed or processed at a later date.

*Fig.63* and *Fig.64* are a couple of screenshots showing VosDemo in

action:


*Fig.63:VosDemo 1*

The selected datum image is displayed to the left, and the incoming camera image to the right


*Fig.64:VosDemo 2*

Here can be seen the resulting difference image overlaid on the Camera image

Considering the individual stages of the transformation algorithm:

When the initial image pixels are grabbed, these are stored as RGB values in a set of 3 dedicated 320x240 arrays, one for each colour

band. These three arrays are then used to calculated the grayscale threshold of the image. Simultaneously, the pixel parameters are inverted to obtain an inverted image matrix as shown in *Fig.65*:

$$T=\frac{\sum(255-(aR+bG+cB))d}{320*240}$$

where T=Threshold
and where the parameters a, b and c are defined in RGB to grayscale conversion

*Fig.65:Threshold Calculation*

According to this threshold value, the calculated pixel will be set to black or to white, and stored in a temporary array.

The multiplication factors *a, b* and *c* are generic values for transformation from colour reference to a grayscale mode, calculated to represent the normal human perception of colour distribution in light. They are, respectively for Red, Green and Blue: 0.3 - 0.58 - 0.12 (See *Fig.66*)[*47*]

$$G=0.3r+0.58g+0.12b$$

*where*
$G$: grayscale value
$r$: red component value
$g$: green component value
$b$: blue component value

*Fig.66:Grayscale Transformation*

This calculation is carried out once for each new image. When a datum image is grabbed, this info will simply be passed over to a

new array.

When an image comparison calculation is carried out, we are simply doing a straightforward subtraction of temporary arrays A and B, so as to speed up the code execution cycle. There are a number of factors slowing down the detection process, which would otherwise not be present on a dedicated system:
Firstly, the image is being grabbed pixel per pixel by the camera interface software, and this is being translated into a jpeg format. Once this is completed, Vosdemo then grabs this composited image and decomposes the image once more into its original pixel structure. Obviously the larger the image, the longer this process will take. Once these pixels are grabbed and the total image threshold calculated (executed in a single cycle, one cycle being 320x240, i.e. 76800 groups of calculations), the threshold then has to be applied (second cycle) and stored (third cycle). The actual image comparison is also carried out in a single cycle, and saving the difference image will be a cut down cycle, as only those pixels showing an actual difference are saved. This represents a total of 384 000 groups of calculations per active comparison.

If the entire system were task dedicated (i.e. hard-coded), the initial transformation into a valid image format , and the subsequent re-transformation into separate pixel structure would be obsolete. We would not be needing each input to be displayed, and thus only

the difference image would be calculated and stored temporarily to be fed into the detecting network.

One factor which is going to affect both systems is the size of the image and the type of camera used. If we opt to carry out detection using a monochrome camera or one with IR capability, we will be losing the entire set of colour information, which represents twice the entire image size, i.e. 2x320x240 or 153,600 chunks of data, where each chunk could vary from being a single bit, to 8 bits of data (considering images varying from purely monochrome data to 256 colour distribution).

Considering the advantage which can be obtained through using IR capability cameras as far as detection is concerned , we can also appreciate the substantial amount of data compression or reduction which can be brought about by their use.

If the thresholding function, whose output is a monochrome array defined purely by the image dimensions and not the camera colour definition, is hard-wired (i.e., implemented through an electronic circuit rather than through computing emulations), then we are considering a situation where the software will be dealing with anything from one third to one twenty-fourth of the amount of data as compared to what it is currently having to cope with. The software will then purely be dealing with the image comparison

cycles.

## 5.4.2 - VosReader

VosDemo is designed to carry out only the initial image comparison, but none of the subsequent processing steps.
This very structured approach is intentional in order to facilitate the performance analysis of the various stages in the entire system.

VosReader is a stand-alone application which uses the data created by VosDemo. VosReader displays the datum and the camera images in two small side windows for purposes of clarity. The main central window is used to display the saved change image, in full size.

The user is then able to manipulate this main file through the use of pre-coded or custom designed filters (which can be stored). The final output can be saved when the user is satisfied about the type of filtering achieved. Currently, the user is limited to filters with an aspect of 3x3 or 5x5, although the range could be increased to include 9x9 pixel filtering.

As can be seen, VosReader is concerned solely with the image post-processing aspect - This is the stage at which we can determine exactly what type of data is going to be fed into a network for

further analysis.

That this analysis is not occurring in real-time here is not a problem - We are at the stage of defining various types and sequences of filtering. VosReader is simply presenting a highly visual experimentation platform, allowing filter effects to be displayed immediately, or corrected if not adequate.

In the initial stages of recognition network development, we will not be running in real-time. The actual network training stage will be requiring large quantities of variable data to be available, and once an adequate filtering algorithm has been decided upon, it would be quite a simple matter to write a separate program capable of dealing with a few hundred or thousand files in a batch manner, without any visual clues, so as to speed up processing time and avoid any unnecessary programming clutter.

*Fig.67* shows VosReader in action:

*Fig.67:VosReader*


Let us now observe the operations sequence of VosReader (*Fig.68*).

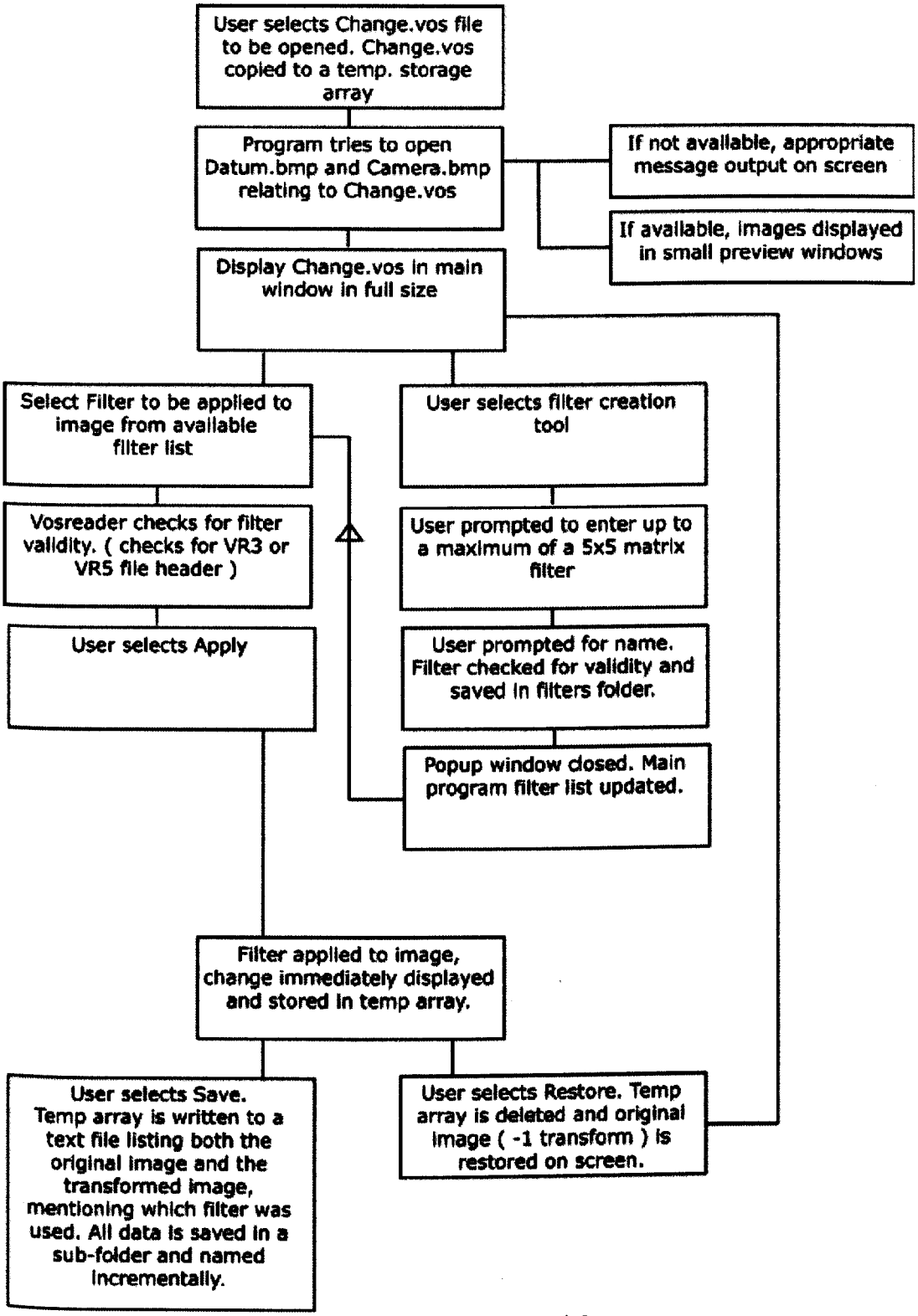



*Fig.68: VosReader Operational Sequence*

Due to the fact that VosReader can immediately display the results

of user-made filters, it becomes quite a powerful tool for this initial development stage.

The actual filtering process in itself causes a number of issues, which are considered here:

When Change.vos (the output of VosDemo) is opened, the contents of the file are copied to a temporary storage array. This array will be used for all further transforms, leaving the original file uncorrupted.

When applying a filter to an image, we face the problem of edge filtering. There are a number of ways to go about this task:

Apply the full filter on the entire image, accepting slight filter degradation at the image edges due to only partial filter activation, as illustrated in *Fig.69*:

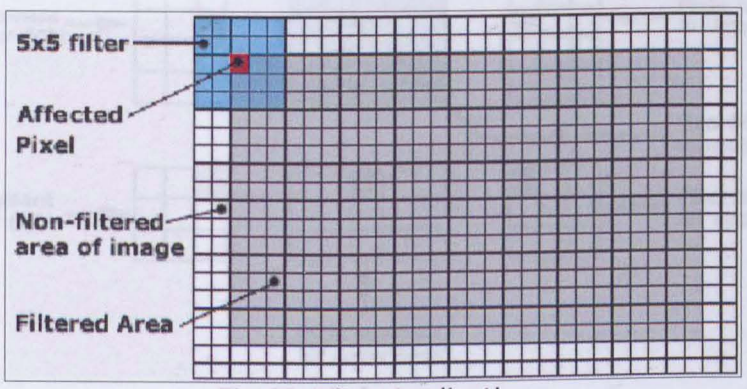



*Fig.69:Filter Application*

This shows a 5x5 filter being applied to the image, but leaving a 2

pixel wide border around the entire image which will not be affected by the selected filter.

Whilst this might not be readily noticeable to the human eye, such an approach could lead to quite serious data loss or data corruption, and is therefore not suitable to the application at hand.

A more appropriate approach would be to use an adaptive filter which is modified when applied to the edges of the image. *Fig.70* shows how the filter itself is split and adapted to be applied to the various image edge segments.

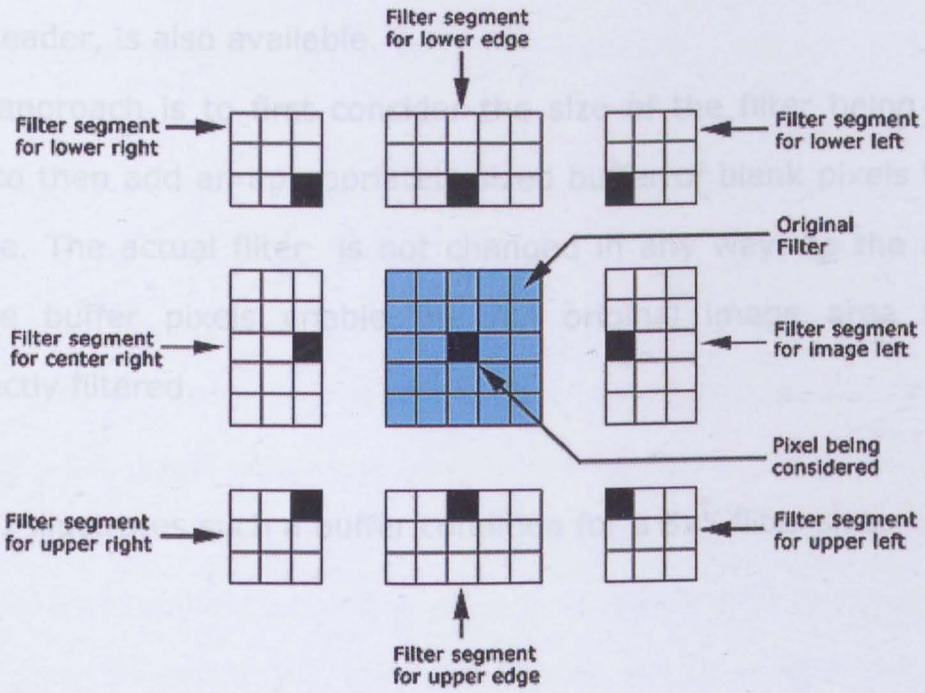



*Fig.70:Adaptive Filtering*

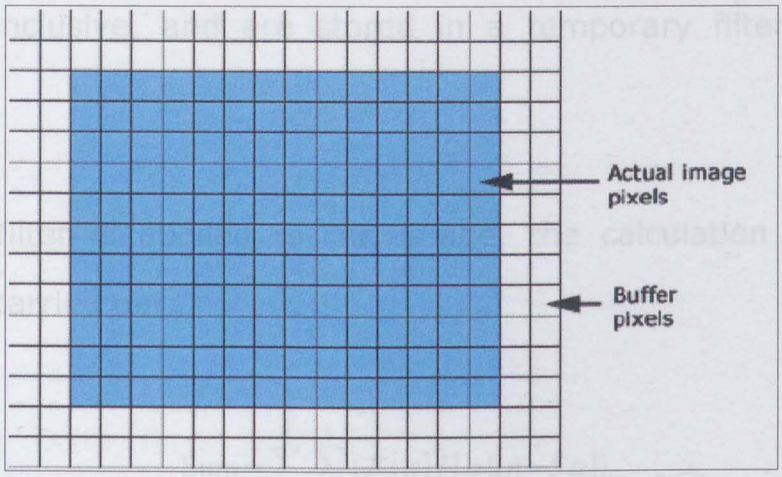
The illustration above is not strictly accurate, as the filter segments shown would only be applicable for actual image edge pixels. The second row of image pixels would require yet a further level of filter segments where the considered pixel would be inset into the filter segment by a single pixel filter column.

As can be observed, the above method, whilst very accurate as far as filter application is concerned, involves some rather convoluted and extensive filter adaptations, which become ridiculous when larger filter sizes are considered.

A third method, the one which was eventually adopted for VosReader, is also available.
The approach is to first consider the size of the filter being used, and to then add an appropriately sized buffer of blank pixels to the image. The actual filter is not changed in any way, as the added image buffer pixels enable the full original image area to be correctly filtered.

*Fig.71* illustrates such a buffer condition for a 5x5 filter size.

*Fig.71:Image Buffer*




In the case of a 3x3 filter, the image buffer size need only be one pixel wide. These buffer adaptations can easily be carried out at runtime, although the option chosen in VosReader was to set a maximum filter size of 5x5, with a fixed image buffer width of 2 pixels. Any smaller filters are automatically resized using null values.

The input to VosReader is a .vos type file, which is purely a text listing of all activated pixels within a given difference image. VosReader transforms this listing into an image matrix with values from -1 to +1. The initial default matrix is set to -1 (all deactivated) and any set pixels within the image map receive a value of +1. This representation allows for the buffer zone to added on using null values. As explained later, the value 0 allows all filter calculations to be cleanly cancelled out, thus not biasing the final filter output for the actual considered image.

All filters in VosReader may have real values, varying between -1

and +1 inclusive, and are stored in a temporary filter array, F [1..25].

When a filter is applied to the image, the calculation shown in *Fig.72* is carried out:

$$\text{Sum} = \sum_{a=0}^{4} \sum_{n=1}^{5} (F[n] I[a][n-5a])$$

*Fig.72:Filter Application*

where the image is stored in an array I[height][width].
As the filter is always applied as a multiplication with the image pixel values, the previously set null value buffer zone has effectively no outcome on the actual image filtering process.

The resultant value of **Sum** is then considered.
If **Sum > 0**, the pixel considered by the calculation (I[2][n-3]) will be set to **+1** (activated), otherwise it will be set to **-1** (deactivated).

To prevent distortions in the filter application, the actual filtered image output is kept separate from the original data. The filter itself is only applied to the original image data, thus preventing cumulative filter effects during a single filter application. Once the filtering process is complete, the original data is completely replaced by the filtered output for use with further filter

applications.

Once a filter has been applied, the user has the option of saving the new resultant image, in which case it will be written, along with the original image data, into a tab delimited text file (allowing for easy editing in a spreadsheet application), listing the name of the original file as well as any filters applied.

VosReader features an undo function to restore one filter step, in case the results are not as expected.
Filters will however work in a cumulative fashion, allowing the user to apply the same filter to an image a number of times in succession to obtain enhanced effects.

## *5.5 - Artificial Data*

### 5.5.1 - General Considerations

At this point, there now exists the possibility to process images in a fairly rapid and flexible way using a combination of both VosDemo and VosReader.
These enable us to distinguish and isolate the differences arising within two separate images, once the said images have been processed for features such as background noise reduction.

We know from experimentation, that the visible image difference can be correctly extracted. It remains to be determined whether the next step can also be successfully carried out: analysing and classifying the resulting image difference.

A number of questions arise when considering the implementation of a neural network as a classifier in this context:

- Can a network correctly define and determine a person, given only variable 2 dimensional data ?
- Will a network be able to consider and compensate for scaling effects and distance factors and be able to distinguish these from pure size differences ?
- How can the most appropriate data for the network be determined?

In order to answer these questions, it is initially necessary to examine which data can, regardless of the final application, be extracted from the resulting difference image at all. It is however important to note that this extracted data must be data which is common to all difference images to be examined.

Due to the way a network functions, we must ensure a certain consistency in the data types. A network cannot be expected to know that the first line of data for one image represents the height of an object, but in the second image this same data line represents, for example, the object surface area. This would simply lead to absolutely unreliable and nonsensical network outputs as the system would attempt to compare totally mismatched lines of data to each other.

## 5.5.2 - Data Parameters

The data which is to be used as network inputs must satisfy the following conditions:

- The quantity of data must be constant.

  Each image must provide the network with the same amount of data, thus providing a constant and previously defined set of inputs. This ensures that data interdependent relationships remain uncorrupted.

- The type of data must be constant for each input.

  For each input, the type of data must remain constant, as the data will most probably be undergoing a preliminary preparation stage. This will be a specialised process for each line of data, thus precluding the option of swapping or mixing data input lines.

- The data range must be definable for each input.

  As it is unlikely that the data extracted from the image will immediately be available in a usable range for a network (-1/+1), each input line will, during its preprocessing phase, have to be scaled down by a certain amount. In order for this to be constantly successful, the maxima and minima of the considered line of data must be previously known for all possible situations. A dynamic process might be used, where data is balanced relatively within a single set, but this would be a much more complex and sensitive approach, as each incoming data line would have to feed both its minimum and maximum values in order for correct analysis to take place. The danger in such an approach is that the otherwise existing link between different data sets is now lost, which could easily lead to data corruption, as the actual internal structure of the analysing network would have to be individually adapted to each new data line.

Although the parameters of this experiment have been broadly laid out, given the known camera specifications, the first sequence of

testing (proving the concept through the use of a classifying neural network) should not be run using pure live data, i.e., images taken directly from a live capture sequence.

The reason for this lies purely in the sheer complexity of such data. An uncontrolled environment provides too many unknown image parameters which might ultimately unknowingly affect the outcome of any testing.

For this primary network testing phase, it will be necessary to generate a set of strictly controlled images where all variables are known. The complexity of this set can then be gradually increased as the system is developed to eventually represent and/or include actual live data.

A broad set of controllable features can then be defined and manipulated according to the level of complexity which can be accepted by the network at any one time.

These features are:

- Type and Colour of Background.

  Controls the background lightness and reflectancy.
- Number of objects present.

  Where an object defines anything not directly linked to the scene background.
- Type of objects present.

  The type of an object is primarily divided into two main classes:

Animate and Inanimate. A rougher classification may also be achieved with the definition: Target or Non-Target.

- Colour of objects.

  The colour of an object as related to the scene background. This will directly affect the ability of the system to accurately locate the said object.

- Object Positions.

  Positions relative to the camera. This is effectively the point of view of the object. This is to be considered mainly for the vertical offset between object and camera, which will lead to more or less severe proportional distortions.

- Pose(s) of person(s) present.

  Especially important during the initial development stage, this is a controllable factor when considering comparisons between for example a crouching or crawling human and an animal such as a large dog.

- Illumination.

  Both overall and point illumination which will affect the scene through direct or indirect shadowing.

- Presence and quantity of noise.

  Purely for artificial data. A controlled quantity of randomly distributed noise can be applied to the image in order to simulate effects such as camera vibration or interference.

- Type of Noise.

  This will determine the distribution type of the noise (even or

random) as well as other items such as colour variations which are more likely to cause image distortion.

The initial experiment which is to be carried out is to evaluate the response of a simple network when presented with simple target and non-target images. The aim is not to determine the noise-resistance of the network by introducing many variable parameters, but rather to simply observe the feasibility of using such an approach.

As a source of data, we could use images from live captures under strictly controlled conditions, but this is likely to introduce a number of uncontrolled variations in positions, lighting and image noise, even were the data to be initially manually "cleaned". A much preferable source of data would be from fully artificially generated images: a scene can be set up digitally with a fixed number of parameters which may then be exactly controlled according to the test requirements.

Metacreations Poser[*93*] was used in this task, as it allows anatomically accurate human modelling whereby the age and sex of the person being modelled can be accurately controlled. Such parameters are important as the profiles of males and females in various age ranges present quite marked differences.

Although it is not necessary at this early stage to model all possible combinations, it is important to evaluate the flexibility of even a simple test network in the way it can adapt to such shape variations

whilst presenting a constant alarm output.

The models produced by Poser[*93*] were then introduced into a controlled scene create in 3D Studio[*96*], which allowed for an exact background and lighting setup.

## *5.6 - Validity Testing*

This initial test or series of tests has the aim of confirming the initial observations made on the processes of image comparison and data extraction using the simple algorithms developed with the initial help of Adobe Photoshop filters.

In order to do this, a tightly defined environment must first be defined, thus allowing ensuing test results to be objectively evaluated.

### 5.6.1 - Test Environment

This initial test relies on observing purely the actual shape or profile of a human as opposed to an assortment of other objects.

The images are presented as inverted shadows, where the background is completely black and the object to be considered is entirely white. This format was selected as it is part of the default settings for the human modelling software Poser.

All external light sources were cancelled out, resulting in a perfectly flat or 2D image with no information on the object distance to camera nor on the scene light source, as shadows are simply not being modelled in order to reduce possible image aberrations to a minimal level.

As the prime task was not to evaluate the accuracy of the separation process, only a model of an adult male was used for this test phase. This human figure was presented in an variety of standing poses, seen from two distinct camera angles representing realistic camera mounting heights (between 2.5 and 3.5 m depending on the actual target aspect).

Negative targets were provided by various modelled objects such as lampposts, chairs, letters of the alphabet as well as simple geometrical shapes in a number of combinations. Care was taken to present negative targets in a variety of positions and with varying area densities within the object limits.

## 5.6.2 - Test Data Set

The final data set for this initial test was comprised of slightly over 2000 images stored in non-compressed monotone bitmap format and using a standard size of 320x240 pixels at 72dpi. This reflects one of the available formats from the Webcam being used for live data capture. This also represents an industry-standard format , representing a CCD element of 230Kpixels.

The storage format of this initial data set will also allow for easy modifications to the original images in order to generate subsequent more complex image sets without running the entire scene generation process again.

With the current data set, the need for an initial image comparison process is avoided, as the object to be observed is already presented in its "pure" form, and we can progress directly to the various phases of target detection, separation and target data extraction and analysis.

## 5.6.3 - Image Evaluation Methods

We have now a valid set of controlled sample data, but as yet no way of analysing the data within these images. A number of approaches can be considered, which may be split into two main groupings:

1. Image pixel analysis.
2. Data feature analysis.

### 1.Image Pixel Analysis

Whilst the overall concept of image pixel analysis is very simple,

requiring little or no data preparation prior to network presentation, it does also have its own intrinsic problems and limitations.

Image pixel analysis roughly involves analysing every single pixel of each image via a dedicated network.

The first and major undesirable feature of such an approach lies in the sheer volume of information to be processed. Considering the image format adopted for this experiment, a 320x240 pixel image results in a total of 76800 individual pixels.
Every single one of these would then require its own input node within an analysis network. This is not only ridiculous in that much non-valid data will be processed, but would also be placing highly exaggerated hardware requirements on the final system. Were such a network to use only a single hidden layer of 10 neurons with a single output flagged to high or low, this would result in a minimum of 76800 *10 + 10 = 768010 multiplication processes per image, assuming ideal conditions.

Quite apart from the pure dimensions of such a network, other problems would also arise within the aspect of data presentation. Unless the target were to cover most of the total image area, the effect of overall scaling would contribute to reducing the final effect of the object on the overall image.

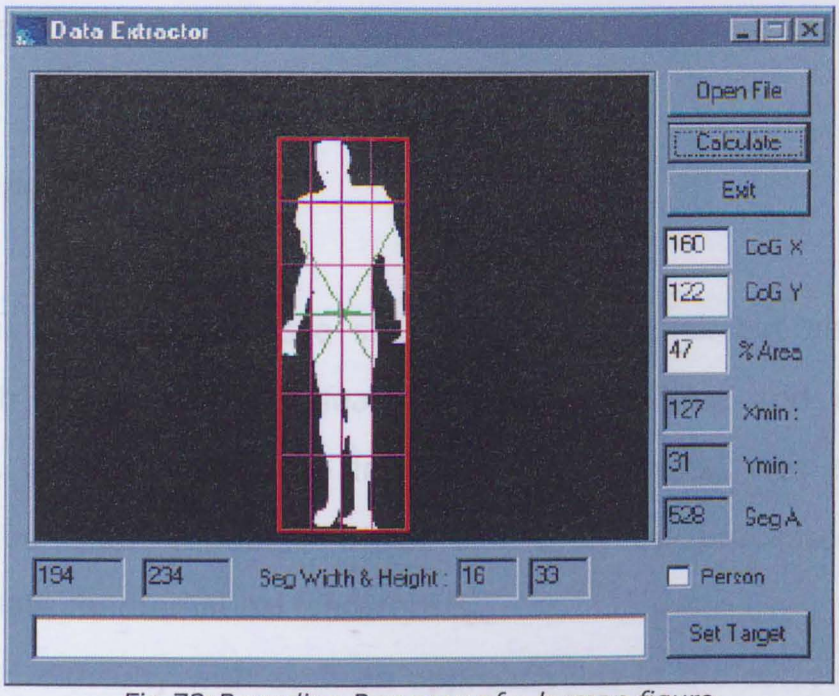



*Fig.73:Bounding Box area of a human figure*

*Fig.73* shows the bounding box around an average human figure.

It has been determined through experimentation (analysis of a few thousand images), that the area actually occupied within this bounding box by the human shape normally varies between 40 and 50%. Thus, even in the unlikely event of a targeted object covering the entire image, the number of affected pixels (for a human figure) would lie roughly at 50%.

This also means that 50% of the data to be evaluated would be unnecessary clutter, reducing the effectiveness of the affected pixels.

Along similar lines, were an image to contain a high level of distributed noise, this could effectively drown-out the influence of a potential target, especially if the latter was to be relatively small as

compared to the overall image area.

## 2.Data Feature Analysis

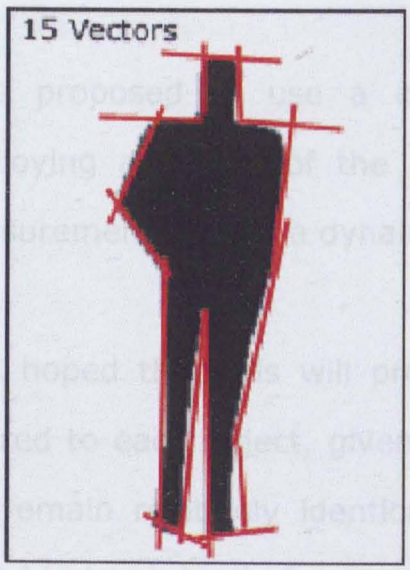
Data Feature Analysis is a slightly more complex but also more thorough approach to evaluating the image data, and basically involves deconstructing the image to be analysed into a set of predefined data variables which can then be processed using appropriate mathematical transforms.

The main difficulty with such an approach lies in the correct choice of data to be extracted and appropriate data processing algorithms. Referring back to a discussion with Dr. P. Rosin [*Appendix F*], the simplest types of data (those extracted directly from the image relationships without any attempts at mapping or interpretation) are often the most reliable, regarding their consistency over a set of images with varying parameters.

In this situation, the types of targets which are processed are constant neither in their aspect to the camera nor in their overall shape properties. This makes it difficult to use a recognition system based on algorithms dependant on vectorial matching, as each object will have to be described using a varying quantity of vectors to perform a satisfactory object description, unless a highly deformable vector template is used. This approach has been discussed earlier, and although it is adopted in many systems, its disadvantages are deemed too high, requiring increased analysis of

the resultant output, whilst the aim of this study is to restrict the processing stages to an acceptable minimum.

Examples of such a vectorial approach can be seen in *Fig.74*, which both present an adult male facing the camera. It can be observed that the slight change in position causes a dramatic increase in the number of vectors required to accurately describe the already much simplified profile given. If we then consider that such poses as crawling or crouching must also be taken into consideration, it becomes obvious why an analysis based on a fully vectorial description of the image becomes unsuitable. Vectorial measurements might still become useful for conditions such as describing the overall direction of an object within the image, but these will then be used in conjunction with other standard measurements.



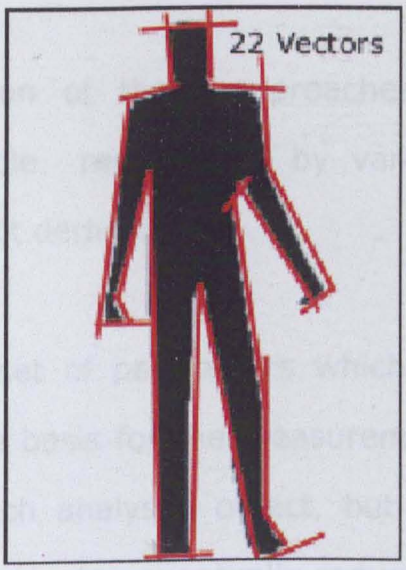



*Fig.74:Two human profiles showing possible vectorial descriptions. Note that the vectors shown here are for illustration purposes only and have been greatly exaggerated to facilitate viewing*

The use of defined templates, such as those used by Tate and Takefuji [*81*], is a further derivation of the vector based approach. This does have the advantage of offering a fixed number of points off each image from which the final vectors can be derived, but also has the disadvantage of having to select a grid with a resolution fine enough to allow an accurate description of the object being studied, and the necessity to provide matching templates to evaluate the actual parameters obtained from these objects. This does seem to be a fairly lengthy approach, finally relying on a fixed and artificially created parameter set (the actual templates) which will be the actual point of failure if a given data set has not been sufficiently described.

It is proposed to use a combination of these approaches by employing a subset of the deformable represented by variable measurements within a dynamic, object derived grid.

It is hoped that this will provide a set of parameters which are tailored to each object, given that the basis for the measurements will remain relatively identical for each analysed object, but also scaled independently for each. This eliminates eventually redundant

data created by many measurement points on a large object, ensuring that the same number of measurement parameters are used, regardless of the observed object dimensions.

An added advantage to considering standard dimensional data is also the relative ease of obtaining such data, which will speed up the entire data extraction phase, an advantage for a system working online.

### 5.6.4 - Image Data Extraction

In order to extract the intended data, it was necessary to develop a number of custom software packages, which will be explained in the following section. The software development process has been intentionally split up into a number of modules which make the entire process much easier to modify, even if the final result is quite far from operating in real-time. These processes are however designed in such a way that they can ultimately be joined up into a single faster and more concise application which will then be able to fulfil the primary requirements of this study.

Let us now consider a rough overview of the entire data extraction process for this initial experimentation series:

- Obtention of bitmap image file, single target object per image.

- Transformation of image to proprietary .VOS format, effectively describing the image in a monotone fashion editable in spreadsheet applications and other custom software.
- VOS file then fed through VosDataExtractor to obtain a maximum number of lines of data from the image's target object. This results in the creation of a .VDF file, which is an annotated text file containing all the extracted data in a standardised but as yet unmodified format.
- VDF files can then be compiled by a custom batch processing utility to be collated into a usable table of data for further analysis.

The first two steps in this process have already been described in detail, and the operation of VosDataExtractor will now be explained. The process is as follows:

## *5.7 - Vos Data Extractor- VDE*

For a given image (using the afore mentioned test conditions specifying images containing a single target and no background noise), the target must first be located. This is done by simply finding the first set pixel within the image by scanning from the top left corner and progressing from left to right and top to bottom.
Each pixel thus found is defined as a new object. Objects which are touching or within a certain distance of each other are then grouped

into a single object.

Once this has been achieved, the furthest extremities of the target object are defined (by scanning each line of the object and determining the minimum and maximum extension values) and placed within a bounding box. All further processing is now carried out only on the set pixels within this newly defined object-dependent boundary.

Within the obtained bounding rectangle, we can then extract a number of measurements on the actual object or the box, such as total box area, area of box set by target, box height and width and object Centroid position. From the Centroid, 6 radial measurements are taken at 60 degrees to one another, specifying the maximum length to the edge of the target. Additionally to this, the bounding box itself is split into an array of 6x4 smaller segments and from each of these segments is extracted an extra set of measurements specifying the segment area and percentage area set by the target.

In this manner, a total of 41 independent lines of data are obtained from each image, the last value being a manually controlled boolean value specifying whether the currently considered target is human or not. This final value is necessary as a reference for later network development work, in order to provide a basic set of training data with known output values.

*Fig.75* illustrates VDE in operation, with certain of the extracted data values overlaid on the actual image target:

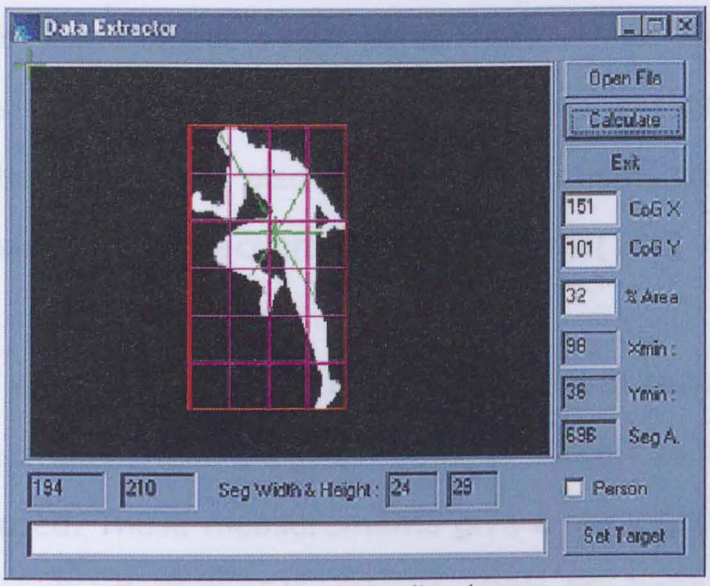



*Fig.75:The main bounding box*

split into its 6x4 grid and the 6 vectors emerging from the Centroid can be clearly seen.

The actual sequence of operations is as follows:

1 - VOS format file opened and stored in a 320x240 array, ranging variables are initialised.

2- Entire image is scanned until a set pixel is encountered. Current coordinates stored in a temp value. Once the entire image has been scanned, the overall object bounding box limits have been defined and stored. Simultaneously, the percentage area of the bounding box occupied by set pixels is evaluated. (this current version is

limited to a single object per image without any form of background noise).

3 - The centroid of the object is now determined in both the X and Y directions. This value can help to determine the orientation of the object within the frame. It is also useful in determining the relative importance of surrounding sections.(see next point)[*100*].

4 - 6 lines are extended from the centroid, arranged radially at 60 degrees to one another, and their last point of contact to the target object is measured. These measurements give a fairly good idea of the overall mass distribution of the object within the bounding box. A similar approach, depending on an object centroid and ensuing radial measurements has been outlined by Tamas Sziranyi [*79*] within the context of motion tracking. These values are used to provide object specific information which can easily be used to distinguish one object from another within a noisy environment.

5 - The entire bounding box is split up into a 4x6 grid of equally sized elements, and for each element a measurement of percentage area set calculation is carried out. This provides a more detailed view of the contents of the total object box. The actual grid size provides a good detail resolution without providing too many different measurements which have to be individually evaluated. This grid, dependent on the actual object dimensions, ensures a

consistent level of measures detail for all objects irrespective of their dimensions, and guarantees the consistency of the extracted data over a full image set, regardless of the position or size of the object to be analysed.

6 - For each image, a final value is manually set, defining whether the target currently considered is to be finally classified as human or not. This value is required during the training stage of a simple network as a final check value, but will not be used as part of the network inputs.

Below is a complete listing of the data taken from each image and the ranges considered for each line of data:

| ***Data Value*** | ***Data Range*** |
|---|---|
| Bounding Box X min | 0-319 |
| Bounding Box Ymin | 0-239 |
| Box Width | 1-320 |
| Box Height | 1-240 |
| Box Area | 1-76800 |
| Centroid Xpos | 0-319 |
| Centroid Ypos | 0-219 |
| Weighting | 1-3 |
| Segment Area | 1-3200 |
| Segment 1-1 | 0-100% |
| ... | ... |
| Segment 4-6 | 0-100% |
| Radial 0 degree | 0-320 |

| Data Value | Data Range |
|---|---|
| Radial 60 degree | 0-271 |
| Radial 120 degree | 0-271 |
| Radial 180 degree | 0-320 |
| Radial 240 degree | 0-271 |
| Radial 300 degree | 0-271 |

It must be noted that these are the pure capture values, which will most probably require some form of preprocessing prior to being used to develop or run a classification network.

The centroid Xpos and centroid Ypos value cover the maximum available data range, as it is entirely possible to detect an object which will be aligned to the edge of the captured image. Although this is unlikely to be representing a target, the data range still needs to be considered as part of the object processing sequences.

## *5.8 - Initial Network Creation and Evaluation*

### 5.8.1 - Data Considerations

Given the previously laid out conditions for a set of artificial images, and the above described data set, we are now at the stage where an initial network might be developed in order to prove the ability to classify human and non-human forms within this scope.

The data which we now have available through VDE is however

unsorted and presented in a raw form, where certain parameters are doubly described and where other parameters might not be using all too obvious scaling and range values.
If a simple network were to be developed using the raw data, it is highly likely that some form of distortion would occur within the network's internal data representations, as certain values with high maxima would be drowning out other more sensitive (and maybe more important) data lines. Simply because 40 odd lines of data have been extracted from each image, does not necessarily signify that all these data lines are going to be crucial to the development of a successful network.

We need to first attempt to map the relationships between the various data lines, determining not only which lines might safely be left out, but also in which manner the remaining data inputs will need to be scaled and transformed in order to present an ideal and balanced set of data to the network. The more we can optimise this initial input data, the more powerful or reliable the resulting network can become, as it will have more internal resources available to actually describing and classifying the data instead of just preparing the raw data to be in a usable form.

## 5.8.2 - Data Preparation

Illustrated in *Fig.76* and *Fig.77* are two plots of raw data extracted

from two different images, the first containing "noise" (i.e., an invalid target), the second containing a valid target (i.e., a human shape). The horizontal axis represents the entire data set as described in the previous chapter, the vertical axis illustrates the current value of each data input.

At this stage, there are very few differences to be seen visually in the data which might be differentiating a valid and an invalid target, Both graphs seems to follow a similar pattern, albeit with widely varying maxima in certain ranges.

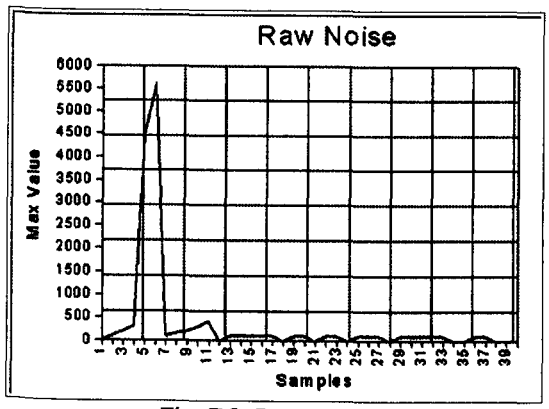



*Fig.76:Raw Noise*

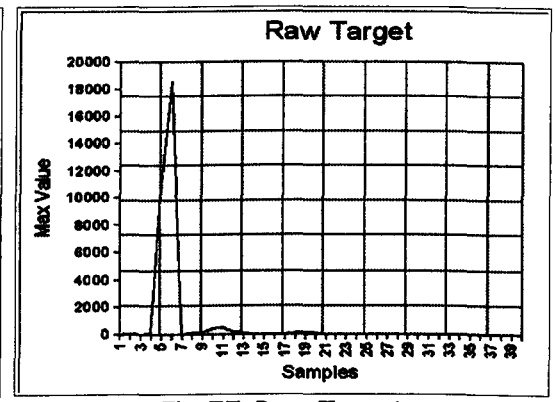



*Fig.77:Raw Target*

To be able to effectively compare the trends of these two different data classes, it is necessary to first establish some common reference value. This can be achieved through a number of scaling techniques. These will be considered in more depth, but in order to gain a quick overview of possible results, the two data sets shown above will now be entirely scaled by converting all measurements into percentages of their possible representation range. Doing this results in a fairly neutral presentation of the data, where each input

line has a chance of exerting a fair share (in this case, 1 part in 40 as the 41st data member is an artificial parameter which was added manually giving the expected network classification, and is used only for validating a training or trained network) of influence on the resulting network.

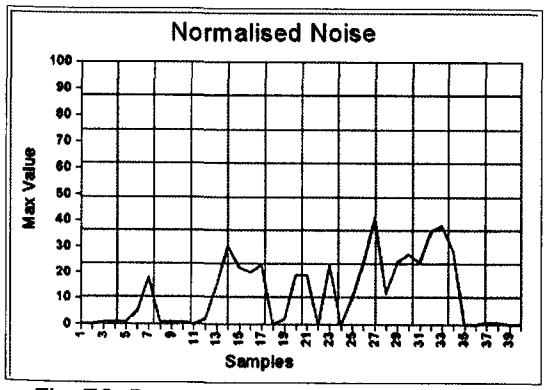



*Fig.78:Percentage Normalised Noise*

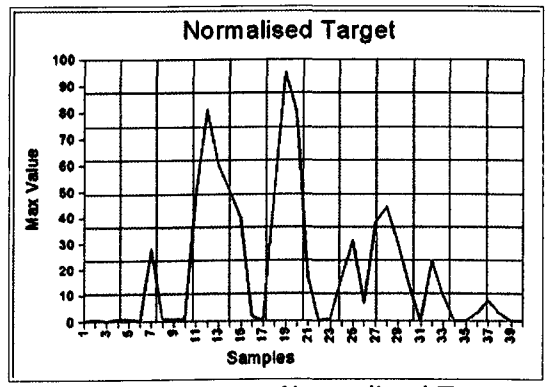



*Fig.79:Percentage Normalised Target*

As seen in *Fig.78* and *Fig.79*, even such a simple data transform results in a greatly enhanced overview of the actual data variations, and the differences between the two data sets can be much better considered.

Whilst this particular approach might not be the best suited to the data currently being considered, it does very well illustrate the general effect of data normalisation, which tends to enhance smaller data values whilst reducing the range of larger more dominant features in the raw data set. Naturally, each input line must be carefully evaluated to determine whether an enhancement or reduction is actually necessary.

For the purpose of this example, only two images are being considered. In order to obtain a realistic impression of the general data trends, it will be necessary to consider, if possible, the entire data set (some 2000 samples currently), splitting the valid and invalid target components to allow for independent analysis processes. Obviously, whichever transforms are eventually decided on, they will have to be taken in a general form which may be applied to either valid or invalid target, as this particular classification is only known during the testing phase. The actual goal of this preprocessing phase is to actually enhance any features signalling a possible valid target , whilst simultaneously reducing the presence of non-valid data.

It is also important to remember at this stage, that the ideal network inputs lie between -1 and + 1, or 0 and +1, depending on the approach taken. This allows for an easy combination of both digital and analogue type inputs using a standardised range with balanced scaling.

For this simple test, it is not necessary to develop any extremely sensitive adaptations, as these will most likely have to be quite heavily modified for the more realistic data sets. It would however be useful to develop a single transform which could be applied to all lines of data.

A generic transform which can be applied in order to enhance small

values whilst hardly modifying larger ones is the logarithmic transform, as illustrated in *Fig.80*:

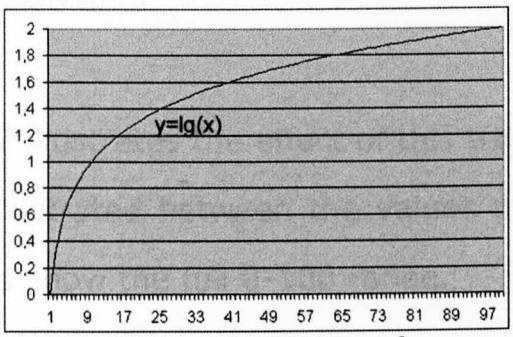



*Fig.80:Logarithmic Transform*

This shows a plot of a logarithmic transform on a constant series between 1 and 100. A similar effect can be achieved using a square root transform, although the final result is a much milder data adaptation. Obviously, a pure logarithmic transform cannot be used on the raw data on hand, as the range includes 0 values.

Having tested a number of approaches using the built-in data preparation tools of Neural Works Pro II, as well as conventional data analysis in MS Excel, a final fairly simple process was arrived at.

The actual equation is shown in *Fig.81*.

$$y = \frac{lg(\sqrt{100\frac{x}{Max}} + 1)}{lg\,11}$$

*Fig.81:Data Transformation Equation*

where:

x - raw data input

Max - maximum range value for current input

y -resultant processed data

the graph in *Fig.82* illustrates the effect of this transform.

The raw data is illustrated between the values of 0 and 12, whilst the other two plots show the full 0-100 range.

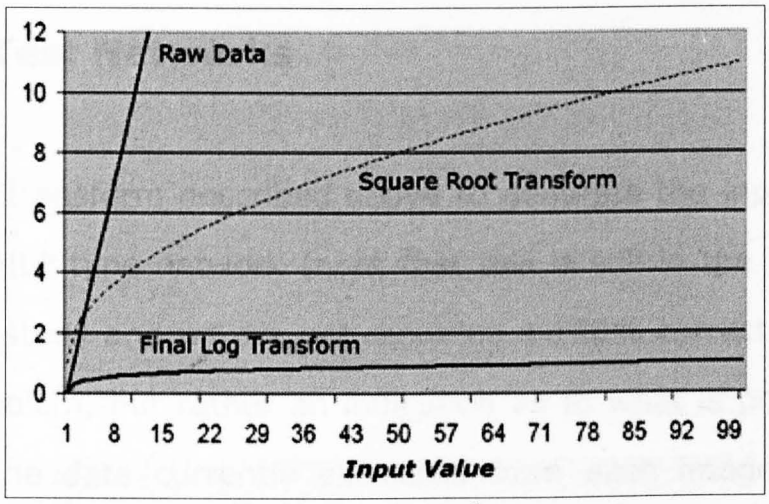



*Fig.82:Data Transforms*

As can be seen the final output is now limited to the 0-1 range

The final transform is actually a combination of 3 different operations: [*Appendix D*]

1. Range normalisation. This transforms the raw data into a percentage value based on each input's current range.
2. Square root transform. This simply cleans up the data by providing a limited degree of smoothing. The +1 feature is to always guarantee a positive value as an outcome to the following logarithmic transform.
3. Log transform. Dramatically enhances low values as compared to larger dominant data lines. The associated lg(11) divide is simply a maximum range divisor which scales the data to lie between the desired 0-1 values.

## 5.8.3 - Test Networks

Using the transform described above to generate the input sets for a simple MLP type network (note that this is still in the range of a feasibility study and we are not requiring a 100% correct resolution to the problem, but rather an indication as to what is possible and whether the data currently extracted from each image object is sufficient to meet our requirements), a number of test networks were generated, which gave highly satisfactory results.

For the purpose of these tests, the full available data range was utilised.

The best results were obtained on a 39-2-3-3-1 network, giving an

RMS error of 0.0852 and a classification rate of 0.9847, although similar results (in the 0.98 range) were also achieved using less complex arrangements such as 39-5-3-1, thus with only two hidden layers. These being MLP's, the notation "39-5-3-1" describes a network using 39 input nodes, a first hidden layer of 5 nodes, a second hidden layer of 3 nodes and a final output of 1 node, providing a 0-1 output.

The training, testing and validation data sets were obtained by splitting the available generated data (over 2000 images) into 3 groups collated from randomly mixed data in such a way as to ensure an even distribution of all data ranges within each set. This was done in order to ensure an ideal representative training set.

*Fig.83* illustrates the logical arrangement of the successful 39-2-3-3-1 network:

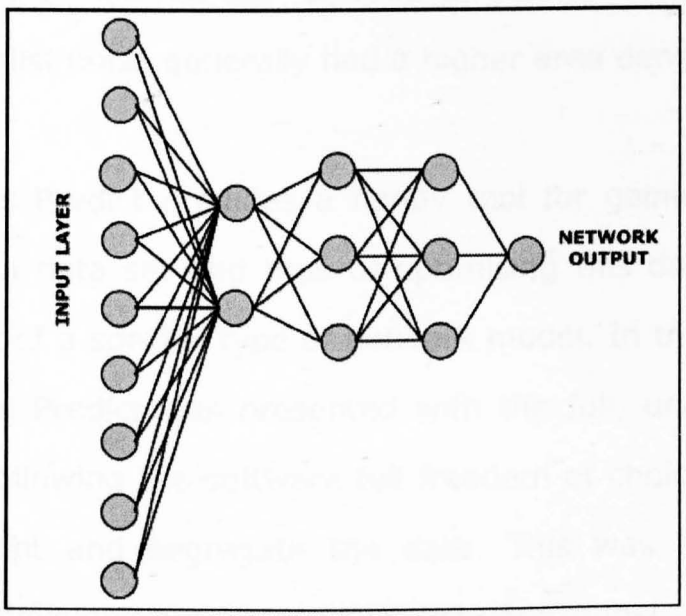



*Fig.83:Network Architecture*

For clarity, not all input nodes have been included. At this stage in the process, the actual values of the weights within the network are not extremely important.

The result which has been provided shows that a 98% correct classification rate can be achieved on relatively unprocessed artificial data and using a relatively simple network architecture, which is sufficient to justify further development on the entire system.

This configuration still represents a very unoptimised network layout, as there has been as yet no attempt to map possible input node commonalities. During the initial (and rapid) data overviews, no real relationships appeared within the dataset, apart from the tendency of the "% area set" of valid targets to lie between 20% and 40%, whilst noise generally had a higher area density.

Neural Works Predict provides a handy tool for gaining a general overview of a data set and thus of optimising this dataset for the development of a specific type of network model. In this initial test, Neural Works Predict was presented with the full, unaltered, data range, thus allowing the software full freedom of choice in the way it would adapt and segregate the data. This was run with the intention of developing a simple MLP network model. Although the

initial test did seem very promising, achieving an initial classification rate of 98%, with most errors lying in over-classification (false positives), it did also raise some concern as to the data members selected by Predict to represent the full data set. Given the total data set of 40 members, Predict chose to omit all but five, retaining the following members:

1 - Box Height

2 - % of Total Area Set

3 - Segment Area

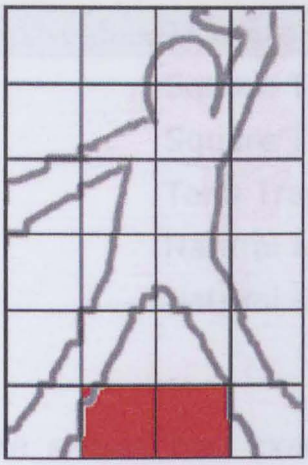
4 - Segment 2-6

5 - Segment 3-6

Obviously, certain members such as Box Width, Area etc. can be interpolated using the selected members, but the last two elements are very strongly dependent on the angle at which the object is being viewed, as well as the stage of movement which the object is currently carrying out. Data members such as centroid or radial measures would have seemed to contain more relevant information.

Given a particular object's bounding box, the two relevant segments causing concern are highlighted in *Fig.84*:

*Fig.84:Selected Segments*



On the entirely artificial image shown above, the relevant segments are highlighted in red. Although this is only a sample illustration, it is easy to see that not every single object held within the bounding box must necessarily be occupying either of these segments. The fact that Predict has selected these for this particular network might well be a reflection of the type of data available in the full training data set, which might be characterised by humans occupying these two segments, and noise generally not occupying them.

For the time being, it is sufficient to be aware of the fact that this selection might not be totally unbiased and is not necessarily representative of a larger data set.

Let us now observe the actual mathematical transforms which Predict selected to carry out on each selected input:

| Parameter | Transformation |
|---|---|
| Line 1 (Box Height) | Square Transform |
| Line 2 (% Area Set) | Square Transform |
| Line 3 (Segment Area) | Tanh Transform |
| Line 4 (Seg 2-6) | Natural Logarithm (Base 10) |
| Line 5 (Seg 3-6) | Natural Logarithm (Base 10) |

In *Fig.85* and *Fig.86* are shown two examples of data prepared using these transforms

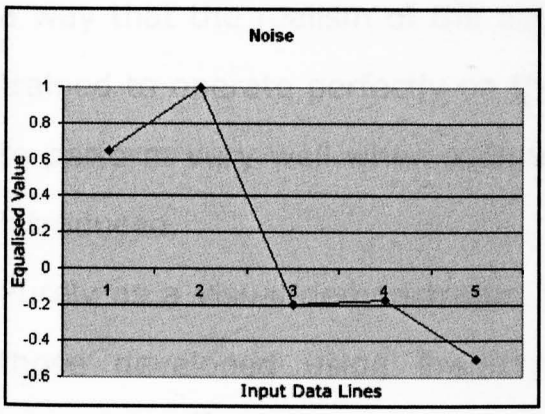



*Fig.85:Noise*

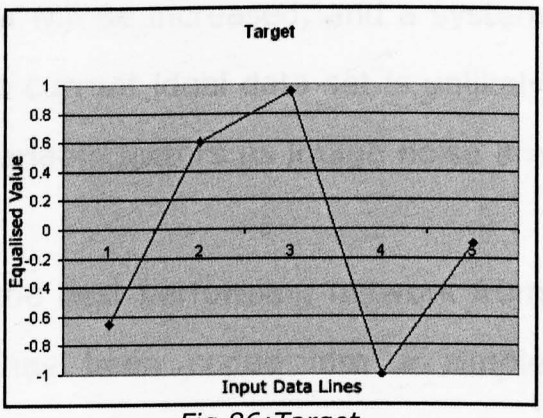



*Fig.86:Target*

It is interesting to note that, at least for these two examples, Predict has established a transformation algorithm which nearly mirrors the trends of the noise data to that of the target data along the vertical axis.

As can be seen, the data transforms selected by Predict are overall relatively simple. This does not imply that the problem being considered is in itself simple, but rather that the raw data as obtained from the image is fairly well aligned to the type of optimisation required by a simple MLP at this stage of the process. As the final system is intended to be running in real-time, the

simplicity of the final data transforms cannot be disregarded.

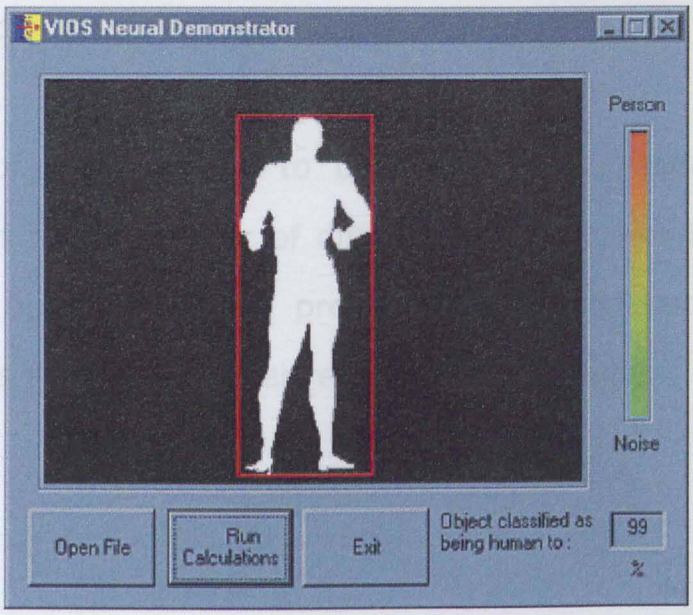
Considering the results achieved using both Neural Works Predict and more conventional manual analysis methods, we can say that the principle of the entire process has been shown to work, at least for this initial set of data. More time could be spent refining the actual data transformation algorithms and network architectures to try and achieve 100% problem resolution, but this would largely be a waste of time, as the data set now needs to be expanded in such a way that the realism of the data will be increased, and a system trained to operate perfectly on the current ideal data set is unlikely to perform very well when such variable factors as image noise are introduced.

Purely as a visual demonstrator, the best performing network from those developed using Predict has been coded into a simple software package.

This program allows any simple VOS format file (one satisfying the initial test conditions of a single object with no surrounding noise) to be rapidly evaluated and classified. This displays the unfiltered network output as a classification percentage.

*Fig.87* shows this demonstrator in use.

*Fig.87:Person Detection*




It can be quite clearly seen how the object detection process has located the person and placed a bounding box around this possible target. The network result can be seen via the graduated classification bar as well as via the pure percentage output, in this case 99%.

It is interesting to note that the currently used network has difficulties in classifying valid targets when the object's total area within the entire image falls below a certain value (approx.<(Image Height/4)). This behaviour could well be a result of the way in which the training, testing and validation data sets were generated from the total data set, possibly with less smaller targets within the validation set, as this was the set used by Predict to train the network. This validation set was selected to be only 10% of the total available data, with Predict left free to select which

lines it would actually use in the validation process.

A number of tests remain to be carried out, especially in the evaluation of the behaviour of differing network architectures, but the results obtained so far prove that the concept in itself is feasible, thus justifying further research in this study.

# 6 - System Development

*If you torture data sufficiently, it will confess to almost anything - Fred Menger*

## *6.1 - Enhanced Data Complexity*

Considering the results obtained from the tests based on the artificially generated data, these do confirm the validity of this study.
The data used for these initial tests was however, very strictly controlled and highly simplified, when compared to real live data, which a final system would have to be working with.

The next development phase for this study is therefore going to be a gradual and controlled increase in the complexity of the data used, whether this be generated artificially or captured from a live source. Each added level of complexity may then be correctly analysed, and incremental improvements to the various data processing algorithms may be introduced to cope with the added system requirements.

## 6.2 - Multiple Targets and Noise

The original set of artificial data used in the system feasibility study featured a single object per image.
In a real life situation, such a situation would be quite rare, as light sources, shadow effects and even spurious background movements could all contribute to presenting the effect of multiple moving "objects" within a single frame, even if only one person were actually present. Additionally, we cannot ignore the situation where more than one person might be in the actual surveillance area, thus creating a multiple target situation.

This is raised in point 1 of the main hypothesis: *Is it possible to develop a system capable of processing the output from an industry-standard camera in order to identify the presence of a person* ***or multiple persons*** *within the image ?*

As mentioned above, spurious background movements, depending on the surveillance location, might also be present in the image. Such movements might be the results of wind blowing through tree branches, broken light reflections on various surfaces, or any type of background movement, small enough not to be classified as a potential target but still large enough to be identified in the surveillance image. These effects are classified as noise, as they do not assist in any way in the actual task of target identification but

rather distort the overall image by possibly masking parts of valid targets or by modifying areas of the image and forcing a then wasteful - to the overall system performance - analysis of the given area.

The difference between the two phenomena, potential target and noise, lies purely in the size and distribution of the effect, but the result in either case is the same: The creation of an area of movement within the image which will have to be correctly identified and subsequently analysed.
Whilst noise can be discarded without further considerations, a potential target will need to be correctly processed. The actual difference lies purely in the definition of the object's relative size in the image, and thus, in this situation using a single uncalibrated camera, in the definition of the overall system's effective surveillance range.

The inclusion of these two items into the general data set involves certain modifications to the overall detection process, as will now be discussed.

## 6.2.1 - Multiple Targets

The inclusion of multiple targets into the data set actually introduces two separate conditions into the detection process. These conditions arise purely from the relative positions of the objects within the image.

When multiple objects are in an image, they can be placed as follows (considering two objects):

1. Objects separated from each other, clearly distinguishable as two objects.
2. Objects not overlapping, but sharing common areas on their bounding boxes.
3. Objects overlapping, thus not easily distinguishable from a single object.
4.

These conditions are illustratedin *Fig.88*, *Fig.89* and *Fig.90.*

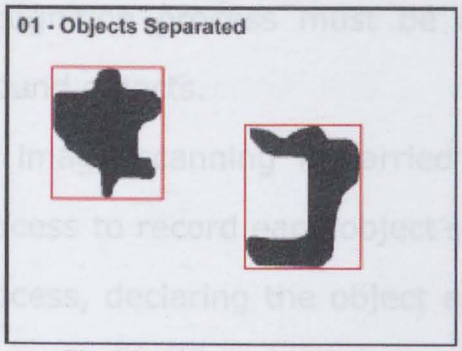



*Fig.88:Objects Separated*

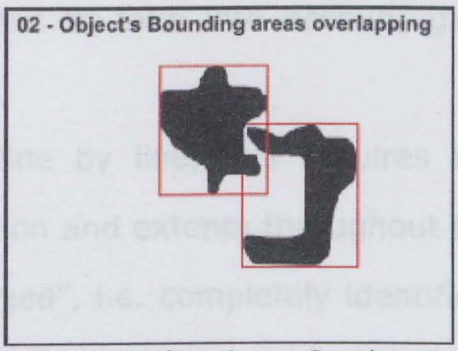



*Fig.89:Bounding Areas Overlapping*

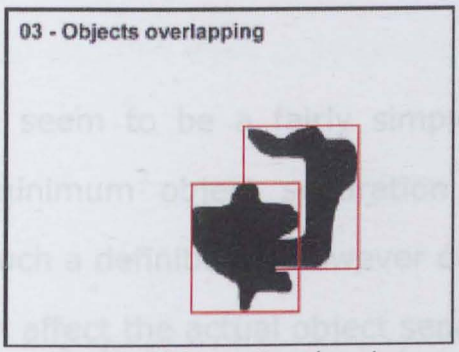



*Fig.90:Objects Overlapping*

Conditions 1(*fig.88*) and 3(*fig.90*) do not present any particular problems to the current detection process, other than having to be adapted to accept more than a single target. Condition 2(*fig.89*) however, needs extra consideration.

Given the current method for identifying an object in the image, an object is defined by finding the first set pixel in each direction, and by these, defining the outer extremities of the object's bounding box.

This system will however always define only a single bounding box (and thus a single object), no matter how many objects are actually present in the image. In order to detect multiple objects, the

recognition process must be modified to take into account gaps around objects.

As image scanning is carried out line by line, this requires the process to record each object's position and extents throughout the process, declaring the object as "closed", i.e. completely identified, once a consistent gap has been found surrounding the entire object from its surroundings.

Initially, this might seem to be a fairly simple process, which it would be if the minimum object separation distance could be precisely defined. Such a definition is however difficult to set, as the following factors can affect the actual object separation:

Distributed Image noise, Noise suppression algorithm, Image shadow artefacts, object's position within the image.

These various factors can also lead to a single object being effectively separated into a number of sub-objects, which must however still be treated as a single potential target.

In order to provide an accurate way of identifying separate objects within the image, a dynamic system should be considered. As the actual object identification is a sequential operation to the image noise reduction, a value might be obtained from the actual average image noise level (which can be easily extrapolated from these calculations) , which can then be used to derive a minimum specific object separation distance. The advantage of using such an

approach resides in the fact that each image is now considered for its own specific conditions.

As explained above, the actual process of identifying and separating multiple objects is very much reliant on the method used to carry out the noise-reduction process on the overall image. Whilst still separated processes, these must be tuned to one another in order to be presenting valid and unbiased or uncorrupted data to the next processing level.

## 6.2.2 - Noise Reduction

There are two types of noise which can occur within an image: distributed noise and local noise.
In the current application, local noise is essentially treated as a potentially valid target and is thus processed and eliminated.

Distributed noise, as mentioned earlier, is a natural occurrence depending on the active surveillance environment, and can be observed in two main forms: distributed movement within an image, such as background tree branch movement, and distributed movement of the entire image, such as that caused by actual camera vibration.

Local noise, while requiring a full detection cycle to be considered

and cancelled, does not generally affect the validity of an image. A high level of distributed noise however, can result in severely corrupted image data, making potential targets difficult to accurately locate and analyse. Due to the type of approach currently being considered, it is necessary, as far as possible, to eliminate image distributed noise whilst retaining the relevant image data.

### 1. Overall Image Movement

Caused by camera vibration, for example.

Overall image movement is relatively simple to compensate. Using a frame by frame image analysis process, pixel-wise movement within an image can be located and the entire image can then be shifted in the appropriate direction and by the appropriate amount to result in a zero difference image comparison outcome.

### 2. Movement within the Image

As caused by wind-induced branch movement.

This is slightly more complex to eliminate, as the movement generally cannot be reduced to a single definable vector. In this case, each detected area of movement (pixel-wide) must be

separately analysed in order to determine the magnitude and direction of the now local movement.

This effectively results in a pixel by pixel analysis of the entire image difference area.

Each difference pixel is shifted by a distance of one pixel in one of 8 directions until either a zero difference is obtained of the pixel-shift directions are exhausted. In the second case, the result then remains as an identified difference at the original position.

*Fig.91* illustrates the pixel-shift directions:

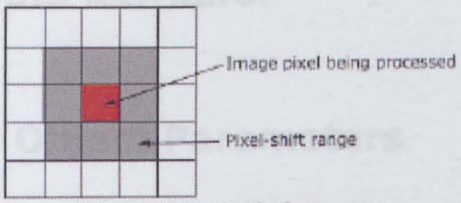



*Fig.91:Pixel Shift Range*

As can be observed, the process for identifying and eliminating movement within the image is essentially a superset of the process utilised to process image-wide movement.

It does however result in a slightly lower resolution output image, as areas within the image that have registered movement can be effectively over processed - hence the necessity to limit this processing to purely areas of image change, and to constrain the pixel's mobility in the compensation range in such a way as to enable correct motion correction whilst preventing the elimination of valid image changes due to potential target motion.

The pixel compensation range must therefore be matched to the current image noise level (which is determined via the immediate image comparison process), in order to provide an accurate noise compensation process. This necessity to dynamically adapt to each individual image was previously mentioned in the process for separating potential targets which have overlapping boundary areas.

## *6.3 - Multiple Data Extractor*

### 6.3.1 - Multiple Object Parameters

In order to test the processes suggested, the code for the original **Data Extractor** was modified to include the various stages of noise control and multiple object capability.

The exact relationship between the average image distributed noise level and the accordingly adjusted minimum object spacing values has not yet been established and is thus still adjusted manually, as a balance value must be obtained which, whilst eliminating a maximum of distributed noise will also introduce the least possible object corruption into the image.

*Fig.92* show the new code in action:

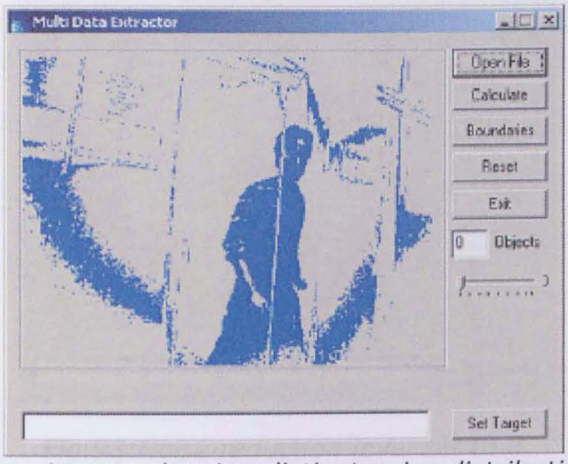



*Fig.92:Base image, showing distinct noise distribution areas.*

In this image, the slider controlling the noise-reduction level can be clearly seen below all the other control buttons.(value currently set to 0).

At this stage, the image has simply been opened but not yet processed, which explains the number of objects located as being zero.

In *Fig.93*, *Fig.94* and *Fig.95* is shown the effect of increasing the minimum object spacing on the actual noise reduction process. The larger squares show areas of the image which will be retained once the noise removal is completed.

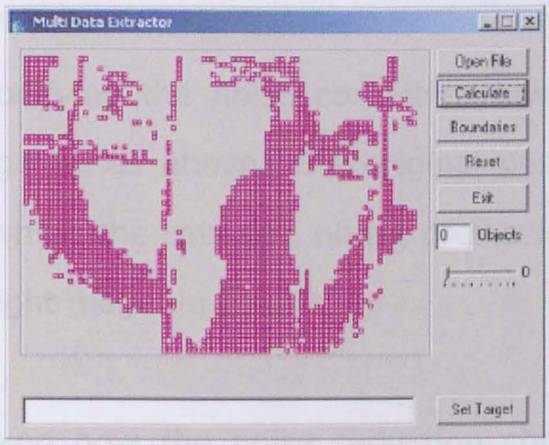



*Fig.93:Noise reduction level set to 0*

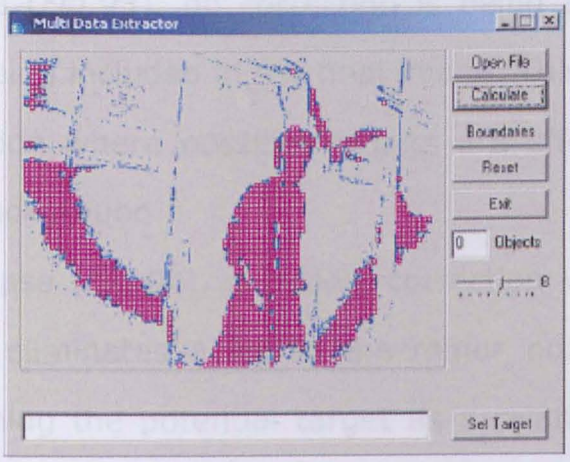



*Fig.94:Noise level reduction set to8*

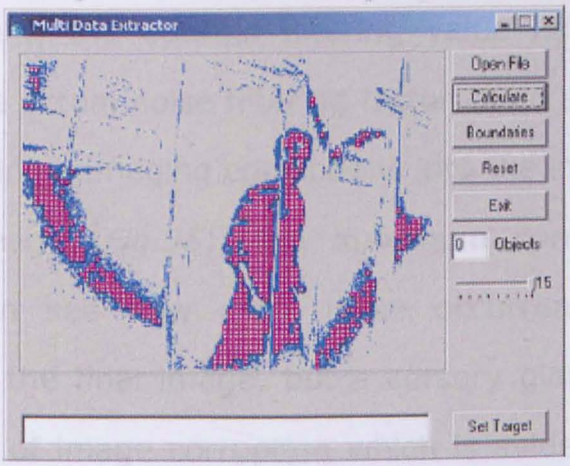



*Fig.95:Noise level reduction set to 15*

The effect of modifying the noise correction level can be clearly seen in the image series above. The original underlying image is shown in blue, whilst the retained pixels of the image have been highlighted as bright magenta blocks.

It can be observed how the noise reduction component actually affects the image integrity.

In the first case (*Fig.93*), no correction is being applied, and too much noise is being included in the final image. This leads to visible object deformation where possible targets are effectively blended into the noise background.

In the second case (*Fig.94*), a middle correction value is applied. This effectively eliminates a lot of the minor noise occurrences, whilst still retaining the potential target as a relatively unmodified object. Larger noise occurrences are somewhat minimised but still appear as distinct potential objects in the filtered image. From a visual assessment, the optimal filtering value has not yet been reached, and the actual noise filtering factor could still be increased without introducing damaging corruptions into the image.

In the third image (*Fig.95*), the maximum correction value is applied. We can see how most noise occurrences have been eliminated from the final image, but a cursory glance also reveals the high degree of image corruption which is evident by the much reduced area of the potential target which has been marked as

worthy of further analysis.

This is an obvious case of overcorrection which could easily result in potential targets being either totally ignored, or corrupted to such a degree that they are no longer recognisable as such.

The same series is now shown with the object detection and separation code in place. Each identified object is shown within its bounding box (*Fig.96, Fig.97, Fig.98*):

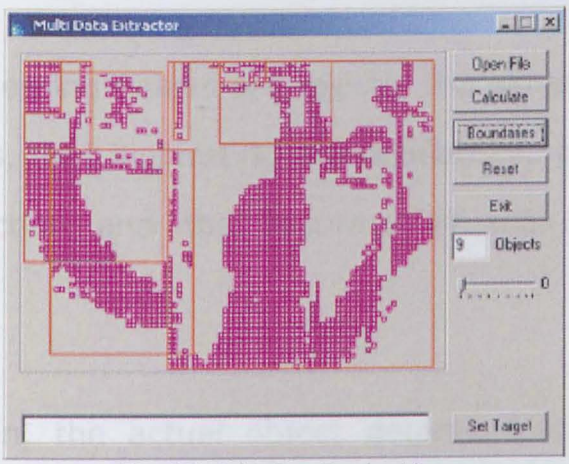



*Fig.96:0 Noise Reduction*

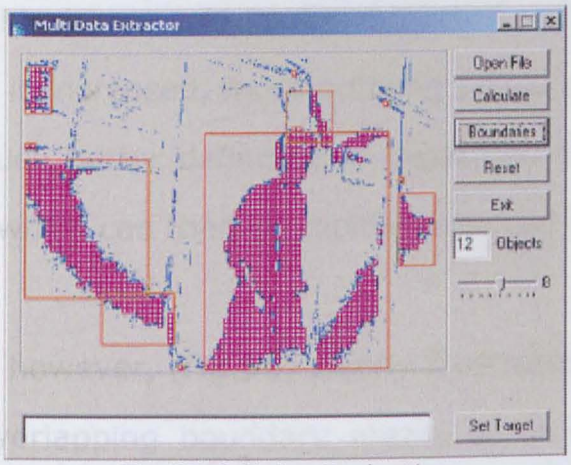



*Fig.97:8 Noise Reduction*

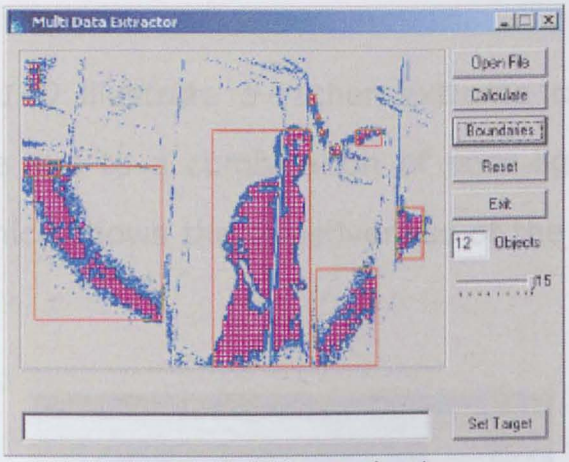



*Fig.98:15 Noise Reduction*

The actual processing values are exactly the same as in the noise-reduction series, thus illustrating the relationship between the level of nose correction and the accuracy of the individual object recognition.

As can be seen, the actual object detection accuracy increases throughout the series, even though the actual object erosion can be clearly seen. This is actually to be expected, as when the noise correction level is increased, more outlying pixels are cancelled out, thus leaving much better defined and separated objects within the image, objects which can then be rapidly detected and separated.

The first image however, (*Fig.96*) clearly illustrates the condition of objects with overlapping boundary areas, or in one case of one object located entirely within the boundary of another object but still correctly identified as an independent object, which would then be further analysed on its own merit.

*Fig.99* and *Fig.100* illustrate a rather extreme case with a fully saturated image due to a combination of poor lighting and cheap camera lens, which shows the effectiveness of the object detection process:

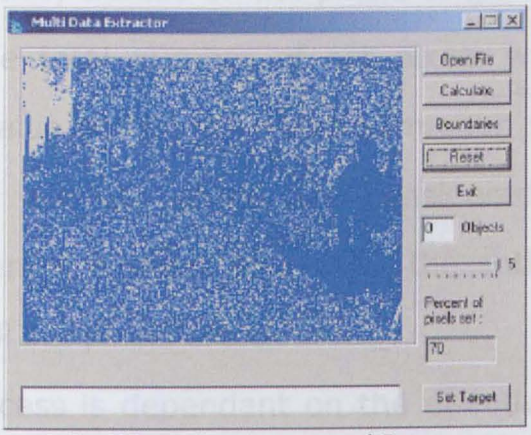



*Fig.99:Noise Saturated Image*

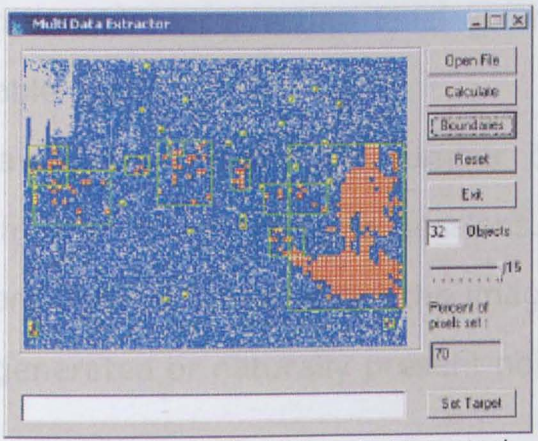



*Fig.100:Saturated Objects Detected*

## 6.3.2 - Deriving the Dynamic Noise Correction Level

The setting to adjust the noise level in each image will have to be based on a dynamic function in order to provide optimal image results.

The actual noise correction process is based on a subsampling technique. Using this, clumps of 4x4 pixels of the original image are considered. If the total number of set pixels within this area is larger than the value set on the noise-correction slider, the area is marked as valid and its pixels are considered to be set and representing potential objects, otherwise, the entire area is judged to be noise and all set pixels are cleared.

As the entire process is dependant on the effective average image saturation, it is necessary to establish the link between the said saturation and the required correction level in order to provide a clean but also usable image.

Measurements have been taken on a number of datasets, both artificial and from live image capture sessions. Although these always only present a single target within the image, they also have either artificially generated or naturally present noise, depending on the data set.

*Fig.101* and *Fig.102* show the noise density over two separate data sets:

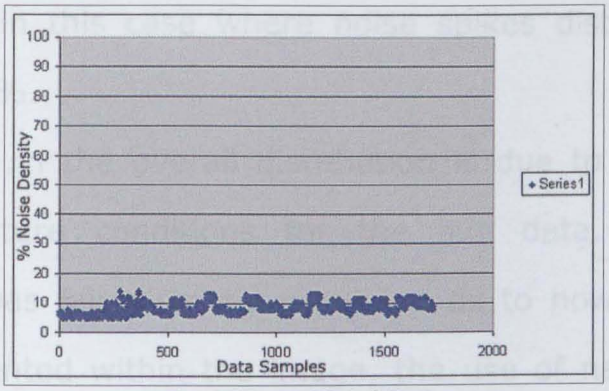



*Fig.101:Artificial Data*

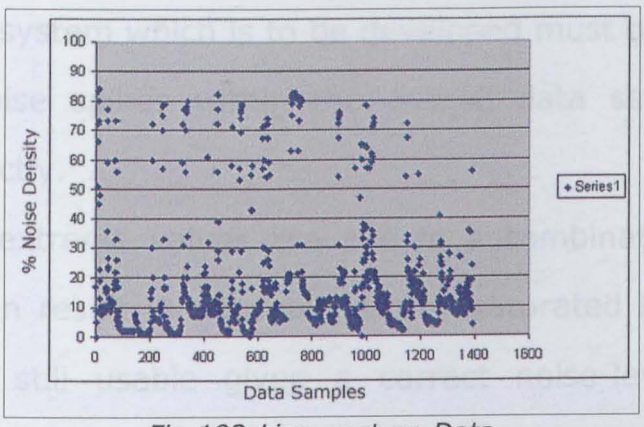



*Fig.102:Live-capture Data*

As can be clearly seen, the average noise density in both cases remains lower than expected.

The average value for the artificial data set (*Fig.101*) is at 8.3% with a minimum at 5% and a maximum value of 14%.

As would be expected, the dataset from the live capture sequence (*Fig.102*) is much less regular, but still follows a similar trend. The average noise density is here at 15.2% *with* a minimum of 0% and a maximum of 84%, the median value, probably a more telling

measurement in this case where noise spikes distort the overall trend, lies at 9%.

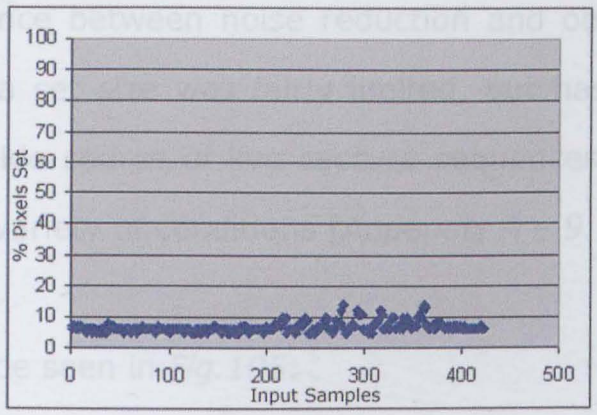
This difference in the overall distribution is due to the much less controlled capture conditions for the live data. Although the environment was fairly strictly regulated as to how many objects could be presented within the image, the use of natural light and the effects of cast shadows are bound to create a less precise data set, which does not however mean that it is a less accurate data set. The final system which is to be developed must be able to cope with such noise spikes within the overall data set in order to function correctly.

Many of the extreme values are due to a combination of factors which do often result in unusually highly saturated images, which are however still usable given a correct noise-level correction adjustment.

In the case of the artificial data set, the number of images analysed was over 1700, and over 1300 for the live-capture sequence.

As could be expected, there is not much difference between the distribution patterns for images with or without valid targets. This is illustrated in *Fig.103* and *Fig.104*:

*Fig.103:No Valid Target*



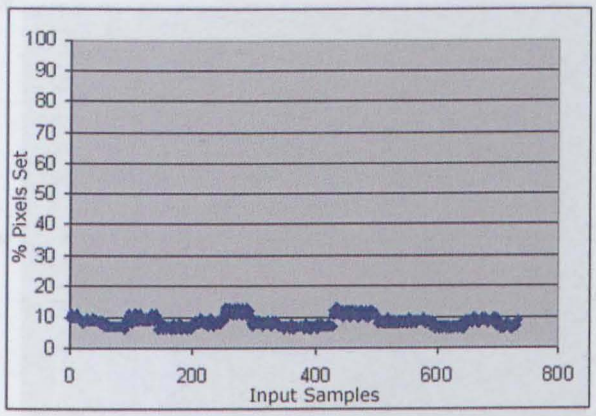




*Fig.104:Valid Target*

*(In both cases, taken from the artificial data set)*

Such a density variation cannot be expected, due to the image capturing process, as objects will change their image space occupation as they vary their distance to the surveillance camera.. This is not the same factor as the object's set area within its own bounding box, which does reflect the nature of the objects currently being examined.

A short evaluation of the noise correction level necessary for various noise density conditions was carried out. As this process has to be carried out manually in order to determine the best point

offering a balance between noise reduction and object corruption, the overall data set size was fairly limited, but has been selected from a fairly wide source of live capture sequences presenting the camera with a variety of conditions [*Appendix A – 9.1*].

The result can be seen in *Fig.105*:

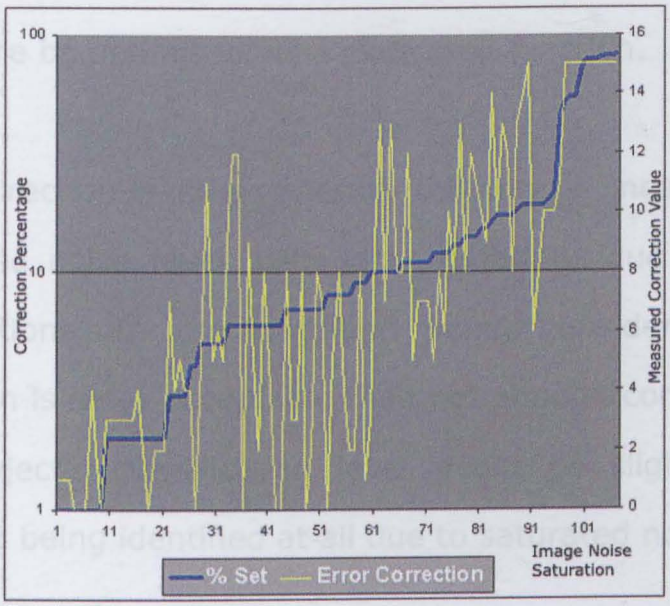



*Fig.105:Required Correction Level*

*%set curve(blue) uses the left hand vertical axis 0-100*

*Error Correction curve(yellow) uses the right hand vertical axis 0-15*

The dark blue line (Cleaner diagonal curve) shows the percentage of image pixels set for each image (i.e., the average noise distribution). This is shown on a logarithmic scale from 0 to 100. The yellow line shows the corresponding minimum noise correction level required to provide a usably defined object using the current

detection process.

Although irregularities are present in this second setting, the trend is to roughly follow the distributed noise level.

Obviously, this presents a calculated value which follows experimentation to obtain the best noise correction value. Such an approach would not be feasible in the final autonomous system, and must therefore be optimised via a dedicated function.

The noise correction level is generally following a linear mapping of the distributed noise level, with local variations due to particular image conditions. In the condition being considered, a slight overcorrection is more acceptable than not enough correction, as at most, an object's classification level might be slightly reduced, instead of not being identified at all due to saturated noise effects.

The graph below shows a linear correction level, as well as a logarithmical ratio. The logarithmic value generally provides too high a correction level, thus reducing the amount of image information available. The ideal solution would follow a middle value between these two paths, not over correcting images with low noise levels, whilst providing an adequate level of filtering for highly saturated images. A suggested curve is illustrated in *Fig.106*:

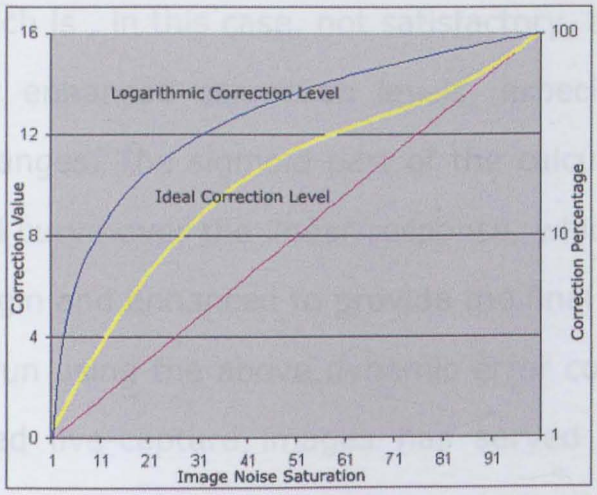



*Fig.106:Potential Error Correction marked in yellow*

*Where the Ideal Correction Level curve(yellow) uses the left hand axis (0-15) and the Logarithmic Correction Level is based on the right hand vertical axis (1-100)*

The function actually applied in this case is shown in *Fig.107*:

$$Y = 0.15X + \left(\frac{(100 - X)}{15}\right)\sin(0.01\Pi)$$

Where Y is the noise correction level (0 to 15)
and X is the Distributed Noise Level (as a percentage)

*Fig.107:Error Correction Function*

As can be seen on the plot (*Fig.106*), this function serves to enhance the correction response in the lower level range, whilst not becoming quite as extreme as a pure logarithmical function (shown in dark blue), in order to preserve the final object integrity. Initially, the noise distribution level is simply scaled down to between 0 and 15 (the total correction range available).This results in a very linear

adaptation, which is , in this case, not satisfactory, as many images require slightly enhanced correction levels, especially in the low noise density ranges. The sigmoid part of the calculation serves to place a sigmoid curve over the linear response, which is then offset towards the origin and enhanced to provide the final desired effect.
A second test run using the above dynamic error correction on 100 randomly picked live-capture images has served to confirm the accuracy and reliability of the process.

The final adjustment curve is shown in *Fig.108*. This reflects the fact that the noise correction value is an accurate integer, thus the correction value is modified in steps of 1:

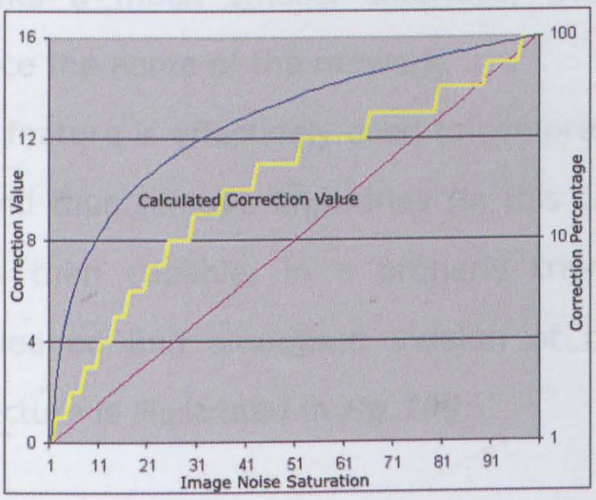



*Fig.108:Final Noise Adjustment*

Applying this function to the overall image results in a dynamically adaptive object detection process for each image, providing an ideal level of noise filtering depending on the image being considered.

## 6.3.3 - "Intelligent" Noise Reduction

As the system being developed is eventually based on the use of a form of neural network, the question can be raised, as to why the noise reduction process is not treated in the same way.

Indeed, work has been carried out using bottleneck architectures for noise reduction in images [*49, 35, 16*]:

A bottleneck architecture is based mostly on an MLP type of network, where the system offers the same number of inputs as outputs. This allows the input pattern to be properly reconstituted at the output stage. However, the middle hidden layer of the network presents a much smaller interface, thus creating the bottleneck, hence the name of the network.

This bottleneck feature is effectively used to compress the incoming data stream, and thus remove impurities (in this case noise). The output layer is then capable, in a properly trained system, of outputting a cleaned and smoothed version of the image. The network architecture is illustrated in *Fig.109*:

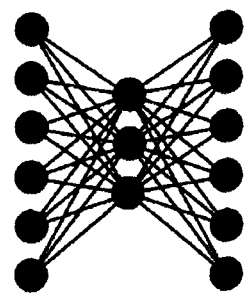


*Fig.109:Bottleneck Network*

The size of the actual input layer is not really dependant on the image size, as the total image will be presented in portions to the network, but must simply be sufficient for the type of noise levels to be encountered.

The advantage of such a system is its great simplicity, and reliability once trained. It does however present a fairly rigid structure, which will be incapable of taking into consideration overall image noise levels and varying image noise levels - the data will always be considered within the network's local area, but not as a factor of a larger image-wide noise distribution ratio.

The system proposed and evaluated using Multiple Data Extractor, whilst considering smaller image sections, does take the overall varying parameters into consideration and accordingly adjusts the actual level of correction to be carried out. This thus preserves the data purity in detailed areas of the image when the overall noise level is fairly low, and only applies a maximum corrective level in cases of total or near total image noise saturation.

## 6.3.4 - Processing Times

The process described above for identifying objects within the image, whilst functioning reliably, is not necessarily the best process for the application being considered. The problems associated with it are:

- Necessity to process the entire image area
  This can be fairly time consuming.
- Object detection depends on suitable noise filtering, thus presenting one extraneous abstraction layer into the entire process.

This has led to the development of a further approach which is less destructive when considering the resulting image data:

## *6.4 - Image Feature Analysis*

In order to evaluate the efficiency of the object detection and noise reduction algorithms,, the following process was developed:

Instead of analysing each image pixel by pixel and measuring an average image saturation level which in turn leads to a noise correction value, the actual area to be considered can be limited by carrying out a linear analysis of the image light variations, as shown in *Fig.110*:

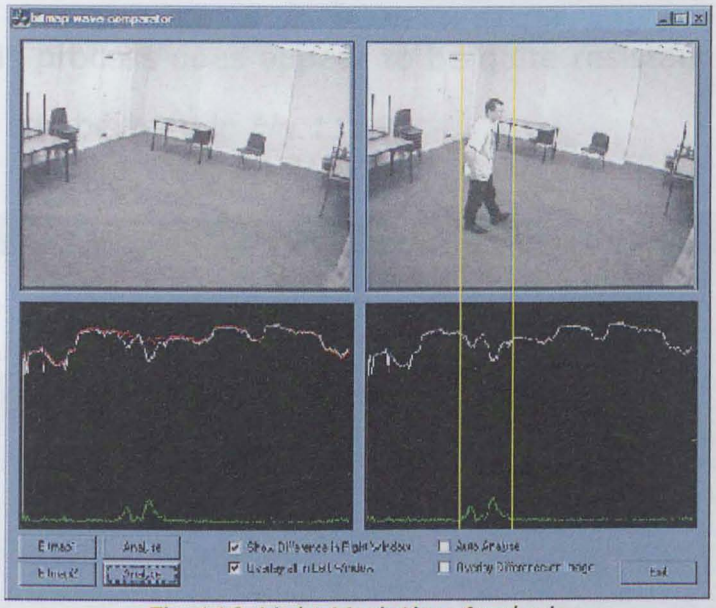


*Fig.110:Light Variation Analysis*

This shows the two images being compared. To the left, the datum image, to the right, the incoming camera image. The actual light curves for each image are shown below (overlaid on the left hand image). The eventual image differences (illustrated by the green line at the bottom of the black window) can be very rapidly located using a simple line analysis of the final difference curve as well as a certain level of momentum, used to gap small areas of no change which might otherwise interrupt a continuous object. It must be noted that the curves displayed here are in their raw format, without any form of optimising, which could otherwise be used to enhance the areas of difference.

As with the initial processes used to compare the datum and incoming images, there does remain the risk of missing areas of

movement due to similar colour tones in the changed areas, although this process does appear to be quite resistant to this type of noise, as can be seen in *Fig.111*:

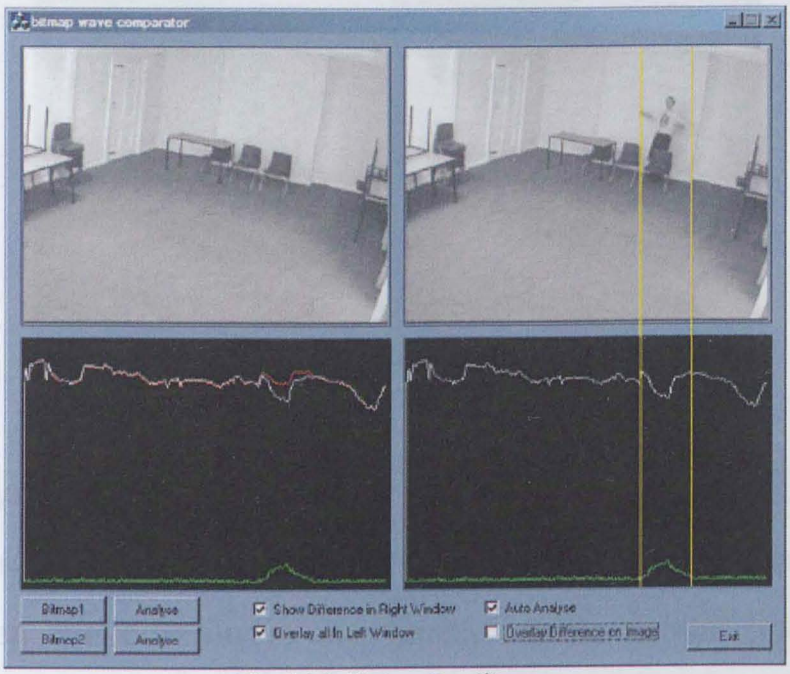
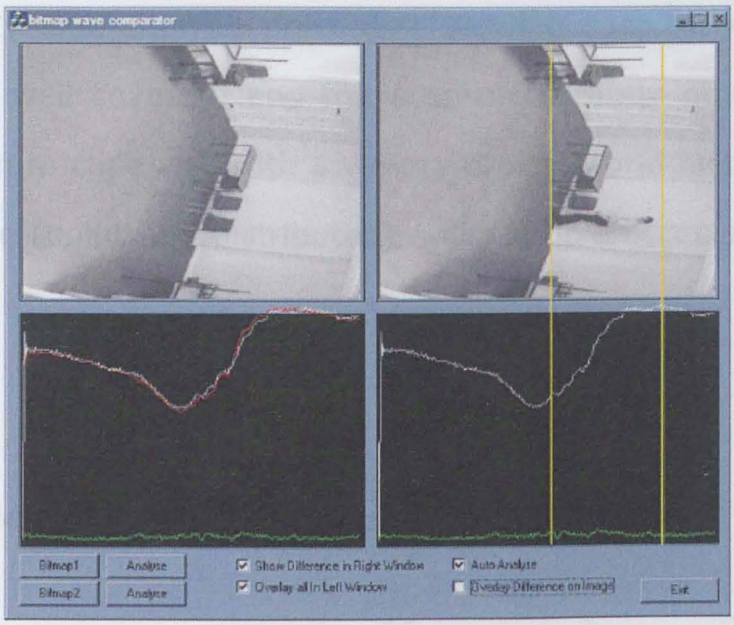



*Fig.111:Noise Resilience*

To test the process capabilities, the images were rotated 90 degrees clockwise before being analysed, in order to remove the fairly large influence of the dark trousers against the otherwise light image, and thus focus the difference detection process on the area where the white shirt is against the white wall. As can be seen, even though the actual difference curve exhibits only minor (visible) changes, the full area of change was correctly identified as shown in *Fig.112*:

*Fig.112:Change Identification*




This analysis approach is valid for images with well defined areas of change and shows the efficiency of the object separation process, even when the differences are hardly visible to the human eye. Actually identifying the object limits will require the standard edge detection as described in the previous sections. For images with few areas of change, this process can be used to rapidly identify and isolate these parts of the image for further processing. If the image is however saturated with change areas, the entire image will have to be analysed anyway as is currently the case.

## *6.5 - Data Selection*

The actual image analysis processes allowing data to be extracted have been well covered, and the system currently developed has been shown to cope well with a variety of real world factors such as image noise (point and distributed) and multiple objects in a single image.

## 6.5.1 - Network Architecture

Throughout the system testing and development phase, use has been made of a fairly simple MLP network, developed on the various sets of artificial images and their resulting data. In a real world situation, this is not the ideal network configuration to use considering the system working conditions.

Due to the structure and training process of an MLP network, it is important to generate a very complete training data set, containing examples of all possible input types and classes, linked to clearly defined output patterns.

In the system at hand, although the number of inputs and the range of each is well known (the input data is from a rigid data extraction procedure guaranteeing a certain consistency), the actual data source is more difficult to control: There is no prerequisite other than a given threshold dimension for the type of object from which data will be extracted and analysed.

The network is thus presented with the data via at least one layer of abstraction (the data extraction and preparation stages), thereby increasing the data complexity by a variable factor depending on a number of dynamic settings used in the processing sequences within this abstraction layer (notably the dynamic noise correction level).

Due to these factors, it is necessary to introduce a fairly loose form of classifier prior to the final decision making step.

The role of such a classifier would, in this case, be a form of data segregation or grouping, depending on the relationships within the object data set. Its role is thus less that of determining the validity of a potential target, but more that of labelling the current dataset according to fairly rough parameters, and thus allowing a following process to actually classify the object considered with more ease, as the initially seemingly random input pattern would have been not only transformed into a more standardised format but will also have received a further set of parameters (the exact number is set by the exact form of this classifier) affirming the probability of the object belonging to a given input class.

## 6.5.2 - Data Pre-Classifier

The actual format of this pre-classifier is yet to be determined, but it must be considered that the data classes to date are highly dynamic. The aim of such a pre-classifier is not to issue a strict boolean type output, but to support the current data set with a certain probability measure as to the data origin.

It is obvious that the full range of potential input sequences is nearly impossible to collect during this development stage. It is however possible to collect and present the development sequences with a fairly wide range of these potential inputs, which should enable the establishment of a dynamic process capable of a certain level of data interpretation and extrapolation.

A SOM (Self Organising Map) type of network offers exactly these advantages. As explained earlier, a SOM is quintessentially a data mapping device. The system is created with a number of default values. As the entire available data set is presented to this network, the various nodes and weights are adjusted in such a way as to create a multidimensional map (the number of dimensions depend on the relationships within the data as well as on the actual architecture of the SOM itself) of the data presented. Values which have not been seen during this quite open training process can easily be analysed to observe their clustering behaviour, and the actual multidimensional position of this data within the network can then be analysed as a probability of belonging to a specific class of

the input dataset.

Whilst this might appear to provide a one step solution to the problem at hand, the actual output from such a network still requires a certain level of interpretation.

As the training process is totally unsupervised (limits to certain dimensional values can be set, but no errors as such occur as the training network is not provided with expected response patterns), the actual data ordering can be following quite a complex curve, depending on the dimensionality of the data to be analysed. This is however predictable to a certain extent, given that the data dimensionality is previously known.

### 6.5.3 - Hybrid Architecture

Using such a front end for rough data separation, the general data patterns can be observed, and the valid areas within the thus created multidimensional space can be categorised.

When an element is then analysed during run-time, the position of this element inside this previously established multidimensional space is taken into consideration to allow a form of classifying, which is however still dependant on a number of interrelated features, i.e. the actual network weights defining the enclosing space. The complexity of these relationships is thus directly linked not only to the network's physical grid size, but also on the dimensionality of the actual input data.

Due to the network structure. These dimensions are however known to reside within certain predefined boundaries, which were used to establish the network training patterns. Due to the training process (and dependant on the volume and distribution of the data actually used for the network training process), the boundaries to the various input classes can be more or less precisely labelled, which then enables the extraction of precisely defined and controlled data parameters as a resultant of this original sorting step.

The data thus extracted is actually very well suited to be directly fed into an MLP type network for a final classification stage. (It must be noted that an MLP is not a necessary final stage, this could very well be replaced by a form of fuzzy classifier, which does however carry the risk of being less adaptive to such a dynamic data set, and also requiring a very complete rule set to be defined, based on historical data inputs which might not be fully representative of the system inputs under real life runtime conditions.)

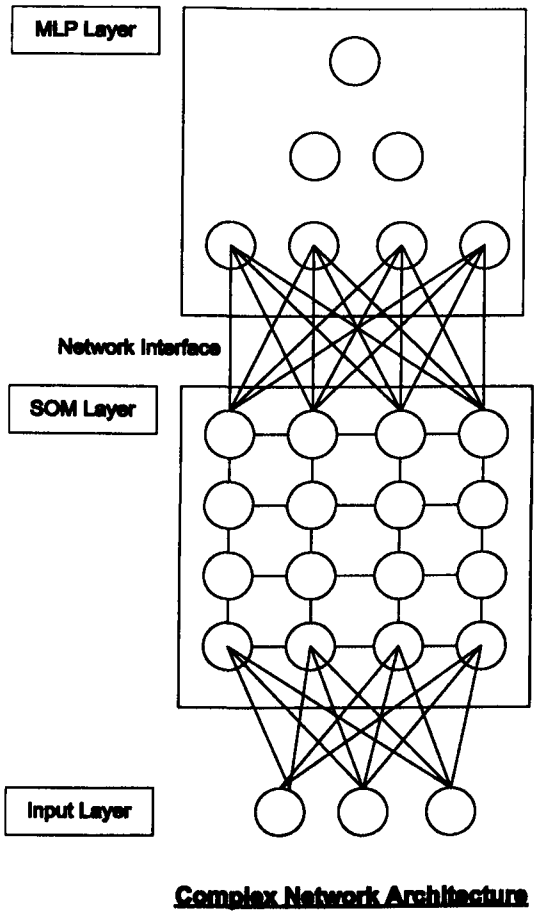



*Fig.113:Complex Network Architecture*

Such a hybrid system offers the advantage of not resulting in any form of data loss between the various stages, as all outputs from the first layer (Processed input values as well as extrapolated data relationships) are passed directly on to the second layer. The data analysis and condensing (reducing the multidimensional values to a single probability output statement) occur within the system, and more complex data input arrangements are also possible, therefore the secondary layer could be provided with the original raw (or only optimised) data as well as receiving the first layers inputs.

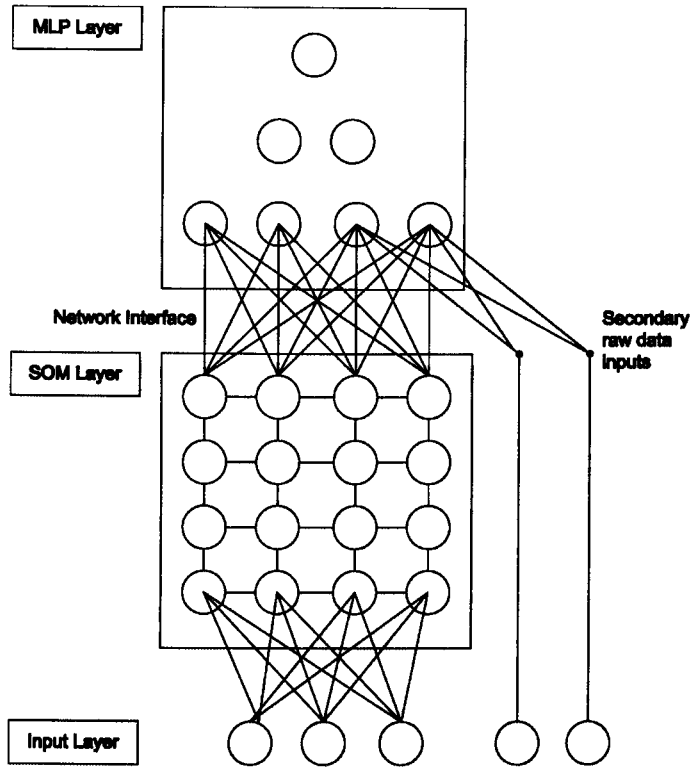



*Fig.114:Complex Network Architecture 2*

Using a similar architecture, it is of course entirely possible to provide the second classifier with entirely different data sets, albeit originating from the same object to be analysed. This is a valid solution when the data preparation required for the different network architectures needs to consider different data relationships for optimal network performance.

In the system currently considered, there is a fairly large pool of data from which ideal inputs can be selected. This does however then present the problem of actual data selection in order to maximise the data relevance to problem solving and minimise

object corruption through an insufficiently descriptive data set.
In the data studies previously carried out on the simulated data sets, certain features appeared to achieve importance levels which were contrary to the data consistency: Rapidly and easily changing data values were selected for final consideration, which led to unnecessary bloating of the final system, as additional network inputs were made available to compensate for the relatively poor quality of these specific data lines.

Using the complex system architecture described above, less important data values can still be considered in the initial network sorting layer, whilst more critical values could be fed directly into the second classifier stage, maybe using a special weighting component to accentuate the relevance of the individual network inputs.

More specifically, the data set which is currently extracted from each potential target has a large section of mapping components, the segment area components. These 24 entries serve to describe the actual spatial distribution of the object within its own boundaries, thus giving a rough evaluation of the real object shape. This subset of the image data is highly sensitive to object aspect changes, whilst still carrying information as to the object class. Due to this highly dynamic component, this subset has so far been partially avoided, even though the principal component analysis of

the entire range of artificial data sets revealed certain of these elements as being of not inconsiderable importance to the overall object description. The system described here presents an ideal way of including this data subset into the final analysis without distracting the final classifier network from more consistent but maybe less dominant data values. To achieve this, the segment area subset can be fed directly into the initial sorting network (this subset is ideally suited to a SOM architecture due to its dimensional components).

## *6.6 - Data Optimisation*

### 6.6.1 - Data Mapping

The data used to date has been fairly generic, optimised for use mainly with MLP type networks.

When considering the initial classifier stage of the system (the SOM layer), the data to be fed into it is already in a relatively well prepared form, as the values are all percentage representations of the area occupied within each zone. A weighting component is not needed, as all the zones have an equal importance, depending on the position of the target within the bounding box, and the value variance is, in all cases, between 0 and 100 (or between 0 and 1 for

an ideally scaled data set).

When examining a number of cases, a general distribution map can be recognised within the segment area pattern, where the central values tend to be highly set, regardless of the condition, simply due to the average object mass distribution.

It is then questionable as to whether all these values are actually necessary to carry out a correct object analysis (leading to a very large and slow network system).

A rapid analysis of the segments data through rapid network generation did to confirm this assumption to a given degree.
In order to test this assumption, a number of small SOM networks were generated.

## 6.6.2 - SOM Generation

The SOM architecture has been selected for its ability to map undefined data into rough data sets following a mathematical process which effectively results in a spatial/mathematical mapping of the data representation areas within the network as defined by the network nodes, based on data feature similarities.

The SOM architecture is ideal for this type of study, as it allows a fairly rapid visualisation of the actual data pattern distribution and representation, without having to actually modify the data in any way (other than a simple log-based optimisation for the SOM structure), and thus risk corrupting the integrated data signals.

The data mapping feature of a SOM network is, however, very sensitive to varying input vector sizes, making some form of data normalisation, necessary, especially in the case of input data vectors with widely varying values.

Due to the training process of a SOM, such an architecture is less suited to generalisation, considering the relationship between the number of data inputs and the width and height of the mapping grid itself - this means that a SOM network can rapidly reach a saturation point where the presented data will no longer be mapped correctly, as map features have been allocated to different input sequences[*80*].

Considering the Segment data available, the range for all the values is between 0 and 100, which can be reduced to a pure percentual representation. The problem with such a normalisation remains however in the comparatively large influence of higher value inputs, which can lead to a distortion of the actual mapping representation. To counter this effect, the data is treated uniformly through a logarithmical optimiser. This serves the purpose of enhancing the

lower data range relative to the larger values, but also of enhancing the difference to very small values. The actual process used is given in *fig.115*.

$$Y = \frac{(lg(X))}{(lg/(100))}$$

*Where* $Y$ *is the prepared data and* $X$ *is the original data*

*Fig.115:Data Normalisation*

For the purpose of this test, the segment arrangements shown in *Fig.116*, *Fig.117*, *Fig.118* and *Fig.119* were used:

| | | | |
|---|---|---|---|
| A | G | M | S |
| B | H | N | T |
| C | I | O | U |
| D | J | P | V |
| E | K | Q | W |
| F | L | R | X |

Centres

*Fig.116:Centres Segments*

| | | | |
|---|---|---|---|
| A | G | M | S |
| B | H | N | T |
| C | I | O | U |
| D | J | P | V |
| E | K | Q | W |
| F | L | R | X |

Extremes

*Fig.117:Extremes Segments*

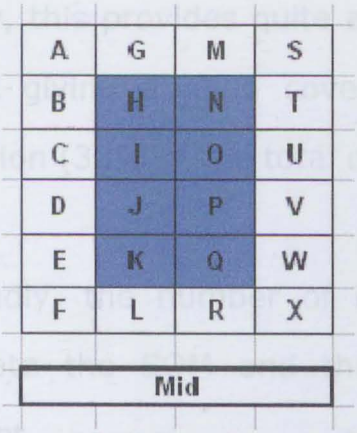



*Fig.118:Middles Segments*

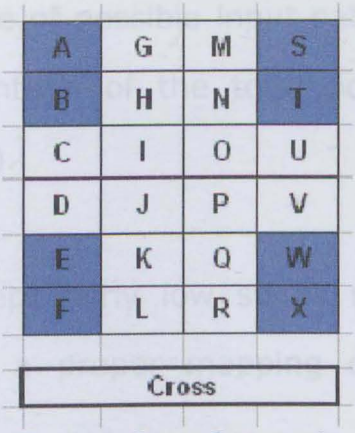



*Fig.119:Cross Segments*

The actual patterns represent the main areas of interest on the segments.

| | |
|---|---|
| Centres(*Fig.116*) | shows the expected head position, as well as feet positions for a standing human and hand positions |
| Extremes(*Fig.117*) | This takes a representative area from the central body portion, as well as possible extreme positions for the hands and feet. |
| Middles (*Fig.118*) | Only takes into account the middle body position. Extremities are ignored |
| Cross (*Fig.119*) | Represents the main areas for hands and feet in given conditions |

In all the patterns, the number of data classes was intentionally limited to 8, for two reasons:

Firstly, this provides quite a good range of possible input patterns, whilst giving a good coverage percentage of the total possible selection (33% of the total data is used).

Secondly, the number of inputs is kept fairly low so as not to saturate the SOM and thus prevent a proper mapping of the dataset.

This combination allows the actual SOM structure to be kept relatively small, which is advantageous to both processing speed and hardware requirements.

In *Fig.120* are shown the variations present within the segment data sets, when comparing target to non-target data.

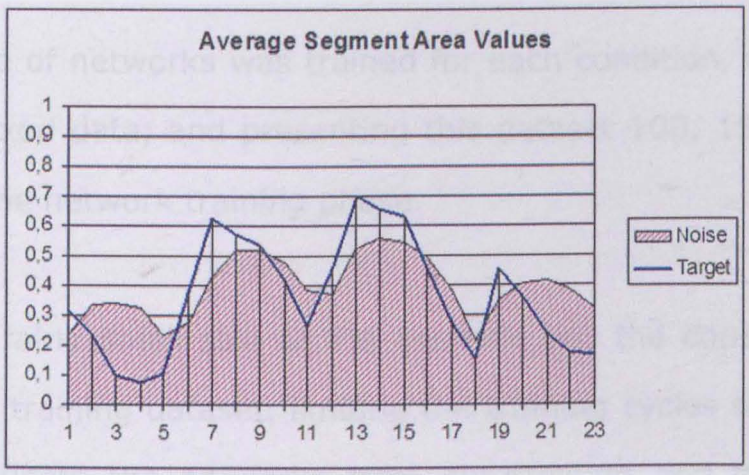



*Fig.120:Average Segment Data Values*

This already shows differences in the data trends for both conditions. To accurately judge this, it is more useful to refer to the

standard deviation measures, as shown in *Fig.121*:

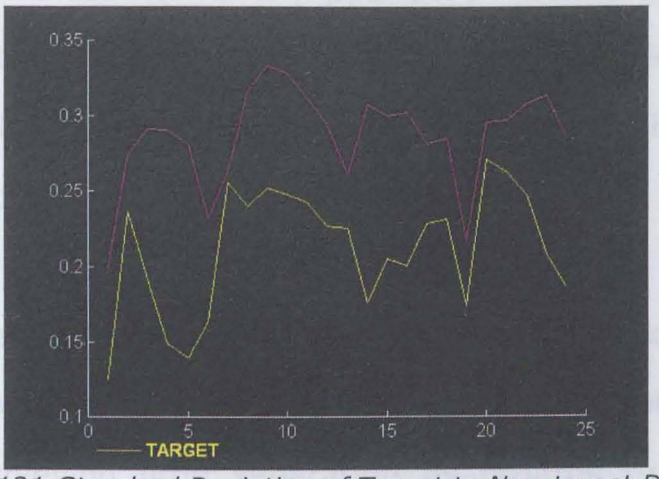



*Fig.121:Standard Deviation of Target to Non-target Data*

The graph above (*Fig.121*) can be seen to confirm the assumption that the most extreme values (the four corners of the bounding box grid) offer the least variation between noise and target. These points are marked by the two large dips in the graph above.

A sequence of networks was trained for each condition, using 1400 lines of mixed data, and presenting this dataset 100, 150 and 200 times for the network training phase.

Given the fairly small size of the network and the comprehensive size of the training dataset, limiting the training cycles to fairly low values prevents the networks from overtraining and thus simply learning the training dataset. For testing purposes, a separate testing dataset of 610 lines of mixed data was reserved.

The results of these networks [*Appendix J - 10x10 SOM Network Tests*] show that the various models did indeed succeed in creating a representation of the data. This can be evaluated visually via the ordering and clustering of the various weight patterns.

When the test data set was used however, the actual level of sorting was found to be very poor throughout, with noise and target data lines achieving little separation as regards to the winning network node:

When the trained network is presented a line of yet unseen data, a calculation is carried out to obtain the vectorial position of this data within the network's hyperspace. The winning node is that closest to this position, and the actual distance between the input and the winning node is taken as a confidence level to the association with this node's previously defined data class or label.

In most of the cases considered here, the winning node and distance for both target and noise data lines was often the same, highlighting the inability of the networks to distinguish between these two data classes, even though the data sorting process had appeared to run successfully.

The situation thus presented is now twofold:

1 - The data used has no relevant information permitting a clear class definition between noise and target to be made.

2 - The data used is too complex to fully model on this type of network, indicating a need for more degrees of movement within the network to allow an accurate segregation between the data classes to be carried out. The modelling done to date was involved mainly in matching the data complexities of a single class. Effectively a case of network saturation.

Given that the segments data being studied here is directly linked to the object geometry and mass distribution, it is unlikely that the separation between noise and target not be represented in the final data set. This is corroborated by the standard deviation curves of the two data sets, which show visible separations.
In order to provide a more accurate mapping model, the size of the SOM network was increased fourfold to 20x20 architecture. The mathematical degrees of freedom within the network are now: 20x20x8 = 3200 (This is a direct measure, without taking into account the relational links between each node in the system).

The same process was now carried out as described previously, training three networks for each condition with learning times of

100, 150 and 200 cycles and random weight initialisation.

Once again, visually, a good degree of data sorting occurred, although now a certain level of clustering could also be observed [*Appendix J - 20x20 SOM Network Tests*]. When observing the actual results using the test data set, a much better data separation was also achieved, showing the network's increased mapping capacity.

On all models except the "Middles" data set, a clear clustering pattern is apparent within the network weights: indeed, the "middles" trained networks also showed the worst separation when using the test data set. This shows that the dataset used does indeed contain the information necessary to providing a good separation between the two classes: target and noise.

When considering the actual results, the best class separations where achieved by the Centres and Extremes Data sets using slightly higher training cycles [*Appendix J – Results*].

Out of this, a new data set was generated, combining certain features of both original sets. The actual layout of this combination set is as shown in *Fig.122*:

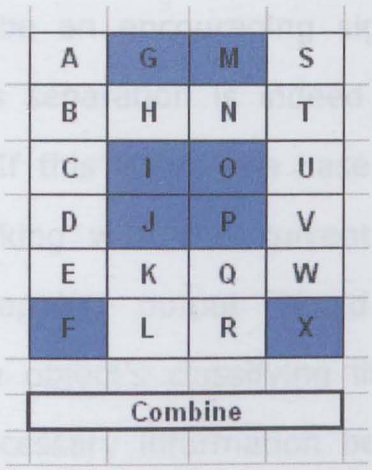



*Fig.122:Combine Segments*

The features selected in this set represent general areas containing the body, head and feet in extreme conditions.

A selection of networks was trained to evaluate the mapping capacities according to various starting parameters. This approach is necessary to obtain the best possible network, as all the initial weight values are randomly set, leading to different minimisation possibilities each time. In addition to this, depending on the actual network topology (Weights distributions within the network space), a local minima can lead a training process to optimise in an incorrect direction, thus blocking any further optimisations which might otherwise have been achieved.

As can be seen [*Appendix J - Network Mapping*], various separation models have been achieved. These charts show the firing rates achieved for the test data sets, expressed as a percentage of the total dataset size.

Although this might be an encouraging sign, it is necessary to evaluate whether this separation is indeed representative of the data classes or not. If this is not the case, the use of this pre-sorting network, working with the current datasets, cannot be justified, as the mapping output would provide no further information as to the object's classifying likelihood - This would simply result in unnecessary information being fed into the next classifying stage, thus tying up a number of resources and compromising the final system performance.

At the very least, this would represent a waste of processing time, which cannot be justified on a realtime system, where the system effectiveness is measured by its update interval.

The actual percentage of target to noise data in the various data sets is as follows:

Centres: 63.80 % of Target conditions.
Combine: 64.43% of Target conditions.
Extremes: 63.80 % of Target conditions.

The most accurate data splitting was generally achieved by the "Extremes" dataset, with a clear separation between the two main classes, and an effective mapping separation of 64% mapped target to 36% mapped noise.

This is close enough to the dataset separation of 63.8% - 36.2% to be considered a valid mapping separation. The incorrectly mapped 0.2% of the cases are due to a number of factors which are essentially due to conditions where the target and noise data present no clear separation, thus necessitating a further classification stage assisted by further data parameters for the condition considered.

The effect achieved through the use of this type of SOM network is effectively a form of data compression via a mapping mechanism:

Starting from a set of 8 unsorted data inputs, the SOM net has optimised these and converted them into a probability measure as to their belonging to a certain class within the data. This particular class in itself is not specified, but using the expected results against the test data set, the classification areas can be roughly estimated. As the output of the SOM network is not intended to be directly interpreted, but will in turn be fed into a further classifying stage using a supervised learning process, it is at this stage not actually necessary to carry out the labelling process, apart from the information it would give as to the reliability of the SOM sorting process.

This probability measure is dependant not only on the actual

winning node coordinates, but also on the vectorial distance to this node, and the associated metric (although these values are indeed related):

The chance of an input data set of belonging to a certain data class decreases as the vectorial (Euclidean) distance to the winning node representing this data class increases. Conversely, the smaller the vectorial distance, the greater the likelihood of the dataset actually belonging to the described class.

There are thus a number of outputs from the SOM network to be considered. These are:

- Winning node x coordinate.
- Winning node y coordinate.
- Vectorial distance from dataset to winning node.
- Metric measure.

In this case, the use of a SOM actually results in a net data compression ratio of 50%, coupled with the advantage of a form of preparatory data mining as to the classification of the data.
This effectively completes the initial development of the preliminary mapping stage.

## 6.6.3 - Classifier

The development of the classifier module is, in a way, a lot easier than the initial data mapper, as the process to be used follows a supervised training method: The system is presented with a set of data inputs and simultaneously with the expected output values.
This makes it very easy to measure the actual performance of the network, the main difficulties lying in the steps of data selection, data preparation and network optimisation.

During the system evaluation phase, the main validation tool used was a series of small backpropagation networks. Indeed, the entire data preparation carried out was to optimise the obtained data for exactly this type of network.

The choice of input data is also partially dependant on the data output by the original mapping layer. Data available from here is:

- Winning X Node.
- Winning Y Node.
- Euclidean Distance to input data.
- Maximum Boundary Measure.
- Metric (Relative measure of Sum of input data to the boundary size).

Data used in previous small classifier nets was:

- Box Width.

- Box Height.
- Centroid X Position.
- Centroid Y Position.
- % of total boundary area set by object.

Also available as possible inputs are the various radial measures, which could be treated in the same way as the individual segment measurements, by processing them through a separate mapper.

The other measurements currently available are either irrelevant (Box X Pos, Box Y Pos), or are represented by other measurements (Box Area, can be calculated using Box Width and Box Height). Whether the classifier will be able to deduce these relationships is not clear, and it might be worth considering some form of preparation.

Considering the values Box Width, Box Height and Box Area, these could be represented as:

1 - Box Height, Box Width.

2 - Box Area.

3 - Box Area, (Box Width/Box Height).

Considering the various methods:

1: Provides the full information, although the link between Height

and Width will not necessarily be recognised. Original data can be reconstructed.

2: Provides the most condensed form of data, however, the object dimensions are lost in the data. Does not allow a full reconstruction of the original data. Original data can be reconstructed, although a scaling problem might arise.

3: This approach provides a comprehensive coverage of the actual data space, giving a condensed form of the object relationships

For the data to be fed into a classifier network, it should ideally be within the range of 0/1 or -1/+1.
If a simple ration is taken of Box Width/Box Height, this will not necessarily be below 1, as for given objects the width might be greater than the height.
Inverting the result in such a case is not acceptable, as the network input order must always remain consistent.
One solution would be to present the data in vectorial form:

$$Max = Max(Width, Height)$$

$$V_{Width} = Width/ Max$$

$$V_{Height} = Height/Max$$

although this does result in two output values, it does still represent an improvement in the description of the data, and provides a

simultaneous normalising effect. Coupled with the normalised Box Area value, a full reconstruction of the original data is still possible, showing that data loss has not occurred.

This does not however solve the problem of providing a single value which will consistently be in the 0-1 range without inverting the presentation order for given circumstances.

The next step would thus be to create an artificial data split at 0.5, where values below this would indicate a normally low ratio (less than 1), and values above a high ratio (over 1) although the techniques used to execute such a transform always lead to a loss of the original data, which is contrary to the entire principle of data optimisation.

## 6.6.4 - Classifier Training

The initial training dataset for the final classifier will then consist of the following input vectors:

- SOM Winning Node X Pos
- SOM Winning Y Node
- SOM Euclidean Distance
- SOM Metric
- Centroid X Pos
- Centroid Y Pos
- Box Area
- Box Width

- Box Height

although none of these will be presented in their raw extraction form.

The types of optimisations are shown below:

**SOM Winning X Node:**

1 in n encoding. n taken as maximum SOM network width.

$D_1$ = Som Winning X Node / 20.

This form of 1/n encoding is generally not recommended, due to the artificial ranking which it creates within the data ranges represented for the particular vector. In this case however, a geometrical relationship already exists due to the topography of the network. It is therefore entirely correct to create this implied ordering which gives an idea as to the actual position within the total grid

**SOM Winning Y Node:**

1 in n encoding. n taken as maximum SOM network height.

$D_2$ = Som Winning Y Node / 20.

**SOM Euclidean Distance to winning node:**

$D_3$ = Distance / Number of SOM inputs

$D_3$ = Distance / 8.

**SOM Metric:**

Metric = Sum of SOM inputs/Max Boundary

Where Max Boundary = 8.

$Metric_{Min} = 0$

$Metric_{Max} = 1$

$D_4$ = Metric.

**Centroid X Position:**

The data extracted represents the absolute position from the image edge. This must be processed to result in a measure relative to the current object's position.

$D_5 = (CXPos - X_{Min})/Width$

**Centroid Y Position:**

This is similar to the X Position.

$D_6 = (CYPos - Y_{Min})/Height$

Which results in a relative percentual measure between 0 and 1.

**Box Width:**

$D_7$=Width/Max(Width,Height)

where Max(Width,Height) relates only to the current input vector, not the entire dataset.

**Box Height:**

$D_8$ = Height/Max(Width,Height)

where Max(Width,Height) relates only to the current input vector, not the entire dataset.

**Box Area:**

$D_9$ = BoxArea/(320x240)

No dynamic scaling is used here, as smaller values are intentionally suppressed in favour of larger objects.

A number of networks were trained [*Appendix A – 9.3*], based on these inputs and a poll of the best performing SOM networks developed previously.

The necessity to develop a number of different models, even when using the same setup parameters, arises from two main factors:

Initially, a networks weights are randomly initialised. This random starting pattern effectively changes the energy states within the network, leading to a unique internal structure for every run.

The second factor which will affect the network development is the type of data used to train it, as well as the order in which the data is presented to the network.

An MLP network is also very flexible in its actual internal structure. As mentioned by L. Tarassenko in "A Guide to Neural Computing applications" [*16*], a three layer architecture is capable of solving

most non-linear problems, the difficulty remains only in selecting the correct number of nodes on each of these layers.

For the network considered here, the input layer has nine neurons, and the output has a single neuron.
As yet, no theory has been developed as to the exact relationship between the network architecture, training data and layer sizes. This process is thus, initially at least, fairly empirical.
Initially, this was set to a guessed value.

The available known data set was split into two randomly selected non-equal sections:
Training (approx. 2/3 of the data).
Validation (approx. 1/3 of the data).

The sets were selected using a random sorting process, in order to obtain representative samples for all conditions in both data sets.
This ensures that all three data sets (training, testing and validation) all have variety of data samples, leading to a balanced network training process and thus optimising the generalisation potential of the final network. This would not be the case if the training data had been severely biased towards a particular type of condition, for example only crouching humans, which would reduce the networks ability to correctly classify conditions it had not encountered during the training phase.

A further Testing set was created using the original noisy data, in order to evaluate the actual generalisation potential of the final networks.

## 6.6.5 - MLP Considerations

The process of training an MLP network involves carefully watching the output error reduction to select the best point to interrupt training, at the stage where the output error is the lowest, however without having the network simply learn the training data and loose its ability to generalise to yet unseen data, a process known as overtraining, or overfitting.

The method used is to present the network with the separate testing data set after a preset number of training epochs. This will be shown to the network in a pure analysis and not training mode, and the output compared to the expected value. This effectively generates a second error plot, which can be monitored in parallel with the first.

The lowest point on the second curve can indicate the ideal training point of the network, as after this the network weights will gradually start to overfit to the training data (*Fig.123*).

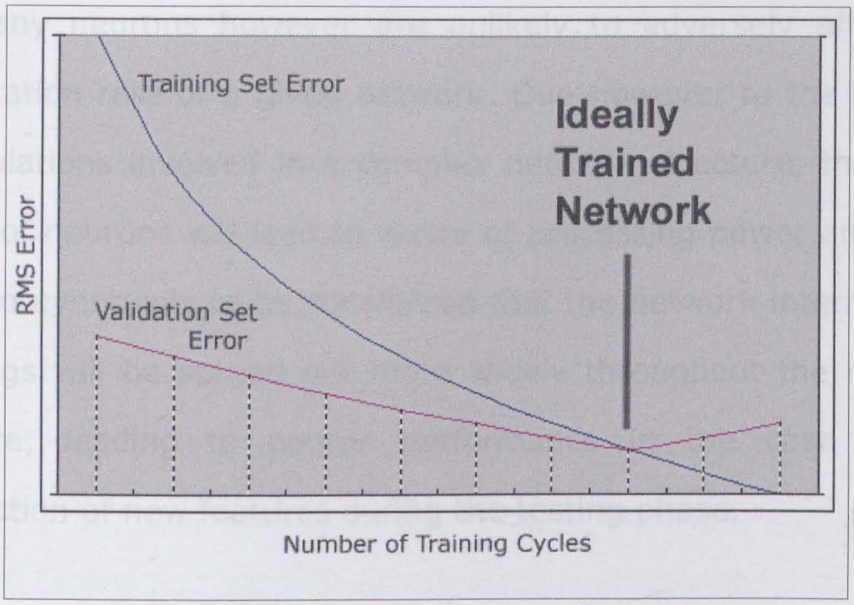



*Fig.123:RMS Training Error against Testing Error*

The simple evaluation of these curves is, however, not really sufficient to determine the best training breaking point. Even though a network might be giving a satisfactory classification rate, the system itself might well be processing redundancy: The actual size of the network might still be optimisable.

The number of neurons on each layer will closely determine the effectiveness of the network.

Too few neurons will cause the network to perform poorly where many different data classes are to be evaluated, as well as causing a poor network performance on new data. Such a fault is however normally to be recognised during the training phase, as the network training error is then unlikely to converge to 0, but will generally retain a fairly high value, depending on the complexity of the training data.

Too many neurons however are unlikely to adversely affect the classification rate of a given network. Due however to the number of calculations involved in a complex network structure, the sheer excess of neurons will lead to waste of processing power and time. It is also eventually to be considered that the network internal data groupings will be spread out more widely throughout the network structure, leading to poorer performance in the case of the introduction of new features during the testing phase.

A network can be gradually pruned, either on an empirical basis, where a limited number of neurons are disabled and the network performance then evaluated, or by considering the firing rates and values of specific neurons. A neuron with a low firing rate (i.e. rarely activated) and low output values (near to zero), can be evaluated as participating little in the network performance. Such a neuron can then be disabled to test its actual contribution to the classification process.
If no deterioration in the network performance is observed on a representative data set (it is important that all data cases be tested, as a given neuron might only fire for a given situation), the selected neuron could be completely erased from the network structure.

Many of the available neural software packages are capable of dealing with the evaluation of such neurons, either automatically, or via user prompts for confirmation.

In network optimisation, the actual ordering of the data inputs can also be of importance [*16*]. If logical or mathematical relationships exist within the input data set, it can assist the network classification to present this data to the network in a sorted fashion.

In this example, the Box Area, Box Height and Box Width values are all linked, and are thus to be presented to the network at adjacent inputs. This also applies to the Centroid X and Y positions.

### 6.6.6 - MLP Training

Using the values presented earlier (SOM_X, SOM_Y, Euclidean Distance, Metric, Centroid_X, Centroid_Y, Box Width, Box Height, Box Area), a number of networks have been developed, based on a three layer structure (2 hidden layers and 1 output layer) and using a sequential training process.

The number of neurons per layer was varied between 4 and 7 for layer 1, and 2 and 5 for layer 2. The output layer was kept consistent with a single output.
A single output is, in this case, sufficient, as the data is being classified into one of two classes (person, or no person). A possibility would be to enlarge the output layer to two neurons, thus dedicating one neuron to each data class, but tests carried out

during the initial SOM tests (linked to small MLP's for performance evaluation) showed no improvement between the two structures. In a number of cases with different MLP structures and a variety of mapping SOM networks, the output error varied only between a few decimal positions, with no consistent trend justifying the increased network complexity.

It must not be forgotten that the final structure is to be kept to the absolute minimum size, in order to enable a rapid translation to a hardware model. The final network size and input data vector size will be instrumental in determining the processing power required in the final system, in this case, a system to be integrated into the actual camera module, thus critical in determining the cost of the entire system.

The networks developed are based on two particular SOM models using the Extremes data set. The best performing SOM networks with training cycles of 200 and 400 epochs, initial learning rates of 0.4 and momentum rates of 0.1. The learning rate and momentum rate values are dynamically updated during the training process in order to optimise the results obtained: Whilst these parameters initially start with quite high values which allow for larger adaptation within the network, their values are gradually reduced to near-zero following an exponential curve.

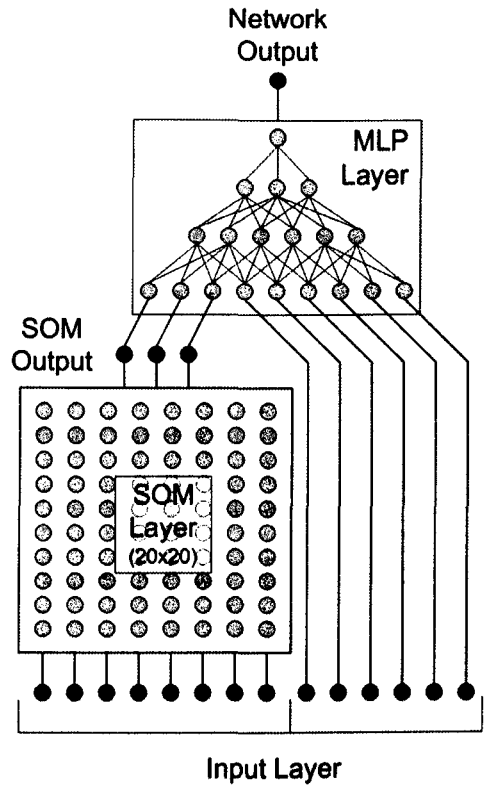



*Fig.124:Final Network Architecture*

*(Note: Not all connections in the networks are shown)*

The complete network structure is shown in *Fig.124.*

The diagram above illustrates the most successful form of network which was developed, using a 20x20 SOM network to process the Extremes segment measurements, the results of this mapper and 6 other inputs from the raw data being in turn fed into an MLP network with the following structure:

Input Layer: 9 Neurons.

Hidden Layer 1: 7 Neurons.

Hidden Layer 2: 4 Neurons.

Output Layer: 1 Neuron

(As the input layer does not contain any adjustment weights, taking the data values directly, this is not considered as an actual adjustment layer of the network).

A number of sequential training sessions were carried out with the said architecture in order to obtain the best starting point, an important feature considering the network weights random initialisation process.

The training interruption parameters were set as follow:
Interruption via testing results or on reaching a minimum RMS error of 0.01 on the training data.

The effectively reached interruption point eventually retained the following values:

Training Data RMS error: 0.087258.
Testing Data error: 0.065488.
overall training period: 97 Epochs.

Even though the target RMS error value had not been reached, the training process was stopped at this stage as further training cycles were only leading to a worsening of the network resolution when

considering the performance of the network on unseen data, thus implying that further training was only leading to an overfitting of the training data set.

The network training was then repeated using the same initialisation point, as the data presentation order had been randomised.

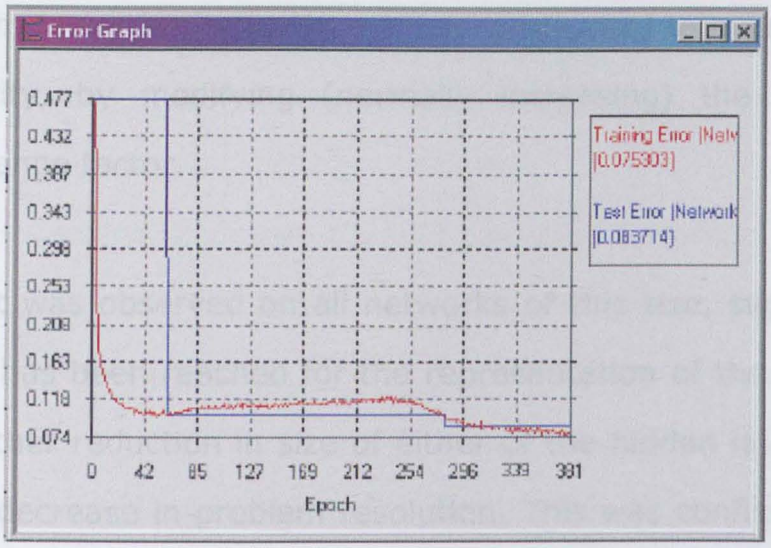
This process was repeated, using variations to the network momentum and learning rate values, in order to optimise the particular set of starting weight values.

The entire process was repeated over a selection of new weight initialisations, which provide not only new parameters to optimise the training run, but an entirely different topology offering different minimisation possibilities.

Carrying out these optimisations over approximately 10 new weight initialisations reduced the final error level by a value 0.007, thus providing nearly a 1% improvement in the network resolution.

This can be seen in *Fig.125*:

*Fig.125:Optimised Network Resolution*




Where the testing data error represents the performance of the network on a random selection of novel data, representative of the full data-range.

As can be seen on the graph above, the crossing point of the two error lines occurs, in this case, on approximately the 310th training epoch, after which point the testing error starts to increase whilst the training error continues to decrease, indicating that the network is now purely learning the training dataset instead of generalising it, and is thus losing its ability to classify novel data sequences.

The graph in *Fig.125* showsn that this is not the first crossing point of the training and validation error lines, which is due to the overcoming of a local minima in the network topology. This is shown by the slight rise in the training error value, followed by a relatively sharp drop from 0.118 to 0.074. Such a local minima is a common feature in most networks which, if not taken into consideration, can provide sub-optimal performance. The

momentum factor in the training sequence is used to overcome this particuliarity, by modifying (normally increasing) the calculated weight change factor.

This effect was observed on all networks of this size, suggesting a limitation has been reached for the representation of the data, and that a further reduction in size of either of the hidden layers would lead to a decrease in problem resolution. This was confirmed when smaller networks were trained but failed to achieve RMS error values lower than 1.2.

Examining the weights of the most successful network also revealed no values close enough to zero to justify cutting a particular connection out of the network structure.

Considering the other option, i.e., enlarging the network structure, a few experiments were carried out using structures varying between 2 and 3 hidden layers, with up to 15 nodes per layer.

Unlike a more restricted network, where the network topology does not allow the network to map the data accurately enough, thus leading to poor problem resolution, the larger a network becomes, the greater the danger of the network overtraining and simply learning the training data set, thus not developing any generalisation rules for unknown data.

This effect was observed on the larger network tests, with the RMS error value dropping down below 0.03, but coupled to an ever increasing value for the test data set error.
This factor seemed to occur immediately when a third hidden layer was introduced, and for a two layer structure, was apparent when the first hidden layer size surpassed the size of the input vector.

Experiments in inverting the layer distributions (first hidden layer smaller than the second hidden layer) were briefly tested, but the problem being considered is not one of data compression or noise suppression, but one of accurate classification. This type of structure is therefore not suited to the type of data analysis expected of the network.

## *6.7 - Conclusion*

Using the current approaches in the stages of data extraction, preparation and analysis, a valid system has been established which permits a satisfactory resolution of the problem at hand.

The addition of a second supportive SOM mapper using the radial object measures might be worth considering, to obtain a yet higher accurate final classification rate, although this would be at the

expense of a slower system response, due to both the data extraction and analysis through the network.

With an overall error rate of 0.0654 on data not seen in the training stage, the final network is pleasantly compact:

7x9 connections on level 1 : 63

7x4 connections on level 2 : 28

4 connections on level 3 : 4

Total MLP connections of 95.

The heavier part of the processing is embodied in the SOM data mapper, with a 20x20 structure and 8 inputs, which results in 3200 connections.

The advantage however remains in the fact that the dynamic adaptations are taking place in the initial image analysis stage, thus not requiring any further training on the part of the networks presented here.

# 7 - Conclusion

*There comes a time in the history of any project when it becomes necessary to shoot the engineers and begin production - MacUser,1990*

The initial reason for starting the development of the Intelligent Optical System was in response to the extremely high rates of false alarms of unattended surveillance systems, and as a way of promoting the use of optical surveillance methods within the home security segment, doing away with the need for trained personnel in both the system setup and the system operation phases.

Initially, the study was to cover the fields of both fire detection and intruder detection. After a period of initial research, the fire detection aspect was dropped as extending the scope of the project too much, and it was decided to concentrate on the field of intruder detection.

Although, due to unforeseen circumstances (Weyrad Ltd. filed for bankruptcy in late 1999 and was subsequently split up and sold to a number of different companies), the product itself was never developed to the stage of a commercial prototype, the system development can be considered to be a success.

From a failure rate (False alarms and mis-classifications) in commercial systems approaching 97%, the IOS system has reached a correct classification rate of 94%, with the largest proportion of the remaining 6% being due to false negatives, using an approach which can be mounted in any type of environment without having to retrain the entire system.

This dynamic adaptation to the system's environment represents a huge advantage for a commercial application, meaning that successful detection can be carried out within a changing environment without any detrimental effects to the actual detection rate, a problem which is commonplace in many automated surveillance processes.

Given the initial conditions, these have been satisfied:

- The final system is capable of operating with no prior knowledge of its environment.
- The system camera installation does not require any special training, any location will do.
- The system is capable of analysing multiple objects in each image, even if these are partially obstructed or affected by other noise.
- The system can operate entirely without operator intervention (even though it is currently in the form of modules, created for ease of development and testing, these can easily be integrated into a single streamlined package, as each module only needs to be started – all calculations are carried out autonomously of

operator intervention.)

- The system has been kept compact to allow for easy integration into a stand-alone product.
- Running on a Pentium II-300Mhz computer running Windows NT, the system is able to run a complete image analysis sequence within 0.0625 seconds. This does depend on the image complexity and the number of objects detected, and is a combined value taken from timing the various separated processes. Once these are integrated into a single streamlined package, running on dedicated hardware, it is assumed that this time would drop approximately by half, thus allowing for a close-enough match to real-time performance for a regular surveillance system.

The final network structure can be observed in Appendix K.

# 8 - Further Studies

*Now that we have all this useful information, it would be nice to be able to do something with it - Unix manual*

The system developed so far, although it uses many dynamic features within the image processing stages, is essentially a static system which analyses one image at a time, and does this completely separately of previous images within a single surveillance sequence.

On the current system, there are two main areas which would benefit of an entirely dynamic, time-based approach:

## *8.1 - Datum Image Setting*

The datum image is currently set at the beginning of a detection sequence, and is then used throughout the surveillance period, using various methods to optimise it in regards to the incoming camera images.

If however, the surveillance conditions are subject to larger changes, the datum image will no longer be presenting an optimal measurement base. In such a situation, it would be advisable to

capture a new datum image, obviously only if no actual object movement is detected in the surveillance area !

Such a new datum image capture could be triggered by one of two types of changes:

- Dramatic overall light level change.

  On each image, the median colour level is measured. If this median level on the camera image is consistently higher or lower by a factor of 10%, a new datum image should be generated. The actual time frame on which to base such a decision should be sufficient to take into account normal variations due to:

  - - Cloud Movement.
  - - Objects temporarily covering the entire camera lens.

  A suggested value would be set to approximately 10 minutes, thus avoiding too many updates from occurring due to changing weather conditions.

  It would thus be sufficient to plot the average median light level difference over the selected time frame to provide an update decision. This is therefore based on an already existing calculation, thus avoiding an excessive extra calculation load from being put on the overall system.

- Constant image difference detection over a given time period.

  Such a condition is likely to occur on an outdoors based

surveillance system, where the actual surveillance area might not be completely controlled. This would apply if the surveillance area was covering an area such as a storage area, where a change in the actual environmental geometry might occur:

- A box might be added to or removed from the surveillance area. Once such an event has been registered by the detection process, as long as it remains present, it need not be repeatedly analysed.

- Using the current system, each area of change within the image is initially stored within its own matrix, providing a number of definitions relating to the geometry of the object.
  It would be possible to store at least a subset of this information, allowing the system to memorise or compare the position of non-target objects within an image.

- Over a predefined time frame, the constant detection of a given object could then lead to a system datum image update, thus removing a source of system slowdown (each object detected leads to a system response penalty, as the object must first be analysed then classified).

The two methods considered essentially lead to the type of effect within the image. One will be an overall light level change, whilst the other will lead to a local effect. In order to combine the processing of these parameters, one method would be to split the

overall image into a number of sub-grids, which can then be analysed for light level differences. This would effectively remove the need to memorise all detected object parameters, depending on the size of the defined grid, as a constant difference within one or more of the subzones would lead to an overall datum image update.

## *8.2 - The Object Classification Process*

The object classification process, although dependant on a number of dynamic features, it itself also basically a static process, analysing each image entirely independently of the previous ones.

For a surveillance system, this is however a disadvantage, as objects within a scene follow dynamic paths: A person might walk behind a car, thus being partially or entirely hidden for a few seconds.

Not only this, but the recognition ratio of an object is likely to change over time as different aspects and thus geometries are presented to the camera.

It is important to note that the final classifier does not output a simple YES/NO condition, but results in a classifier percentage level of confidence, in this case a six-digit precision value, which will

fluctuate for each frame of an image, depending on the object position and geometry as well as the image noise level (The noise correction algorithm does affect the final data extracted for a given object within the image).

It is therefore advisable to provide a confidence level tracking value for each identified object within the image [*66, 61*]. Using such a function, the previous classifier output, or an average value of the previous classifier outputs over a given number of frames, can be provided as an extra input to the final classifying stage, thus providing a form of bias which can help in situations where an object is temporarily lost of partially hidden. Such a process could also assist in eliminating obvious non-target objects before these are processed by the classifier, although such an approach is slightly more dangerous, as a valid target could well initially be classified as 0% valid if it is not within the actual detection range. This would then provide an artificial bias to a non-target output, which would then require a number of successful target classifications to reach a non-biased situation, unless the actual network output were considered on a logarithmical basis, thereby leading to a much larger contribution by valid targets as compared to non-valid objects.

The overall mechanism of such a process can be seen in *Fig.126*:

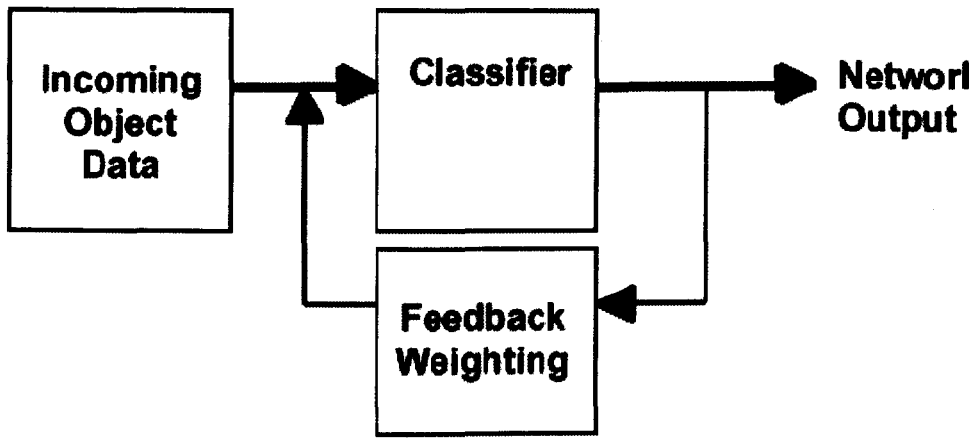



*Fig.126:Feedback Process*

Such an approach would then open the way to additional features such as object trajectory prediction, which would provide extra information as to the nature of the object detected. [*17*].

# 9 - Appendix A

## 9.1 - Noise Analysis

| IMAGE NAME | % PIXELS SET | NOISE CORRECTION LEVEL |
|---|---|---|
| capture0038.vos | 45 | 11 |
| capture0048.vos | 8 | 0 |
| capture0058.vos | 10 | 3 |
| capture0068.vos | 17 | 10 |
| capture0078.vos | 10 | 9 |
| capture0088.vos | 12 | 5 |
| capture0098.vos | 18 | 13 |
| capture0108.vos | 17 | 13 |
| capture0116.vos | 3 | 7 |
| capture0124.vos | 2 | 3 |
| capture0132.vos | 2 | 3 |
| capture0140.vos | 2 | 3 |
| capture0148.vos | 2 | 3 |
| capture0156.vos | 2 | 3 |
| capture0164.vos | 2 | 3 |
| capture0172.vos | 7 | 8 |
| capture0182.vos | 3 | 4 |
| capture0192.vos | 2 | 4 |
| capture0202.vos | 1 | 1 |
| capture0212.vos | 1 | 1 |
| capture0222.vos | 1 | 1 |
| capture0232.vos | 15 | 10 |
| capture0242.vos | 6 | 10 |
| capture0252.vos | 6 | 12 |
| capture0260.vos | 6 | 12 |
| capture0268.vos | 10 | 13 |
| capture0276.vos | 78 | 15 |
| capture0284.vos | 5 | 5 |
| capture0292.vos | 3 | 5 |
| capture0300.vos | 2 | 2 |
| capture0308.vos | 1 | 0 |
| capture0316.vos | 1 | 0 |
| capture0326.vos | 1 | 0 |
| capture0336.vos | 2 | 0 |
| capture0346.vos | 72 | 15 |
| capture0356.vos | 15 | 9 |
| capture0366.vos | 20 | 10 |
| capture0376.vos | 10 | 7 |
| capture0386.vos | 7 | 3 |
| capture0396.vos | 7 | 3 |
| capture0404.vos | 9 | 2 |
| capture0412.vos | 7 | 8 |
| capture0420.vos | 8 | 5 |
| capture0428.vos | 17 | 12 |
| capture0436.vos | 12 | 8 |
| capture0444.vos | 13 | 8 |
| capture0452.vos | 8 | 8 |
| capture0460.vos | 9 | 8 |
| capture0470.vos | 11 | 8 |
| capture0480.vos | 14 | 8 |
| capture0490.vos | 4 | 4 |
| capture0500.vos | 2 | 2 |
| capture0510.vos | 2 | 2 |
| capture0520.vos | 2 | 2 |
| capture0530.vos | 16 | 14 |
| capture0540.vos | 5 | 11 |
| capture0548.vos | 10 | 13 |
| capture0556.vos | 53 | 15 |
| capture0564.vos | 78 | 15 |
| capture0572.vos | 3 | 4 |
| capture0580.vos | 1 | 4 |
| capture0588.vos | 1 | 2 |
| capture0596.vos | 1 | 0 |
| pic009.vos | 6 | 0 |
| pic017.vos | 7 | 0 |
| pic025.vos | 9 | 1 |
| pic033.vos | 11 | 12 |
| pic041.vos | 12 | 6 |
| pic049.vos | 19 | 14 |
| pic057.vos | 54 | 15 |
| pic065.vos | 19 | 15 |
| pic073.vos | 6 | 9 |
| pic083.vos | 6 | 5 |
| pic093.vos | 11 | 5 |
| pic103.vos | 8 | 4 |
| pic113.vos | 7 | 2 |
| pic123.vos | 6 | 2 |
| pic133.vos | 19 | 6 |
| pic143.vos | 19 | 8 |
| pic153.vos | 82 | 15 |
| pic161.vos | 81 | 15 |
| pic169.vos | 81 | 15 |
| pic177.vos | 78 | 15 |
| pic185.vos | 80 | 15 |
| pic193.vos | 17 | 8 |
| pic201.vos | 14 | 12 |
| pic209.vos | 14 | 11 |
| pic217.vos | 7 | 8 |
| pic227.vos | 6 | 8 |
| pic237.vos | 5 | 5 |
| pic247.vos | 6 | 5 |
| pic257.vos | 7 | 7 |
| pic267.vos | 11 | 7 |
| pic277.vos | 11 | 7 |
| pic287.vos | 12 | 10 |
| pic297.vos | 5 | 6 |
| pic305.vos | 5 | 5 |
| pic313.vos | 10 | 8 |
| pic321.vos | 54 | 15 |
| pic329.vos | 19 | 10 |
| pic337.vos | 11 | 7 |
| pic345.vos | 22 | 10 |
| pic353.vos | 13 | 13 |
| pic361.vos | 8 | 2 |
| pic371.vos | 4 | 0 |
| pic381.vos | 6 | 0 |
| pic391.vos | 6 | 0 |

## 9.2 - Image Subtraction Results

The following images illustrate a few cases of image substraction using the developed object detectionand noise reduction algorithms, as well as a classification using the first test MLP network. Although fully trained on artificial data, these images show a satisfactory performance on real data.

The sequences show the datum image, the incoming camera image to be analysed and the final object recognition with initial network classification, where the classification range is a percentage of certainty from 0 to 100 that the target considered is valid.


*Fig.127:Sequence 01*

*Fig.127* shows a fairly clean capture, where the resultant difference is very clean and a high classification is reached.


*Fig.128:Sequence 02*

*Fig.128* shows a slightly more difficult condition with ground shadows and similar colour bands.


*Fig.129:Sequence 03*

*Fig.129* shows a situation with a high level of distributed noise due to light casting. Note how the noise reduction algorithm serves to correctly isolate the valid target.


*Fig.130:Sequence 04*

*Fig.130* shows a more difficult situation where the target has identical colour shades to the background, creating local loss of

difference. The noise correction algortihm helps to counteract this effect, succesfully identifying the final object.


*Fig.131:Sequence 05*

*Fig.131* illustrates a difficult environment with many reflections and shadows. Partial loss of difference due to ground shadows on the target legs.


*Fig.132:Sequence 06*

*Fig.132* shows a similar situation to fig.131, however with more distributed noise on the right and lower edges of the image.


*Fig.133:Sequence 07*

*Fig.133* illustrates a condition with multiple targets, two valid and one non valid, as correctly classified by the initial network. Note the loss of the first targets' torso due to colour similarieties with the background, and the ensueing lower classification value.


*Fig.134:Sequence 08*

*Fig.134* shows a fairly straightforward target analysis with multiple occlusions in the torso area due to background patterns.


*Fig.135:Sequence 09*

*Fig.135* is again a fairly straightforward analysis on a smaller object.


*Fig.136:Sequence 10*

*Fig.136* illustrates a condition with severe shadowing, which is combined with the actual target into a single object with partial noise cleaning – Classification is still correct.


*Fig.137:Sequence 11*

*Fig.137* illustrates a condition with quite a high noise level due to light and shadow casting. The single valid target in the frame is correctly classified.


*Fig.138:Sequence 12*

*Fig.138* illustrates a case of an invalid target with a high local noise value due to surface reflections.


*Fig.139:Sequence 13*

*Fig.139* illustrates the case of an invalid moving target, couple with noise artifacts due to light/shadow casting. It is interesting to note that the noise artifact is not classified as well as the actual invalid target(the dog).


*Fig.140:Sequence 14*

*Fig.140* shows a valid and an invalid target in the same frame, with correct detection and classification for both.


*Fig.141:Sequence 15*

*Fig.141* shows two correctly identified and classified valid targets in a single frame.Note that shadows have been eliminated via the noise correction procedure.

## 9.3 - MLP Network Evaluations example

The exampe below shows an evaluation of a number of MLP-type networks on a same data set but with different architectures and weight initialisations.

| Results | False Positives | | | | | | Train Mode | | |
|---|---|---|---|---|---|---|---|---|---|
| | Linear | RMS | Normal | RMS | Architecture | Cycles | Sequential | Random | Replace |
| Net 1 | | | | | | | | | |
| 01-1 | 87,24 | 87,24 | | | 10-6-3-2 | 30000 | X | | |
| 01-2 | 87,24 | 87,24 | | | 10-6-3-2 | 25000 | X | | X |
| 02 | | | | | | | | | |
| | | | | | | | | | |
| Net 2 | | | | | | | | | |
| 01 | 94,42 | 94,42 | 85,71 | 85,71 | 10-6-3-2 | 30000 | | | |
| 02 | 92,19 | 87,88 | 85,71 | 55,26 | 10-6-2 | 20000 | X | | |
| | | | | | | | | | |
| Net 3 | | | | | | | | | |
| 01-1 | 76,71 | 77,35 | | 0,68 | 10-5-3-2 | 20000 | X | | X |
| 01-2 | 76,71 | 77,35 | | 0,7 | 10-5-3-2 | 30000 | X | | |
| '02-1 | 72,89 | 73,68 | 94,11 | 96,97 | 10-8-2 | 24400 | X | | X |
| '02-2 | 78,79 | 78,15 | 3 | 2,9 | 10-8-2 | 20000 | X | | X |
| 02-3 | 91,97 | 91,23 | 75 | 76,36 | 10-8-2 | 30000 | X | | X |
| | | | | | | | | | |
| Net 4 | | | | | | | | | |
| 01 | 83,73 | 83,57 | 92,15 | 91,26 | 10-5-3-2 | 34500 | | | |
| 02-1 | 78,47 | 78,47 | 0,74 | 0,74 | 10-6-4-2 | 30000 | X | | |
| 02-2 | 85,96 | 85,96 | 5,68 | 5,68 | 10-6-4-2 | 29000 | | X | X |
| 02-3 | 78,31 | 78,31 | 0,73 | 0,73 | 10-6-4-2 | 60000 | | X | X |
| 03-1 | 86,28 | N/A | | | 10-6-4-1 | 20000 | | X | X |
| 03-2 | 82,62 | N/A | | | 10-6-4-1 | 60000 | | X | X |
| 04-1 | 82,93 | N/A | | | 10-5-1 | 50000 | | X | X |
| 04-2 | 85,33 | N/A | | | 10-5-1 | 20000 | | | |
| 04-3 | 85,33 | N/A | | | 9-5-1 | 20000-lsn | | | |
| 04-4 | 73,68 | N/A | | | 10-5-1 | 30000 | X | | |
| 04-5 | 49,6 | N/A | | | 10-5-1 | 12100 | X | | |
| | | | | | | | | | |
| Net 5 | | | | | | | | | |
| 01-1 | 94,58 | 94,42 | 38,25 | 37,14 | 10-6-3-2 | 30000 | X | | |
| 01-2 | 83,25 | 80,7 | 5,71 | 4,95 | 10-6-3-2 | 30001 | | X | |
| 02-1 | 81,02 | 81,02 | 98,3 | 98,3 | 10-7-4-2 | 29500 | X | | |
| 02-2 | 87,56 | 87,56 | 6,41 | 6,41 | 10-7-4-2 | 30000 | | X | |
| 02-3 | 86,76 | 86,76 | 12,05 | 12,05 | 10-7-4-2 | 30001 | | X | X |
| 02-4 | 94,74 | 94,74 | 63,63 | 63,63 | 10-7-4-2 | 30002 | X | | |
| 02-5 | 88,68 | 88,68 | 8,45 | 8,45 | 10-7-4-2 | 50000 | | X | |
| 03-1 | 83,57 | 83,57 | 2,94 | 2,91 | 10-8-5-2 | 30000 | | X | |
| 03-2 | 83,09 | 83,09 | 5,66 | 5,66 | 10-8-5-2 | 30001 | | X | X |
| 03-3 | 94,74 | 94,74 | 63,63 | 63,63 | 10-8-5-2 | 30002 | X | | X |
| | | | | | | | | | |
| Net 6 | | | | | | | | | |
| 01-1 | 48,01 | 48,01 | 100,00 | 100,00 | 10-4-3-2 | 20000 | X | | |
| 01-2 | 95,06 | 95,06 | 19,35 | 19,35 | 10-4-3-2 | 30000 | X | | |
| 01-3 | 67,94 | 67,94 | 95,52 | 95,52 | 10-4-3-2 | 20001 | X | | X |
| 01-4 | 95,37 | 95,37 | 37,93 | 37,93 | 10-4-3-2 | 30001 | X | | X |
| 01-5 | 95,69 | 95,69 | 14,81 | 14,81 | 10-4-3-2 | 20002 | | X | |
| 01-6 | 95,22 | 95,22 | 36,67 | 36,67 | 10-4-3-2 | 40002 | | X | |
| 01-7 | 95,69 | 95,53 | 40,74 | 39,29 | 10-4-3-2 | 20003 | | X | X |
| 01-8 | 94,74 | 94,74 | 15,15 | 15,15 | 10-4-3-2 | 60003 | | X | X |
| 02-1 | 48,01 | 48,01 | 100,00 | 100,00 | 10-3-2-2 | 20000 | X | | |
| 02-2 | 95,53 | 95,53 | 25,00 | 25,00 | 10-3-2-2 | 20001 | | X | |
| **02-3** | **95,53** | **95,53** | **53,57** | **53,57** | **10-3-2-2** | **20002** | | **X** | **X** |
| 02-4 | 51,99 | 51,99 | 0,00 | 0,00 | 10-3-2-2 | 30000 | X | | |
| **02-5** | **94,58** | **94,58** | **11,76** | **11,76** | **10-3-2-2** | **30001** | | **X** | |
| 02-6 | 94,74 | 94,74 | 18,18 | 18,18 | 10-3-2-2 | 30002 | | X | X |

# 10 - Appendix B – Relevant British Standards

BS 5839: Fire Detection and Alarm Systems for Buildings

BS 7230: Theft Detection Systems

BS 7807: Fire and Security Integrated Systems

BS 820: Anti-Burglar measures in Buildings

BS 5446: Components of Automatic Fire Alarm Systems for Residential Premises

BS 4737: Intruder Alarm Systems

BS 5979: Code of Practice for Remote Centres for Alarm Systems

BS 6799: Code of Practice for Wire-free Intruder Alarm Systems

# 11 - Appendix C – Specialised Software Packages Used

## *11.1 - Neural Modelling*

- Neural Works Pro II
- Neusciences Neuframe
- TLearn v 1.03

## *11.2 - Data Analysis*

- Neural Works Predict
- SPSS

## *11.3 - Artificial Data Modelling*

- Macromedia Poser I
- 3D Studio Max
- Adobe Photoshop

## *11.4 - Code Generation*

- Microsoft Visual C++ v5

## *11.5 - Main Self-written Packages*

**- Bitmap Wave Comparator**

Analyses the colour waves of two bitmaps for rapid change detection.

Includes auto adjustment to varying light levelsover the images

**- Cheat Office**

Image analysis deomonstrator showing the process of image feature extraction as an executive toy, allowing an office background to be cancelled and replaced by any specified image.

**- Data Extractor**

Extracts required data parameters from a VOS format file

**- Multiple Data Extractor**

Extracts required data parameters from a VOS format file. Can deal with image noise and multiple objects

**- Neural Demo**

Small demontrator of a simple MLP network for human classification. Works directly on VOS format files.

**- Results Filter**

Package for adjusting the varying data formats from the TLearn package, for further use in Excel.

**- SOM Trainer**

Self Organising Map generation and training software with visual display of resulting network structures.

**- VIOS Neural Demonstrator**

More advanced neural demonstrator including image processing algorithms for object detection and noise cancellation.

**- VosDemo**

Simple demonstrator package for grabbing and comparing camera images. Also capable of saving the direct difference.

**- VosReader**

VOS manipulating package, allows user defined filters to be applied to a VOS format file.

**- VosViewer**

Rapid viewer for VOS format files.

**- Weyrad Demo**

Active demonstrator of image processing algorithms. offers sequential timed or manual activation, saves the resulting VOS files. Used for mass data generation.

# 12 - Appendix D - Data Pre-processing techniques, a Summary

## *12.1 - Scaling / Normalising*

The resulting value Y from an input X is calculated by:

$$Y = ( X-Xmin)/Xrange$$

This type of scaling is used for continuous variables, where each variable has its own separate dynamic range, and where the distribution of the values within the variable's range is fairly even.

## *12.2 - Angular Transforms*

Each input is transformed into an angular representation in radians, and a sin or cosine of the resulting angle is taken. This is then combined with the vector length to give a two component representation of complex data pattern.

Well suited to periodic variables, or for frequency analysis within determined ranges.

## *12.3 - Zero-Mean Unit Variance*

Used for continuous data, this method is applied to a complete data set and be applied either by row or by column (either by input set or by variable set).
The actual transform is as follows:

$$Y = (X\text{-mean})/\text{Standard Deviation}$$

## *12.4 - Binary Coding*

This techniques is useful for categorical variables where a scaling transform would artificially accentuate certain items.
The input can be coded using various methods, some of which are:

1 in n: 100, 010, 001

Gray Scaling: 000, 001, 011, 010, 110...

hermometer: 100, 110, 111

Continuous: 1/n, 2/n, 3/n....n/n

This does generally imply that a single data line becomes represented by a number of lines, defined by the encoding method chosen.

## *12.5 - Vector Augmentation*

This method can be applied to multidimensional data vectors, and is used to extract either the size or the direction of the data, depending on the method chosen.

### 12.5.1 - Method 1, used when the vector size is critical

Given an input vector $E'=(e_1,e_2,e_3..e_n)$

E′ is calculated for all input vectors, where

$$||E'|| = \sqrt{(\Sigma e_i^2)}=1$$

A value N is then chosen such that N>E′ ( N=1.1E′) and a new entry d to the input vector is calculated such that

$$d = \sqrt{(N^2-||E'||^2)}$$

This results in a new vector $E''=(d,e_1,e_2,e_3..e_n)$

The final data vector is then considered as

$$E = E''/N$$

## 12.5.2 - Method 2, used when the direction of the vector is critical

E′ is calculated as above, but the final data vector is obtained by :

$$E = E' / ||E'||$$

# 13 - Appendix E – Image Stabilising Methods

Two main methods exist for image vibration supression :

1 - Mechanical, using stabilisors or compensators.

2 – Software, by calcuting image movement and introducing a corrective vector.

In *Fig.142* is shown a modern mechanical system, as built in professional digital cameras.

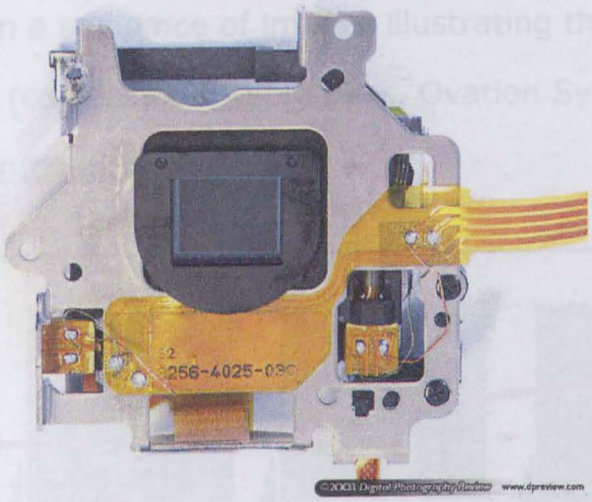


Fig.142:Image Stabiliser, Courtesy of "Digital Photography Review", source Konika Minolta.

In such a system, a motion sensor is used to capture the type of motion, and the entire lens and capture device is then moved accordingly to compensate for the movement. This approach is rapid and effective, but also highly expensive, and dependent on a

combination of mechanical components which are subject to normal wear.

Software compensation depends on comparing successive image frames to determine the size and direction of movement. Many systems limit themselves to comparing a specific area of the image (i.e. The center) in order to speed up the process, and then attempt to lock onto a recogniseable feature of an appropriate size.
Although this method is computationally more expensive, it is cheaper to implement and does not really on any mechanical systems which are prone to failure.
In *Fig.143* is shown a sequence of images illustrating the software correction process (courtesy of Stable Eyes, Ovation Systems Ltd. http://www.ovation.co.uk)

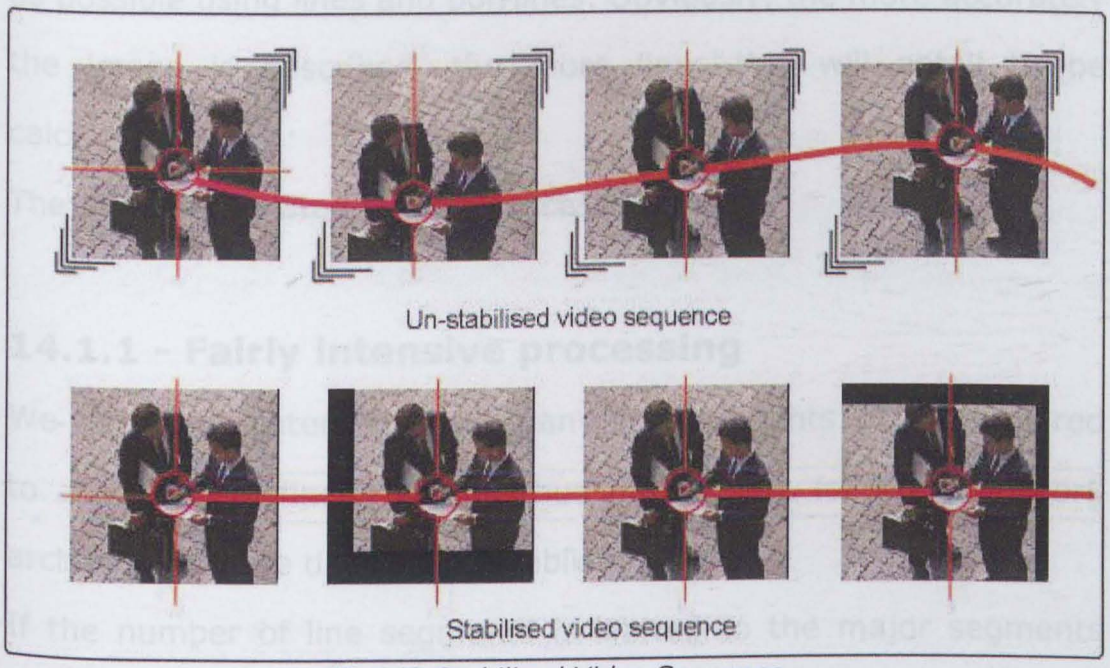



*Fig.143:Stabilised Video Sequence*

# 14 - Appendix F – A Meeting with Dr. Paul Rosin, 19.11.98

Dr. Paul Rosin from Uxbridge was approached to lend some expert opinion, having many years of experience in image feature extraction. This is the stage which effectively converts the filtered image data into a data set useable in a neural architecture.

Throughout the meeting, he outlined a number of different methods for image feature extraction and matching, which will be reviewed below:

## *14.1 - Line matching*

This approach consists of trying to describe the data as accurately as possible using lines and polylines. Obviously, the more accurately the image is described, the more lines this will entail to be calculated.

The problems related to such processing are:

### 14.1.1 - Fairly intensive processing

We cannot predetermine how many line segments will be required to accurately define a shape, thus making the following network architecture more difficult to establish.

If the number of line segments is limited to the major segments only, there might and probably will, be a loss of important

description data.

As the shape is described more accurately, the calculations become more and more unreliable and prone to error, as each line is calculated with reference to its successor.

### 14.1.2 - Shape Properties

We are here considering calculating a fixed number of relationships within the image. These could be features such as max height and width, positioning of Centroid, area calculation, perimeter measurement, generic displacement vectors and global image positioning.

Certain of these properties are unreliable as a basis for object recognition, for example, perimeter measurements can very easily be influenced by noise in the shape, in this case, area measurement is much more relevant and reliable. Obviously, many of these measurements will be scale dependent, but will be interlinked. The use of a graduated camera would make measurements easier, giving full scalar information, but distance can be compiled from the data interaction.

Certain shape properties will also be heavily influenced by image resolution, thus requiring a standardised set of image input parameters.

Generally, the first processing algorithm must be the best and most reliable, as any errors occurring at this stage will be fed right

through any further processing and be exaggerated at every single stage. The final cumulative error and data loss can become quite large for complex algorithms.

# 15 - Appendix G – Infra-Red Imaging

As mentioned in the previous section, an experimental capture was also run using a small board camera with limited IR response.
A number of different situations were examined, with various lighting set ups, including no lighting at all apart from the onboard IR LED's.

Generally, the quality of the images obtained was quite high, with accurate and sharp object outlines and little or no blurring over the focal range (.5m -> 10m). The effects of the IR response were very interesting. All following comments are valid within the range of the IR illuminance only:

Under purely artificial lighting conditions, most shadows cast from objects were altogether cancelled out. Although clearly visible to the naked eye under the trial conditions, they did not appear either on the monitor used at the time, nor on the final saved film. Objects in the cast shadow suffered from no loss of definition in any way.

Under natural light conditions, this shadow cancellation was very much reduced, to the point of being practically non existent, however, the sharper images more than compensated for the loss of this 'feature'.

Under no light or low light conditions, the effect of the IR illumination and response was most marked. Although objects tended to lose any tonal information, their general outlines were enhanced. At this stage, it is not the colour of the object which is affecting its visibility, but the material it is made of, and also its heat absorbance capacity. For example, polished black leather shoes appeared as near white when within the IR range.

To explain the marked difference in shadow elimination between natural and artificial lighting conditions, our assumption is that the wavelength of the artificial lights (regular white fluorescent tubes) have a much lower red component. Indeed, these types of lights are normally balanced nearer towards the blue end of the spectrum, due to the process of phosphorus excitation which they employ. Any shadows created under these conditions will therefore be colder than the same shadows cast by natural light. If we were to use regular incandescent bulbs as our light source, we might very well find this shadow elimination property much reduced, as such bulbs have a higher red response.

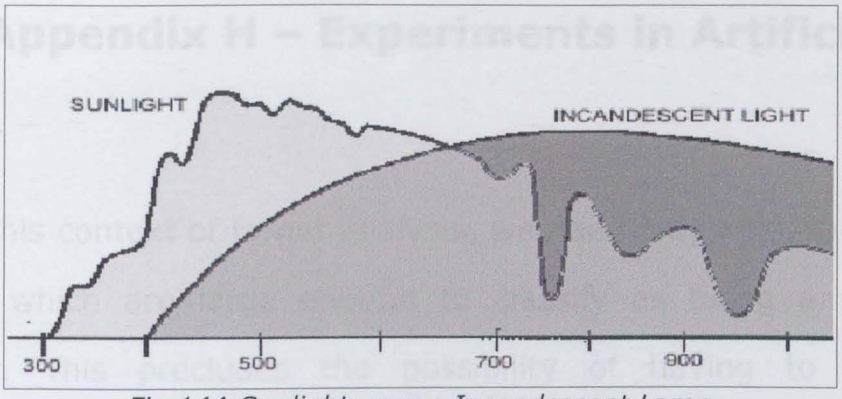



*Fig.144:Sunlight versus Incandescent Lamp*

*Fig.144* shows the spectrum for sunlight and an incandescent light. The higher intensity of the IR component for the light bulb can be clearly observed. The values on the scale are in Nm (Nanometers), with human vision ranging from about 320 to just over 700, IR being at the higher end of the scale.

# 16 - Appendix H – Experiments in Artificial Data

Wihtin this context of target analysis, we need only analyse moving objects which are large enough to classify as being a possible intruder. This precludes the possibility of having to analyse inanimate objects or random movements. Natural motion due to leaves moving etc are delt with in the preprocessing stage. The only possible targets which then remain are human and large animal.

Having developed the system with human training sets, we know that performance in that respect is adequate fro this first network. We have however not presented it with any animal data.

For the sake of ease of use, the room modelled in 3DStudio in the previous sections was used, with a mesh of a large sized dog, and a number of scenes were thus created, as can be seen in *Fig.145*:

*Fig.145:Dog Mesh*

The final analysis of the data showed certain interesting characteristics:

When presented side on, the dog was generally classified as being between 40% and 60% positive target. However, presenting the dog in a frontal view caused a dramatic rise in the recognition level, with the final output lying between 70% and 85%.

Although the classification never rose above 90%, it is still sufficiently high to raise concerns as to the current network validity. If we observe the front profile of the dog (*Fig.146*), we can see that it very closely resembles the ideal human profile, although obviously smaller. Within this system, we are however never taking

account of scale, only of the direct screen size of objects.

*Fig.146: Highlighted outline of dog seen in a frontal pose*

This aspect of changing target resolution as the object moves relative to the camera also highlights the benefits which could be obtained in using some form of history, or time tracking. If each frame's target resolution were recorded, and each object tracked (parameters such as Centroid displacement and relative size could be used) the final recognition output would not be a straight network calculation but would be a summation of past responses. This final result could also be calculated using an intelligent system, or might just be a straight averaging calculation.

# 17 - Appendix I – Contour Analysis Considerations

Whilst the experimentation sequences described in the previous sections have proved most successful, we are starting to investigate a different technique for identifying an intruder.

The current approach of identifying every moving object within the image, and analysing each of these separately can be quite a lengthy process. In addition to this, non-human objects could be misclassified, leading to false alarm conditions. We could however analyse the entire difference image as a set of fixed contours, which we would then attempt to map onto objects in the image. If we have generic contour maps for a standing human and a crouching human, and maybe also for similar objects, such as a dog walking, we could present each of these to the image and compare the resulting correlation values to determine wether or not the target is human.

This approach requires only a number of adjustable templates to be stored in memory. These can then be deformed within set limits to map onto the image.

Similar work has been carried out by the Universities of Leeds and Reading, with their Vehicle Tracker and People Tracker. When

observing their work, the only detection flaw resided in the fact that the person had to be completely visible before the system could carry out a positive identification. Once the person had been picked up, they could then be partially obscured whilst still being tracked, due to the use of incremental time information in the accuracy of the mapping. Whilst this work made use of extensive calculations which could not be used in a real time system, the approach was interesting, and could be adapted to a neural system with a SOM network.

Using this approach, a direct analysis could be made of the incoming camera image, without going through the stage of image comparison. If the image is reduced to tonal value contours, we can then apply the templates directly to this. This might be a much faster way of analysing an image, with a corresponding lower loss of detail.

An analysis was carried out on the capture sequences, to determine a generic standing human shape, and a generic crouching human shape. The results can be seen in *Fig.147* and *Fig.148*. As we can see, the human form can fairly easily be reduced to a number of simple geometric shapes, with variable mathematical relationships within and around these shapes varying according to the aspect of the target to the camera and the pose of the target.

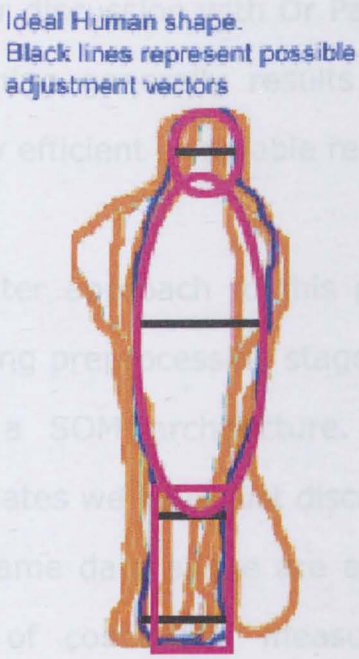



*Fig.147:Human Shape Analysis*

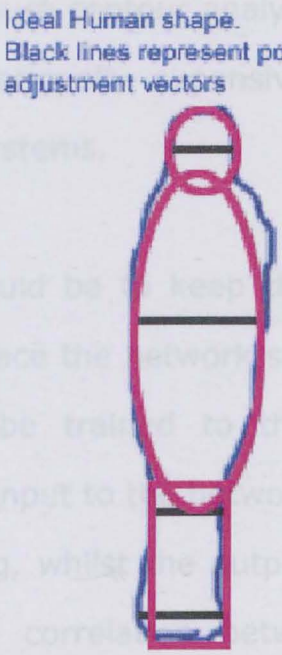



*Fig.148:Ideal Human Shape*

Further experimentation with Adobe Photoshop has shown that pure image tonal level analysis might be insufficient to extract suitable contours without the need for extensive reconstruction through data extrapolation . This can be seen in *Fig.149*:

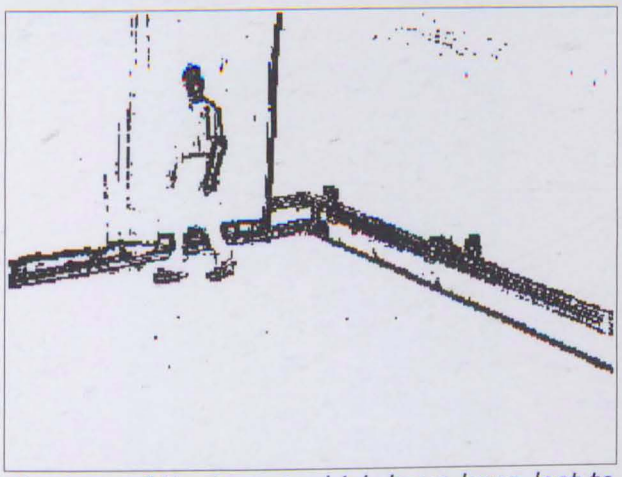


*Fig.149:Note the legs of the target which have been lost to the contours*

In our discussion with Dr Paul Rosin , pure contour analysis via line matching generally results in computationally expensive but not highly efficient or reliable recognition systems.

A better approach to this problem would be to keep the existing working preprocessing stages, and replace the network section only with a SOM architecture. This can be trained to the stylised templates we have just discussed. The input to the network remains the same data as we are already using, whilst the output is some type of confidence measure, or the correlation between each existing template and the image object being analysed. In this way, we can use a type of voting structure, and can also plot the varying likelihood of the target being human over a certain time span. Such a feature could assist in the constant detection of a target being partially obscured and deformed through spurious shadow effects or through the target being momentarily obscured by other objects in the image.

# 18 - Appendix J – Development Images

## *18.1 - 10x10 SOM Network Tests*

The following images show the final network node mapping and weight distributions for a 10x10 SOM network after the given number of training cycles.

### 18.1.1 - Center Data Set

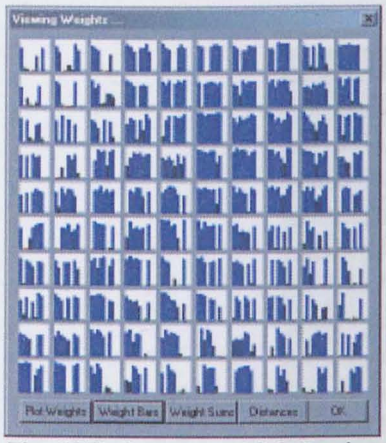



*Fig.150:Center - 100 Cycles*

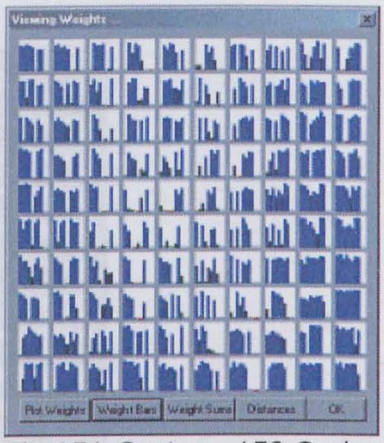



*Fig.151:Center - 150 Cycles*

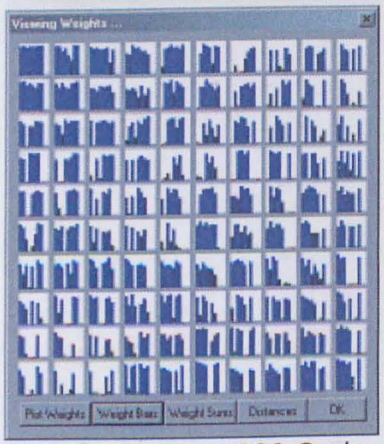



*Fig.152:Center - 200 Cycles*

## 18.1.2 - Cross Data Set

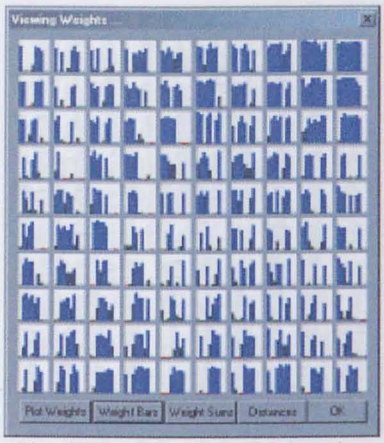



*Fig.153:Cross - 100 Cycles*

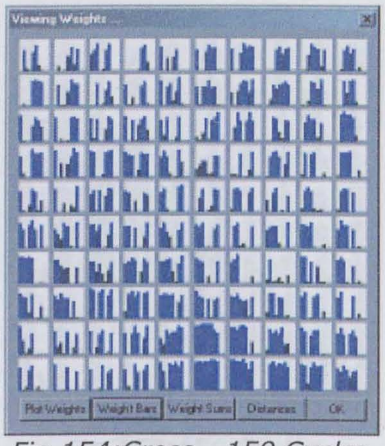



*Fig.154:Cross - 150 Cycles*

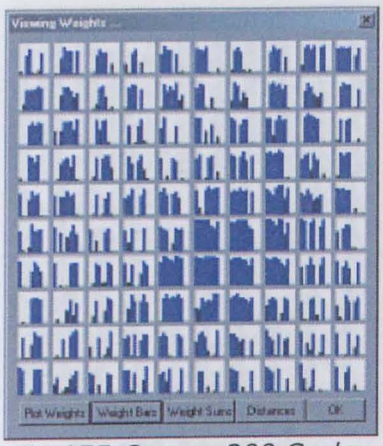



*Fig.155:Cross - 200 Cycles*

## 18.1.3 - Extremes Data Set

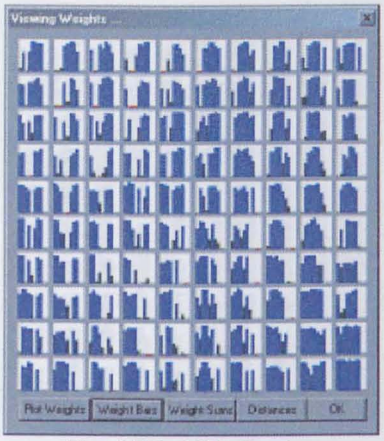



*Fig.156:100 Cycles*

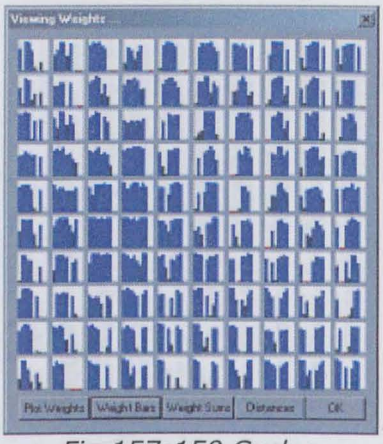



*Fig.157:150 Cycles*

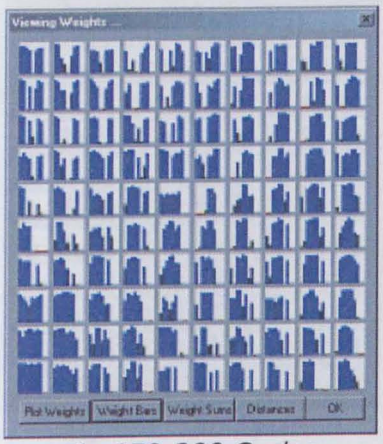



*Fig.158:200 Cycles*

## 18.1.4 - Middle Data Set


*Fig.159:100 Cycles*

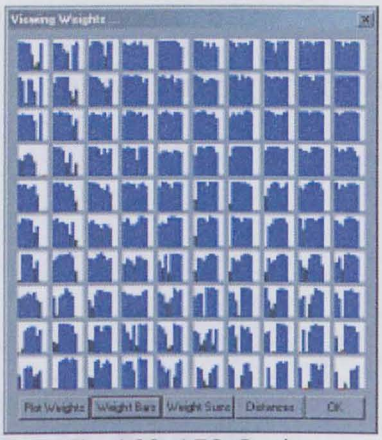



*Fig.160:150 Cycles*

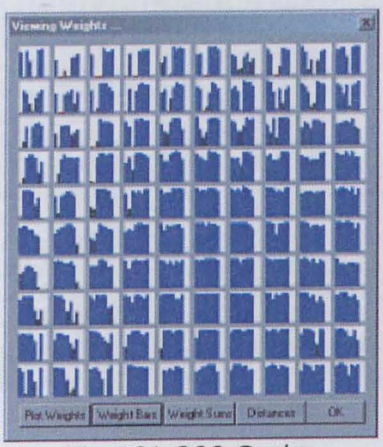



*Fig.161:200 Cycles*

## *18.2 - 20x20 SOM Network Tests*

The following images show the final network node mapping and weight distributions for a 20x20 SOM network after the given number of training cycles, showing a much better data space resolution.

### 18.2.1 - Centres Data Set

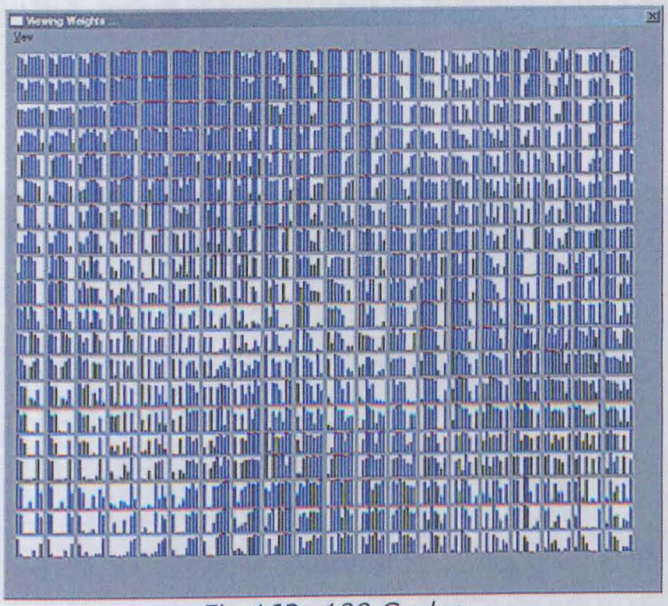



*Fig.162: 100 Cycles*

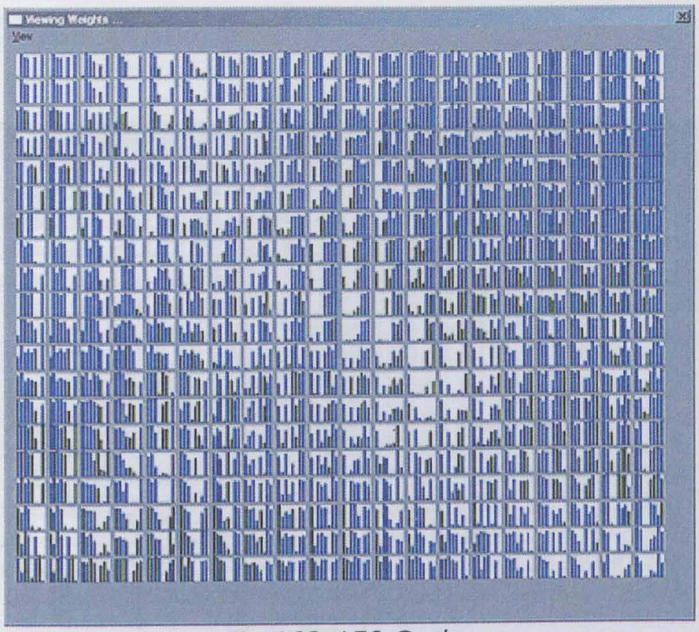



*Fig.163:150 Cycles*

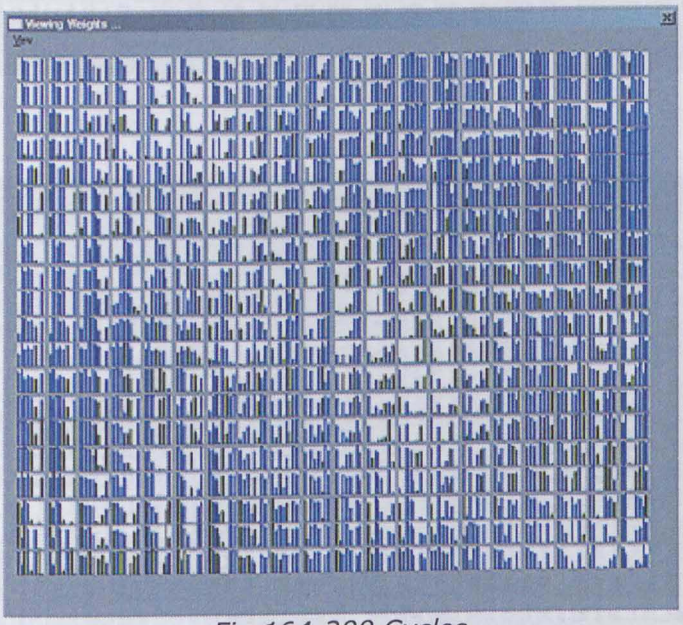



*Fig.164:200 Cycles*

## 18.2.2 - Cross Data Set

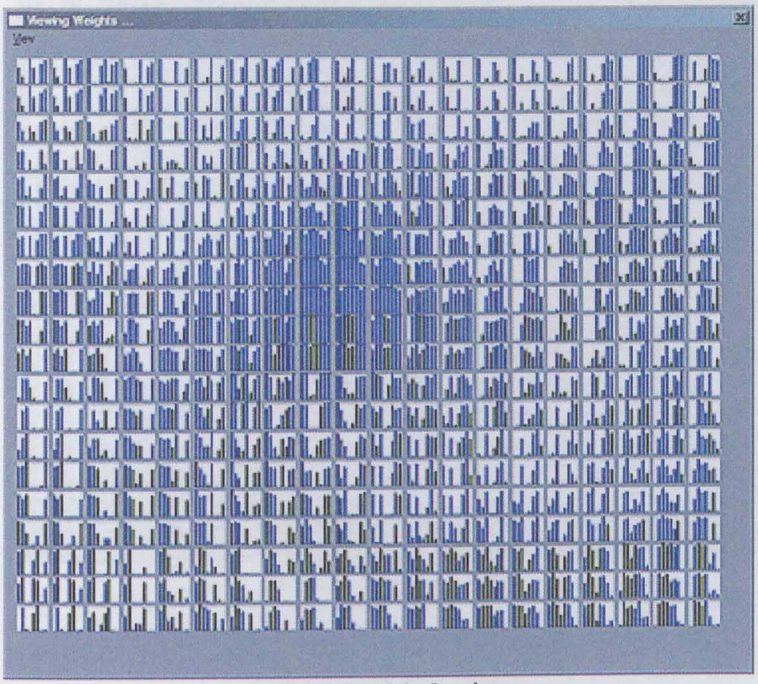



*Fig.165:100 Cycles*

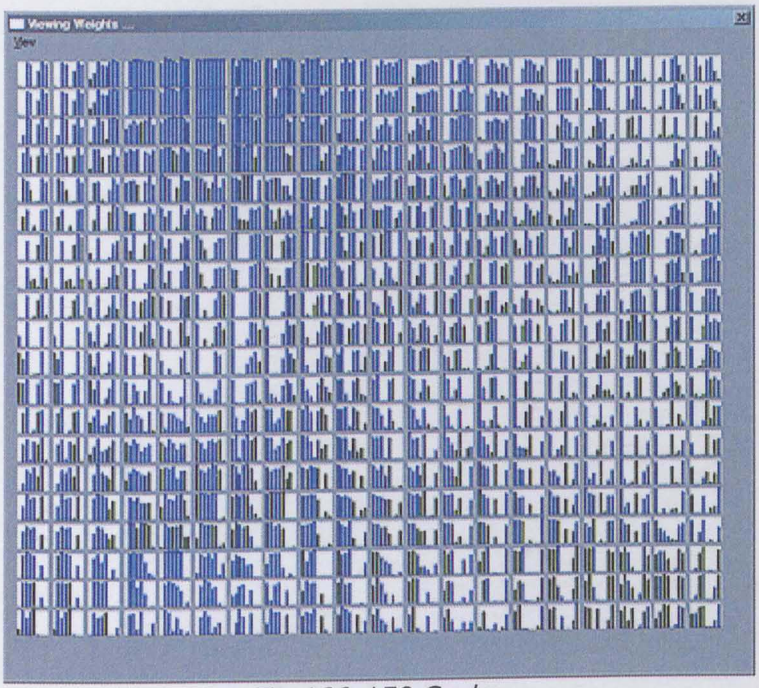



*Fig.166:150 Cycles*

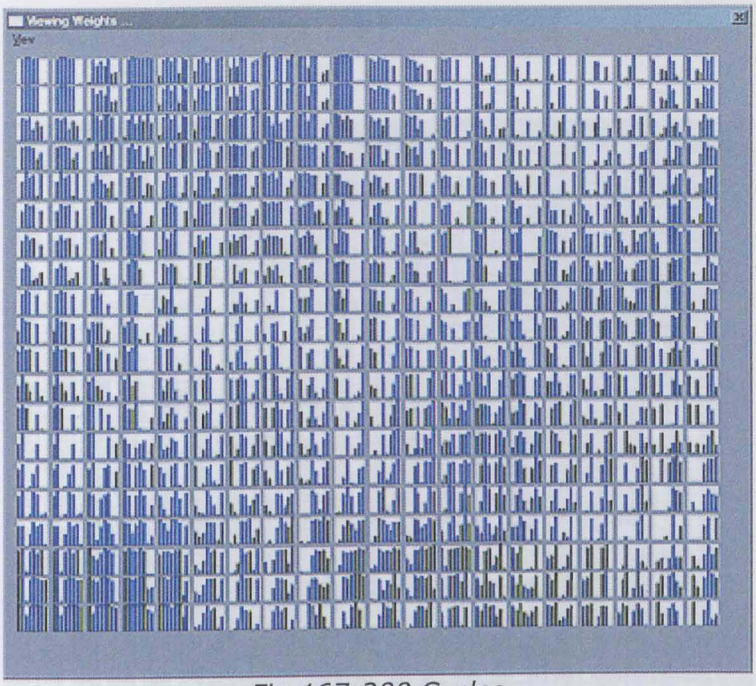



*Fig.167:200 Cycles*

## 18.2.3 - Extremes Data Set

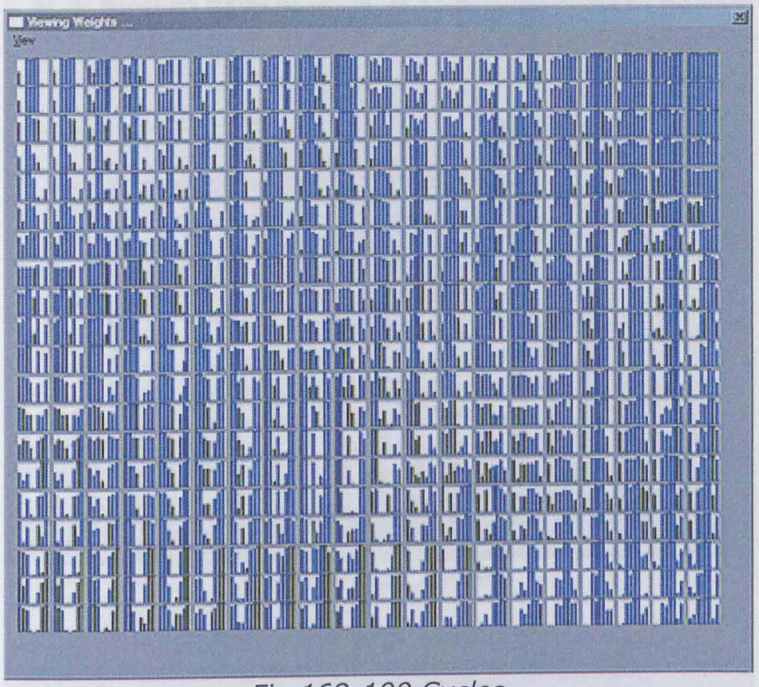



*Fig.168:100 Cycles*

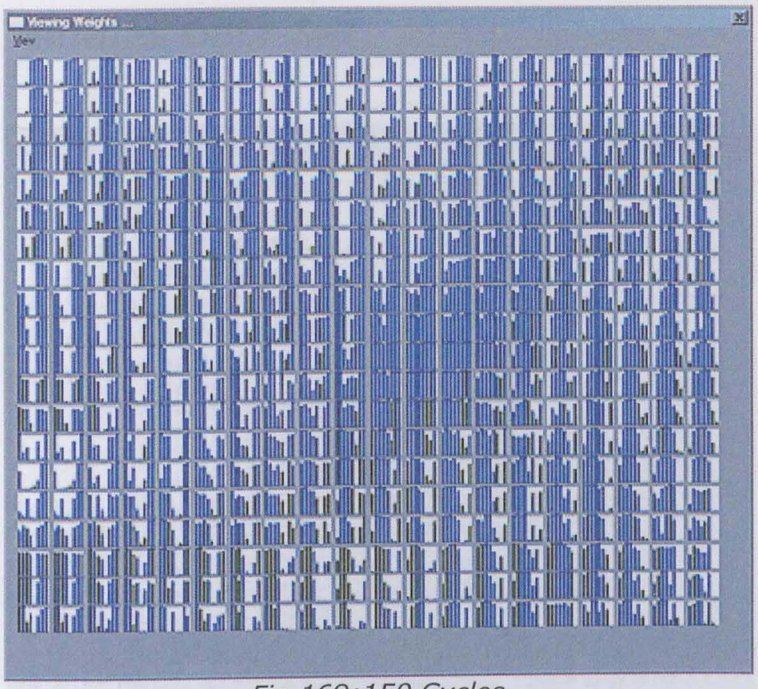



*Fig.169:150 Cycles*

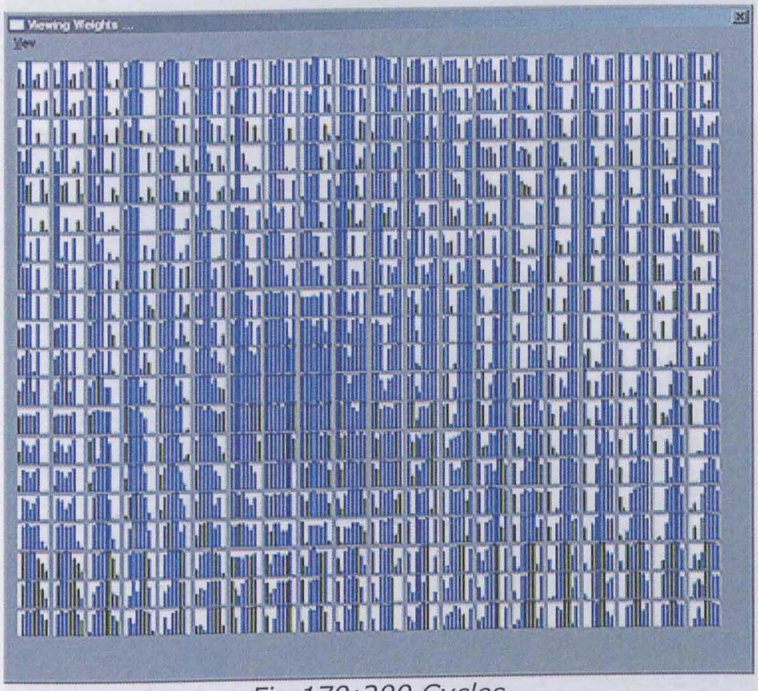



*Fig.170:200 Cycles*

## 18.2.4 - Middles Data Set

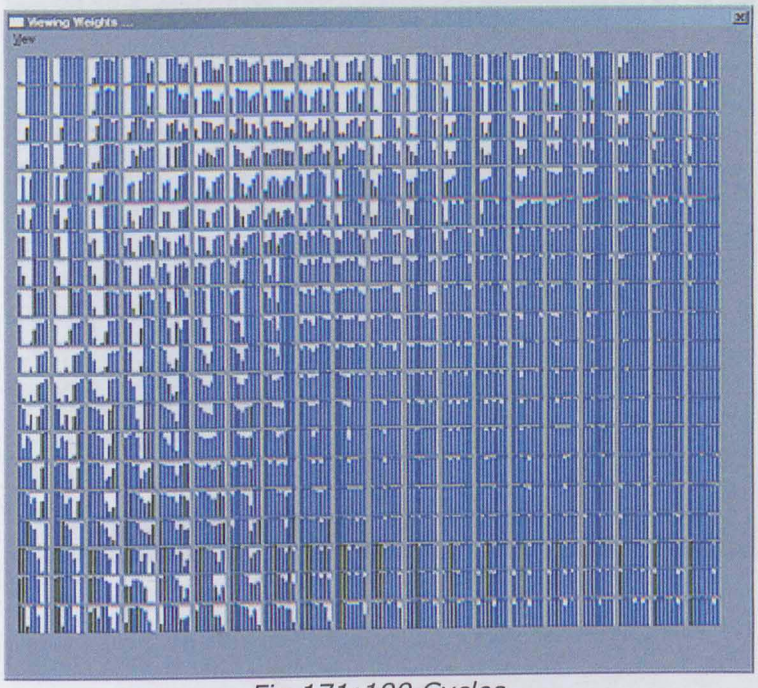



*Fig.171:100 Cycles*

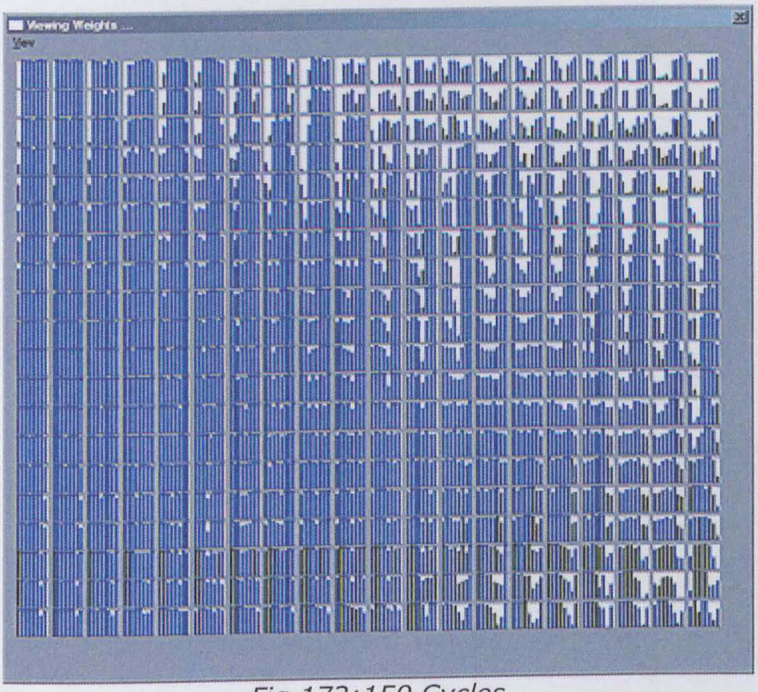



*Fig.172:150 Cycles*

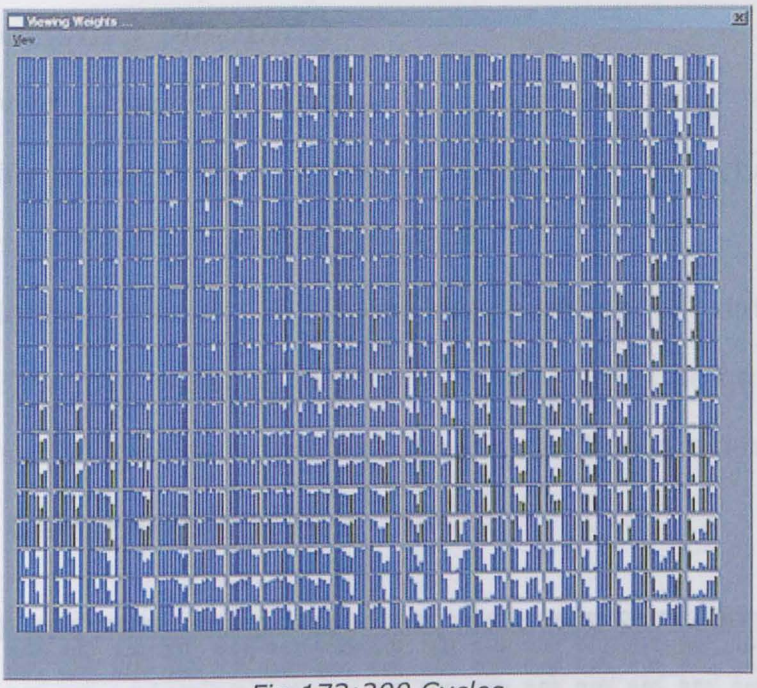



*Fig.173:200 Cycles*

## *18.3 - Network Mapping*

The following images illustrate tha data mapping of trained networks for a single data instance, illustrating the acheived data separations within the network structure.

The highlighted entries illustrate the number of data classes mapped within the network, with the actual values being the levels of confidence of class attachment of the input data presented.

### 18.3.1 - Centres Data Set

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| 1 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| 2 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| 3 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| 4 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| 5 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| 6 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| 7 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| 8 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| 9 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.18 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| 10 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| 11 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.83 | 0.00 | 0.00 | 0.00 | 0.00 |
| 12 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| 13 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| 14 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| 15 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| 16 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| 17 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| 18 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| 19 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |

*Fig.174:300 Cycles*

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| 1 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| 2 | 0.81 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| 3 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| 4 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| 5 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| 6 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| 7 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| 8 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| 9 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| 10 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| 11 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| 12 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| 13 | 0.00 | 0.00 | 0.19 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| 14 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| 15 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| 16 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| 17 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| 18 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| 19 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |

*Fig.175:400 Cycles*

## 18.3.2 - Combined Data Set

| | 0.00 | 1.00 | 2.00 | 3.00 | 4.00 | 5.00 | 6.00 | 7.00 | 8.00 | 9.00 | 10.00 | 11.00 | 12.00 | 13.00 | 14.00 | 15.00 | 16.00 | 17.00 | 18.00 | 19.00 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| 1.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| 2.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| 3.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| 4.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| 5.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| 6.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| 7.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| 8.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| 9.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| 10.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.42 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| 11.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.40 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| 12.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| 13.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| 14.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| 15.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| 16.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.19 |
| 17.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| 18.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| 19.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |

*Fig.176:300 Cycles*

| | 0.00 | 1.00 | 2.00 | 3.00 | 4.00 | 5.00 | 6.00 | 7.00 | 8.00 | 9.00 | 10.00 | 11.00 | 12.00 | 13.00 | 14.00 | 15.00 | 16.00 | 17.00 | 18.00 | 19.00 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| 1.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| 2.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| 3.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| 4.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| 5.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| 6.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| 7.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| 8.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| 9.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| 10.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| 11.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| 12.00 | 0.29 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| 13.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| 14.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| 15.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| 16.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.72 |
| 17.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| 18.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| 19.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |

*Fig.177:400 Cycles*

## 18.3.3 - Extremes Data Set

| | 0.00 | 1.00 | 2.00 | 3.00 | 4.00 | 5.00 | 6.00 | 7.00 | 8.00 | 9.00 | 10.00 | 11.00 | 12.00 | 13.00 | 14.00 | 15.00 | 16.00 | 17.00 | 18.00 | 19.00 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| 1.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| 2.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| 3.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| 4.00 | 0.64 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| 5.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| 6.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| 7.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| 8.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| 9.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| 10.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| 11.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| 12.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| 13.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.36 |
| 14.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| 15.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| 16.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| 17.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| 18.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| 19.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |

*Fig.178:200 Cycles*

| | 0.00 | 1.00 | 2.00 | 3.00 | 4.00 | 5.00 | 6.00 | 7.00 | 8.00 | 9.00 | 10.00 | 11.00 | 12.00 | 13.00 | 14.00 | 15.00 | 16.00 | 17.00 | 18.00 | 19.00 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.64 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| 1.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| 2.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| 3.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| 4.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| 5.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| 6.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| 7.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| 8.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| 9.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| 10.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| 11.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| 12.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| 13.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| 14.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| 15.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| 16.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| 17.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| 18.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| 19.00 | 0.00 | 0.00 | 0.36 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |

*Fig.179:400 Cycles*

# 19 - Appendix K – Final System Structure

Below is a listing of the final network weights for the selected architecture .

## *19.1 - SOM Layer*

SOMNET

Number of inputs: 8

SOM map size:20x20

Learning Rate Max (initial) value: 0.4

Learning rate Min value: 0.01

Minimum Neighborhood: 0

Number of training epochs: 200

**Weight Values:**

**Row 0**

0.742983 0.809236 0.829152 0.782873 0.755458 0.664619
0.663736 0.647426 0.80132 0.974913 0.956669 0.929929
0.964242 0.880896 0.952623 0.953795 0.977781 1
0.958066 0.956557

0.724673 0.908545 0.782919 0.897536 0.794244 0.660787
0.700337 0.767314 0.77855 0.949909 0.94846 0.91471
0.900149 0.95155 0.995684 0.942028 0.960366 0.987284
0.983374 0.980562

0.889565 0.998207 0.962176 0.96495 0.837316 0.674574
0.718395 0.885702 0.931431 0.917144 0.921331 0.916599
0.839779 0.992556 0.992069 0.948128 0.969409 0.993051
0.996107 0.953173

0.900831 0.992445 0.932285 0.913082 0.545246 0
0.0315898 0.119517 4.36351e-005 0 0.18445
0.539591 0.789764 0.933946 0.965259 0.981593 0.956445
0.919925 1 0.952105

0.994801 0.980288 0.658888 0.278495 0.537529 0.690643
0.743922 0.859326 0.826908 0.880834 0.90153 0.893127
0.818834 0.934781 0.896473 0.960657 0.925879 0.89282
0.954752 0.96648

0.970825 0.911677 0.790883 0.793988 0.770213 0.767493

0.817974 0.912238 0.913098 0.935301 0.523414 0.529922
0.523678 0.581866 0.479821 0.982723 0.953821 0.919925
0.924608 0.998196

0.921702 0.895987 0.962827 0.940526 0.896549 0.873726
0.902742 0.950376 0.999908 0.919067 0.910365 0.9926
0.97887 0.889695 0.614608 0.602774 0.734451 0.87037
0.956537 0.992053

0.106051 0.0485273 0.00119537 0.142474 0.285244
0.557698 0.696085 0.952879 0.937552 0.879641 0.944501
0.974052 0.999349 0.876683 0.99819 0.935636 0.951999
0.987747 0.954223 0.997037

**Row 1**

0.750437 0.887746 0.881645 0.908422 0.78605 0.687292
0.690106 0.666212 0.865606 0.961826 0.966529 0.921067
0.996645 0.965412 0.93334 0.956935 0.979016 0.987181
0.864925 0.955033

0.782133 0.904879 0.93295 0.904713 0.835354 0.693511
0.69897 0.707464 0.85963 0.906578 0.934849 0.957166
0.998837 0.930364 0.993605 0.957616 0.860325 0.863271
0.904361 0.991723

0.939337 0.998547 0.993582 0.952606 0.796585 0.663158
0.615224 0.706035 0.866096 0.86443 0.927909 0.871592
0.914966 0.887409 0.976479 0.958299 0.965697 0.978478
0.895336 0.999995

0.941099 0.99449 0.946083 0.948587 0.781705 0.0875249 0
0.00521966 0.01989 0.000330516
0.240472 0.502841 0.722048 0.801672 0.852412 0.925796
0.946277 0.978993 0.974297 0.981267

0.954595 0.957756 0.814906 0.605185 0.636634 0.711096
0.731199 0.683329 0.806941 0.809529 0.733137 0.844344
0.799582 0.688329 0.765881 0.861715 0.950649 0.856771
0.999748 0.893465

0.848206 0.830267 0.811311 0.654478 0.830384 0.761148
0.789892 0.977914 0.889823 0.723786 0.650995 0.721013
0.714181 0.66746 0.899684 0.930871 0.979097 1
0.990943 0.973971

0.513817 0.702729 0.93401 0.666631 0.928136 0.875331
0.896196 0.991348 0.975143 0.99911 0.825372 0.880629
0.935304 0.77922 0.727198 0.702119 0.884464 1
0.95408 0.974598

0.0741963 0.000974041 0.0279821 0.0010605 0.239649
0.549138 0.690106 0.976405 0.945818 0.998344 0.945171
0.978329 0.961312 0.895119 0.907349 0.943447 0.966558
0.999435 0.899106 0.858373

**Row 2**

0.762596 0.927543 0.969389 0.797226 0.708148 0.911609
0.73259 0.934628 0.995402 0.953144 0.880334 0.935961
0.991411 0.906262 0.981837 0.952773 0.961046 0.984027

0.962424 0.974955

0.749172 0.83878 0.917886 0.783656 0.709346 0.85384
0.850598 0.721957 0.941254 0.963123 0.914786 0.963349
0.957995 0.984149 0.95065 0.972832 0.884243 0.774856
0.858779 0.997071

0.809385 0.997081 0.859508 0.826783 0.692282 0.667434
0.451692 0.604479 0.777572 0.830949 0.940828 0.892933
0.934766 0.960482 0.911502 0.929838 0.915667 0.967862
0.973466 0.99024

0.813318 0.989903 0.903695 0.791094 0.662492 0.00647031
0.00381984 0.000102088 0
0.116407 0.336091 0.532513 0.816974 0.936834 0.908572
0.815083 0.90449 0.955936 0.965131 0.957709

0.888747 0.930137 0.90357 0.882643 0.870505 0.696368
0.73449 0.730369 0.585396 0.591631 0.398297 0.582641
0.423828 0.611788 0.627497 0.811839 0.904898 0.999027
0.96104 0.930185

0.436081 0.539579 0.414679 0.721289 0.42787 0.7679
0.820348 0.871185 1 0.846445 0.61192 0.754701
0.761317 0.808215 0.923221 0.914292 0.99469 0.995025
0.967738 0.857186

0.43437 0.794517 0.932254 0.954452 0.755078 0.884452
0.871385 0.982815 1 0.998852 0.969075 0.972817
0.951257 0.761699 0.655738 0.806989 0.992903 0.999592
0.976871 0.735893

0 0.000901167 0.0636823 0.00259277
0.156836 0.559278 0.841262 0.931335 0.99977 0.967013
0.959208 0.992285 0.941549 0.907715 0.966217 0.988593
0.961633 0.979034 0.871887 0.746737

**Row 3**

0.936043 0.917144 0.775097 0.896062 0.8523 0.95159
0.938766 0.961244 0.962163 0.973232 0.850589 0.983455
0.987681 0.822503 0.941779 0.949104 0.937129 0.916941
0.923402 0.926957

0.896688 0.722952 0.767003 0.869318 0.807564 0.945437
0.943172 0.918772 0.886221 0.982476 0.956062 0.987449
0.980724 0.963938 0.971214 0.968133 0.863523 0.807136
0.838157 0.939491

0.996546 0.96807 0.655012 0.5212 0.365503 0.151614
0.157207 0.322104 0.532745 0.712121 0.580598 0.921593
0.892203 0.947143 0.907019 0.73736 0.804107 0.930004
0.966166 0.965687

0.944134 0.956059 0.668492 0.339149 0.520208 0.0438943
0.0123401 0.1154 0.035495 0.0181068 0.376942 0.412081
0.692525 0.890311 0.938767 0.818411 0.862664 0.913867
0.957588 0.957904

0.911337 0.743754 0.86629 0.788997 0.726465 0.69534
0.684486 0.689374 0.532821 0.445397 0.28632 0.0375876

0.250167 0.401899 0.588094 0.732648 0.903797 0.964851
0.97785 0.943361

0.113866 0.0804849 0.153148 0.0118031 0.292478 0.657851
0.841155 0.973472 0.899777 0.643303 0.948457 0.71046
0.677637 0.733669 0.952454 0.925815 0.889075 0.971443
0.95113 0.923189

0.612847 0.903626 0.910581 0.936234 0.880691 0.696541
0.88371 0.934225 0.953877 0.943974 0.982064 0.880721
0.861056 0.874684 0.941452 0.931291 0.990643 0.966221
0.95861 0.96651

7.83605e-005 0.000309194 0.0113453 0.00698481
0.264946 0.684521 0.855251 0.958475 0.965073
0.89013 0.945294 0.965814 0.867205 0.688298 0.843198
0.997593 0.94632 0.95986 0.854999 0.491454

**Row 4**

0.94001 0.9362 0.915471 0.810865 0.868304 0.93947
0.849777 0.962329 0.896239 0.977943 0.966892 0.905855
0.640668 0.889587 0.949611 0.775443 0.77408 0.734897
0.635247 0.855632

0.926748 0.926746 0.883382 0.817906 0.791811 0.644785
0.782168 0.899226 0.719167 0.920715 0.909548 0.904888
0.690153 0.772776 0.761305 0.691168 0.830273 0.807714
0.998522 0.815318

0.95464 0.944917 0.656328 0.000343663 0.0061447
0.000596269 0 1.29811e-005 0.000774757
0.353227 0.774312 0.741052 0.672458 0.765212
0.712421 0.828769 0.770697 0.834546 0.991999 0.999618

0.895559 0.89832 0.713928 0.47311 0.637823 0.355064
6.57931e-006 0.0365681 0.000267175 0.188415
0.050488 0.54407 0.661473 0.793286 0.925438 0.760433
0.594292 0.820375 0.968977 0.991586

0.94395 0.868434 0.794409 0.514778 0.751324 0.742577
0.775642 0.598193 0.0395431 0.00394424 0.0189213
3.0861e-005 3.10741e-006 0.270457 0.665103
0.697997 0.834976 0.942599 0.953336 0.93706

0.0207014 0.00612785 0.115964 2.51167e-005
0.0086166 0.541436 0.816786 0.983327 0.860988 0.824862
0.759013 0.763861 0.695248 0.865802 0.909582 0.975498
0.959304 0.995928 0.906255 0.892546

0.946801 0.935191 0.916115 0.858008 0.813834 0.939416
0.939778 0.983954 0.80862 0.757004 0.837583 0.883922
0.748749 0.86581 0.941007 0.925893 0.918522 0.901786
0.789775 0.768259

7.25048e-006 0.152644 0.124157 0.000758961
0.414225 0.976561 0.970631 0.985986 0.93766 0.98993
0.855528 0.991845 0.890598 0.895756 0.947195 0.955675
0.973425 0.973985 0.846122 0.708113

**Row 5**

0.920271 0.963853 0.869767 0.80416 0.895576 0.896468
0.916267 0.884615 0.964809 0.79412 0.916187 0.913808
0.94947 0.785295 0.796679 0.956471 0.720685 0.538532
0.255639 0.25356

0.943 0.923755 0.887062 0.870434 0.906506 0.882687
0.842121 0.797635 0.671162 0.522883 0.53228 0.603386
0.546428 0.593926 0.617831 0.555305 0.706689 0.897203
0.777286 0.878024

0.969519 0.886316 0.61517 0.305864 0.0824473 0.0245414
0.0874239 0 1.39026e-005 0.246322 0.914259 0.791
0.854878 0.591824 0.580278 0.789804 0.89277
0.930053 0.881844 0.937137

0.893983 0.927108 0.750877 0.670799 0.813467 0.604303
0.271947 1.22935e-005 0.058464 0.00122278
6.76604e-005 0.506977 0.781016 0.847128 0.93295
0.544996 0.714595 0.843009 0.983015 0.963691

0.907558 0.895027 0.799817 0.744597 0.881623 0.666056
0.649927 0.325255 0.0698491 0.0748718 4.47103e-006 0
1.43339e-005 0.0803574 0.693456 0.892809
0.921806 0.903015 0.90928 0.959215

1.2686e-005 0.536559 0.25613 0.30322 0.103849
0.615869 0.668479 0.892702 0.769599 0.912269 0.867277
0.854471 0.955222 0.908125 0.866596 0.973787 0.834098
0.956087 0.473537 0.909072

0.905655 0.935682 0.803194 0.721393 0.842164 0.67337
0.793493 0.939968 0.40028 0.970253 0.851633 0.856275
0.928672 0.745063 0.964015 0.954264 0.95859 0.955454
0.800324 0.806666

0.429446 0.493484 0.434479 0.623745 0.865074 0.896656
0.968646 0.975599 0.924368 0.947822 0.966854 0.96126
0.978548 0.901406 0.749189 0.890215 0.89715 0.961181
0.920027 0.819492

**Row 6**

0.968724 0.926589 0.920745 0.748319 0.872137 0.937745
0.949199 0.895931 0.823202 0.859556 0.843906 0.983055
0.897716 0.877977 0.838177 0.639704 0.672801 0.464627
5.18936e-005 5.81189e-005

0.955136 0.942849 0.899061 0.873746 0.887125 0.879095
0.756424 0.739878 0.569419 0.441068 0.288064 0.183395
0.0395849 0.217029 0.23229 0.429839 0.658259 0.665874
0.753913 0.989731

0.987336 0.883578 0.713722 0.477035 0.328404 0.307262
0.0499345 0.067381 0.334608 0.564104 0.829491 0.937117
0.636543 0.771739 0.761813 0.809557 0.696812 0.784005
0.88874 0.980452

0.96804 0.809981 0.702584 0.702442 0.71094 0.689263
0.355838 0.0381338 0.0160283 0.0311371 0.0994819 0.461207
0.733546 0.80705 0.805553 0.70873 0.776131 0.886367

0.942853 0.952179

0.926577 0.900256 0.784578 0.784604 0.698456 0.591364
0.726775 0.4035 0.51783 0.408734 0.501091 0.052407
0.000897236 0.56011 0.828309 0.971142 0.885076
0.98386 0.924953 0.955991

1.32527e-005 0.000262547 0.47567 0.466763
0.327854 0.238561 0.0411949 0.468916 0.589366 0.813514
0.832661 0.980537 0.938848 0.930137 0.923386 0.920259
0.828247 0.975799 0.994026 0.812085

0.783514 0.774412 0.632084 0.634222 0.54765 0.430599
0.884423 0.808252 0.87792 0.862491 0.934371 0.861849
0.884777 0.854372 0.862937 0.932056 0.796068 0.954529
0.942919 0.50829

0.740051 0.74648 0.732805 0.735819 0.87286 0.943921
0.880361 0.923337 0.937403 0.946445 0.950855 0.965991
0.983732 0.916938 0.874753 0.663141 0.916561 0.720609
0.913978 0.73799

**Row 7**

0.934896 0.772798 0.429449 0.792228 0.934525 0.963256
0.955122 0.835641 0.777273 0.715023 0.940432 0.813865
0.93625 0.952204 0.868314 0.698523 0.343038 0.00185278
0.00189838 0.0197544

0.960976 0.560181 0.871686 0.897716 0.933388 0.92686
0.911367 0.693939 0.476472 0.355848 0.00179162
0.000489541 0.000504343 0.000391338 0
0.180192 0.783634 0.817607 0.862388 0.726869

0.997746 0.983818 0.82003 0.616595 0.701 0.627997
0.555034 0.0637632 0.525349 0.858786 0.842656 0.839199
0.82941 0.875617 0.606381 0.73358 0.558365 0.586716
0.712403 0.905318

0.962773 0.962958 0.870657 0.6901 0.631357 0.265385
0.238046 7.9607e-005 0.0198979 7.63551e-006
0.160551 0.530429 0.786263 0.789589 0.903414 0.80816
0.665758 0.935862 0.867435 0.924792

0.939236 0.896805 0.707899 0.822298 0.561894 0.777353
0.752672 0.763708 0.742744 0.77102 0.777632 0.750674
0.510347 0.979131 0.994034 0.977708 0.970363 0.916891
0.888488 0.946342

2.69919e-005 0.00196564 0.282488 0.288415
0.148298 0.0224374 0.0150122 0 0.407163 0.704171
0.879286 0.849201 0.872916 0.929036 0.992041 0.942923
0.871126 0.837033 0.806486 0.974843

0.439681 0.71425 0.746165 0.744766 0.599121 0.740941
0.737899 0.940221 0.959858 0.973926 0.990364 0.876386
0.948274 0.764059 0.976095 0.844279 0.900958 0.806334
0.814409 0.870328

0.75106 0.766779 0.945775 0.817836 0.921764 0.817569
0.83452 0.934284 0.909323 0.965915 0.965546 0.893773

0.94832 0.917114 0.916775 0.893293 0.889335 0.942447
0.699582 0.607758

**Row 8**

0.946215 0.978938 0.832793 0.991639 0.867847 0.961585
0.977196 0.890051 0.703006 0.866814 0.821325 0.909694
0.888902 0.690091 0.672797 0.557806 0.0139085 0.0751958
0.00157948 0.0609335

0.99738 0.991249 0.975838 0.950729 0.931191 0.949307
0.971909 0.759477 0.370335 0.216718 0.00482776
3.60326e-006 0.000334715 0.000125219
2.65568e-006 0.0188618 0.532981 0.85773 0.812709
0.720021

0.999676 0.974375 0.922548 0.845528 0.86706 0.923779
0.777796 0.370385 0.787218 0.661557 0.75382 0.917774
0.673642 0.836647 0.89345 0.587557 0.0739206 0.204087
0.671734 0.946278

0.950725 0.945776 0.804884 0.720854 0.399759 0.0259235
0.00425821 0.00786877 0.00334694
0.00956786 0.00807414 0.39586 0.659659
0.768838 0.83011 0.961706 0.726842 0.876951 0.996373
0.986427

0.892345 0.653702 0.639016 0.798426 0.947093 0.882441
0.524166 0.731105 0.851181 0.743814 0.982615 0.976021
0.909547 0.932893 0.971316 0.966458 0.869512 0.891631
0.972364 0.948118

0.0349388 0.00119253 0.249418 3.88476e-005 0
2.13233e-006 0.00663955 0.0155036
0.0982184 0.774033 0.961296 0.861323 0.890863 0.945585
0.939688 0.990519 0.980664 0.972892 0.866578 0.805439

0.632581 0.748348 0.814619 0.968199 0.820606 0.927333
0.704494 0.941333 0.929675 0.892704 0.930634 0.830005
0.850251 0.978138 0.718097 0.969733 0.898371 0.922256
0.79651 0.842946

0.9871 0.952976 0.97127 0.940064 0.783306 0.800625
0.751486 0.947706 0.651195 0.718478 0.891476 0.756433
0.538633 0.846183 0.928572 0.914134 0.890703 0.720344
0.407672 0.0618451

**Row 9**

0.967468 0.905302 0.9448 0.93259 0.915028 0.990616
0.955139 0.973371 0.776632 0.690106 0.716676 0.882177
0.961506 0.57381 0.000141461 0.0352523 9.69822e-006
0.00785306 0.010077 0.056103

0.947143 0.848969 0.923674 0.807603 0.96107 0.987952
0.95442 0.8109 0.457999 0 0 0.00117685
0.0830806 0.0178848 0.0309448 0.00813594
0.0305919 0.434348 0.731934 0.0878512

0.973854 0.939931 0.886055 0.848469 0.917048 0.967426
0.855021 0.854268 0.795137 0.778151 0.779921 0.885042

0.804487 0.796816 0.733938 0.428092 0.201719 0.031055
0.0168783 0.744761

0.909068 0.896725 0.952793 0.758286 0.499805 0.0855554
0.00305541 0.00421644 0.0234857 0
0.00189148 0.601018 0.744732 0.766709 0.883903
0.826794 0.87951 0.937928 0.97152 0.834237

0.598585 0.455275 0.571984 0.404345 0.478435 0.394564
0.862985 0.780684 0.772854 0.680865 0.967577 0.91998
0.996316 0.985138 0.970144 0.800389 0.2626 0.858293
0.991807 0.961862

0.388713 0.215879 0.0526075 0.00910445 0
0.0118976 0.225755 0.433482 0.524225 0.66111 0.944322
0.958377 0.938594 0.962381 0.980372 0.972624 0.977571
0.936273 0.935971 0.815616

0.52422 0.782289 0.856268 0.848038 0.830902 0.97702
0.796441 0.762292 0.665117 0.639378 0.9721 0.361648
0.891215 0.893026 0.936049 0.969781 0.966547 0.183462
0.646981 0.829296

0.941345 0.929577 0.931948 0.945963 0.697869 0.974831
0.905783 0.914391 0.803249 0.650515 0.606203 0.512145
5.22785e-005 0.607481 0.832448 0.972958 0.800775
0.801706 0.0115412 0.0168976

**Row 10**

0.976445 0.959139 0.826051 0.958976 0.981505 0.968443
0.905371 0.836092 0.754415 0.772224 0.74471 0.891268
0.553474 0.331905 0.000259981 0.102686 0.0505553
0.00139313 0.468687 0.516395

0.968301 0.96211 0.941119 0.945484 0.928991 0.95167
0.972828 0.897346 0.67692 0.43017 0.312164 0.0197363
0.0368704 0.0710305 0.000224959 0.0909371 0.00118344
0.00229014 0.0692255 0.0736205

0.798432 0.829173 0.96927 0.878332 0.76923 0.849157
0.898725 0.725153 0.811407 0.840628 0.839128 0.928905
0.824539 0.751806 0.660424 0.332352 0.00192269
0.000943107 0.253626 0.0521408

0.886351 0.907538 0.929716 0.862702 0.852048 0.566221
0.119878 0.000796413 0.279443 0.0273152 0.0571988
0.0720801 0.385041 0.540932 0.662133 0.735921 0.81402
0.739465 0.846243 0.523914

0.173538 0.165592 0.321266 0.179735 0 0.309481
0.775252 0.848718 0.84825 0.783694 0.835183 0.958987
0.959013 0.961184 0.954738 0.715838 0.71052 0.801516
0.838372 0.911644

0.386458 0.353476 0.310211 0.127217 0 0.122549
0.432541 0.53915 0.616014 0.786111 0.858366 0.985199
0.960014 0.952448 0.934206 0.949994 0.945811 0.879673 0.8777
0.77371

0.90611 0.894916 0.758183 0.873278 0.991135 0.873339

0.807646 0.555479 0.525168 0.551028 0.61843 0.736145
0.855311 0.924504 0.884201 0.954066 0.966949 0.754954
0.166897 0.876844

0.986235 0.985237 0.985301 0.959465 0.964709 0.930314
0.994009 0.996427 0.956984 0.683858 0.580324 0.0672747
0.413941 0.609311 0.822582 0.837121 0.989067 0.600298
0.0354399 0

**Row 11**

0.995164 0.965013 1 0.956812 0.992539 0.99089
0.932909 0.917131 0.91566 0.934057 0.689482 0.237669
0.121205 0.0769973 0.259167 0.0656044 0.572608 0.598617
0.847608 0.936119

0.974099 0.97378 0.999447 0.968094 0.990478 0.94513
0.951824 0.882155 0.867896 0.934404 0.772073 0.00701746
0.0233641 0.0757652 0.148894 0.0301173 0.500635
0.465386 0.65011 0.68786

0.73262 0.802035 0.981946 0.894382 0.969085 0.963331
0.858619 0.733928 0.771755 0.894791 0.962356 0.704298
0.790568 0.839445 0.544222 0.0598326 0.21474 0.201342
0.0379553 0.00128605

0.885669 0.912253 0.959068 0.910699 0.966154 0.704377
0.326586 0.0487002 0.0318139 0.117555 0.00772872 0
0.084521 0.300777 0.357334 0.500998 0.603475
0.716372 0.832392 0.914449

0 0.0287945 0.0117881 0.0207035 0.00119392
0.023168 0.377674 0.268194 0.407682 0.768544 0.957141
0.97912 0.926062 0.933873 0.942964 0.92745 0.963371
0.814446 0.804623 0.729314

0.422549 0.437417 0.358959 0.21152 0.0344505 0.272444
0.412719 0.642088 0.744651 0.876107 0.974954 0.982195
0.954637 0.946615 0.90939 0.925219 0.890452 0.834159
0.825734 0.85838

0.995613 0.909372 0.726857 0.813407 0.712401 0.789329
0.356218 0.214324 0.218123 0.490997 0.593902 0.952172
0.899503 0.889561 0.94898 0.927397 0.97057 0.77237
0.883107 0.839967

0.993386 0.960465 1 0.955167 0.993441 0.929622
0.735773 0.929026 0.873842 0.765396 0.520871 0.419353
0.758204 0.850354 0.743125 0.880358 0.710725 0.800535
0.467699 0.0288322

**Row 12**

0.877382 0.944729 0.871969 0.858784 0.888293 0.930249
0.945074 0.946807 0.934941 0.917312 0.713487 0.114622 0
0.0233486 0.409101 0.617772 0.77703 0.78152
0.988348 0.927768

0.927541 0.969377 0.990053 0.999141 0.992421 0.938641
0.866895 0.919135 0.898481 0.932146 0.73894 0.180027 0
0.010337 0.0451504 0.0952756 0.00337171

0.637535 0.986554 0.956988

0.530055 0.827678 0.980954 0.975733 0.984006 0.924092
0.696509 0.683802 0.83724 0.927711 0.890938 0.786929
0.76386 0.643963 0.352014 0.0981411 0.00140254
0.266495 0.00220791 0.0296609

0.878798 0.944727 0.937195 0.939143 0.891156 0.634407
0.212269 0 0.0671856 0.0171538 0.000887448 0
0 0.0276614 0.129109 0.17556 0.511559
0.698359 0.822356 0.887869

0 0 0.00025515 0.0305126 0.00276362
0.00193582 0.0188775 0.00842628
0.563247 0.751251 0.928466 0.90787 0.92134 0.815608
0.868125 0.880416 0.943675 0.790779 0.841013 0.723411

0.741481 0.683125 0.473397 0.367534 0.39187 0.459679
0.696554 0.70798 0.823706 0.899037 0.828448 0.896177
0.946252 0.858555 0.785411 0.822771 0.92991 0.78228
0.821295 0.976345

0.80382 0.804409 0.845964 0.599959 0.357434 0.0431185
0.0671634 0.0309985 0.0910972 0.223429 0.309639 0.723398
0.867938 0.983165 0.951335 0.955113 0.927237 0.725027
0.786817 0.958679

0.902946 0.949476 0.721885 0.99995 0.945404 0.900607
0.920108 0.923401 0.899666 0.818485 0.709708 0.760717
0.832715 0.945355 0.816639 0.748329 0.911309 0.858408
0.888322 0.744794

**Row 13**

0.957316 0.968903 0.96719 0.957517 0.939559 0.826075
0.978719 0.98724 0.876914 0.926871 0.711001 0.000282851
0 0.0572788 0.648029 0.734594 0.835027
0.915404 0.956015 0.87271

0.940832 0.989535 0.964375 0.984337 0.974453 0.827462
0.933349 0.573203 0.847942 0.94618 0.780336 0.00104383
0.00302046 0.0640642 0.00299649 0
0.533256 0.939409 0.965724 0.880481

0.643544 0.987113 0.945281 0.889242 0.991332 0.776163
0.652106 0.608922 0.8131 0.948572 0.902987 0.740994
0.768433 0.672594 0.0147762 0 0.244071 0.442526
0.190971 0

0.865279 0.913357 0.8052 0.908419 0.965909 0.712509
0.445071 0.0813209 0.0171415 0.115549 0.0723882 0
0.000489164 0.0103707 0.00280631 0
0.505181 0.836442 0.900864 0.869765

0 0.0182457 0.000167387 0.000453558
0.0292516 0.000310515 0.000541416 0.147492
0.551751 0.847829 0.837887 0.842962 0.869173 0.858485
0.811635 0.850183 0.814767 0.650815 0.923205 0.877173

0.926809 0.84451 0.657176 0.52605 0.496634 0.589762
0.769075 0.858004 0.484896 0.867386 0.837164 0.828282

0.664945 0.631605 0.495374 0.850342 0.826761 0.739179
0.970298 0.986629

0.938195 0.84685 0.658708 0.453347 0.000100434
0.000302321 0.000146685 0.000793905
0.00531168 0 0.134478 0.456983 0.906167
0.901555 0.889913 0.982208 0.885168 0.842611 0.989562
0.987168

0.85426 0.895866 0.666097 0.678337 0.914923 0.89498
0.936207 0.892106 0.982749 0.951431 0.902456 0.787438
0.905304 0.888801 0.936074 0.674322 0.871365 0.994939
0.932813 0.963585

**Row 14**

0.965027 0.983572 0.938241 0.728597 0.983279 0.920132
0.727599 0.923135 0.797013 0.581254 0.106642 0
0.162764 0.00391511 0.515898 0.645825 0.817918
0.990757 0.913433 0.898755

0.84774 0.887586 0.992442 0.995665 0.996323 0.876325
0.726116 0.931062 0.857537 0.739506 0.85899 0.387357
0.34383 0.288925 0.103429 0.126219 0.484869 0.966544
0.88627 0.788892

0.908021 0.833155 0.975374 0.95247 0.980748 0.850473
0.703843 0.896252 0.801941 0.869451 0.819628 0.790065
0.732967 0.669848 0.219927 0 0.212853 0.44744
0.137222 0

0.938411 0.972868 0.92637 0.97517 0.955085 0.819359
0.852377 0.629838 0.468292 0.319996 0.082734 0.502486
0.235651 0.0488863 0.00671232 0 0.381185
0.771129 0.77869 0.629785

8.77704e-005 0.000390185 0 0.000108833
0 0.000120388 0.229233 0.365501
0.778907 0.806336 0.638043 0.570383 0.489793 0.744506
0.620918 0.724869 0.694938 0.91409 0.82799 0.958058

0.978977 0.95552 0.86931 0.743279 0.917327 0.797366
0.677764 0.735687 0.800711 0.698513 0.623005 0.623239
0.390701 0.00207056 0.598819 0.799435 0.82686
0.932328 0.947999 0.996038

0.957097 0.810784 0.462483 0.22337 2.78439e-006 0
0 0 0.00187503 0.0344075
0.129258 0.189381 0.463588 0.866228 0.861708 0.84434
0.899145 0.97059 0.962954 0.99331

0.750777 0.831371 0.807775 0.805185 0.864266 0.955428
0.985852 0.799515 0.871138 0.85959 0.882248 0.236876
0.576892 0.888507 0.722867 0.758369 0.808843 0.998669
0.978906 0.984456

**Row 15**

0.835431 0.887506 0.962064 0.964107 0.844264 0.910624
0.959267 0.974712 0.532326 0 3.98018e-005 0
0.290986 0.680753 0.542466 0.627636 0.839291

0.824126 0.910485 0.99285

0.881862 0.986791 0.908995 0.968146 0.963047 0.936532
0.907793 0.99724 0.714459 0.680821 0.000549417
0.437198 0.522439 0.499999 0.285172 0.150515 0.533426
0.649493 0.843288 0.95778

0.8332 0.948003 0.792128 0.916705 0.942351 0.927266
0.949767 0.945981 0.858249 0.63939 0.841062 0.794981
0.703172 0.661093 0.25139 0 0.212773 0.284124
0.0390796 0.0607946

0.946269 0.962118 0.734252 0.951426 0.875238 0.913175
0.912673 0.794946 0.785304 0.639391 0.874486 0.659631
0.347918 5.35387e-005 0.0346699 0 0.518911
0.644613 0.741733 0.442923

0.0795829 0.00396779 3.4173e-005 0.000397895
6.11499e-006 0.191355 0.373156 0.561867
0.670589 0.671229 0.948736 0.368831 0.329831 0.000126657
0.117819 0 0.468751 0.521507 0.634426
0.566421

0.941168 0.965582 0.889418 0.957823 0.973398 0.830054
0.659598 0.55583 0.50929 0.738529 0.111786 0.627644
0.264334 0 0.471457 0.784101 0.832311 0.90393
0.888091 0.999365

0.741891 0.550742 0.438608 0.151228 2.62754e-006 0
0 0 0.0145153 4.139e-006 0.165362
1.8519e-005 0.31202 0.438203 0.6631 0.715682
0.872215 0.955515 0.920613 0.999591

0.953177 0.922396 0.986657 0.779927 0.933228 0.916026
0.905481 0.993686 0.828112 0.738562 0.624117 0
0.455309 0.84621 0.570218 0.389076 0.821665 0.995089
0.973529 0.999268

**Row 16**

0.428903 0.870806 0.934191 0.922072 0.956497 0.89685
0.913776 0.86754 0.703442 0.598604 0.482463 0.52649
0.588699 0.804128 0.720372 0.882054 0.776755 0.810266
0.795732 0.964775

0.984773 0.864709 0.912027 0.972461 0.998256 0.855615
0.985108 0.921669 0.955869 0.656768 0.555677 0.778633
0.621349 0.641259 0.232167 0.0235128 0.121855 0.167269
0.867787 0.903958

0.971971 0.916855 0.941486 0.950169 0.976908 0.908437
0.944647 0.946025 0.962403 0.86075 0.813871 0.64679
0.614503 0.499264 0.225784 0.0272932 0.0622384 0.00120888
0.000518342 0.101247

0.822277 0.93312 0.921782 0.913334 0.801084 0.92595
0.975446 0.915664 0.699248 0.641222 0.563697 0.359369
0.495186 0.585245 0.0193013 0.0512663 0.410562 0.81979
0.692237 0.855158

0.0143444 0.300771 0.300986 0.251476 0.322424 0.370916

0.522158 0.598236 0.776669 0.823604 0.805698 0.630974
0.130685 0.00503741 0.0246785 0.00409707
0.104769 0.414067 0.396338 0.266277

0.908499 0.832596 0.899034 0.901371 0.873159 0.862985
0.749047 0.471207 0.433889 0.13986 0.103978 0.00290895
0.137984 0.0455656 0.615922 0.843382 0.843511
0.817757 0.830996 0.930376

0.64804 0.547984 0.386992 0.174825 0.00149981 0
0 7.87842e-006 0.00631615
0.047024 0.0430437 0.00165895 0.149012 0.333235
0.463282 0.797354 0.789924 0.929837 0.895307 0.944828

0.915784 0.791486 0.839319 0.892344 0.965436 0.908678
0.896708 0.815426 0.78758 0.698409 0.617678 0.278211
0.437707 0.772114 0.624624 0.915496 0.734394 0.927091
0.999616 0.969648

**Row 17**

0.560032 0.774724 0.902236 0.893907 0.88462 0.837166
0.886884 0.901555 0.738047 0.734079 0.683226 0.829529
0.79929 0.831788 0.872921 0.883784 0.899671 0.86182
0.906956 0.957271

0.879413 0.85205 0.999898 0.997798 0.909566 0.989998
0.990118 0.916787 0.998849 0.807875 0.675893 0.823326
0.683698 0.530695 0.00253291 0.107541 0.000163838
0.342256 0.833892 0.945416

0.88163 0.976832 0.955085 0.947093 0.95195 0.946416
0.96069 0.91917 0.996764 0.879858 0.801147 0.872343
0.475005 0.442119 0.46893 0.165786 0.0766972 0.0603441
7.75411e-005 0.000816793

0.93094 0.89816 0.937143 0.900049 0.874377 0.945805
0.962391 0.930185 0.975713 0.644614 0.579728 0.0059929
0.674136 0.469869 0.00332156 0.465617 0.689244
0.681682 0.842049 0.602367

0.455661 0.443317 0.46773 0.469529 0.486743 0.593255
0.69553 0.875277 0.888752 0.813796 0.818306 0.751631 0
0.0258322 0.115638 0.0192837 0.000181241
0.162604 0.000562188 0.00395561

0.885852 0.680078 0.867391 0.771576 0.956681 0.931574 0.7367
0.568264 0.291156 0.132134 0 0.0496621 0
0.233759 0.721963 0.831849 0.914839 0.891938
0.96074 0.706293

0.557075 0.412259 0.377182 0.19526 3.37119e-005
1.65335e-006 0.0008443 2.17797e-006 0.00109715
0.0512152 0.00580124 8.58378e-005 0
0.0389195 0.000905185 0.394094 0.72367
0.774964 0.881857 0.841173

0.936956 0.873736 0.885695 0.827527 0.786342 0.941993
0.874075 0.849195 0.711203 0.678979 0.675802 0.829882
0.399225 0.56989 0.865231 0.925754 0.916156 0.946409

0.964538 0.94087

**Row 18**

0.829264 0.835922 0.84616 0.94825 0.921175 0.936111
0.685558 0.664221 0.555898 0.76815 0.880943 0.976588
0.939887 0.915694 0.922126 0.823478 0.873642 0.840451
0.86182 0.749311

0.94746 0.956619 0.990494 0.992811 0.990106 0.973097
0.91422 0.99941 0.991727 0.843294 0.887588 0.952292
0.535505 0.384693 0.252388 0.0793095 0.332349 0.517415
0.901822 0.926159

0.945165 0.943124 0.927992 0.989218 0.954801 0.98881
0.943355 0.995831 0.997601 0.910249 0.910008 0.701132
0.605427 0.445481 0.394916 0.00331333 0.266579
0.446079 9.13963e-005 0

0.96501 0.906072 0.926978 0.895214 0.7843 0.977389
0.975257 0.980062 0.967313 0.743171 0.684645 0.586925
0.731201 0.611049 0.619888 0.69497 0.739333 0.7749
0.938152 0.614431

0.58735 0.63726 0.66148 0.693353 0.787103 0.89756
0.940895 0.990553 0.86807 0.827462 0.826858 0.819528 0.596
0.403259 0.393175 0.000150132 0 0
0 0

0.753661 0.79603 0.811121 0.885913 0.79704 0.925209
0.895021 0.938934 0.389308 0.230519 0.0475383 0.000118924
0.553752 0.68968 0.858757 0.739162 0.854297
0.942204 0.944528 0.568847

0.618603 0.426553 0.289674 0.179949 1.36622e-005
4.13062e-006 0.0214942 0.000478044 0.331976
0.0791712 0.0773675 0.00154219 0.106947 0.0877716
0.0965148 0.00332852 0.295521 0.464947 0.498249
0.879594

0.750962 0.818506 0.856373 0.946854 0.920258 0.865503
0.819289 0.449776 0.470556 0.52397 0.773583 0.670728
0.880226 0.817688 0.937999 0.969894 0.929402 0.83911
0.963866 0.950218

**Row 19**

0.937104 0.797958 0.837057 0.76941 0.777791 0.931619
0.145596 0.178606 0.664011 0.901289 0.941531 0.992615
0.962738 0.959322 0.919561 0.791199 0.869857 0.901596
0.733528 0.577128

0.903537 0.903363 0.925444 0.940816 0.799013 0.944855
0.962061 0.940417 0.752603 0.924417 0.973991 0.990481
0.514923 0.31182 0.243571 0.000664393 0.564191
0.87411 0.797198 0.718178

0.988708 0.972282 0.884463 0.926532 0.759774 0.998597
0.925311 0.933846 0.948686 0.939898 0.98067 0.970405
0.62243 0.472948 0.433852 0.596479 0.502497 0.0594165
0.0451007 0

0.957325 0.986518 0.760391 0.637386 0.792052 0.999212
0.988192 0.958589 0.908788 0.956352 0.968571 0.765185
0.737725 0.719547 0.730921 0.718641 0.826801 0.689597
0.74663 0.753645

0.871294 0.985675 0.835355 0.777096 0.821619 0.944582
0.934859 0.930139 0.932142 0.744833 0.786226 0.874756
0.682502 0.593622 0.448535 0 0 0 0
0

0.643583 0.92713 0.914394 0.883278 0.866106 0.88004
0.910839 0.880648 0.608584 0.241225 0.0763791 0.0235763
0.628654 0.8932 0.873923 0.892901 0.889935 0.722969
0.718021 0.669542

0.465663 0.42956 0.340458 1.45699e-005 0.00036217
0.0490995 8.05294e-005 0.0570215 0.09342
0.131067 0.000282264 0.1254 0.122962 0.12967
0.101538 0.118305 0.109273 0.0894796 0.462799 0.661592

0.649126 0.696708 0.829593 0.840561 0.924965 0.576054
0.721598 0.116841 0.0594525 0.131157 0.679131 0.990157
0.956128 0.952822 0.94319 0.925347 0.898803 0.973106
0.804733 0.647698

## *19.2 - MLP Layer*

NumberOfInputs = 9;
NumberOfOutputs = 1;
NumberOfLayers = 4;
Layer sizes:

Input Layer: 9
Hidden Layer 1: 7 + Bias
Hidden Layer 2: 3 + Bias
Output Layer: 1

Transfer Function : Sigmoid
Training Method: Sequential
Initial Learning Rate: 0.2
Initial Momentum: 0.8
Triained for 97 epochs

**Weights per layer per node:**

Weights:
Layer 2

| | | |
|---|---|---|
| Node 1 | Bias is | -1.16695377441978 |
| Node 1 | Weight from 1 is | 0.287749382606513 |
| Node 1 | Weight from 2 is | 6.68779653502263 |
| Node 1 | Weight from 3 is | 2.12380318025296 |
| Node 1 | Weight from 4 is | -2.99195778125491 |
| Node 1 | Weight from 5 is | -2.79258647191585 |
| Node 1 | Weight from 6 is | -0.813940272990039 |
| Node 1 | Weight from 7 is | -0.9933645895901 |
| Node 1 | Weight from 8 is | -0.993318215932311 |
| Node 1 | Weight from 9 is | 0.833333455198896 |
| Node 2 | Bias is | 0.490869824141896 |
| Node 2 | Weight from 1 is | -0.400118053183008 |
| Node 2 | Weight from 2 is | -3.79257307516019 |
| Node 2 | Weight from 3 is | -0.92824726602038 |

| | | |
|---|---|---|
| Node 2 | Weight from 4 is | 1.38639761223755 |
| Node 2 | Weight from 5 is | 2.32751524737333 |
| Node 2 | Weight from 6 is | -0.518193799649914 |
| Node 2 | Weight from 7 is | -0.167100691183909 |
| Node 2 | Weight from 8 is | 0.283150766230358 |
| Node 2 | Weight from 9 is | -0.919194069687385 |
| Node 3 | Bias is | 0.462423248578962 |
| Node 3 | Weight from 1 is | -1.02146462143204 |
| Node 3 | Weight from 2 is | -4.85562040553977 |
| Node 3 | Weight from 3 is | -0.634953235901165 |
| Node 3 | Weight from 4 is | 1.81905614370662 |
| Node 3 | Weight from 5 is | 2.99874088792403 |
| Node 3 | Weight from 6 is | -0.514936487972874 |
| Node 3 | Weight from 7 is | 0.165035939614431 |
| Node 3 | Weight from 8 is | 0.734926667956098 |
| Node 3 | Weight from 9 is | -0.349716852733153 |
| Node 4 | Bias is | -0.251549161583715 |
| Node 4 | Weight from 1 is | -0.20429322142147 |
| Node 4 | Weight from 2 is | -1.8120072178301 |
| Node 4 | Weight from 3 is | -0.428959914282485 |
| Node 4 | Weight from 4 is | 0.437774619554934 |
| Node 4 | Weight from 5 is | 0.667978125317098 |
| Node 4 | Weight from 6 is | -0.608167141256914 |
| Node 4 | Weight from 7 is | -0.410622260324976 |
| Node 4 | Weight from 8 is | 0.421968397606879 |
| Node 4 | Weight from 9 is | -0.510107005826963 |
| Node 5 | Bias is | 0.260565806085845 |
| Node 5 | Weight from 1 is | 0.13882193385232 |
| Node 5 | Weight from 2 is | -1.99748635878574 |
| Node 5 | Weight from 3 is | -0.755544646206114 |
| Node 5 | Weight from 4 is | 0.0575742524314847 |
| Node 5 | Weight from 5 is | 1.2838943241021 |
| Node 5 | Weight from 6 is | -0.35320445650789 |
| Node 5 | Weight from 7 is | -0.19338942524826 |
| Node 5 | Weight from 8 is | -0.077286083531646 |
| Node 5 | Weight from 9 is | -0.457640242412167 |

Node 6 Bias is 0.672870926034741
Node 6 Weight from 1 is -0.620512114843159
Node 6 Weight from 2 is -3.93227024052153
Node 6 Weight from 3 is -0.43580820259583
Node 6 Weight from 4 is 1.1964255424759
Node 6 Weight from 5 is 2.14471444777331
Node 6 Weight from 6 is -0.486088569238042
Node 6 Weight from 7 is -0.35192936109598
Node 6 Weight from 8 is -0.160725731350153
Node 6 Weight from 9 is -0.271990436990789
Node 7 Bias is -0.855929725364306
Node 7 Weight from 1 is 0.913092592130773
Node 7 Weight from 2 is 5.22940063718065
Node 7 Weight from 3 is 0.978445705530449
Node 7 Weight from 4 is -2.73469503393611
Node 7 Weight from 5 is -3.1816814868644
Node 7 Weight from 6 is 0.274046120320427
Node 7 Weight from 7 is -0.54343677253497
Node 7 Weight from 8 is -0.873690620372073
Node 7 Weight from 9 is 1.15417676137488
Layer 3
Node 1 Bias is -0.37972958555123
Node 1 Weight from 1 is -0.48781192969308
Node 1 Weight from 2 is 0.3245452147136
Node 1 Weight from 3 is -0.202350774731255
Node 1 Weight from 4 is -0.123909787703083
Node 1 Weight from 5 is 0.119192312507584
Node 1 Weight from 6 is 0.278067920189425
Node 1 Weight from 7 is -1.0262302778878
Node 2 Bias is -0.502822254680326
Node 2 Weight from 1 is 4.4655146412411
Node 2 Weight from 2 is -2.46537370502533
Node 2 Weight from 3 is -3.08536094836032
Node 2 Weight from 4 is -1.47204977253768
Node 2 Weight from 5 is -1.43296850744446
Node 2 Weight from 6 is -2.40489109847024

| | | |
|---|---|---|
| Node 2 | Weight from 7 is | 3.41765232849167 |
| Node 3 | Bias is | -0.239005008952351 |
| Node 3 | Weight from 1 is | -3.58265830122547 |
| Node 3 | Weight from 2 is | 2.19699217640588 |
| Node 3 | Weight from 3 is | 2.94417315657475 |
| Node 3 | Weight from 4 is | 0.972994265465855 |
| Node 3 | Weight from 5 is | 1.19805491826407 |
| Node 3 | Weight from 6 is | 2.42635076476544 |
| Node 3 | Weight from 7 is | -2.75230672757084 |
| Layer 4 | | |
| Node 1 | Bias is | -1.49952950899378 |
| Node 1 | Weight from 1 is | 0.464671423180393 |
| Node 1 | Weight from 2 is | -8.71082800308455 |
| Node 1 | Weight from 3 is | 6.30597300806232 |

# 20 - Appendix L – Software Source Code

## 20.1 - Som Trainer

```
// Som TrainerDlg.cpp : implementation file
//

#include "stdafx.h"
#include "Som Trainer.h"
#include "Som TrainerDlg.h"
#include "DlgProxy.h"
#include "ViewWeightsDlg.h"
#include "SomHeader.h"
#include <iomanip.h>


#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

/////////////////////////////////////////////////////////////////////////////
// CAboutDlg dialog used for App About

class CAboutDlg : public CDialog
{
public:
  CAboutDlg();

// Dialog Data
  //{{AFX_DATA(CAboutDlg)
  enum { IDD = IDD_ABOUTBOX };
  //}}AFX_DATA
  // ClassWizard generated virtual function overrides
  //{{AFX_VIRTUAL(CAboutDlg)
  protected:
  virtual void DoDataExchange(CDataExchange* pDX);    // DDX/DDV support
  //}}AFX_VIRTUAL

// Implementation
protected:
  //{{AFX_MSG(CAboutDlg)
  //}}AFX_MSG
  DECLARE_MESSAGE_MAP()
};

CAboutDlg::CAboutDlg() : CDialog(CAboutDlg::IDD)
{
  //{{AFX_DATA_INIT(CAboutDlg)
  //}}AFX_DATA_INIT
}

void CAboutDlg::DoDataExchange(CDataExchange* pDX)
{
  CDialog::DoDataExchange(pDX);
  //{{AFX_DATA_MAP(CAboutDlg)
  //}}AFX_DATA_MAP
}

BEGIN_MESSAGE_MAP(CAboutDlg, CDialog)
  //{{AFX_MSG_MAP(CAboutDlg)
```

```
    // No message handlers
  //}}AFX_MSG_MAP
END_MESSAGE_MAP()

/////////////////////////////////////////////////////////////////////////////
// CSomTrainerDlg dialog

IMPLEMENT_DYNAMIC(CSomTrainerDlg, CDialog);

CSomTrainerDlg::CSomTrainerDlg(CWnd* pParent /*=NULL*/)
  : CDialog(CSomTrainerDlg::IDD, pParent)
{
  //{{AFX_DATA_INIT(CSomTrainerDlg)
  m_FileName = _T("");
  m_FileSize = 0;
  m_MapHeight = 0;
  m_MapWidth = 0;
  m_MaxTrain = 0;
  m_MinNeighbour = 0;
  m_NumInputs = 0;
  m_MaxLearn = 0.0;
  m_MinLearn = 0.0;
  m_SaveName = _T("");
  //}}AFX_DATA_INIT
  // Note that LoadIcon does not require a subsequent DestroyIcon in Win32
  m_hIcon = AfxGetApp()->LoadIcon(IDR_MAINFRAME);
  m_pAutoProxy = NULL;
}

//*****************************************************************
//   Declaring Instance of SOM

SOM SomNet;

int CheckAll = 0;


//*****************************************************************


CSomTrainerDlg::~CSomTrainerDlg()
{
  // If there is an automation proxy for this dialog, set
  //  its back pointer to this dialog to NULL, so it knows
  //  the dialog has been deleted.
  if (m_pAutoProxy != NULL)
    m_pAutoProxy->m_pDialog = NULL;
}

void CSomTrainerDlg::DoDataExchange(CDataExchange* pDX)
{
  CDialog::DoDataExchange(pDX);
  //{{AFX_DATA_MAP(CSomTrainerDlg)
  DDX_Text(pDX, IDC_FILE_EDIT, m_FileName);
  DDX_Text(pDX, IDC_FILESIZE_EDIT, m_FileSize);
  DDX_Text(pDX, IDC_MAPHEIGHT_EDIT, m_MapHeight);
  DDX_Text(pDX, IDC_MAPWIDTH_EDIT, m_MapWidth);
  DDX_Text(pDX, IDC_MAXTRAIN_EDIT, m_MaxTrain);
  DDX_Text(pDX, IDC_MINNEIGH_EDIT, m_MinNeighbour);
  DDX_Text(pDX, IDC_NUMINPUTS_EDIT, m_NumInputs);
  DDX_Text(pDX, IDC_MAXLEARN_EDIT, m_MaxLearn);
  DDX_Text(pDX, IDC_MINLEARN_EDIT, m_MinLearn);
  DDX_Text(pDX, IDC_SAVE_EDIT, m_SaveName);
  //}}AFX_DATA_MAP
}

BEGIN_MESSAGE_MAP(CSomTrainerDlg, CDialog)
  //{{AFX_MSG_MAP(CSomTrainerDlg)
```

```
  ON_WM_SYSCOMMAND()
  ON_WM_DESTROY()
  ON_WM_PAINT()
  ON_WM_QUERYDRAGICON()
  ON_WM_CLOSE()
  ON_BN_CLICKED(IDC_FILESELECT_BUTTON, OnFileselectButton)
  ON_BN_CLICKED(IDC_TRAIN_BUTTON, OnTrainButton)
  ON_BN_CLICKED(IDC_CHECK_BUTTON, OnCheckButton)
  ON_COMMAND(ID_FILE_EXIT, OnFileExit)
  ON_COMMAND(ID_FILE_LOADNETWORK, OnFileLoadnetwork)
  ON_COMMAND(ID_FILE_SAVENATWORK, OnFileSavenatwork)
  ON_COMMAND(ID_NETWOK_SAVEWEIGHTS, OnNetwokSaveweights)
  ON_COMMAND(ID_NETWOK_VIEWWEIGHTS, OnNetwokViewweights)
  ON_COMMAND(ID_TEST_SAVERESULTS, OnTestSaveresults)
  ON_COMMAND(ID_TEST_TESTNETWORK, OnTestTestnetwork)
  ON_COMMAND(ID_TEST_VIEWRESULTS, OnTestViewresults)
  ON_EN_CHANGE(IDC_FILE_EDIT, OnChangeFileEdit)
  ON_EN_CHANGE(IDC_FILESIZE_EDIT, OnChangeFilesizeEdit)
  ON_EN_CHANGE(IDC_MAXLEARN_EDIT, OnChangeMaxlearnEdit)
  ON_EN_CHANGE(IDC_MAXTRAIN_EDIT, OnChangeMaxtrainEdit)
  ON_EN_CHANGE(IDC_MINLEARN_EDIT, OnChangeMinlearnEdit)
  ON_EN_CHANGE(IDC_MINNEIGH_EDIT, OnChangeMinneighEdit)
  ON_BN_CLICKED(IDC_TESTSET_BUTTON, OnTestsetButton)
  ON_BN_CLICKED(IDC_TESTSAVE_BUTTON, OnTestsaveButton)
  //}}AFX_MSG_MAP
END_MESSAGE_MAP()

/////////////////////////////////////////////////////////////////////////////
// CSomTrainerDlg message handlers

BOOL CSomTrainerDlg::OnInitDialog()
{
  CDialog::OnInitDialog();
  // Add "About..." menu item to system menu.
  // IDM_ABOUTBOX must be in the system command range.
  ASSERT((IDM_ABOUTBOX & 0xFFF0) == IDM_ABOUTBOX);
  ASSERT(IDM_ABOUTBOX < 0xF000);

  CMenu* pSysMenu = GetSystemMenu(FALSE);
  if (pSysMenu != NULL)
  {
    CString strAboutMenu;
    strAboutMenu.LoadString(IDS_ABOUTBOX);
    if (!strAboutMenu.IsEmpty())
    {
      pSysMenu->AppendMenu(MF_SEPARATOR);
      pSysMenu->AppendMenu(MF_STRING, IDM_ABOUTBOX, strAboutMenu);
    }
  }

  // Set the icon for this dialog.  The framework does this automatically
  //  when the application's main window is not a dialog
  SetIcon(m_hIcon, TRUE);                                    // Set big icon
  SetIcon(m_hIcon, FALSE);                    // Set small icon

  // TODO: Add extra initialization here
  m_MapHeight = 10;
  m_MapWidth = 10;
  m_NumInputs = SOM_NUMBER_INPUTS;
  m_MaxLearn = 0.4;
  m_MinLearn = 0.01;
  m_MinNeighbour = 0;
  m_MaxTrain = 4000;

  UpdateData(FALSE);
  return TRUE;  // return TRUE  unless you set the focus to a control
}
```

```
void CSomTrainerDlg::OnSysCommand(UINT nID, LPARAM lParam)
{
  if ((nID & 0xFFF0) == IDM_ABOUTBOX)
  {
    CAboutDlg dlgAbout;
    dlgAbout.DoModal();
  }
  else
  {
    CDialog::OnSysCommand(nID, lParam);
  }
}

void CSomTrainerDlg::OnDestroy()
{
  WinHelp(0L, HELP_QUIT);
  CDialog::OnDestroy();
}

// If you add a minimize button to your dialog, you will need the code below
//  to draw the icon.  For MFC applications using the document/view model,
//  this is automatically done for you by the framework.

void CSomTrainerDlg::OnPaint()
{
  if (IsIconic())
  {
    CPaintDC dc(this); // device context for painting
    SendMessage(WM_ICONERASEBKGND, (WPARAM) dc.GetSafeHdc(), 0);

    // Center icon in client rectangle
    int cxIcon = GetSystemMetrics(SM_CXICON);
    int cyIcon = GetSystemMetrics(SM_CYICON);
    CRect rect;
    GetClientRect(&rect);
    int x = (rect.Width() - cxIcon + 1) / 2;
    int y = (rect.Height() - cyIcon + 1) / 2;

    // Draw the icon
    dc.DrawIcon(x, y, m_hIcon);
    }
    else
    {
      CDialog::OnPaint();
    }
}

// The system calls this to obtain the cursor to display while the user drags
//  the minimized window.
HCURSOR CSomTrainerDlg::OnQueryDragIcon()
{
  return (HCURSOR) m_hIcon;
}

// Automation servers should not exit when a user closes the UI
//  if a controller still holds on to one of its objects.  These
//  message handlers make sure that if the proxy is still in use,
//  then the UI is hidden but the dialog remains around if it
//  is dismissed.

void CSomTrainerDlg::OnClose()
{
   if (CanExit())
     CDialog::OnClose();
}

void CSomTrainerDlg::OnOK()
{
```

```
  if (CanExit())
    CDialog::OnOK();
}

void CSomTrainerDlg::OnCancel()
{
  if (CanExit())
    CDialog::OnCancel();
}

BOOL CSomTrainerDlg::CanExit()
{
  // If the proxy object is still around, then the automation
  //  controller is still holding on to this application.  Leave
  //  the dialog around, but hide its UI.
  if (m_pAutoProxy != NULL)
  {
    ShowWindow(SW_HIDE);
    return FALSE;
  }

  return TRUE;
}

void CSomTrainerDlg::OnFileselectButton()
{
  // TODO: Add your control notification handler code here
  CFileDialog dlg(TRUE,"*.txt","*.txt",NULL);
  dlg.DoModal();
  m_FileName = dlg.GetPathName();
  m_TestFile = m_FileName;
  UpdateData(FALSE);

  if(!SomNet.SetFileName(m_FileName))
  {
    MessageBox("File doesn't exist, or is being used. Please reselect","File Selection
Error",MB_ICONERROR);
    m_FileName = "";
    UpdateData(FALSE);
  }
}


void CSomTrainerDlg::OnQuitButton()
{
  // TODO: Add your control notification handler code here
  OnOK();
}


void CSomTrainerDlg::OnTrainButton()
{
  // TODO: Add your control notification handler code here
  if(CheckAll!=1)
  {
    MessageBox("Please initialise all parameters","Initialisation Error",MB_ICONWARNING);
    return;
  }
  BeginWaitCursor();
  CProgressCtrl* ProgControl;
  ProgControl=  (CProgressCtrl*) GetDlgItem(IDC_TRAIN_PROGRESS);
  ProgControl->SetRange(0,m_MaxTrain);
  for(int loop = 0; loop< m_MaxTrain; loop++)
  {
    if(!SomNet.RunNet())
    {
      loop = m_MaxTrain;
      EndWaitCursor();
```

```
      MessageBox("Network training Failed","Network Error",MB_ICONERROR);
      return;
    }
    SomNet.IncCycle();
    ProgControl->SetPos(loop);
  }
  EndWaitCursor();
  MessageBox("Training Completed","Network Training",MB_ICONINFORMATION);
  ProgControl->SetPos(0);
}

void CSomTrainerDlg::OnCheckButton()
{
  // TODO: Add your control notification handler code here
  UpdateData(TRUE);
  SomNet.InitCurCycle();
  if(m_FileName=="")
  {
  MessageBox("You must select a training file","Initialisation Error", MB_ICONERROR);
    return;
  }
  if(!SomNet.SetDataLines(m_FileSize))
  {
    MessageBox("Training File Size must be a positive value","Initialisation Error",MB_ICONERROR);
    //m_FileSize= 0;
    //UpdateData(FALSE);
    return;
  }
  if(!SomNet.CheckFileSize())
  {
    MessageBox("Training file size does not match actual file","Initialisation Error",MB_ICONERROR);
    //m_FileSize = 0;
    //UpdateData(FALSE);
    return;
  }
  SomNet.SetNumInputs();
  SomNet.SetMapSize();
  if(!SomNet.SetRates(m_MinLearn, m_MaxLearn))
  {
    MessageBox("Learning Rates must be positive,\nMax Learning rate must be larger value","Initialisation
Error",MB_ICONERROR);
    m_MaxLearn = 0.4;
    m_MinLearn = 0.01;
    UpdateData(FALSE);
    return;
  }
  if(!SomNet.SetMinNeighbour(m_MinNeighbour))
  {
    MessageBox("Min Neighbourhood must be positive","Initialisation Error",MB_ICONERROR);
    m_MinNeighbour = 0;
    UpdateData(FALSE);
    return;
  }
  if(!SomNet.SetMaxCycles(m_MaxTrain))
  {
    MessageBox("MaxTraining Cycles must be a positive Integer","Initialisation Error",MB_ICONERROR);
    m_MaxTrain = 2000;
    UpdateData(FALSE);
    return;
  }
  SomNet.CalcMaxNeighbour();
  SomNet.CalcCurNeighbour();
  SomNet.CalcCurRate();
  SomNet.RandomWeights();
  SomNet.InitFilePos();

 MessageBox("All parameters initialised, ready for training","Network Ready",MB_ICONINFORMATION);
  CheckAll = 1;
```

```
}

void CSomTrainerDlg::OnFileExit()
{
  // TODO: Add your command handler code here
  OnOK();
}

void CSomTrainerDlg::OnFileLoadnetwork()
{
  // TODO: Add your command handler code here
  CString LoadFileName;
  CFileDialog dlg(TRUE,"vsf","*.vsf",OFN_OVERWRITEPROMPT,"Vios SOM File *.vsf||Text File *.txt",NULL);
  dlg.DoModal();
  LoadFileName = dlg.GetPathName();
  m_NetFile = LoadFileName;
  ifstream LoadFile(LoadFileName,ios::nocreate);

  char *Header = "SOMNET";
  char *Temp="";
  LoadFile>>Temp;
  if (strcmp(Header,Temp)!=0)
  {
    MessageBox("File Header Mismatch");
    return;
  }
  LoadFile >> m_NumInputs;
  LoadFile >> m_MapWidth;
  LoadFile >> m_MapHeight;
  LoadFile >> m_MaxLearn;
  LoadFile >> m_MinLearn;
  LoadFile >> m_MinNeighbour;
  LoadFile >> m_MaxTrain;

  char *Temp1="";
  int Temp2;
  double Temp3;
  int loop1, loop2, loop3;

  for (loop2 = 0; loop2<m_MapHeight; loop2++)
  {
    LoadFile >> Temp1;
    LoadFile >> Temp2;

    for (loop3 = 0; loop3<m_NumInputs; loop3++)
    {
      for ( loop1 = 0; loop1<m_MapWidth; loop1++)
      {
        LoadFile >>Temp3;
        SomNet.SetWeight(loop1, loop2, loop3, Temp3);
      }
    }
  }
  LoadFile.close();

  SomNet.SetMapSize();
  SomNet.SetNumInputs();
  SomNet.SetMaxCycles(m_MaxTrain);
  SomNet.SetRates(m_MinLearn, m_MaxLearn);
  SomNet.SetMinNeighbour(m_MinNeighbour);
  MessageBox("Network Loaded");
  UpdateData(FALSE);
}

void CSomTrainerDlg::OnFileSavenatwork()
{
  // TODO: Add your command handler code here
   int loop1, loop2, loop3;
```

```
  CString SaveFileName;

  CFileDialog dlg(FALSE,"vsf","*.vsf",OFN_OVERWRITEPROMPT,"Vios SOM File *.vsf||Text File *.txt",NULL);
  dlg.DoModal();
  SaveFileName = dlg.GetPathName();
  ofstream SaveFile(SaveFileName);
  SaveFile << "SOMNET\n";
  SaveFile << m_NumInputs << endl;
  SaveFile << m_MapWidth << "\t" << m_MapHeight << endl;
  SaveFile << m_MaxLearn << "\t" << m_MinLearn << endl;
  SaveFile << m_MinNeighbour << endl;
  SaveFile << m_MaxTrain << endl;
  for (loop2 = 0; loop2<m_MapHeight; loop2++)
  {
    SaveFile << "Row " << loop2<<endl;
    for (loop3 = 0; loop3<m_NumInputs; loop3++)
    {
      for ( loop1 = 0; loop1<m_MapWidth; loop1++)
      {
        SaveFile << SomNet.GiveWeight(loop1,loop2,loop3) << "\t";
      }
      SaveFile << endl;
    }
  }
  SaveFile.close();

}

void CSomTrainerDlg::OnNetwokSaveweights()
{
  // TODO: Add your command handler code here
  int loop1, loop2, loop3;

  CFileDialog dlg(FALSE,"vwf","*.vwf",OFN_OVERWRITEPROMPT,"Vios Weights File *.vwf||Text File
*.txt",NULL);
  dlg.DoModal();
  CString SaveFileName;
  SaveFileName = dlg.GetPathName();
  ofstream SaveFile(SaveFileName);

  for (loop2 = 0; loop2<m_MapHeight; loop2++)
  {
    SaveFile << "Row " << loop2<<endl;
    for (loop3 = 0; loop3<m_NumInputs; loop3++)
    {
      for ( loop1 = 0; loop1<m_MapWidth; loop1++)
      {
        SaveFile << SomNet.GiveWeight(loop1,loop2,loop3) << "\t";
      }
      SaveFile << endl;
    }
  }
  SaveFile.close();
}


void CSomTrainerDlg::OnNetwokViewweights()
{
  // TODO: Add your command handler code here
  ViewWeightsDlg m_dlg;
  int loop1, loop2, loop3;

  m_dlg.MaxCycles = m_MaxTrain;
  m_dlg.MaxLines = m_FileSize;

  for(loop1 = 0; loop1<10; loop1++)
  {
```

```
    for(loop2 = 0; loop2<10; loop2++)
    {
      for (loop3 = 0; loop3<5; loop3++)
      {
        m_dlg.WeightVals[loop2][loop1][loop3] = SomNet.GiveWeight(loop2, loop1, loop3);
      }
      m_dlg.FireRates[loop2][loop1] = SomNet.GiveFire(loop2, loop1);
    }
  }
  for(loop1= 0; loop1<5; loop1++)
  {
    m_dlg.Inputs[loop1] = SomNet.GiveData(loop1);
  }
  m_dlg.Test = 0;
  m_dlg.DoModal();
}

void CSomTrainerDlg::OnTestSaveresults()
{
  // TODO: Add your command handler code here

}

void CSomTrainerDlg::OnTestTestnetwork()
{
  // TODO: Add your command handler code here
  ViewWeightsDlg m_dlg;
  int loop1, loop2, loop3;

  if(m_FileName=="")
  {
    MessageBox("You must select a training file","Initialisation Error", MB_ICONERROR);
    return;
  }
     SomNet.SetDataLines(1);
    m_FileSize = 1;
    UpdateData(FALSE);
    SomNet.InitFilePos();
    SomNet.ReadDataLine();
//  MessageBox(m_FileName);
    char Text[500];
    sprintf(Text,"Values are : %f - %f - %f - %f - %f",SomNet.GiveData(0),SomNet.GiveData(1),
SomNet.GiveData(2),SomNet.GiveData(3),SomNet.GiveData(4) );
  MessageBox(Text);

  SomNet.CalcWinner();
  m_dlg.Test = 1;
  m_dlg.WinX = SomNet.GiveWinnerX();
  m_dlg.WinY = SomNet.GiveWinnerY();
  m_dlg.MaxCycles = m_MaxTrain;
  m_dlg.MaxLines = m_FileSize;
  double Max = 0;
  double Min = 100;
  for(loop1 = 0; loop1<10; loop1++)
  {
    for(loop2 = 0; loop2<10; loop2++)
    {
      for (loop3 = 0; loop3<5; loop3++)
      {
        m_dlg.WeightVals[loop2][loop1][loop3] = SomNet.GiveWeight(loop2, loop1, loop3);
        if(m_dlg.WeightVals[loop2][loop1][loop3]>Max) Max = m_dlg.WeightVals[loop2][loop1][loop3];
        if(m_dlg.WeightVals[loop2][loop1][loop3]<Min) Min = m_dlg.WeightVals[loop2][loop1][loop3];
      }
      m_dlg.FireRates[loop2][loop1] = SomNet.GiveFire(loop2, loop1);
    }
  }
  m_dlg.MaxWeight = Max;
  m_dlg.MinWeight = Min;
```

```
  Max = 0;
  Min = 100;
  for(loop1= 0; loop1<5; loop1++)
  {
    m_dlg.Inputs[loop1] = SomNet.GiveData(loop1);
    if(m_dlg.Inputs[loop1]>Max) Max = m_dlg.Inputs[loop1];
    if(m_dlg.Inputs[loop1]<Min) Min = m_dlg.Inputs[loop1];
  }
  m_dlg.MaxIn = Max;
  m_dlg.MinIn = Min;

  m_dlg.Test = 1;
  m_dlg.DoModal();
  m_dlg.Test = 0;
  m_FileName = " ";
  UpdateData(FALSE);
}

void CSomTrainerDlg::OnTestViewresults()
{
  // TODO: Add your command handler code here

}

void CSomTrainerDlg::OnChangeFileEdit()
{
  CheckAll = 0;
}

void CSomTrainerDlg::OnChangeFilesizeEdit()
{
  CheckAll = 0;
}

void CSomTrainerDlg::OnChangeMaxlearnEdit()
{
  CheckAll = 0;
}

void CSomTrainerDlg::OnChangeMaxtrainEdit()
{
  CheckAll = 0;
}

void CSomTrainerDlg::OnChangeMinlearnEdit()
{
   CheckAll = 0;
}

void CSomTrainerDlg::OnChangeMinneighEdit()
{
  CheckAll = 0;
}

void CSomTrainerDlg::OnSavetestButton()
{
   // TODO: Add your control notification handler code here

}

void CSomTrainerDlg::OnTestsetButton()
{
  // TODO: Add your control notification handler code here
  int loop,loop1;
  BeginWaitCursor();
  UpdateData(TRUE);
```

```
  char store;
  int count=1;
  ifstream TestFile(m_FileName,ios::nocreate);
  while(TestFile)
  {
    TestFile.get(store);
    if(store=='\n')
      count++;
  }
  TestFile.close();
  m_FileSize = count;

  SomNet.SetDataLines(m_FileSize);
  UpdateData(FALSE);
  SomNet.InitFilePos();

  ofstream OutFile(m_SaveName);
  OutFile << "Results from "<< m_NetFile << endl;
  OutFile << "Tested using "<< m_TestFile << endl << endl;
  OutFile << "WinX\tWinY\tVectorial Distance\tBoundary Size\tMetric\n\n";
  CProgressCtrl* ProgControl;
  ProgControl=  (CProgressCtrl*) GetDlgItem(IDC_TRAIN_PROGRESS);

  ProgControl->SetRange(0,m_FileSize);
  ProgControl->SetPos(0);
  for(loop = 0; loop<m_FileSize; loop++)
  {
    SomNet.ReadDataLine();

//    char Text[500];
//    sprintf(Text,"Values are : %f - %f - %f - %f - %f",SomNet.GiveData(0),SomNet.GiveData(1),
SomNet.GiveData(2),SomNet.GiveData(3),SomNet.GiveData(4) );
//    MessageBox(Text);
    SomNet.CalcWinner();
    double Temp = 0;
    double Distance = 0;
    //calc distance from input to winning node
    double InputsSum = 0;
    for(loop1= 0; loop1<SOM_NUMBER_INPUTS; loop1++)
    {
  Temp = SomNet.GiveWeight(SomNet.GiveWinnerX(),SomNet.GiveWinnerY(),loop1) - SomNet.GiveData
(loop1);
      Distance += sqrt(pow(Temp,2));
      Temp = 0;
    }
    for(loop1 = 0; loop1<SOM_NUMBER_INPUTS; loop1++)
      InputsSum += SomNet.GiveData(loop1);
    //calc wining node's boundary distance
    double Boundary = 0;
    double d[8];
    for(int loop=0; loop<8; loop++)
      d[loop] = 0;
    int Tot = 8;
    if(SomNet.GiveWinnerY()!=0)
    {
  if(SomNet.GiveWinnerX()!=0) d[0] = CalcDistance(SomNet.GiveWinnerX()-1, SomNet.GiveWinnerY()-1);
      else Tot--;
      d[1] = CalcDistance(SomNet.GiveWinnerX(), SomNet.GiveWinnerY()-1);
  if(SomNet.GiveWinnerX()!=9) d[2] = CalcDistance(SomNet.GiveWinnerX()+1, SomNet.GiveWinnerY()-1);
      else Tot--;
    }
    else Tot -= 3;
 if(SomNet.GiveWinnerX()!=0) d[3] = CalcDistance(SomNet.GiveWinnerX()-1, SomNet.GiveWinnerY());
    else Tot--;
  if(SomNet.GiveWinnerX()!=9) d[4] = CalcDistance(SomNet.GiveWinnerX()+1, SomNet.GiveWinnerY());

    else Tot--;
    if(SomNet.GiveWinnerY()!=9)
```

```
    {
  if(SomNet.GiveWinnerX()!=0) d[0] = CalcDistance(SomNet.GiveWinnerX()-1, SomNet.GiveWinnerY()+1);
       else Tot--;
       d[1] = CalcDistance(SomNet.GiveWinnerX(), SomNet.GiveWinnerY()+1);
 if(SomNet.GiveWinnerX()!=9) d[2] = CalcDistance(SomNet.GiveWinnerX()+1, SomNet.GiveWinnerY()+1);
       else Tot--;
    }
    else Tot -=3;
    //find average distance, and take half !
    double Avge = 0;
    //for(loop = 0; loop <8; loop++)
     //Avge += d[loop];
     //Avge  = 0.5*(double)(Avge/Tot);
    Avge = 0;
    for(loop = 0; loop<8; loop++)
    {
      if(d[loop]>Avge) Avge = d[loop];
    }

    //Calc Metric
    double Metric = InputsSum/Avge;


    //now save the data
 OutFile << SomNet.GiveWinnerX() <<"\t"<<SomNet.GiveWinnerY()
<<"\t"<<Distance<<"\t"<<Avge<<"\t"<<Metric<<endl;
    ProgControl->SetPos(loop);
  }
  OutFile.close();
  EndWaitCursor();
  MessageBox("Testing Completed","SOM Trainer",MB_OK);
}


void CSomTrainerDlg::OnTestsaveButton()
{
  // TODO: Add your control notification handler code here
  CFileDialog Saver(FALSE,"txt","*.txt",OFN_OVERWRITEPROMPT,"Results File|*.txt||",NULL);
  if(Saver.DoModal()==IDOK)
  {
    m_SaveName = Saver.GetPathName();
    UpdateData(FALSE);
  }
}


double CSomTrainerDlg::CalcDistance(int X, int Y)
{
  double Vals[5];
  double Weight1, Weight2;
  for(int loop = 0; loop<5; loop++)
  {
    Weight1 = SomNet.GiveWeight(SomNet.GiveWinnerX(),SomNet.GiveWinnerY(),loop);
    Weight2 = SomNet.GiveWeight(X,Y,loop);
    Vals[loop] = sqrt(pow(Weight2-Weight1,2));
  }
  double retval = 0;
  for(loop = 0; loop <5; loop++)
    retval += Vals[loop];
  return(retval);
}


// Som TrainerDlg.h : header file
```

```
//


#if !defined(AFX_SOMTRAINERDLG_H__952466AB_ACC2_11D3_B160_8BFB919D1E24__INCLUDED_)
#define AFX_SOMTRAINERDLG_H__952466AB_ACC2_11D3_B160_8BFB919D1E24__INCLUDED_

#if _MSC_VER >= 1000
#pragma once
#endif // _MSC_VER >= 1000


class CSomTrainerDlgAutoProxy;

/////////////////////////////////////////////////////////////////////////////
// CSomTrainerDlg dialog

class CSomTrainerDlg : public CDialog
{
  DECLARE_DYNAMIC(CSomTrainerDlg);
  friend class CSomTrainerDlgAutoProxy;

// Construction
public:
  double CalcDistance(int X,int Y);
  CString m_TestFile;
  CString m_NetFile;
  CSomTrainerDlg(CWnd* pParent = NULL);       // standard constructor
  virtual ~CSomTrainerDlg();

// Dialog Data
  //{{AFX_DATA(CSomTrainerDlg)
  enum { IDD = IDD_SOMTRAINER_DIALOG };
  CString m_FileName;
  int   m_FileSize;
  int   m_MapHeight;
  int   m_MapWidth;
  int   m_MaxTrain;
  int   m_MinNeighbour;
  int   m_NumInputs;
  doublem_MaxLearn;
  doublem_MinLearn;
  CString m_SaveName;
  //}}AFX_DATA
  // ClassWizard generated virtual function overrides
  //{{AFX_VIRTUAL(CSomTrainerDlg)
   protected:
  virtual void DoDataExchange(CDataExchange* pDX);            // DDX/DDV support
  //}}AFX_VIRTUAL

// Implementation
protected:
  CSomTrainerDlgAutoProxy* m_pAutoProxy;
  HICON m_hIcon;

   BOOL CanExit();

  // Generated message map functions
  //{{AFX_MSG(CSomTrainerDlg)
  virtual BOOL OnInitDialog();
  afx_msg void OnSysCommand(UINT nID, LPARAM lParam);
  afx_msg void OnDestroy();
   afx_msg void OnPaint();
  afx_msg HCURSOR OnQueryDragIcon();
  afx_msg void OnClose();
  virtual void OnOK();
  virtual void OnCancel();
```

```
  afx_msg void OnFileselectButton();
  afx_msg void OnLoadButton();
  afx_msg void OnQuitButton();
  afx_msg void OnSaveButton();
  afx_msg void OnTrainButton();
  afx_msg void OnCheckButton();
  afx_msg void OnFileExit();
  afx_msg void OnFileLoadnetwork();
  afx_msg void OnFileSavenatwork();
  afx_msg void OnNetwokSaveweights();
  afx_msg void OnNetwokViewweights();
  afx_msg void OnTestSaveresults();
  afx_msg void OnTestTestnetwork();
  afx_msg void OnTestViewresults();
  afx_msg void OnChangeFileEdit();
   afx_msg void OnChangeFilesizeEdit();
  afx_msg void OnChangeMaxlearnEdit();
  afx_msg void OnChangeMaxtrainEdit();
  afx_msg void OnChangeMinlearnEdit();
   afx_msg void OnChangeMinneighEdit();
  afx_msg void OnSavetestButton();
  afx_msg void OnTestsetButton();
  afx_msg void OnTestsaveButton();
  //}}AFX_MSG
  DECLARE_MESSAGE_MAP()
};

//{{AFX_INSERT_LOCATION}}
// Microsoft Developer Studio will insert additional declarations immediately before the previous line.

#endif // !defined
(AFX_SOMTRAINERDLG_H__952466AB_ACC2_11D3_B160_8BFB919D1E24__INCLUDED_)



// ViewWeightsDlg.cpp : implementation file
//

#include "stdafx.h"
#include "Som Trainer.h"
#include "ViewWeightsDlg.h"
#include "Som TrainerDlg.h"
#include <math.h>

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif


/////////////////////////////////////////////////////////////////////////////
// ViewWeightsDlg dialog


ViewWeightsDlg::ViewWeightsDlg(CWnd* pParent /*=NULL*/)
  : CDialog(ViewWeightsDlg::IDD, pParent)
{
 //{{AFX_DATA_INIT(ViewWeightsDlg)
  // NOTE: the ClassWizard will add member initialization here
 //}}AFX_DATA_INIT
}


void ViewWeightsDlg::DoDataExchange(CDataExchange* pDX)
{
```

```
 CDialog::DoDataExchange(pDX);
 //{{AFX_DATA_MAP(ViewWeightsDlg)
  // NOTE: the ClassWizard will add DDX and DDV calls here
 //}}AFX_DATA_MAP
}


BEGIN_MESSAGE_MAP(ViewWeightsDlg, CDialog)
 //{{AFX_MSG_MAP(ViewWeightsDlg)
 ON_BN_CLICKED(IDC_PLOT_BUTTON, OnPlotButton)
 ON_BN_CLICKED(IDC_SUM_BUTTON, OnSumButton)
 ON_BN_CLICKED(IDC_BARS_BUTTON, OnBarsButton)
 ON_WM_MOUSEMOVE()
 ON_BN_CLICKED(IDC_FIRING_BUTTON, OnFiringButton)
 //}}AFX_MSG_MAP
END_MESSAGE_MAP()

/////////////////////////////////////////////////////////////////////////////
// ViewWeightsDlg message handlers

BOOL ViewWeightsDlg::OnInitDialog()
{
 CDialog::OnInitDialog();

 // TODO: Add extra initialization here
 OnPlotButton();
 Invalidate();
 return TRUE;  // return TRUE unless you set the focus to a control
               // EXCEPTION: OCX Property Pages should return FALSE
}




void ViewWeightsDlg::OnOK()
{
 // TODO: Add extra validation here
 CDialog::OnOK();
}


void ViewWeightsDlg::OnPaint()
{

 // Do not call CDialog::OnPaint() for painting messages
}



void ViewWeightsDlg::OnPlotButton()
{
 // TODO: Add your control notification handler code here
  CClientDC dc(this);
  int StartX;
  int StartY;

  CPen RectPen;
  RectPen.CreatePen(PS_SOLID,1,RGB(255,255,255));
  CPen* pOriginalPen;
  pOriginalPen = dc.SelectObject(&RectPen);

  for(StartY = 10; StartY<360; StartY+=35)
  {
   for (StartX = 10; StartX<360; StartX+=35)
   {
    Rectangle(dc,StartX,StartY,StartX+30,StartY+30);
```

```
    }
  }

  if (Test==1)
  {

    CPen WinPen;
    WinPen.CreatePen(PS_SOLID,4,RGB(255,100,0));
    pOriginalPen = dc.SelectObject(&WinPen);
    StartX = 10 + (35 * WinX);
    StartY = 10 + (35 * WinY);
    Rectangle(dc,StartX, StartY, StartX + 30, StartY + 30);
  }

  CPen PlotPen;
  PlotPen.CreatePen(PS_SOLID,1,RGB(255,0,0));
  pOriginalPen = dc.SelectObject(&PlotPen);

  int Posx, Posy;

  for(StartY = 10; StartY<360; StartY+=35)
  {
    for (StartX = 10; StartX<360; StartX+=35)
    {
      Posx = StartX;
      Posy = StartY + 29;
      MoveToEx(dc,Posx,Posy,NULL);

      for (int loop3 = 0; loop3<NumIns; loop3++)
      {
       Posy = StartY + 30 - (int)(30*WeightVals[(StartX-10)/35][(StartY-10)/35][loop3]);
       LineTo(dc,Posx,Posy);
       Posx += int(35/NumIns);
      }
    }
  }


}

void ViewWeightsDlg::OnSumButton()
{
 // TODO: Add your control notification handler code here
 CClientDC dc(this);
 int StartX;
 int StartY;
 int loop3;
 double Sum;
 int loop1;
 int loop2;


 for(StartY = 10; StartY<360; StartY+=35)
 {
  for (StartX = 10; StartX<360; StartX+=35)
  {
    Sum = 0;
    double Dist = 0;
    for(loop3 = 0; loop3<NumIns; loop3++)
    {
     double Temp = WeightVals[(StartX-10)/35][(StartY-10)/35][loop3];
     Sum += sqrt(pow(Temp,2));
     //Dist += WeightVals[(StartX-10)/35][(StartY-10)/35][loop3] - Inputs[loop3];
    }
    //Sum = (int)(Dist*51);
    int Mult = 255/(NumIns*MaxWeight);
    for (loop1 = 0; loop1<30; loop1++)
    {
```

```
    for (loop2 = 0; loop2<30; loop2++)
    {
     SetPixel(dc,StartX + loop1, StartY+loop2,RGB(0,(int)(Mult*Sum),(int)(Mult*Sum)));
    }
   }
  }
 }
 if (Test==1)
  {

   CPen WinPen;
   WinPen.CreatePen(PS_SOLID,4,RGB(255,100,0));
   CPen* pOriginalPen;
   pOriginalPen = dc.SelectObject(&WinPen);

   StartX = 10 + (35 * WinX);
   StartY = 10 + (35 * WinY);
   MoveToEx(dc, StartX, StartY, NULL);
   LineTo(dc, StartX+30, StartY);
   LineTo(dc, StartX+30, StartY+30);
   LineTo(dc, StartX, StartY + 30);
   LineTo(dc, StartX, StartY);
  }


}

void ViewWeightsDlg::OnBarsButton()
{
  // TODO: Add your control notification handler code here
  CClientDC dc(this);
  int StartX;
  int StartY;
  int loop3;
  int loop1;
  int loop2;

  CPen RectPen;
  RectPen.CreatePen(PS_SOLID,1,RGB(255,255,255));
  CPen* pOriginalPen;
  pOriginalPen = dc.SelectObject(&RectPen);

  for(StartY = 10; StartY<360; StartY+=35)
  {
    for (StartX = 10; StartX<360; StartX+=35)
    {
      Rectangle(dc,StartX,StartY,StartX+30,StartY+30);
    }
  }


  for(StartY = 10; StartY<360; StartY+=35)
  {
    for (StartX = 10; StartX<360; StartX+=35)
    {
      for(loop3 = 0; loop3<NumIns; loop3++)
      {
        for(loop1 = 0; loop1<NumIns+1;loop1++)
        {
          for (loop2 = 0; loop2<(int)(30*WeightVals[(StartX-10)/35][(StartY-10)/35][loop3]); loop2++)
          {
            SetPixel(dc,StartX + (NumIns+1)*loop3 + loop1, StartY + 30 - loop2,RGB(0,50,(int)
(255*WeightVals[(StartX-10)/35][(StartY-10)/35][loop3])));
          }
        }
      }
    }
  }
```

```
  if (Test==1)
  {
    CPen WinPen;
    WinPen.CreatePen(PS_SOLID,4,RGB(255,100,0));
    CPen* pOriginalPen;
    pOriginalPen = dc.SelectObject(&WinPen);
    StartX = 10 + (35 * WinX);
    StartY = 10 + (35 * WinY);
    MoveToEx(dc, StartX, StartY, NULL);
    LineTo(dc, StartX+30, StartY);
    LineTo(dc, StartX+30, StartY+30);
    LineTo(dc, StartX, StartY + 30);
    LineTo(dc, StartX, StartY);
  }


}


void ViewWeightsDlg::OnFiringButton() //distances
{
  // TODO: Add your control notification handler code here
  int StartX;
  ClientDC dc(this);
  int StartY;
  int loop3;
  double Sum;
  int loop1;
  int loop2;
  int Mult;

  double Res1 = abs(MaxWeight-MinIn);
  double Res2 = abs(MaxIn - MinWeight);
  if(Res1>Res2)  Mult = 255/(NumIns*Res1);
  else Mult = 255/(NumIns*Res2);


  for(StartY = 10; StartY<360; StartY+=35)
  {
    for (StartX = 10; StartX<360; StartX+=35)
    {
      Sum = 0;
      double Dist = 0;
      for(loop3 = 0; loop3<NumIns; loop3++)
      {

        double Temp= WeightVals[(StartX-10)/35][(StartY-10)/35][loop3] - Inputs[loop3];
        Sum += sqrt(pow(Temp,2));
      }
      Sum = (int)(Sum*Mult);
      for (loop1 = 0; loop1<30; loop1++)
      {
        for (loop2 = 0; loop2<30; loop2++)
        {
          SetPixel(dc,StartX + loop1, StartY+loop2,RGB(Sum,0,0));//(int)(51*Sum)));
        }
      }
    }
  }
  if (Test==1)
  {
    CPen WinPen;
    WinPen.CreatePen(PS_SOLID,4,RGB(255,100,0));
    CPen* pOriginalPen;
    pOriginalPen = dc.SelectObject(&WinPen);
    StartX = 10 + (35 * WinX);
    StartY = 10 + (35 * WinY);
    MoveToEx(dc, StartX, StartY, NULL);
```

```
    LineTo(dc, StartX+30, StartY);
    LineTo(dc, StartX+30, StartY+30);
    LineTo(dc, StartX, StartY + 30);
    LineTo(dc, StartX, StartY);

    double WinDist=0;
    for(int loop = 0; loop <NumIns; loop++)
    {
      double Temp = WeightVals[WinX][WinY][loop] - Inputs[loop];
      WinDist += sqrt(pow(Temp,2));
    }
    char Text[500];
    sprintf(Text,"Vectorial Distance is : %f - Mult :%i",WinDist,Mult);
    MessageBox(Text);
  }


}




#if !defined(AFX_VIEWWEIGHTSDLG_H__A3E1F763_ACF0_11D3_B160_8BFB919D1E24__INCLUDED_)
#define AFX_VIEWWEIGHTSDLG_H__A3E1F763_ACF0_11D3_B160_8BFB919D1E24__INCLUDED_

#if _MSC_VER >= 1000
#pragma once
#endif // _MSC_VER >= 1000
// ViewWeightsDlg.h : header file
//


const int NumIns=8;
/////////////////////////////////////////////////////////////////////////////
// ViewWeightsDlg dialog

class ViewWeightsDlg : public CDialog
{
// Construction
public:

 ViewWeightsDlg(CWnd* pParent = NULL);   // standard constructor
 double WeightVals[10][10][NumIns];
 int FireRates[10][10];
 int MaxCycles;
 int MaxLines;
 double Inputs[NumIns];
 int Test;
 int WinX;
 int WinY;
 double MaxWeight;
 double MinWeight;
 double MaxIn;
 double MinIn;


// Dialog Data
 //{{AFX_DATA(ViewWeightsDlg)
 enum { IDD = IDD_WEIGHTS_DIALOG };
  // NOTE: the ClassWizard will add data members here
 //}}AFX_DATA


// Overrides
 // ClassWizard generated virtual function overrides
 //{{AFX_VIRTUAL(ViewWeightsDlg)
 protected:
```

```
 virtual void DoDataExchange(CDataExchange* pDX);    // DDX/DDV support
 //}}AFX_VIRTUAL

// Implementation
protected:

 // Generated message map functions
 //{{AFX_MSG(ViewWeightsDlg)
 afx_msg void OnPaint();
 virtual BOOL OnInitDialog();
 virtual void OnOK();
 afx_msg void OnPlotButton();
 afx_msg void OnSumButton();
 afx_msg void OnBarsButton();
 afx_msg void OnFiringButton();
 //}}AFX_MSG
 DECLARE_MESSAGE_MAP()
};
//{{AFX_INSERT_LOCATION}}
// Microsoft Developer Studio will insert additional declarations immediately before the previous line.

#endif // !defined
(AFX_VIEWWEIGHTSDLG_H__A3E1F763_ACF0_11D3_B160_8BFB919D1E24__INCLUDED_)




/*************************************************************************\
|      |
| Header code for SOM network                                    |
|      |
|---------------------------------------------------------------  |
|
|
\*************************************************************************/


#include <fstream.h>
#include <math.h>
#include <stdlib.h>
#include <time.h>

const int SOM_NUMBER_INPUTS = 8;


class SOM
{
  //designed for a 10x10 network with 5 inputs, to adapt change the
  //size of the arrays Weight[][][], Winner[][] and Input[]
  //Designed to read it's info from an ASCII file

private :
  double MaxRate;  //Maximum Learning Rate
  double MinRate;  //Minimum Learning Rate
  double CurRate;  //Current Learning Rate
  int MaxCycles;  //Max number of training cycles
  int CurCycle;  //Current cycle value

  int MaxNeighbour; //Maximum neighbourhood
  int MinNeighbour; //Minimum Neighbourhood
  int CurNeighbour; //Current Neighbourhood

  double Weight[10][10][SOM_NUMBER_INPUTS];
  int Fire[10][10];

  int WinnerX;
  int WinnerY;
```

```
  double Error;

  int MapWidth;
  int MapHeight;
  int NumInputs;
  double Input[SOM_NUMBER_INPUTS];

  int DataLines;   //No of lines of data in file to be read
  CString FileName;   //Name of file to be presented
  int FilePos;  //Current Position of file pointer


public :

  int SetDataLines(int);          //return 0 for error, 1 for success
  int CheckFileSize(); //returns 1 if size is equal to DataLines, else 0
  int SetFileName(CString);       //checks for file existance and returns 1 for success, else 0
  int ReadDataLine(); //reads a row of data from file and stores it in Input[]. If the data
  //is not >=0 and <=1, returns a zero
  double GiveData(int);           //returns the value of Input[int]
  int GiveWidth(){return MapWidth;}
  int GiveHeight(){return MapHeight;}
  double GiveRate(){return CurRate;}
  int GiveNeighbour(){return CurNeighbour;}
  int GiveMaxNeighbour(){return MaxNeighbour;}
  int GiveCycle(){return CurCycle;}
  int GiveWinnerX(){return WinnerX;}
  int GiveWinnerY(){return WinnerY;}
  double GiveWeight(int X,int Y, int I){return Weight[X][Y][I];}
  void DisplayWeights();
  double GiveError(){return Error;}
  int GiveFire(int x, int y){return Fire[x][y];}


  void InitCurCycle(){CurCycle = 0;}
  int SetMaxCycles(int);          //sets the value of MaxCycles, returns 1 on success, 0 on fail
  int SetMinNeighbour(int);//sets the value of MinNeighbour, returns 1 on success
  int SetRates(double, double);  //sets the maximum and minimum learning rate values
  void InitFilePos(){FilePos = 0;}
  void SetNumInputs(){NumInputs=SOM_NUMBER_INPUTS;}
  void SetMapSize(){MapHeight=10;MapWidth=10;}
  void SetWeight(int x,int y, int z, double val)
  {
    Weight[x][y][z] = val;
  }

  void CalcMaxNeighbour();       //Calculates the maximum neighbourhood
  void CalcCurNeighbour();       //calculates the current neighbourhood
  void CalcCurRate(); //Calculates the curretn learning rate
  void RandomWeights();          //Initialises the network weights to random values
  void IncCycle(){CurCycle++;}
  void CalcWinner();
  void CalcError(); //Runs network through one sequence of data for all nodes
  //Call ReadDataLine(), CalcCurNeighbour(), CalcCurRate()
  //before applying this function.
  int RunNet();  //returns 1 on success, 0 on fail
  //Handles all calls required by CalcError()
  //calls CalcError()
  int InitNet(char*,int,int,int,double,double);//Call this to initialize the network
  //variables sent are :FileName,Number of Data Lines in File
  //No of Training Cycles, Min Neighbourhood, Max Learning Rate
  //min Learning Rate
};

//###############################################################
//###############################################################
```

```
//******************** END OF CLASS BODY ***********************

int SOM::SetDataLines(int Number)
{
  if (Number>0)
  {
    DataLines = Number;
    return(1);
  }
  else return 0;
}

//*********************************************************************

int SOM::SetFileName(CString Name)
{
  ifstream TestFile(Name,ios::nocreate,filebuf::sh_read);
  if (TestFile)
  {
    FileName = Name;
    TestFile.close();
    return 1;
  }
 else
  {
    TestFile.close();
    return 0;
  }
}

//*********************************************************************

int SOM::CheckFileSize()
{
   char store;
  int count=1;
  ifstream TestFile(FileName,ios::nocreate);
   while(TestFile)
  {
    TestFile.get(store);
    // TestFile.seekg(1,ios::cur);
    if(store=='\n')
    count++;
  }

  TestFile.close();

  FilePos = 0;
  if (count== DataLines)
    return 1;
  else return 0;

}

//*********************************************************************

int SOM::ReadDataLine()
{
  int loop;

  ifstream DataFile(FileName,ios::nocreate,filebuf::sh_read);
  DataFile.seekg(FilePos,ios::beg);

  for (loop = 0; loop <NumInputs; loop++)
  {
    DataFile >> Input[loop];
    if ((Input[loop]<0) || (Input[loop]>1))
      return 0;
```

```
    DataFile.seekg(1,ios::cur);
    FilePos = DataFile.tellg();
  }
  DataFile.close();

//  ofstream CheckFile("Checkfile.txt");
//  for(loop = 0; loop < NumInputs; loop++)
//  {
//    CheckFile << Input[loop] << " - ";
//  }
//  CheckFile.close();
  return 1;
}

//*****************************************************************

double SOM::GiveData(int Pos)
{
  return (Input[Pos]);
}

//*****************************************************************


int SOM::SetMaxCycles(int max)
{
  if (max>0)
  {
    MaxCycles = max;
    return 1;
  }
  else return 0;
}

//*****************************************************************

int SOM::SetMinNeighbour(int min)
{
  if (min>=0)
  {
     MinNeighbour = min;
    return 1;
  }
  else return 0;
}

//*****************************************************************


int SOM::SetRates(double min, double max)
{
  int check1 = 0;
  int check2 = 0;
  int check3 = 0;

  if (max>=min) check1 = 1;
  if ((min>=0) && (min<=1)) check2 = 1;
  if ((max>=0) && (max<=1)) check3 = 1;

  if ((check1==1)&&(check2==1)&&(check3==1))
  {
     MinRate = min;
    MaxRate = max;
    return 1;
  }
   else return 0;
}
```

```
//*************************************************************

void SOM::CalcMaxNeighbour()
{
  div_t div_result;
  int MapSize= MapWidth*MapHeight;
  div_result = div((int)(sqrt(MapSize)), 2);
  MaxNeighbour = div_result.quot;
}

//*************************************************************

void SOM::CalcCurNeighbour()
{
  int NeighDiff;
  double CycleDiff;

  NeighDiff = MaxNeighbour - MinNeighbour;
  CycleDiff = (MaxCycles - CurCycle);

   CurNeighbour = (int)(MinNeighbour + ((NeighDiff)*(pow((CycleDiff/MaxCycles),2))));
}

//*************************************************************

void SOM::CalcCurRate()
{
  double RateDiff;
  double CycleDiff;
  RateDiff = MaxRate - MinRate;
  CycleDiff = MaxCycles - CurCycle;

  CurRate = MinRate + ((RateDiff)*(pow((CycleDiff/MaxCycles),2)));
}

//*************************************************

void SOM::RandomWeights()
{
  int loop1, loop2, loop3;
  srand((unsigned)time(NULL));

  for (loop1=0; loop1<MapHeight; loop1++)
  {
    for (loop2 = 0; loop2<MapWidth; loop2++)
    {
      for (loop3 = 0; loop3<NumInputs; loop3++)
      {
        Weight[loop2][loop1][loop3] = (float)( rand())/RAND_MAX;
      }
      Fire[loop2][loop1] = 0;
    }
  }
}

//*************************************************************************

void SOM::CalcWinner()
{
  int loop1;
  int loop2;
  int loop3;
  double Dist=0;
```

```
  double Result=0;
  double SmallDistance = 50;    //Set it to the maximum value
  WinnerX = 10;
  WinnerY = 10;

  //First present the line of data to the network, and calculate the node with
  //the lowest activation level

  for (loop2 = 0; loop2<MapHeight;loop2++)
  {
    for (loop1 = 0; loop1<MapWidth; loop1++)
    {
      Result = 0;
      Dist = 0;
      for (loop3 = 0; loop3<NumInputs; loop3++)
      {
        Dist = (Weight[loop1][loop2][loop3] - Input[loop3]);
        Result += pow(Dist,2);
      }
      if (Result < SmallDistance)
      {
        WinnerX = loop1;                    //stores location of winning node
        WinnerY = loop2;
        SmallDistance = Result;
        Fire[loop1][loop2]++;
      }
    }
  }
}


//*************************************************************************


void SOM::CalcError()
{
  int loop1;
  int loop2;
  int loop3;
  double Dist=0;

  double Result=0;
  double SmallDistance = 50;    //Set it to the maximum value
  WinnerX = 10;
  WinnerY = 10;

  //First present the line of data to the network, and calculate the node with
  //the lowest activation level
  for (loop2 = 0; loop2<MapHeight;loop2++)
  {
    for (loop1 = 0; loop1<MapWidth; loop1++)
    {
      Result = 0;
      Dist = 0;
      for (loop3 = 0; loop3<NumInputs; loop3++)
      {
        Dist = (Weight[loop1][loop2][loop3] - Input[loop3]);
        Result += pow(Dist,2);
      }
      if (Result < SmallDistance)
      {
        WinnerX = loop1;                    //stores location of winning node
        WinnerY = loop2;
        SmallDistance = Result;
        Fire[loop1][loop2]++;
      }
```

```
    }
  }

  //Now run the error through the network
  //Checks that the neighbiurhood doesn't run over the map boundaries
  //Adjust the weight values for each Input/Node

  double Correc = 0;
  int StartX, StartY, EndX, EndY;

  if ((WinnerX - CurNeighbour)<0) StartX = 0;
  else StartX = WinnerX - CurNeighbour;
  if ((WinnerX + CurNeighbour)>=MapWidth) EndX = MapWidth-1;
  else EndX = WinnerX + CurNeighbour;
  if ((WinnerY - CurNeighbour)<0) StartY = 0;
  else StartY = WinnerY - CurNeighbour;
  if ((WinnerY + CurNeighbour)>=MapHeight) EndY = MapHeight-1;
  else EndY = WinnerY + CurNeighbour;


  for (loop2 = StartY; loop2 <= EndY; loop2++)
  {
    for (loop1 = StartX; loop1<=EndX; loop1++)
    {
      for (loop3 = 0; loop3<NumInputs; loop3++)
      {
        Dist = 0;
        Correc = 0;
        Dist = (Weight[loop1][loop2][loop3] - Input[loop3]);
        Correc = (CurRate*Dist);
        //if ((Correc>=-1) && (Correc<=0))
        Weight[loop1][loop2][loop3] -= Correc;
        if ( Weight[loop1][loop2][loop3]>1) Weight[loop1][loop2][loop3] = 1;
        if ( Weight[loop1][loop2][loop3]<0) Weight[loop1][loop2][loop3] = 0;
        if ( Weight[loop1][loop2][loop3]<0.000001) Weight[loop1][loop2][loop3] = 0;
       }
    }
  }
}


//*********************************************************************

int SOM::RunNet()
{
  int loop;
  for (loop = 0; loop<DataLines; loop++)
  {
    if(ReadDataLine())
    {
      CalcCurNeighbour();
      CalcCurRate();
      CalcError();
   }
    else
      return 0;
  }
  FilePos = 0;
  return 1;
}

//*************************************************************

int SOM::InitNet(char *filename, int numlines, int traincycles,int minneighbour,
      double maxrate, double minrate)
{
 CurCycle = 0;
 if (!SetFileName(filename)) return 0;
```

```
 if (!SetDataLines(numlines)) return 0;
 if (!CheckFileSize()) return 0;
 SetNumInputs();
 SetMapSize();
 if (!SetRates(minrate, maxrate)) return 0;
 if (!SetMinNeighbour(minneighbour)) return 0;
 if (!SetMaxCycles(traincycles)) return 0;
 CalcMaxNeighbour();
 CalcCurNeighbour();
 CalcCurRate();
 RandomWeights();
 InitFilePos();

 return 1;
}


//************************************************************

void SOM::DisplayWeights()
{
  double Sum = 0;
  int loop1, loop2, loop3;
  for (loop2 = 0; loop2< MapHeight; loop2++)
  {
    for ( loop1 = 0; loop1<MapWidth; loop1++)
    {
      for (loop3 = 0; loop3 < NumInputs; loop3++)
      {
        Sum+= Weight[loop1][loop2][loop3];
      }
      cout <<Sum<<"\t";
      Sum = 0;
    }
    cout <<"\n";

  }
}
```

# 20.2 - Results Filter

```
// ResultsFilterDlg.cpp : implementation file
//
```

```
#include "stdafx.h"
#include "ResultsFilter.h"
#include "ResultsFilterDlg.h"
#include <fstream.h>

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

/////////////////////////////////////////////////////////////////////////////
// CAboutDlg dialog used for App About

class CAboutDlg : public CDialog
{
public:
 CAboutDlg();

// Dialog Data
 //{{AFX_DATA(CAboutDlg)
 enum { IDD = IDD_ABOUTBOX };
 //}}AFX_DATA

 // ClassWizard generated virtual function overrides
 //{{AFX_VIRTUAL(CAboutDlg)
 protected:
 virtual void DoDataExchange(CDataExchange* pDX);    // DDX/DDV support
 //}}AFX_VIRTUAL

// Implementation
protected:
 //{{AFX_MSG(CAboutDlg)
 //}}AFX_MSG
 DECLARE_MESSAGE_MAP()
};

CAboutDlg::CAboutDlg() : CDialog(CAboutDlg::IDD)
{
 //{{AFX_DATA_INIT(CAboutDlg)
 //}}AFX_DATA_INIT
}

void CAboutDlg::DoDataExchange(CDataExchange* pDX)
{
 CDialog::DoDataExchange(pDX);
 //{{AFX_DATA_MAP(CAboutDlg)
 //}}AFX_DATA_MAP
}

BEGIN_MESSAGE_MAP(CAboutDlg, CDialog)
 //{{AFX_MSG_MAP(CAboutDlg)
  // No message handlers
 //}}AFX_MSG_MAP
END_MESSAGE_MAP()

/////////////////////////////////////////////////////////////////////////////
// CResultsFilterDlg dialog

CResultsFilterDlg::CResultsFilterDlg(CWnd* pParent /*=NULL*/)
 : CDialog(CResultsFilterDlg::IDD, pParent)
{
 //{{AFX_DATA_INIT(CResultsFilterDlg)
 m_Source = _T("");
 //}}AFX_DATA_INIT
 // Note that LoadIcon does not require a subsequent DestroyIcon in Win32
 m_hIcon = AfxGetApp()->LoadIcon(IDR_MAINFRAME);
```

```
}

void CResultsFilterDlg::DoDataExchange(CDataExchange* pDX)
{
 CDialog::DoDataExchange(pDX);
 //{{AFX_DATA_MAP(CResultsFilterDlg)
 DDX_Text(pDX, IDC_SOURCE_EDIT, m_Source);
 //}}AFX_DATA_MAP
}

BEGIN_MESSAGE_MAP(CResultsFilterDlg, CDialog)
 //{{AFX_MSG_MAP(CResultsFilterDlg)
 ON_WM_SYSCOMMAND()
 ON_WM_PAINT()
 ON_WM_QUERYDRAGICON()
 ON_BN_CLICKED(IDC_PROCESS_BUTTON, OnProcessButton)
 ON_BN_CLICKED(IDC_SELECT_BUTTON, OnSelectButton)
 //}}AFX_MSG_MAP
END_MESSAGE_MAP()

/////////////////////////////////////////////////////////////////////////////
// CResultsFilterDlg message handlers

BOOL CResultsFilterDlg::OnInitDialog()
{
  CDialog::OnInitDialog();

  // Add "About..." menu item to system menu.

  // IDM_ABOUTBOX must be in the system command range.
  ASSERT((IDM_ABOUTBOX & 0xFFF0) == IDM_ABOUTBOX);
  ASSERT(IDM_ABOUTBOX < 0xF000);

  CMenu* pSysMenu = GetSystemMenu(FALSE);
  if (pSysMenu != NULL)
  {
    CString strAboutMenu;
    strAboutMenu.LoadString(IDS_ABOUTBOX);
    if (!strAboutMenu.IsEmpty())
    {
      pSysMenu->AppendMenu(MF_SEPARATOR);
      pSysMenu->AppendMenu(MF_STRING, IDM_ABOUTBOX, strAboutMenu);
    }
  }

  // Set the icon for this dialog.  The framework does this automatically
  //  when the application's main window is not a dialog
  SetIcon(m_hIcon, TRUE);                                    // Set big icon
  SetIcon(m_hIcon, FALSE);                    // Set small icon

  // TODO: Add extra initialization here

  return TRUE;  // return TRUE  unless you set the focus to a control
}

void CResultsFilterDlg::OnSysCommand(UINT nID, LPARAM lParam)
{
  if ((nID & 0xFFF0) == IDM_ABOUTBOX)
  {
    CAboutDlg dlgAbout;
    dlgAbout.DoModal();
  }
  else
  {
    CDialog::OnSysCommand(nID, lParam);
  }
}
```

```
// If you add a minimize button to your dialog, you will need the code below
//  to draw the icon.  For MFC applications using the document/view model,
//  this is automatically done for you by the framework.

void CResultsFilterDlg::OnPaint()
{
  if (IsIconic())
  {
    CPaintDC dc(this); // device context for painting
    SendMessage(WM_ICONERASEBKGND, (WPARAM) dc.GetSafeHdc(), 0);

    // Center icon in client rectangle
    int cxIcon = GetSystemMetrics(SM_CXICON);
    int cyIcon = GetSystemMetrics(SM_CYICON);
    CRect rect;
    GetClientRect(&rect);
    int x = (rect.Width() - cxIcon + 1) / 2;
    int y = (rect.Height() - cyIcon + 1) / 2;

    // Draw the icon
    dc.DrawIcon(x, y, m_hIcon);
  }
  else
  {
    CDialog::OnPaint();
  }
}

// The system calls this to obtain the cursor to display while the user drags
//  the minimized window.
HCURSOR CResultsFilterDlg::OnQueryDragIcon()
{
  return (HCURSOR) m_hIcon;
}

void CResultsFilterDlg::OnProcessButton()
{
  ifstream InFile(m_Source,ios::nocreate);
  ofstream OutFile("Temp.tmp");
  if(!InFile)
  {
    MessageBox("File not found !");
    return;
  }
  while(InFile)
  {
  }
  InFile.close();
  Outfile.close();

}

void CResultsFilterDlg::OnSelectButton()
{
  CfileDialog m_Open(FALSE,"txt","*.txt",OFN_FILEMUSTEXIST,"Results File (*.txt)|*.txt||",NULL);
  if(m_Open.DoModal()==IDOK)
  {
    m_Source = m_Open.GetPathName();
    UpdateData(FALSE);
  }
}
```

# 20.3 - Bitmap Wave Comparator

```
// bitmap wave comparatorDlg.cpp : implementation file
//

#include "stdafx.h"
#include "bitmap wave comparator.h"
#include "bitmap wave comparatorDlg.h"
#include <fstream.h>
#include <math.h>

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

/////////////////////////////////////////////////////////////////////////////
// CAboutDlg dialog used for App About

class CAboutDlg : public CDialog
{
public:
 CAboutDlg();

// Dialog Data
 //{{AFX_DATA(CAboutDlg)
 enum { IDD = IDD_ABOUTBOX };
 //}}AFX_DATA

 // ClassWizard generated virtual function overrides
 //{{AFX_VIRTUAL(CAboutDlg)
 protected:
 virtual void DoDataExchange(CDataExchange* pDX);    // DDX/DDV support
 //}}AFX_VIRTUAL

// Implementation
protected:
 //{{AFX_MSG(CAboutDlg)
 //}}AFX_MSG
 DECLARE_MESSAGE_MAP()
};

CAboutDlg::CAboutDlg() : CDialog(CAboutDlg::IDD)
{
 //{{AFX_DATA_INIT(CAboutDlg)
 //}}AFX_DATA_INIT
}

void CAboutDlg::DoDataExchange(CDataExchange* pDX)
{
 CDialog::DoDataExchange(pDX);
 //{{AFX_DATA_MAP(CAboutDlg)
 //}}AFX_DATA_MAP
}

BEGIN_MESSAGE_MAP(CAboutDlg, CDialog)
 //{{AFX_MSG_MAP(CAboutDlg)
  // No message handlers
 //}}AFX_MSG_MAP
END_MESSAGE_MAP()

/////////////////////////////////////////////////////////////////////////////
```

```
// CBitmapwavecomparatorDlg dialog

CBitmapwavecomparatorDlg::CBitmapwavecomparatorDlg(CWnd* pParent /*=NULL*/)
 : CDialog(CBitmapwavecomparatorDlg::IDD, pParent)
{
 //{{AFX_DATA_INIT(CBitmapwavecomparatorDlg)
 m_DiffCheck = FALSE;
 m_OverlayCheck = FALSE;
 m_AutoCheck = FALSE;
 m_Image = FALSE;
 //}}AFX_DATA_INIT
 // Note that LoadIcon does not require a subsequent DestroyIcon in Win32
 m_hIcon = AfxGetApp()->LoadIcon(IDR_MAINFRAME);
}

void CBitmapwavecomparatorDlg::DoDataExchange(CDataExchange* pDX)
{
 CDialog::DoDataExchange(pDX);
 //{{AFX_DATA_MAP(CBitmapwavecomparatorDlg)
 DDX_Check(pDX, IDC_DIFF_CHECK, m_DiffCheck);
 DDX_Check(pDX, IDC_OVERLAY_CHECK, m_OverlayCheck);
 DDX_Check(pDX, IDC_AUTO_CHECK, m_AutoCheck);
 DDX_Check(pDX, IDC_IMAGE_CHECK, m_Image);
 //}}AFX_DATA_MAP
}

BEGIN_MESSAGE_MAP(CBitmapwavecomparatorDlg, CDialog)
 //{{AFX_MSG_MAP(CBitmapwavecomparatorDlg)
 ON_WM_SYSCOMMAND()
 ON_WM_PAINT()
 ON_WM_QUERYDRAGICON()
 ON_BN_CLICKED(IDC_B1ANALYSE_BUTTON, OnB1analyseButton)
 ON_BN_CLICKED(IDC_B2ANALYSE_BUTTON, OnB2analyseButton)
 ON_BN_CLICKED(IDC_BITMAP1_BUTTON, OnBitmap1Button)
 ON_BN_CLICKED(IDC_BITMAP2_BUTTON, OnBitmap2Button)
 ON_BN_CLICKED(IDC_QUIT_BUTTON, OnQuitButton)
 ON_BN_CLICKED(IDC_DIFF_CHECK, OnDiffCheck)
 ON_BN_CLICKED(IDC_OVERLAY_CHECK, OnOverlayCheck)
 ON_BN_CLICKED(IDC_AUTO_CHECK, OnAutoCheck)
 ON_BN_CLICKED(IDC_IMAGE_CHECK, OnImageCheck)
 //}}AFX_MSG_MAP
END_MESSAGE_MAP()

/////////////////////////////////////////////////////////////////////////////
// CBitmapwavecomparatorDlg message handlers

BOOL CBitmapwavecomparatorDlg::OnInitDialog()
{
 CDialog::OnInitDialog();

 // Add "About..." menu item to system menu.

 // IDM_ABOUTBOX must be in the system command range.
 ASSERT((IDM_ABOUTBOX & 0xFFF0) == IDM_ABOUTBOX);
 ASSERT(IDM_ABOUTBOX < 0xF000);

 CMenu* pSysMenu = GetSystemMenu(FALSE);
 if (pSysMenu != NULL)
 {
  CString strAboutMenu;
  strAboutMenu.LoadString(IDS_ABOUTBOX);
  if (!strAboutMenu.IsEmpty())
  {
   pSysMenu->AppendMenu(MF_SEPARATOR);
   pSysMenu->AppendMenu(MF_STRING, IDM_ABOUTBOX, strAboutMenu);
  }
 }
```

```
 // Set the icon for this dialog.  The framework does this automatically
 //  when the application's main window is not a dialog
 SetIcon(m_hIcon, TRUE);                                    // Set big icon
 SetIcon(m_hIcon, FALSE);                          // Set small icon

 // TODO: Add extra initialization here

 return TRUE;  // return TRUE  unless you set the focus to a control
}

void CBitmapwavecomparatorDlg::OnSysCommand(UINT nID, LPARAM lParam)
{
 if ((nID & 0xFFF0) == IDM_ABOUTBOX)
 {
  CAboutDlg dlgAbout;
  dlgAbout.DoModal();
 }
 else
 {
  CDialog::OnSysCommand(nID, lParam);
 }
}

// If you add a minimize button to your dialog, you will need the code below
//  to draw the icon.  For MFC applications using the document/view model,
//  this is automatically done for you by the framework.

void CBitmapwavecomparatorDlg::OnPaint()
{
 if (IsIconic())
 {
  CPaintDC dc(this); // device context for painting

  SendMessage(WM_ICONERASEBKGND, (WPARAM) dc.GetSafeHdc(), 0);

  // Center icon in client rectangle
  int cxIcon = GetSystemMetrics(SM_CXICON);
  int cyIcon = GetSystemMetrics(SM_CYICON);
  CRect rect;
  GetClientRect(&rect);
  int x = (rect.Width() - cxIcon + 1) / 2;
  int y = (rect.Height() - cyIcon + 1) / 2;

  // Draw the icon
  dc.DrawIcon(x, y, m_hIcon);
 }
 else
 {
  CPaintDC dc(this);
  int loop, loop2;
  for(loop = 0; loop<320; loop++)
  {
   for(loop2 = 0; loop2<240; loop2++)
   {
    int Gray = Bitmap1[loop][loop2];
    dc.SetPixel(loop+13, loop2+13, RGB(Gray, Gray, Gray));
   }
  }

  for(loop = 0; loop<320; loop++)
  {
   for(loop2 = 0; loop2<240; loop2++)
   {
    int Gray = Bitmap2[loop][loop2];
    dc.SetPixel(loop+350, loop2+13, RGB(Gray, Gray, Gray));
   }
  }
```

```
CPen RedPen,WhitePen,GreenPen,YellowPen;
RedPen.CreatePen(PS_SOLID,1,RGB(255,0,0));
WhitePen.CreatePen(PS_SOLID,1,RGB(255,255,255));
GreenPen.CreatePen(PS_SOLID,1,RGB(0,255,0));
YellowPen.CreatePen(PS_SOLID,1,RGB(255,255,0));
CPen* pOriginalPen;


dc.MoveTo(15,480);
dc.LineTo(333,480);
dc.MoveTo(352,480);
dc.LineTo(669,480);

pOriginalPen = dc.SelectObject(&RedPen);
dc.MoveTo(13,480);
for(loop = 0; loop<320; loop++)
{
 dc.LineTo(13+loop, 480-Sum[0][loop]/240);
}

pOriginalPen = dc.SelectObject(&WhitePen);
dc.MoveTo(350,480);
for(loop = 0; loop<320; loop++)
{
 dc.LineTo(350+loop, 480-Sum[1][loop]/240);
}
int Diff;
int MeanLevel = (int)(sqrt(pow(Mean1-Mean2,2)));
if(m_DiffCheck)
{
 pOriginalPen = dc.SelectObject(&GreenPen);
 dc.MoveTo(350,480);
 for(loop = 0; loop<320; loop++)
 {
  Diff = (int)(sqrt(pow(Sum[0][loop]/240-Sum[1][loop]/240,2)));

  //if (Diff<MeanLevel) Diff = MeanLevel + Diff;
  //Diff = (int)(Diff- sqrt(pow(Mean1 - Mean2,2)));
  dc.LineTo(350+loop, 480-Diff);//RGB(255,0,0));
 }
}

if(m_OverlayCheck)
{
 pOriginalPen = dc.SelectObject(&WhitePen);
 dc.MoveTo(13,480);
 for(loop = 0; loop<320; loop++)
 {
  dc.LineTo(13+loop, 480-Sum[1][loop]/240);
 }
 if(m_DiffCheck)
 {
  pOriginalPen = dc.SelectObject(&GreenPen);
  dc.MoveTo(13,480);
  for(loop = 0; loop<320; loop++)
  {
   Diff = (int)(sqrt(pow(Sum[0][loop]/240-Sum[1][loop]/240,2)));
  // Diff = abs(Diff- sqrt(pow(Mean1/240 - Mean2/240,2)));
  // if (Diff<MeanLevel) Diff = MeanLevel + Diff;
   dc.LineTo(13+loop, 480-Diff);//RGB(255,0,0));
  }
 }
}

if(m_Image)
{
 pOriginalPen = dc.SelectObject(&YellowPen);
 int Summed=0;
```

```
    int Diff;
    int Max=0;
    int MaxRef = 0;

    for(loop = 0; loop<320; loop++)
    {
     Diff = (int)(sqrt(pow(Sum[0][loop]/240-Sum[1][loop]/240,2)));
     Summed+= Diff;
     if (Diff>Max)
     {
      Max = Diff;
      MaxRef = loop;
     }

    }
   // if((MaxRef<0)||(MaxRef>320)) MaxRef = 0;
    Summed = Summed/320;

    int Xmin= 0;
    int Xmax = 0;
    loop = MaxRef;
    int Gap = 0;
    do
    {
     Diff = (int)(sqrt(pow(Sum[0][loop]/240-Sum[1][loop]/240,2)));
     if(Diff<MeanLevel) Gap++;
     loop--;
     if(loop==0) Gap = 4;
    }
    while (Gap<4);
    Xmin = loop+4;

    loop = MaxRef;
    Gap =0;
    do
    {
     Diff = (int)(sqrt(pow(Sum[0][loop]/240-Sum[1][loop]/240,2)));
     //if(Diff<Summed) Gap++;
     if(Diff<MeanLevel) Gap++;
     loop++;
     if(loop==320) Gap = 4;
    }
    while (Gap<4);
    Xmax = loop-4;

    dc.MoveTo(350 + Xmin,13);
    dc.LineTo(350 + Xmin,253);
    dc.MoveTo(350 + Xmax,13);
    dc.LineTo(350 + Xmax,253);
    dc.MoveTo(350+Xmin, 270);
    dc.LineTo(350+Xmin, 490);
    dc.MoveTo(350+Xmax,270);
    dc.LineTo(350+Xmax,490);
   }

   CDialog::OnPaint();
  }
}

// The system calls this to obtain the cursor to display while the user drags
//  the minimized window.
HCURSOR CBitmapwavecomparatorDlg::OnQueryDragIcon()
{
 return (HCURSOR) m_hIcon;
}

void CBitmapwavecomparatorDlg::OnB1analyseButton()
{
```

```
 // TODO: Add your control notification handler code here
 int loop, loop2;
 //Zero all values
 for(loop = 0; loop<320; loop++)
 {
  Sum[0][loop] = 0;
 }
 Mean1 = 0;
 for(loop = 0; loop<320; loop++)
 {
  for ( loop2 = 0; loop2<240; loop2++)
  {
    //Add current pixel value to sum
    Sum[0][loop] += Bitmap1[loop][loop2];
  }
  Mean1+=Sum[0][loop]/240;
 }
 Mean1 = Mean1/320;
 InvalidateRect(CRect(13,250,333,500));
}

void CBitmapwavecomparatorDlg::OnB2analyseButton()
{
 // TODO: Add your control notification handler code here
 int loop, loop2;
 for(loop = 0; loop<320; loop++)
 {
  Sum[1][loop] = 0;
 }
 Mean2 = 0;
 for(loop = 0; loop<320; loop++)
 {
  for ( loop2 = 0; loop2<240; loop2++)
  {
    Sum[1][loop] += Bitmap2[loop][loop2];
  }
  Mean2+=Sum[1][loop]/240;
 }
 Mean2 = Mean2/320;
 InvalidateRect(CRect(350,250,670,500));
 if(m_OverlayCheck) InvalidateRect(CRect(13,250,333,500));
}

void CBitmapwavecomparatorDlg::OnBitmap1Button()
{
 // TODO: Add your control notification handler code here
 CFileDialog OpenFile(TRUE,".bmp","*.bmp");
 if(OpenFile.DoModal()==IDOK)
 {
  CString Temp = OpenFile.GetPathName();
  OpenBitmap(0,Temp);
  int loop;

  for ( loop = 0; loop<320; loop++)
  {
    Sum[0][loop] = 0;
  }
  InvalidateRect(CRect(13,13,363,253));
  if(m_AutoCheck) OnB1analyseButton();
 }
}

void CBitmapwavecomparatorDlg::OnBitmap2Button()
{
 // TODO: Add your control notification handler code here
 CFileDialog OpenFile(TRUE,".bmp","*.bmp");
 if(OpenFile.DoModal()==IDOK)
 {
```

```
  CString Temp = OpenFile.GetPathName();
  OpenBitmap(1,Temp);
  int loop;
  for(loop = 0; loop<320; loop++)
  {
   Sum[1][loop] = 0;
  }
  InvalidateRect(CRect(350,13,670,253));
  if(m_AutoCheck) OnB2analyseButton();
 }

}



DWORD ReadLong(fstream InFile)
{
 BYTE a1,a2,a3,a4;
 DWORD ReturnVal;

 InFile >> a1;
 InFile >> a2;
 InFile >> a3;
 InFile >>a4;

 ReturnVal =(DWORD)( a1 + a2*256 + a3*pow(256,2) + a4*pow(256,3));

 return ReturnVal;
}


WORD ReadShort(fstream InFile)
{
 BYTE a1,a2;
 WORD ReturnVal;

 InFile >> a1;
 InFile >> a2;

 ReturnVal = (WORD)(a1 + a2*256);

 return ReturnVal;
}


void CBitmapwavecomparatorDlg::OnQuitButton()
{
 // TODO: Add your control notification handler code here
 OnOK();
}


void CBitmapwavecomparatorDlg::OpenBitmap(BOOL Source, CString FileName)
{
 // TODO: Add your command handler code here
 BeginWaitCursor();

 fstream InFile;
 InFile.open(FileName,ios::binary|ios::in);

 WORD Header;
 Header = ReadShort(InFile);
 if(Header!=(('M'<<8) + 'B')) //Expect to read BM as first two bytes
 {
  EndWaitCursor();
  return;
 }
```

```
DWORD FileSize = ReadLong(InFile);
WORD Res1 = ReadShort(InFile);
WORD Res2 = ReadShort(InFile);
DWORD ImageOffset = ReadLong(InFile);
DWORD HeaderSize = ReadLong(InFile);
DWORD Width = ReadLong(InFile);
DWORD Height = ReadLong(InFile);
if (ReadShort(InFile) !=1)
{
 EndWaitCursor();
 return;
}
WORD ColDepth = ReadShort(InFile);
DWORD Compression = ReadLong(InFile);
DWORD ImageSize = ReadLong(InFile);
DWORD XPelsMeter = ReadLong(InFile);
DWORD YPelsMeter = ReadLong(InFile);
DWORD Colours = ReadLong(InFile);
DWORD ImpColours = ReadLong(InFile);

if(Compression!=0) //Not designed for compressed bitmaps
{
 EndWaitCursor();
 return;
}
if(Width!=320)
{
 EndWaitCursor();
 return;
}
if(Height!=240)  //only for a 320x240 bitmap!
{
 EndWaitCursor();
 return;
}
if(ColDepth!=24)   //only for 24bit bitmaps
{
 EndWaitCursor();
 return;
}

unsigned char PixelR, PixelG, PixelB;
int loop, loop2;


InFile.seekg(ImageOffset,ios::beg);              //set file pointer to start of image data

for(loop =0; loop<240; loop++)
{
 for (loop2 = 0; loop2<320; loop2++)
 {
   PixelB = InFile.get();
   PixelG = InFile.get();
   PixelR = InFile.get();

   if(Source) Bitmap2[loop2][240-loop-1] = (unsigned _int8)((PixelR)*0.3 + (PixelG)*0.59 + (PixelR)
*0.11);
   else Bitmap1[loop2][240-loop-1] = (unsigned _int8)((PixelR)*0.3 + (PixelG)*0.59 + (PixelR)*0.11);

 }
 }
 InFile.close();

 EndWaitCursor();
}


void CBitmapwavecomparatorDlg::OnDiffCheck()
```

```
{
 // TODO: Add your control notification handler code here
 UpdateData(TRUE);
 InvalidateRect(CRect(350,250,670,500));
 InvalidateRect(CRect(13,250,333,500));
}

void CBitmapwavecomparatorDlg::OnOverlayCheck()
{
 // TODO: Add your control notification handler code here
 UpdateData(TRUE);
 InvalidateRect(CRect(13,250,333,500));
}

void CBitmapwavecomparatorDlg::OnAutoCheck()
{
 // TODO: Add your control notification handler code here
 UpdateData(TRUE);
}

void CBitmapwavecomparatorDlg::OnImageCheck()
{
 // TODO: Add your control notification handler code here
 UpdateData(TRUE);
 InvalidateRect(CRect(350,13,670,253));
 InvalidateRect(CRect(350,260,670,500));
}
```

```
// bitmap wave comparatorDlg.h : header file
//

#if !defined
(AFX_BITMAPWAVECOMPARATORDLG_H__CD5D3756_275C_11D4_93BE_0060084F84CD__INCLUDED_)
#define
AFX_BITMAPWAVECOMPARATORDLG_H__CD5D3756_275C_11D4_93BE_0060084F84CD__INCLUDED_

#if _MSC_VER >= 1000
#pragma once
#endif // _MSC_VER >= 1000

/////////////////////////////////////////////////////////////////////////////
// CBitmapwavecomparatorDlg dialog

class CBitmapwavecomparatorDlg : public CDialog
{
// Construction
public:
 int Mean2;
 int Mean1;
 int Sum[2][320];
 unsigned _int8 Bitmap1[320][240];
 unsigned _int8 Bitmap2[320][240];
 void OpenBitmap(BOOL,CString);
 CBitmapwavecomparatorDlg(CWnd* pParent = NULL);          // standard constructor

// Dialog Data
 //{{AFX_DATA(CBitmapwavecomparatorDlg)
 enum { IDD = IDD_BITMAPWAVECOMPARATOR_DIALOG };
 BOOL m_DiffCheck;
 BOOL m_OverlayCheck;
 BOOL m_AutoCheck;
 BOOL m_Image;
 //}}AFX_DATA

 // ClassWizard generated virtual function overrides
 //{{AFX_VIRTUAL(CBitmapwavecomparatorDlg)
 protected:
 virtual void DoDataExchange(CDataExchange* pDX);          // DDX/DDV support
 //}}AFX_VIRTUAL
```

```
// Implementation
protected:
 HICON m_hIcon;

 // Generated message map functions
 //{{AFX_MSG(CBitmapwavecomparatorDlg)
 virtual BOOL OnInitDialog();
 afx_msg void OnSysCommand(UINT nID, LPARAM lParam);
 afx_msg void OnPaint();
 afx_msg HCURSOR OnQueryDragIcon();
 afx_msg void OnB1analyseButton();
 afx_msg void OnB2analyseButton();
 afx_msg void OnBitmap1Button();
 afx_msg void OnBitmap2Button();
 afx_msg void OnQuitButton();
 afx_msg void OnDiffCheck();
 afx_msg void OnOverlayCheck();
 afx_msg void OnAutoCheck();
 afx_msg void OnImageCheck();
 //}}AFX_MSG
 DECLARE_MESSAGE_MAP()
};

//{{AFX_INSERT_LOCATION}}
// Microsoft Developer Studio will insert additional declarations immediately before the previous line.

#endif // !defined
(AFX_BITMAPWAVECOMPARATORDLG_H__CD5D3756_275C_11D4_93BE_0060084F84CD__INCLUDED_)
```

# 20.4 - Neural Demo

```
// Neural Demo 2Dlg.cpp : implementation file
//

#include "stdafx.h"
#include "Neural Demo 2.h"
#include "Neural Demo 2Dlg.h"
#include <fstream.h>
#include <math.h>
#include "m_Dialog1.h"
#include "HELPDIALOG.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

/////////////////////////////////////////////////////////////////////////////
// CAboutDlg dialog used for App About

class CAboutDlg : public CDialog
{
public:
 CAboutDlg();

// Dialog Data
 //{{AFX_DATA(CAboutDlg)
 enum { IDD = IDD_ABOUTBOX };
 //}}AFX_DATA

 // ClassWizard generated virtual function overrides
 //{{AFX_VIRTUAL(CAboutDlg)
 protected:
 virtual void DoDataExchange(CDataExchange* pDX);    // DDX/DDV support
 //}}AFX_VIRTUAL

// Implementation
protected:
 //{{AFX_MSG(CAboutDlg)
 //}}AFX_MSG
 DECLARE_MESSAGE_MAP()
};

CAboutDlg::CAboutDlg() : CDialog(CAboutDlg::IDD)
{
 //{{AFX_DATA_INIT(CAboutDlg)
 //}}AFX_DATA_INIT
}

void CAboutDlg::DoDataExchange(CDataExchange* pDX)
{
 CDialog::DoDataExchange(pDX);
 //{{AFX_DATA_MAP(CAboutDlg)
 //}}AFX_DATA_MAP
}

BEGIN_MESSAGE_MAP(CAboutDlg, CDialog)
 //{{AFX_MSG_MAP(CAboutDlg)
  // No message handlers
 //}}AFX_MSG_MAP
END_MESSAGE_MAP()

/////////////////////////////////////////////////////////////////////////////
```

```
// CNeuralDemo2Dlg dialog

CNeuralDemo2Dlg::CNeuralDemo2Dlg(CWnd* pParent /*=NULL*/)
 : CDialog(CNeuralDemo2Dlg::IDD, pParent)
{
 //{{AFX_DATA_INIT(CNeuralDemo2Dlg)
 m_Objects = 0;
 //}}AFX_DATA_INIT
 // Note that LoadIcon does not require a subsequent DestroyIcon in Win32
 m_hIcon = AfxGetApp()->LoadIcon(IDR_MAINFRAME);
}

void CNeuralDemo2Dlg::DoDataExchange(CDataExchange* pDX)
{
 CDialog::DoDataExchange(pDX);
 //{{AFX_DATA_MAP(CNeuralDemo2Dlg)
 DDX_Text(pDX, IDC_OBJECTS_EDIT, m_Objects);
 //}}AFX_DATA_MAP
}

BEGIN_MESSAGE_MAP(CNeuralDemo2Dlg, CDialog)
 //{{AFX_MSG_MAP(CNeuralDemo2Dlg)
 ON_WM_SYSCOMMAND()
 ON_WM_PAINT()
 ON_WM_QUERYDRAGICON()
 ON_COMMAND(ID_MENU_ABOUT, OnMenuAbout)
 ON_COMMAND(ID_MENU_BOUNDARIES, OnMenuBoundaries)
 ON_COMMAND(ID_MENU_EXIT, OnMenuExit)
 ON_COMMAND(ID_MENU_FILTER, OnMenuFilter)
 ON_COMMAND(ID_MENU_HELP, OnMenuHelp)
 ON_COMMAND(ID_MENU_NETWORK, OnMenuNetwork)
 ON_COMMAND(ID_MENU_OPEN, OnMenuOpen)
 //}}AFX_MSG_MAP
END_MESSAGE_MAP()

/////////////////////////////////////////////////////////////////////////////
// CNeuralDemo2Dlg message handlers

//**************************************************************************



int Stored[320][240];
CString Title;
int sharp[80][60];
int counter;
int Pos = 4;
int Draw1=0,Draw2=0;
float NetOut=0;
float NetInput[5];

struct Object
{
 int Xmin, Xmax;
 int Xmini[60], Xmaxi[60];
 int Ymin, Ymax;
};


Object Box[100];


BOOL CNeuralDemo2Dlg::OnInitDialog()
{
 CDialog::OnInitDialog();
```

```
// Add "About..." menu item to system menu.

// IDM_ABOUTBOX must be in the system command range.
ASSERT((IDM_ABOUTBOX & 0xFFF0) == IDM_ABOUTBOX);
ASSERT(IDM_ABOUTBOX < 0xF000);

CMenu* pSysMenu = GetSystemMenu(FALSE);
if (pSysMenu != NULL)
{
 CString strAboutMenu;
 strAboutMenu.LoadString(IDS_ABOUTBOX);
 if (!strAboutMenu.IsEmpty())
 {
   pSysMenu->AppendMenu(MF_SEPARATOR);
   pSysMenu->AppendMenu(MF_STRING, IDM_ABOUTBOX, strAboutMenu);
 }
}

// Set the icon for this dialog.  The framework does this automatically
//  when the application's main window is not a dialog
SetIcon(m_hIcon, TRUE);                                    // Set big icon
SetIcon(m_hIcon, FALSE);                    // Set small icon

// TODO: Add extra initialization here

return TRUE;  // return TRUE  unless you set the focus to a control
}

void CNeuralDemo2Dlg::OnSysCommand(UINT nID, LPARAM lParam)
{
 if ((nID & 0xFFF0) == IDM_ABOUTBOX)
 {
  CAboutDlg dlgAbout;
  dlgAbout.DoModal();
 }
 else
 {
  CDialog::OnSysCommand(nID, lParam);
 }
}

// If you add a minimize button to your dialog, you will need the code below
//  to draw the icon.  For MFC applications using the document/view model,
//  this is automatically done for you by the framework.

void CNeuralDemo2Dlg::OnPaint()
{
 if (IsIconic())
 {
  CPaintDC dc(this); // device context for painting

  SendMessage(WM_ICONERASEBKGND, (WPARAM) dc.GetSafeHdc(), 0);

  // Center icon in client rectangle
  int cxIcon = GetSystemMetrics(SM_CXICON);
  int cyIcon = GetSystemMetrics(SM_CYICON);
  CRect rect;
  GetClientRect(&rect);
  int x = (rect.Width() - cxIcon + 1) / 2;
  int y = (rect.Height() - cyIcon + 1) / 2;

  // Draw the icon
  dc.DrawIcon(x, y, m_hIcon);
 }
 else
 {
  CPaintDC dc(this);
  int loop, loop2;
```

```
 if (Draw1)
 {
  for(loop = 0; loop<320; loop++)
  {
   for (loop2 = 0; loop2<240; loop2++)
   {
    if(Stored[loop][loop2])
      SetPixel(dc,loop+12,loop2+18,RGB(0,100,200));
    else SetPixel(dc,loop+12,loop2+18,RGB(0,0,0));
   }
  }
 }
 if (Draw2)
 {
  for (loop = 1; loop<=counter; loop++)
  {
   CPen BoxPen;
   BoxPen.CreatePen(PS_SOLID,1,RGB(255,255,0));
   CPen* pOriginalPen;
   pOriginalPen = dc.SelectObject(&BoxPen);
   MoveToEx(dc,4*Box[loop].Xmin+12, 4*Box[loop].Ymin+18,NULL);
   LineTo(dc, 4*Box[loop].Xmax+16, 4*Box[loop].Ymin+18);
   LineTo(dc, 4*Box[loop].Xmax+16, 4*Box[loop].Ymax+22);
   LineTo(dc, 4*Box[loop].Xmin+12, 4*Box[loop].Ymax+22);
   LineTo(dc, 4*Box[loop].Xmin+12, 4*Box[loop].Ymin+18);
  }
 }


 CDialog::OnPaint();
 }
}

// The system calls this to obtain the cursor to display while the user drags
//  the minimized window.
HCURSOR CNeuralDemo2Dlg::OnQueryDragIcon()
{
 return (HCURSOR) m_hIcon;
}

void CNeuralDemo2Dlg::OnMenuOpen()
{
 // TODO: Add your command handler code here
 char FileName[500];
 char FileTitle[100];
 int count = 0;
 int loop, loop2;

 for ( loop=0; loop < 320; loop++)
 {
  for ( loop2 = 0; loop2 < 240 ; loop2++)
  {
   Stored[loop][loop2]=0;
  }
 }
 Draw1 = 0;
 Draw2 = 0;
 UpdateData(FALSE);
 InvalidateRect(CRect(12,18,332,258), FALSE);


 OPENFILENAME ofn;
 ofn.lStructSize = sizeof(OPENFILENAME);

 ofn.hwndOwner = m_hWnd;;
 ofn.hInstance = NULL;

 ofn.lpstrFilter = TEXT("VosDemo files *.vos\0*.vos\0\0");
```

```
 ofn.lpstrCustomFilter = NULL;
 ofn.nMaxCustFilter = 0;
 ofn.nFilterIndex = 1;
 ofn.lpstrFile = FileName;
 ofn.nMaxFile = 500;
 ofn.lpstrFileTitle = FileTitle;
 ofn.nMaxFileTitle = 99;
 ofn.lpstrInitialDir = NULL;
 ofn.lpstrTitle = "Open VOS file";
 ofn.Flags = OFN_FILEMUSTEXIST;
 ofn.lpstrDefExt = "BMP";
 ofn.lCustData = NULL;
 ofn.lpfnHook = NULL;
 ofn.lpTemplateName = NULL;

 FileName[0] = '\0';

 if (GetOpenFileName(&ofn))
 {

  ifstream file_in(FileName);
  int value1;
  int value2;

  do
  {
   file_in >> value1;
   file_in >> value2;

   Stored[value1][value2] = 1;
   count++;
  }
  while ( file_in.eof() == 0);
  file_in.close();

  for (loop = 0; loop <320; loop++)
  {
   for (loop2 = 0; loop2 <240; loop2++)
   {
    if (Stored[loop][loop2] != 1)
    {
     Stored[loop][loop2] = 0;
    }
   }
  }
  Draw1 = 1;
  Pos = 6 + (int)(count/5000);
  InvalidateRect(CRect(12,18,332,258), FALSE);
 }
}




void CNeuralDemo2Dlg::OnMenuFilter()
{
 // TODO: Add your command handler code here
 m_Dialog1 m_dlg;
 m_dlg.m_Value = Pos;
 m_dlg.DoModal();
 Pos = m_dlg.m_Value;
}
```

```
void CNeuralDemo2Dlg::OnMenuBoundaries()
{
 // TODO: Add your command handler code here
 int loop, loop2,loop3,loop4;
 int count = 0;
 int Mem = 0;
 int gap = 0, Set = 0;
 //OnClearButton();

 for (loop2=0; loop2<60; loop2++)
 {
  for ( loop=0; loop<80; loop++)
  {
    sharp[loop][loop2] = 0;
    count = 0;
    for (loop3=0; loop3<4; loop3++)
    {
     for ( loop4 = 0; loop4<4; loop4++)
     {
      if (Stored[4*loop+loop3][4*loop2+loop4]==1)
        count++;
     }
    }
    if (count >Pos)
    {
     sharp[loop][loop2] = 1;
    }
  }
 }
 counter = 0;

 for (loop = 0; loop<100; loop++)
 {
  Box[loop].Xmin = 80;
  Box[loop].Ymin = 60;
  Box[loop].Xmax = 0;
  Box[loop].Ymax = 0;
 }

 //Check every line in the image
 loop=0;
 for(loop2=0; loop2<60; loop2++)
 {
  Set = 0;
  Mem = 0;
  gap = 0;

  for (loop=0; loop<80; loop++)
  {
    if (sharp[loop][loop2]==1)
    {
     counter++;
     Box[counter].Xmin = loop;
     Box[counter].Ymin = loop2;
     Box[counter].Ymax = loop2;
     Box[counter].Xmax = loop;
     Box[counter].Xmini[loop2] = loop;
     Box[counter].Xmaxi[loop2] = loop;

     //Previous line check for position matching
     for(loop3 = 0; loop3<counter; loop3++)
     {
      if((loop>=(Box[loop3].Xmin-2))&&(loop<=(Box[loop3].Xmax+2))&&
        ((loop2-Box[loop3].Ymax)<3)&&(loop3!=counter)&&
        (loop2>0))
      {
        Set = 1;
```

```
      Mem = loop3;
     }
    }
    //End of line check

    while((gap<3)&&(loop<80))
    {
     loop++;
     if(sharp[loop][loop2]==1)
     {
      gap = 0;
      Box[counter].Xmax = loop;
      Box[counter].Xmaxi[loop2] = loop;

      //Previous line check for position matching
      for(loop3 = 0; loop3<counter; loop3++)
      {
       if((loop>=Box[loop3].Xmin-2)&&(loop<=Box[loop3].Xmax+2)
        &&(loop2-Box[loop3].Ymax<3)&&(loop3!=counter)&&(counter>0))
       {
        Set = 1;
        Mem = loop3;
       }
      }
      //End of line check
     }
     else gap++;
    }
    gap = 0;

    //Matching correction code, updates matched object
    //and deleted new object created
    if (Set==1)
    {
     if(Box[counter].Xmin<Box[Mem].Xmin) Box[Mem].Xmin = Box[counter].Xmin;
     if(Box[counter].Xmax>Box[Mem].Xmax) Box[Mem].Xmax = Box[counter].Xmax;
     Box[Mem].Ymax = loop2;
     Box[counter].Xmin = 80;
     Box[counter].Xmax = 0;
     Box[counter].Ymin = 60;
     Box[counter].Ymax = 0;
     Box[Mem].Xmaxi[loop2] = Box[counter].Xmaxi[loop2];
     Box[Mem].Xmini[loop2] = Box[counter].Xmini[loop2];
     counter--;
     Set = 0;
     Mem = 0;
    }
   }
  }
  loop=0;
 }
 //Box size filtering code, checking for minimum target size

 int IsZero = 0;

 for ( loop = 1; loop<=counter; loop++)
 {
  if (IsZero==1)
  {
   loop--;
   IsZero = 0;
  }
  int TArea = ((Box[loop].Xmax - Box[loop].Xmin)*(Box[loop].Ymax-Box[loop].Ymin));
  if (TArea <74)
  {
   for (int loop1 = loop; loop1<=counter; loop1++)
   {
    Box[loop1] = Box[loop1+1];
```

```
    if (loop==1) IsZero = 1;
    else loop--;
   }
   counter--;
  }
 }

 m_Objects = counter;
 UpdateData(FALSE);
 Draw2 = 1;
 InvalidateRect(CRect(12,18,333,259), FALSE);
}


void CNeuralDemo2Dlg::OnMenuNetwork()
{
 // TODO: Add your command handler code here

 int value1;
 int value2;
 int Xmax, Xmin, Ymax, Ymin;
 int Xtemp=0, Ytemp=0, m_Area=0, m_X=0, m_Y=0;
 int loop, loop2, loop3, loop4;

 int Width=0, Height=0,Weighting=0,BoxArea=0;
 int SegWidth=0, SegHeight=0,SegSet[4][6],SegAcc=0,SegArea=0, x_max=0, y_max=0;
 int TempWidth=0, TempHeight=0;
 int CountX=0, CountY=0;


 //************** NECESSARY VARIABLE INITIALISATIONS FOR BATCH PROCESS ********

  value1 = 0;
  value2 = 0;

  Xtemp=0;
  Ytemp=0;
  m_Area=0;
  m_X=0;
  m_Y=0;

 //************* START OF CALCULATIONS ****************************************

 for ( loop = 1; loop <= counter; loop++)
 {
  //******** CALCULATE BOX WIDTH< HEIGHT & AREA *********


  Width = 0;
  Height = 0;
  Weighting = 0;
  BoxArea = 0;

  Xmin = 4*Box[loop].Xmin;
  Xmax = 4*Box[loop].Xmax;
  Ymin = 4*Box[loop].Ymin;
  Ymax = 4*Box[loop].Ymax;

  Width = Xmax - Xmin;
  Height = Ymax - Ymin;
  BoxArea = Width * Height;

  //****************** C of G Calculation ************
  //checks if both the Stored value and the sharp value is set
  //then checks that the pixel is within the Xmin - Xmax range for the
  //particular line and object.
```

```
int Dist1=0, Dist2=0, Xacc=0, Yacc=0, loop1, TotArea=0, Yval=1;

for (loop1= Xmin; loop1<=Xmax+4; loop1++)
{
 for (loop2 = Ymin; loop2 <=Ymax+4; loop2++)
 {
  if (((Stored[loop1][loop2]==1) && (sharp[loop1/4][loop2/4]==1))
   && ((loop1>= 4* Box[loop].Xmini[loop2/4]) && (loop1 <= (4+4*Box[loop].Xmaxi[loop2/4]))))
  {
   TotArea++;
   Xacc += Dist1;
   Yacc += Dist2;
  }
  Dist1++;
  Yval++;
 }
 Dist1 = 0;
 Yval = 1;
 Dist2++;
}

div_t Xpos, Ypos;
Xpos = div(Yacc, TotArea);
Ypos = div(Xacc, TotArea);

if (Xpos.rem>=(TotArea/2)) Xpos.quot++;
if (Ypos.rem>=(TotArea/2)) Ypos.quot++;


m_X = Xpos.quot + Xmin;
m_Y = Ypos.quot + Ymin;


//*************** END OF C of G CALCULATION ************

//*************** SEGMENT AREA CALC ********************
//The set rectangle will be split up into an 3*4 array of equal
//segments, on which % set pixels calcs are carried out.

SegWidth=0;
SegHeight=0;
SegAcc=0;
SegArea=0;
x_max=0;
y_max=0;
TempWidth=0;
TempHeight=0;

div_t h,w;
h = div(Width, 4);
w = div(Height, 6);
SegWidth = h.quot;
SegHeight = w.quot;

SegArea = SegWidth * SegHeight;


for ( loop1 = 0; loop1 <4; loop1++)
{
 for ( loop2 = 0; loop2<6; loop2++)
 {
  x_max = (loop1 + 1) * SegWidth;
  y_max = (loop2 + 1) * SegHeight;
  if ( loop1 ==3)
  {
   x_max = Width;
   TempWidth = Width - (3 * SegWidth);
```

```
    }
    else TempWidth = SegWidth;
    if (loop2 ==5)
    {
     y_max = Height;
     TempHeight = Height - (5 * SegHeight);
    }
    else TempHeight = SegHeight;
    SegAcc = 0;
    for (loop3=(loop1*SegWidth); loop3<x_max;loop3++)
    {
     for (loop4=(loop2*SegHeight); loop4<y_max;loop4++)
     {
       if ((Stored[Xmin+loop3][Ymin+loop4]==1)
        && (sharp[(Xmin +loop3)/4][(Ymin + loop4)/4]==1))
         //&& ( loop1>= 4*Box[loop].Xmini[loop])
          //&& ( loop1 <= 4*Box[loop].Xmaxi[loop]))
       {
        SegAcc++;
       }
     }
    }
    SegSet[loop1][loop2] = 0;
    SegSet[loop1][loop2]=(int)((100 * SegAcc)/(TempWidth * TempHeight));

    SegAcc=0;
   }
  }

//***************** END OF SEGMENT AREA CALCULATION ****************


m_Area = (TotArea*100/BoxArea);
for (loop4 = 0; loop4<5; loop4++)
{

  NetInput[loop4] = 0;
}


NetInput[0] = (float)(pow(((Ymax-Ymin)*(-0.00880381)+2.86238),2)*0.351784 - 1.343);
NetInput[1] = (float)(pow((m_Area*(-0.0377996)+3.71819),2)*0.224719 - 1.0225);
NetInput[2] = (float)(tanh((SegArea*0.00110947)+1.52922)*28.2527 - 26.875);
NetInput[3] = (float)(log((SegSet[1][5]*0.156775)+3.46737)*1.17052 - 2.456);
NetInput[4] = (float)(log((SegSet[2][5]*0.156775)+3.15382)*1.14088 - 2.311);



for (loop4 = 0; loop4<5; loop4++)
{
  NetInput[loop4] = (float)((NetInput[loop4]+1)*.5);
}

//NETWORK CODE HERE
float Xsum7 = 0;
float Xout7 = 0;
NetOut = 0;
/* Generating code for PE 0 in layer <Hidden1> (3) */
   Xsum7  = (float)(9.4248371 + (-9.7122078) * NetInput[0] + (-0.82052672) * NetInput[1] +

    (-4.1746411) * NetInput[2] + (-1.6331787) * NetInput[3] + (-2.0222914) * NetInput[4]);
   /* Generating code for PE 0 in layer <Hidden1> (3) */
 Xout7 = (float)(tanh( Xsum7 ));


    NetOut  = (float)((0.15872838) + (0.04405418) * NetInput[0] + (-0.12671886) * NetInput[1] +
     (-0.21494821) * NetInput[2] + (-0.029009042) * NetInput[3] + (-0.015902856) * NetInput[4] +
(0.91813892) * Xout7);
```

```
    /* De-scale and write output from network */
  NetOut = (float)(NetOut * (0.625) + (0.5));

  //END OF NETWORK CODE


  int Temp = (int)(100*NetOut);
  if (Temp>100) Temp = 100;
  if (Temp<0) Temp = 0;

  char Text[5];
  //Text[] = "";
  itoa(Temp,Text,10);
  CClientDC dc(this);
  dc.DrawText(Text,CRect(m_X,m_Y,m_X+30,m_Y+15),NULL);

  UpdateData(FALSE);
 }




}

void CNeuralDemo2Dlg::OnMenuExit()
{
 // TODO: Add your command handler code here
 OnOK();
}


void CNeuralDemo2Dlg::OnMenuHelp()
{
 HELPDIALOG m_dlg;
 m_dlg.DoModal();
}


void CNeuralDemo2Dlg::OnMenuAbout()
{
 CAboutDlg m_dlg;
 m_dlg.DoModal();
}
```

# 20.5 - Chloride Demo

```
// Chloride DemoDlg.cpp : implementation file
//

#include "stdafx.h"
#include "Chloride Demo.h"
#include "Chloride DemoDlg.h"
#include <math.h>
#include "ViewDlg.h"
#include <time.h>
#include <direct.h>


#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

/////////////////////////////////////////////////////////////////////////////
// CAboutDlg dialog used for App About

struct BitmapStruct
{
 CString Name;
 unsigned _int8 Gray[320][240];                    //hold bitmap grayscale info
 DWORD Sum;
 int ThresholdLevel;//keeps track of the total image intensity
        //divide by 76800 to get the grayscale median
 DWORD FileSize;
 DWORD Width, Height;          //Width and Height of the Bitmap in Pixels
 DWORD Compression;                              //Specifies the type of compression used
 DWORD ImageOffset;                                                          //Number of bytes from
start of file to image data
 WORD ColDepth;  //type of Bitmap
 DWORD Colours;
 DWORD ImpColours;
};


struct ImageObject
{
 unsigned short int Xmin;
 unsigned short int Xmax;
 unsigned short int Ymin;
 unsigned short int Ymax;
 unsigned short int Xmini[60];
 unsigned short int Xmaxi[60];

 float NetInput[5];
 unsigned _int8 Result;

 int SegArea[4][6];
};


BitmapStruct BitmapDatum, BitmapCamera;
BOOL Difference[320][240];
BOOL Sharp[80][60];
int NetworkThreshold;
ImageObject Box[100];
```

```
class CAboutDlg : public CDialog
{
public:
 CAboutDlg();

// Dialog Data
 //{{AFX_DATA(CAboutDlg)
 enum { IDD = IDD_ABOUTBOX };
 //}}AFX_DATA

 // ClassWizard generated virtual function overrides
 //{{AFX_VIRTUAL(CAboutDlg)
 protected:
 virtual void DoDataExchange(CDataExchange* pDX);    // DDX/DDV support
 //}}AFX_VIRTUAL

// Implementation
protected:
 //{{AFX_MSG(CAboutDlg)
 //}}AFX_MSG
 DECLARE_MESSAGE_MAP()
};

CAboutDlg::CAboutDlg() : CDialog(CAboutDlg::IDD)
{
 //{{AFX_DATA_INIT(CAboutDlg)
 //}}AFX_DATA_INIT
}

void CAboutDlg::DoDataExchange(CDataExchange* pDX)
{
 CDialog::DoDataExchange(pDX);
 //{{AFX_DATA_MAP(CAboutDlg)
 //}}AFX_DATA_MAP
}

BEGIN_MESSAGE_MAP(CAboutDlg, CDialog)
 //{{AFX_MSG_MAP(CAboutDlg)
  // No message handlers
 //}}AFX_MSG_MAP
END_MESSAGE_MAP()

////////////////////////////////////////////////////////////////////////////
// CChlorideDemoDlg dialog

CChlorideDemoDlg::CChlorideDemoDlg(CWnd* pParent /*=NULL*/)
 : CDialog(CChlorideDemoDlg::IDD, pParent)
{
 //{{AFX_DATA_INIT(CChlorideDemoDlg)
 m_DatumName = _T("");
 m_CameraName = _T("");
 m_Detail = _T("");
 m_Batch = FALSE;
 m_Savew = _T("");
 //}}AFX_DATA_INIT
 // Note that LoadIcon does not require a subsequent DestroyIcon in Win32
 m_hIcon = AfxGetApp()->LoadIcon(IDR_MAINFRAME);
}

void CChlorideDemoDlg::DoDataExchange(CDataExchange* pDX)
{
 CDialog::DoDataExchange(pDX);
 //{{AFX_DATA_MAP(CChlorideDemoDlg)
 DDX_Text(pDX, IDC_DATUM_EDIT, m_DatumName);
 DDX_Text(pDX, IDC_CAMERA_EDIT, m_CameraName);
 DDX_Text(pDX, IDC_DETAIL_EDIT, m_Detail);
 DDX_Check(pDX, IDC_BATCH_CHECK, m_Batch);
```

```
 DDX_Text(pDX, IDC_SAVE_EDIT, m_Savew);
 //}}AFX_DATA_MAP
}

BEGIN_MESSAGE_MAP(CChlorideDemoDlg, CDialog)
 //{{AFX_MSG_MAP(CChlorideDemoDlg)
 ON_WM_SYSCOMMAND()
 ON_WM_PAINT()
 ON_WM_QUERYDRAGICON()
 ON_COMMAND(MENU_FILE_EXIT, OnFileExit)
 ON_COMMAND(MENU_OPEN_DATUM, OnOpenDatum)
 ON_COMMAND(MENU_FILE_OPENCAMERA, OnFileOpencamera)
 ON_COMMAND(ID_CALCULATIONS_IMAGEDIFFERENCE, OnCalculationsImagedifference)
 ON_COMMAND(ID_SAVE_SAVEDIFFERENCE, OnSaveSavedifference)
 ON_COMMAND(ID_SAVE_SAVESHARP, OnSaveSavesharp)
 ON_COMMAND(ID_CALCULATIONS_NOISEFILTERING, OnCalculationsNoisefiltering)
 ON_COMMAND(ID_CALCULATIONS_OBJECTBOUNDARIES, OnCalculationsObjectboundaries)
 ON_COMMAND(ID_CALCULATIONS_NETWORKANALYSIS, OnCalculationsNetworkanalysis)
 ON_COMMAND(ID_SAVE_SAVEOBJECTS, OnSaveSaveobjects)
 ON_COMMAND(ID_SAVE_SAVEVDFSPEROBJECTINFO, OnSaveSavevdfsperobjectinfo)
 ON_COMMAND(ID_SETTINGS_VIEWOPTIONS, OnSettingsViewoptions)
 ON_BN_CLICKED(IDC_START_BUTTON, OnStartBatchButton)
 ON_BN_CLICKED(IDC_BATCH_CHECK, OnBatchCheck)
 ON_BN_CLICKED(IDC_NONE_RADIO, OnNoneRadio)
 ON_BN_CLICKED(IDC_ALL_RADIO, OnAllRadio)
 ON_BN_CLICKED(IDC_PROMPT_RADIO, OnPromptRadio)
 ON_BN_CLICKED(IDC_SAVE_BUTTON, OnSaveButton)
 ON_COMMAND(MENU_HELP_ABOUT, OnHelpAbout)
 //}}AFX_MSG_MAP
END_MESSAGE_MAP()

/////////////////////////////////////////////////////////////////////////////
// CChlorideDemoDlg message handlers

BOOL CChlorideDemoDlg::OnInitDialog()
{
 CDialog::OnInitDialog();

 // Add "About..." menu item to system menu.

 // IDM_ABOUTBOX must be in the system command range.
 ASSERT((IDM_ABOUTBOX & 0xFFF0) == IDM_ABOUTBOX);
 ASSERT(IDM_ABOUTBOX < 0xF000);

 CMenu* pSysMenu = GetSystemMenu(FALSE);
 if (pSysMenu != NULL)
 {
  CString strAboutMenu;
  strAboutMenu.LoadString(IDS_ABOUTBOX);
  if (!strAboutMenu.IsEmpty())
  {
   pSysMenu->AppendMenu(MF_SEPARATOR);
   pSysMenu->AppendMenu(MF_STRING, IDM_ABOUTBOX, strAboutMenu);
  }
 }

 // Set the icon for this dialog.  The framework does this automatically
 //  when the application's main window is not a dialog
 SetIcon(m_hIcon, TRUE);                                    // Set big icon
 SetIcon(m_hIcon, FALSE);                          // Set small icon

 // TODO: Add extra initialization here

 ShowDiff = FALSE;
 ShowSharp = FALSE;
 ShowBox = FALSE;
 ShowNet = FALSE;
 NumBoxes = 0;
```

```
 BatchNone = TRUE;
 BatchPrompt = FALSE;
 BatchAll = FALSE;


 return TRUE;  // return TRUE  unless you set the focus to a control
}

void CChlorideDemoDlg::OnSysCommand(UINT nID, LPARAM lParam)
{
 if ((nID & 0xFFF0) == IDM_ABOUTBOX)
 {
  CAboutDlg dlgAbout;
  dlgAbout.DoModal();
 }
 else
 {
  CDialog::OnSysCommand(nID, lParam);
 }
}

// If you add a minimize button to your dialog, you will need the code below
//  to draw the icon.  For MFC applications using the document/view model,
//  this is automatically done for you by the framework.

void CChlorideDemoDlg::OnPaint()
{
 if (IsIconic())
 {
  CPaintDC dc(this); // device context for painting

  SendMessage(WM_ICONERASEBKGND, (WPARAM) dc.GetSafeHdc(), 0);

  // Center icon in client rectangle
  int cxIcon = GetSystemMetrics(SM_CXICON);
  int cyIcon = GetSystemMetrics(SM_CYICON);
  CRect rect;
  GetClientRect(&rect);
  int x = (rect.Width() - cxIcon + 1) / 2;
  int y = (rect.Height() - cyIcon + 1) / 2;

  // Draw the icon
  dc.DrawIcon(x, y, m_hIcon);
 }
 else
 {
  CPaintDC dc(this);

  int loop, loop2;

  CPen SharpPen;
  SharpPen.CreatePen(PS_SOLID,1,RGB(255,0,255));

  //draw Datum and Camera images
  for(loop2 = 0; loop2<240; loop2++)
  {
   for(loop = 0; loop<320; loop++)
   {
    int val = 256-BitmapCamera.Gray[loop][loop2];
    dc.SetPixel(15+loop, 15+loop2,RGB(val,val,val));//draw Camera
    val = 256-BitmapDatum.Gray[loop][loop2];
    dc.SetPixel(350+loop/2, 15+loop2/2,RGB(val,val,val));//draw Datum
    if (ShowDiff)
     if(Difference[loop][loop2])
     dc.SetPixel(15+loop, 15+loop2,RGB(255,255,0));          //draw Difference

   }
  }
```

```
  if(ShowSharp)
  {
   CPen* pOriginalPen;
   pOriginalPen = dc.SelectObject(&SharpPen);
   for (loop2 = 0; loop2<60; loop2++)
   {
    for(loop = 0; loop<80; loop++)
    {
     if(Sharp[loop][loop2]==TRUE)
       Rectangle(dc,(15+loop*4),(15+loop2*4),(19+loop*4),(19+loop2*4));
    }
   }
  }

  if ((ShowBox==TRUE)||(ShowNet==TRUE))
  {
   for (loop = 1; loop<=NumBoxes; loop++)
   {
    CPen BoxPen;
    BoxPen.CreatePen(PS_SOLID,2,RGB(0,255,0));
    CPen* pOriginalPen;
    pOriginalPen = dc.SelectObject(&BoxPen);
    dc.SetBkColor(RGB(0,0,0));
    if(ShowBox)
    {
     dc.SetBkMode(TRANSPARENT);
     dc.SetTextColor(RGB(0,255,0));
     char Text[2];
     itoa(loop,Text,10);
     MoveToEx(dc,4*Box[loop].Xmin+15, 4*Box[loop].Ymin+15,NULL);
     LineTo(dc, 4*Box[loop].Xmax+19, 4*Box[loop].Ymin+15);
     LineTo(dc, 4*Box[loop].Xmax+19, 4*Box[loop].Ymax+19);
     LineTo(dc, 4*Box[loop].Xmin+15, 4*Box[loop].Ymax+19);
     LineTo(dc, 4*Box[loop].Xmin+15, 4*Box[loop].Ymin+15);
     dc.DrawText(Text,CRect(4*Box[loop].Xmin+15,4*Box[loop].Ymin+15,4*Box[loop].Xmin+20,4*Box
[loop].Ymin+20),DT_NOCLIP);
    }
    if(ShowNet)
    {
     dc.SetBkMode(OPAQUE);
     dc.SetTextColor(RGB(255,0,255));
     char Text[5];
     itoa(Box[loop].Result,Text,10);
     int m_X = 4*Box[loop].Xmax+15;
     int m_Y = 4*Box[loop].Ymin+15;
     dc.DrawText(Text,CRect(m_X-10,m_Y,m_X,m_Y+15),DT_NOCLIP);
    }
   }
  }


  CDialog::OnPaint();
 }
}

// The system calls this to obtain the cursor to display while the user drags
//  the minimized window.
HCURSOR CChlorideDemoDlg::OnQueryDragIcon()
{
 return (HCURSOR) m_hIcon;
}

void CChlorideDemoDlg::OnFileExit()
{
 // TODO: Add your command handler code here
 OnOK();
```

```
}


////////////////////////////////////////////////////////////////////
/*********************************************************************/
/**************** FILE OPENING OPERATIONS ************************/
/*********************************************************************/
////////////////////////////////////////////////////////////////////


DWORD ReadLong(fstream InFile)
{
 BYTE a1,a2,a3,a4;
 DWORD ReturnVal;

 InFile >> a1;
 InFile >> a2;
 InFile >> a3;
 InFile >>a4;

 ReturnVal =(DWORD)( a1 + a2*256 + a3*pow(256,2) + a4*pow(256,3));

 return ReturnVal;
}

WORD ReadShort(fstream InFile)
{
 BYTE a1,a2;
 WORD ReturnVal;

 InFile >> a1;
 InFile >> a2;

 ReturnVal = (WORD)(a1 + a2*256);

 return ReturnVal;
}


void CChlorideDemoDlg::OnOpenDatum()
{
 // TODO: Add your command handler code here
 CFileDialog FileDlg(TRUE,"bmp","*.bmp");
 if(FileDlg.DoModal()==IDCANCEL) return;
 BeginWaitCursor();
 m_DatumName = FileDlg.GetPathName();
 m_Detail = "";
 UpdateData(FALSE);

 BitmapDatum.Name = FileDlg.GetFileName();
 int Pos = m_DatumName.Find(BitmapDatum.Name);
 m_DatumDirectory = m_DatumName.Left(Pos);

 fstream InFile;
 InFile.open(m_DatumName,ios::binary|ios::in);

 WORD Header;
 Header = ReadShort(InFile);
 if(Header!=(('M'<<8) + 'B')) //Expect to read BM as first two bytes
 {
  m_Detail = "File Header incorrect";
  UpdateData(FALSE);
  EndWaitCursor();
  return;
 }
```

```
BitmapDatum.FileSize = ReadLong(InFile);
WORD Res1 = ReadShort(InFile);
WORD Res2 = ReadShort(InFile);
BitmapDatum.ImageOffset = ReadLong(InFile);
DWORD HeaderSize = ReadLong(InFile);
BitmapDatum.Width = ReadLong(InFile);
BitmapDatum.Height = ReadLong(InFile);
if (ReadShort(InFile) !=1)
{
 m_Detail = "File has more than one colour plane";
 UpdateData(FALSE);
 EndWaitCursor();
 return;
}
BitmapDatum.ColDepth = ReadShort(InFile);
BitmapDatum.Compression = ReadLong(InFile);
DWORD ImageSize = ReadLong(InFile);
DWORD XPelsMeter = ReadLong(InFile);
DWORD YPelsMeter = ReadLong(InFile);
BitmapDatum.Colours = ReadLong(InFile);
BitmapDatum.ImpColours = ReadLong(InFile);

if(BitmapDatum.Compression!=0)          //Not designed for compressed bitmaps
{
 m_Detail = "Bitmap File is Compressed";
 UpdateData(FALSE);
 EndWaitCursor();
 return;
}
if(BitmapDatum.Width!=320)
{
 m_Detail = "Image Width is not 320 pixels";
 UpdateData(FALSE);
 EndWaitCursor();
 return;
}
if(BitmapDatum.Height!=240)             //only for a 320x240 bitmap!
{
 m_Detail = "File's Height is not 240 pixels";
 UpdateData(FALSE);
 EndWaitCursor();
 return;
}
if(BitmapDatum.ColDepth!=24)                    //only for 24bit bitmaps
{
 m_Detail = "File is not a 24bit RGB image";
 UpdateData(FALSE);
 EndWaitCursor();
 return;
}

unsigned char PixelR, PixelG, PixelB;
int loop, loop2;

BitmapDatum.Sum = 0;
InFile.seekg(BitmapDatum.ImageOffset,ios::beg);         //set file pointer to start of image data

for(loop =0; loop<240; loop++)
{
 for (loop2 = 0; loop2<320; loop2++)
 {
   PixelB = InFile.get();
   PixelG = InFile.get();
   PixelR = InFile.get();

   BitmapDatum.Gray[loop2][240-loop-1] = (unsigned _int8)((255-PixelR)*0.3 + (255-PixelG)*0.59 +
(255-PixelR)*0.11);
   BitmapDatum.Sum += BitmapDatum.Gray[loop2][240-loop-1];
```

```
    Difference[loop2][loop] = FALSE;
    Sharp[loop2/4][loop/4] = FALSE;
   }
  }
  InFile.close();

  NumBoxes = 0;

  //Finished reading the bitmap file, now calculate threshold
  //and threshold version of image map

  BitmapDatum.ThresholdLevel = (int)(BitmapDatum.Sum/76800);
  EndWaitCursor();
  Invalidate();
}


void CChlorideDemoDlg::OnFileOpencamera()
{
  // TODO: Add your command handler code here
  CFileDialog FileDlg(TRUE,"bmp","*.bmp");
  if(FileDlg.DoModal()==IDCANCEL) return;
  BeginWaitCursor();
  m_CameraName = FileDlg.GetPathName();
  m_Detail = "";
  UpdateData(FALSE);

  BitmapCamera.Name = m_CameraName;
  clock_t a = clock();


  fstream InFile;
  InFile.open(m_CameraName,ios::binary|ios::in);

  WORD Header;
  Header = ReadShort(InFile);
  if(Header!=(('M'<<8) + 'B')) //Expect to read BM as first two bytes
  {
   m_Detail = "File Header incorrect";
   UpdateData(FALSE);
   EndWaitCursor();
   return;
  }
  BitmapCamera.FileSize = ReadLong(InFile);
  WORD Res1 = ReadShort(InFile);
  WORD Res2 = ReadShort(InFile);
  BitmapCamera.ImageOffset = ReadLong(InFile);
  DWORD HeaderSize = ReadLong(InFile);
  BitmapCamera.Width = ReadLong(InFile);
  BitmapCamera.Height = ReadLong(InFile);
  if (ReadShort(InFile) !=1)
  {
   m_Detail = "File has more than one colour plane";
   UpdateData(FALSE);
   EndWaitCursor();
   return;
  }
  BitmapCamera.ColDepth = ReadShort(InFile);
  BitmapCamera.Compression = ReadLong(InFile);
  DWORD ImageSize = ReadLong(InFile);
  DWORD XPelsMeter = ReadLong(InFile);
  DWORD YPelsMeter = ReadLong(InFile);
  BitmapCamera.Colours = ReadLong(InFile);
  BitmapCamera.ImpColours = ReadLong(InFile);

  if(BitmapCamera.Compression!=0)          //Not designed for compressed bitmaps
  {
   m_Detail = "Bitmap File is Compressed";
```

```
 UpdateData(FALSE);
 EndWaitCursor();
 return;
 }
 if(BitmapCamera.Width!=320)
 {
 m_Detail = "Image Width is not 320 pixels";
 UpdateData(FALSE);
 EndWaitCursor();
 return;
 }
 if(BitmapCamera.Height!=240)                    //only for a 320x240 bitmap!
 {
 m_Detail = "File's Height is not 240 pixels";
 UpdateData(FALSE);
 EndWaitCursor();
 return;
 }
 if(BitmapCamera.ColDepth!=24)                              //only for 24bit bitmaps
 {
 m_Detail = "File is not a 24bit RGB image";
 UpdateData(FALSE);
 EndWaitCursor();
 return;
 }

 unsigned char PixelR, PixelG, PixelB;
 int loop, loop2;

 BitmapCamera.Sum = 0;
 InFile.seekg(BitmapCamera.ImageOffset,ios::beg);          //set file pointer to start of image data

 for(loop =0; loop<240; loop++)
 {
  for (loop2 = 0; loop2<320; loop2++)
  {
    PixelB = InFile.get();
    PixelG = InFile.get();
    PixelR = InFile.get();

    BitmapCamera.Gray[loop2][240-loop-1] = (unsigned _int8)((255-PixelR)*0.3 + (255-PixelG)*0.59 +
(255-PixelR)*0.11);
    BitmapCamera.Sum += BitmapCamera.Gray[loop2][240-loop-1];
    Difference[loop2][loop] = FALSE;
    Sharp[loop2/4][loop/4] = FALSE;
  }
 }
 InFile.close();

 clock_t b = clock();
 float  secs = (float)((int)((b-a)*100000/CLOCKS_PER_SEC))/100000;
 char Text[100];
 sprintf(Text,"Image loaded in %.3f seconds",secs);
 m_Detail = Text;
 UpdateData(FALSE);

 NumBoxes = 0;

 //Finished reading the bitmap file, now calculate threshold
 //and threshold version of image map

 BitmapCamera.ThresholdLevel =(int)(BitmapCamera.Sum/76800);
 EndWaitCursor();
 Invalidate();
}
```

```
void CChlorideDemoDlg::OpenFile(CString Name)
{
 BeginWaitCursor();
 m_CameraName = Name;
 m_Detail = "";
 UpdateData(FALSE);

 BitmapCamera.Name = m_CameraName;
 clock_t a = clock();


 fstream InFile;
 InFile.open(m_CameraName,ios::binary|ios::in);

 WORD Header;
 Header = ReadShort(InFile);
 if(Header!=(('M'<<8) + 'B')) //Expect to read BM as first two bytes
 {
  m_Detail = "File Header incorrect";
  UpdateData(FALSE);
  EndWaitCursor();
  return;
 }
 BitmapCamera.FileSize = ReadLong(InFile);
 WORD Res1 = ReadShort(InFile);
 WORD Res2 = ReadShort(InFile);
 BitmapCamera.ImageOffset = ReadLong(InFile);
 DWORD HeaderSize = ReadLong(InFile);
 BitmapCamera.Width = ReadLong(InFile);
 BitmapCamera.Height = ReadLong(InFile);
 if (ReadShort(InFile) !=1)
 {
  m_Detail = "File has more than one colour plane";
  UpdateData(FALSE);
  EndWaitCursor();
  return;
 }
 BitmapCamera.ColDepth = ReadShort(InFile);
 BitmapCamera.Compression = ReadLong(InFile);
 DWORD ImageSize = ReadLong(InFile);
 DWORD XPelsMeter = ReadLong(InFile);
 DWORD YPelsMeter = ReadLong(InFile);
 BitmapCamera.Colours = ReadLong(InFile);
 BitmapCamera.ImpColours = ReadLong(InFile);

 if(BitmapCamera.Compression!=0)              //Not designed for compressed bitmaps
 {
  m_Detail = "Bitmap File is Compressed";
  UpdateData(FALSE);
  EndWaitCursor();
  return;
 }
 if(BitmapCamera.Width!=320)
 {
  m_Detail = "Image Width is not 320 pixels";
  UpdateData(FALSE);
  EndWaitCursor();
  return;
 }
 if(BitmapCamera.Height!=240)                 //only for a 320x240 bitmap!
 {
  m_Detail = "File's Height is not 240 pixels";
  UpdateData(FALSE);
  EndWaitCursor();
  return;
 }
```

```
 if(BitmapCamera.ColDepth!=24)                          //only for 24bit bitmaps
 {
  m_Detail = "File is not a 24bit RGB image";
  UpdateData(FALSE);
  EndWaitCursor();
  return;
 }

 unsigned char PixelR, PixelG, PixelB;
 int loop, loop2;

 BitmapCamera.Sum = 0;
 InFile.seekg(BitmapCamera.ImageOffset,ios::beg);       //set file pointer to start of image data

 for(loop =0; loop<240; loop++)
 {
  for (loop2 = 0; loop2<320; loop2++)
  {
    PixelB = InFile.get();
    PixelG = InFile.get();
    PixelR = InFile.get();

    BitmapCamera.Gray[loop2][240-loop-1] = (unsigned _int8)((255-PixelR)*0.3 + (255-PixelG)*0.59 +
(255-PixelR)*0.11);
    BitmapCamera.Sum += BitmapCamera.Gray[loop2][240-loop-1];
    Difference[loop2][loop] = FALSE;
    Sharp[loop2/4][loop/4] = FALSE;
  }
 }
 InFile.close();

 clock_t b = clock();
 float  secs = (float)((int)((b-a)*100000/CLOCKS_PER_SEC))/100000;
 char Text[100];
 sprintf(Text,"Image loaded in %.3f seconds",secs);
 m_Detail = Text;
 UpdateData(FALSE);

 NumBoxes = 0;

 //Finished reading the bitmap file, now calculate threshold
 //and threshold version of image map

 BitmapCamera.ThresholdLevel =(int)(BitmapCamera.Sum/76800);
 EndWaitCursor();
 Invalidate();
}


////////////////////////////////////////////////////////////////////////
/*********************************************************************/
/***************  IMAGE CALCULATION OPERATIONS *********************/
/*********************************************************************/
////////////////////////////////////////////////////////////////////////




void CChlorideDemoDlg::DoJitter()
{
 //*********************************************************************************
 //Following Code generates the thresholded array of the incomming image //
 //first sequence, image as is
 int loop, loop2, Space, Diff,count = 0;
 int m_Mult = 5;
```

```
Space = (int)sqrt(pow(BitmapCamera.ThresholdLevel - BitmapDatum.ThresholdLevel,2));
for (loop = 0; loop <320; loop ++)
{
 for (loop2 = 0; loop2 < 240; loop2 ++)
 {
  //image jitter correction code
  //can compensate by 1 pixel in a certain direction
  //Normal image
  Diff = (int)sqrt(pow((BitmapCamera.Gray[loop][loop2] - BitmapDatum.Gray[loop][loop2]),2));
  if (Diff > (m_Mult*Space)) //m_Mult adjusts the sensitivity
  {
   // If a difference is detected, the pixel is shifted
   // around by a distance of one pixel to evaluate if
   // this is due to a small wind movement. The 9 pixels
   // surrounding the current pixel will be evaluated
   // Every image pixel can be shifted in a different direction

   //Image shifted up and left
   if ((loop<319)&&(loop2<239))
   {
    Diff = (int)sqrt(pow((BitmapCamera.Gray[loop+1][loop2+1] - BitmapDatum.Gray[loop][loop2]),2));
    if (Diff > (m_Mult*Space)) //m_Mult adjusts the sensitivity

    {
     //Image shifted up
     if (loop2<239)
     {
      Diff = (int)sqrt(pow((BitmapCamera.Gray[loop][loop2+1] - BitmapDatum.Gray[loop][loop2]),2));
      if (Diff > (m_Mult*Space)) //m_Mult adjusts the sensitivity
      {
       //Image shifted up and right
       if ((loop>0)&&(loop2<239))
       {
        Diff = (int)sqrt(pow((BitmapCamera.Gray[loop-1][loop2+1] - BitmapDatum.Gray[loop][loop2]),
2));
        if (Diff > (m_Mult*Space)) //m_Mult adjusts the sensitivity
        {
         //Image shifted left
         if (loop<319)
         {
          Diff = (int)sqrt(pow((BitmapCamera.Gray[loop+1][loop2] - BitmapDatum.Gray[loop][loop2]),
2));
          if (Diff > (m_Mult*Space)) //m_Mult adjusts the sensitivity
          {
           //Image shifted right
           if (loop>0)
           {
            Diff = (int)sqrt(pow((BitmapCamera.Gray[loop-1][loop2] - BitmapDatum.Gray[loop][loop2]),
2));
            if (Diff > (m_Mult*Space)) //m_Mult adjusts the sensitivity
            {
             //Image shifted down and left
             if ((loop<319)&&(loop2>0))
             {
              Diff = (int)sqrt(pow((BitmapCamera.Gray[loop+1][loop2-1] - BitmapDatum.Gray[loop]
[loop2]),2));
              if (Diff > (m_Mult*Space)) //m_Mult adjusts the sensitivity
              {
               //Image shifted down
               if (loop2>0)
               {
                Diff = (int)sqrt(pow((BitmapCamera.Gray[loop][loop2+1] - BitmapDatum.Gray[loop]
[loop2]),2));
                if (Diff > (m_Mult*Space)) //m_Mult adjusts the sensitivity
                {


                 //Image shifted down and right
```

```
                    if ((loop>0)&&(loop2>0))
                    {
                     Diff = (int)sqrt(pow((BitmapCamera.Gray[loop-1][loop2-1] - BitmapDatum.Gray[loop]
[loop2]),2));
                     if (Diff > (m_Mult*Space))  //m_Mult adjusts the sensitivity
                     {
                      //If none of the pixel shifts Difference in a difference exclusion
                      //the pixel is marked as a set movement difference
                      Difference[loop][loop2] = TRUE;
                      count++;
                     }

                    }
                   }
                  }
                 }
                }
               }
              }
             }
            }
           }
          }
         }
        }
       }
      }
     }
    // If any one of the pixel shifts Differences in a difference
    //exclusion, the pixel is marked as non altered, ie only
    //a small movement due to wind or BitmapCamera.Gray vibration

    else Difference[loop][loop2] = FALSE;
   }
  }
// NetworkThreshold = 6 + (int)(count/5000);
  NetworkThreshold = 6 + (int)(count/2222);
  if (NetworkThreshold >15 ) NetworkThreshold = 15;
}
```

```
void CChlorideDemoDlg::DoSharp()
{
 int loop,loop2, loop3, loop4;
 int count = 0;

 for (loop2=0; loop2<60; loop2++)
 {
  for ( loop=0; loop<80; loop++)
  {
   Sharp[loop][loop2] = FALSE;
   count = 0;
   for (loop3=0; loop3<4; loop3++)
   {
    for ( loop4 = 0; loop4<4; loop4++)
    {
     if (Difference[4*loop+loop3][4*loop2+loop4])
       count++;
    }
   }
   if (count >NetworkThreshold)
   {
    Sharp[loop][loop2] = TRUE;
   }
```

```
  }
 }
}



int CChlorideDemoDlg::DoObjects()
{
 int counter = 0;
 int loop, loop1, loop2, loop3;
 int Mem, Set, gap;

 for (loop = 0; loop<100; loop++)
 {
  Box[loop].Xmin = 80;
  Box[loop].Ymin = 60;
  Box[loop].Xmax = 0;
  Box[loop].Ymax = 0;
 }

 //Check every line in the image
 loop=0;
 for(loop2=0; loop2<60; loop2++)
 {
  Set = 0;
  Mem = 0;
  gap = 0;

  for (loop=0; loop<80; loop++)
  {
   if (Sharp[loop][loop2]==TRUE)
   {
    counter++;
    Box[counter].Xmin = loop;
    Box[counter].Ymin = loop2;
    Box[counter].Ymax = loop2;
    Box[counter].Xmax = loop;
    Box[counter].Xmini[loop2] = loop;
    Box[counter].Xmaxi[loop2] = loop;

    //Previous line check for position matching
    for(loop3 = 0; loop3<counter; loop3++)
    {
     if((loop>=(Box[loop3].Xmin-2))&&(loop<=(Box[loop3].Xmax+2))&&
       ((loop2-Box[loop3].Ymax)<3)&&(loop3!=counter)&&
       (loop2>0))
     {
      Set = 1;
      Mem = loop3;
     }
    }
    //End of line check

    while((gap<3)&&(loop<80))
    {
     loop++;
     if(Sharp[loop][loop2]==1)
     {
      gap = 0;
      Box[counter].Xmax = loop;
      Box[counter].Xmaxi[loop2] = loop;

      //Previous line check for position matching
      for(loop3 = 0; loop3<counter; loop3++)
      {
```

```
        if((loop>=Box[loop3].Xmin-2)&&(loop<=Box[loop3].Xmax+2)
         &&(loop2-Box[loop3].Ymax<3)&&(loop3!=counter)&&(counter>0))
        {
         Set = 1;
         Mem = loop3;
        }
       }
       //End of line check
      }
      else gap++;
     }
     gap = 0;

     //Matching correction code, updates matched object
     //and deleted new object created
     if (Set==1)
     {
      if(Box[counter].Xmin<Box[Mem].Xmin) Box[Mem].Xmin = Box[counter].Xmin;
      if(Box[counter].Xmax>Box[Mem].Xmax) Box[Mem].Xmax = Box[counter].Xmax;
      Box[Mem].Ymax = loop2;
      Box[counter].Xmin = 80;
      Box[counter].Xmax = 0;
      Box[counter].Ymin = 60;
      Box[counter].Ymax = 0;
      Box[Mem].Xmaxi[loop2] = Box[counter].Xmaxi[loop2];
      Box[Mem].Xmini[loop2] = Box[counter].Xmini[loop2];
      counter--;
      Set = 0;
      Mem = 0;
     }
    }
   }
   loop=0;
  }
  //Box size filtering code, checking for minimum target size

  int IsZero = 0;

  for ( loop = 1; loop<=counter; loop++)
  {
   if (IsZero==1)
   {
    loop--;
    IsZero = 0;
   }
   int TArea = ((Box[loop].Xmax - Box[loop].Xmin)*(Box[loop].Ymax-Box[loop].Ymin));
   if (TArea <74)
   {
    for (loop1 = loop; loop1<=counter; loop1++)
    {
     Box[loop1] = Box[loop1+1];
     if (loop==1) IsZero = 1;
     else loop--;
    }
    counter--;
   }
  }
  return counter;
}


void CChlorideDemoDlg::GetNetInputs()
{
 int loop,loop1,loop2,loop3, loop4;
 int Width, Height;
 int SegWidth,SegHeight,SegAcc,SegArea;
```

```
for ( loop = 0; loop <= NumBoxes; loop++)
{
 //*************** SEGMENT AREA CALC ********************
 //The set rectangle will be split up into an 3*4 array of equal
 //segments, on which % set pixels calcs are carried out.

 SegWidth=0;
 SegHeight=0;
 SegAcc=0;
 SegArea=0;
 int x_max=0;
 int y_max=0;
 int TempWidth=0;
 int TempHeight=0;
 Width = 0;
 Height = 0;
 SegArea = 0;
 SegAcc = 0;

 Width = 4*Box[loop].Xmax - 4*Box[loop].Xmin;
 Height = 4*Box[loop].Ymax - 4*Box[loop].Ymin;

 div_t h,w;
 w = div(Width, 4);
 h = div(Height, 6);
 SegWidth = w.quot;
 SegHeight = h.quot;

 SegArea = SegWidth * SegHeight;


 for ( loop1 = 0; loop1 <4; loop1++)
 {
  for ( loop2 = 0; loop2<6; loop2++)
  {
   x_max = (loop1 + 1) * SegWidth;
   y_max = (loop2 + 1) * SegHeight;
   if ( loop1 ==3)
   {
    x_max = Width;
    TempWidth = Width - (3 * SegWidth);
   }
   else TempWidth = SegWidth;
   if (loop2 ==5)
   {
    y_max = Height;
    TempHeight = Height - (5 * SegHeight);
   }
   else TempHeight = SegHeight;
   SegAcc = 0;
   for (loop3=(loop1*SegWidth); loop3<x_max;loop3++)
   {
    for (loop4=(loop2*SegHeight); loop4<y_max;loop4++)
    {
     if ((Difference[4*Box[loop].Xmin+loop3][4*Box[loop].Ymin+loop4])
      && (Sharp[(Box[loop].Xmin +loop3)][(Box[loop].Ymin + loop4)]))
       //&& ( loop1>= 4*Box[loop].Xmini[loop])
        //&& ( loop1 <= 4*Box[loop].Xmaxi[loop]))
     {
      SegAcc++;
     }
    }
   }
   Box[loop].SegArea[loop1][loop2] = 0;
   Box[loop].SegArea[loop1][loop2]=(int)((100 * SegAcc)/(TempWidth * TempHeight));

   SegAcc=0;
  }
```

```
  }
 }
}



void CChlorideDemoDlg::RunNet()
{
  //NETWORK CODE HERE
 int loop;
 float Output;
 for(loop = 0; loop<NumBoxes; loop++)
 {
  float Xsum7 = 0;
  float Xout7 = 0;
  Output = 0;
  /* Generating code for PE 0 in layer <Hidden1> (3) */
    Xsum7  = (float)(9.4248371 + (-9.7122078) * Box[loop].NetInput[0] + (-0.82052672) * Box[loop].
NetInput[1] +
     (-4.1746411) * Box[loop].NetInput[2] + (-1.6331787) * Box[loop].NetInput[3] + (-2.0222914) * Box
[loop].NetInput[4]);
    /* Generating code for PE 0 in layer <Hidden1> (3) */
  Xout7 = (float)(tanh( Xsum7 ));

   Output  = (float)((0.15872838) + (0.04405418) * Box[loop].NetInput[0] + (-0.12671886) * Box[loop].
NetInput[1] +
     (-0.21494821) * Box[loop].NetInput[2] + (-0.029009042) * Box[loop].NetInput[3] + (-0.015902856)
* Box[loop].NetInput[4] +
   (0.91813892) * Xout7);

    /* De-scale and write output from network */
  Output = (float)(Output * (0.625) + (0.5));

  //END OF NETWORK CODE
  Box[loop].Result = (int)(100*Output);
  if (Box[loop].Result>100) Box[loop].Result = 100;
  if (Box[loop].Result<0) Box[loop].Result = 0;
 }
}



//****************** MENU SELECTIONS *********************************//



void CChlorideDemoDlg::OnCalculationsImagedifference()
{
 // TODO: Add your command handler code here
 BeginWaitCursor();
 clock_t a = clock();
 DoJitter();
 clock_t b = clock();
 float  secs = (float)((int)((b-a)*100000/CLOCKS_PER_SEC))/100000;
 char Text[100];
 sprintf(Text,"Dynamic Network Threshold : %i   Processing time was %.3f
seconds",NetworkThreshold,secs);
 m_Detail = Text;
 UpdateData(FALSE);
 EndWaitCursor();
 if(ShowDiff==TRUE) Invalidate();
}


void CChlorideDemoDlg::OnCalculationsNoisefiltering()
{
 // TODO: Add your command handler code here
```

```
 BeginWaitCursor();
 clock_t a = clock();
 DoJitter();
 DoSharp();
 clock_t b = clock();
 float  secs = (float)((int)((b-a)*100000/CLOCKS_PER_SEC))/100000;
 char Text[100];
 sprintf(Text,"Dynamic Network Threshold : %i   Processing time was %.3f
seconds",NetworkThreshold,secs);
 m_Detail = Text;
 UpdateData(FALSE);
 EndWaitCursor();
 if((ShowDiff) || (ShowSharp)) Invalidate();
}


void CChlorideDemoDlg::OnCalculationsObjectboundaries()
{
 // TODO: Add your command handler code here
 BeginWaitCursor();
 clock_t a = clock();
 DoJitter();
 DoSharp();
 NumBoxes = DoObjects();
 clock_t b = clock();
 float  secs = (float)((int)((b-a)*100000/CLOCKS_PER_SEC))/100000;
 char Text[200];
 sprintf(Text,"Dynamic Network Threshold : %i   %i Object(s) identified in image  Processing time was %.3f
seconds",NetworkThreshold,NumBoxes,secs);
 m_Detail = Text;
 UpdateData(FALSE);
 EndWaitCursor();
 if((ShowDiff) || (ShowSharp) || (ShowBox)) Invalidate();
}


void CChlorideDemoDlg::OnCalculationsNetworkanalysis()
{
 // TODO: Add your command handler code here
 BeginWaitCursor();
 clock_t a = clock();
 DoJitter();
 DoSharp();
 NumBoxes = DoObjects();
 GetNetInputs();
 RunNet();
 clock_t b = clock();
 float  secs = (float)((int)((b-a)*100000/CLOCKS_PER_SEC))/100000;
 char Text[200];
 sprintf(Text,"Dynamic Network Threshold : %i   %i Object(s) identified in image  Processing time was %.3f
seconds",NetworkThreshold,NumBoxes,secs);
 m_Detail = Text;
 UpdateData(FALSE);
 EndWaitCursor();
 if((ShowDiff) || (ShowSharp) || (ShowBox) || (ShowNet)) Invalidate();
}


////////////////////////////////////////////////////////////////////
/*****************************************************************/
/**************** FILE SAVING OPERATIONS ***************************/
/*****************************************************************/
////////////////////////////////////////////////////////////////////
```

```
void CChlorideDemoDlg::OnSaveSavedifference()
{
 // TODO: Add your command handler code here
 int Pos = BitmapCamera.Name.Find("bmp");
 if (Pos==-1) Pos = BitmapCamera.Name.Find("BMP");
 CString SaveName;
 SaveName = BitmapCamera.Name.Left(Pos);
 SaveName = SaveName + "vos";
 CFileDialog FileDlg(FALSE,"vos",SaveName);
 if(FileDlg.DoModal()==IDOK)
 {
  BeginWaitCursor();
  CString SavePath = FileDlg.GetPathName();
  DoJitter();
  ofstream FileOut(SavePath);
  if(!FileOut)
  {
   m_Detail = "File write error - Info could not be saved";
   UpdateData(FALSE);
   EndWaitCursor();
   return;
  }
  int loop, loop2;
  for (loop = 0; loop< 320; loop++)
  {
   for (loop2 = 0; loop2<240; loop2++)
   {
    if (Difference[loop][loop2])
     FileOut << loop<<"\t"<<loop2<<"\n";
   }
  }
  FileOut.close();
  m_Detail = SavePath + " written successfully";
 }
 EndWaitCursor();
 UpdateData(FALSE);
}


void CChlorideDemoDlg::OnSaveSavesharp()
{
 // TODO: Add your command handler code here

}


void CChlorideDemoDlg::OnSaveSaveobjects()
{
 // TODO: Add your command handler code here

}

void CChlorideDemoDlg::OnSaveSavevdfsperobjectinfo()
{
 // TODO: Add your command handler code here

}



////////////////////////////////////////////////////////////////////////
/**********************************************************************/
/********************* SOFTWARE SETTINGS ******************************/
/**********************************************************************/
////////////////////////////////////////////////////////////////////////
```

```
void CChlorideDemoDlg::OnSettingsViewoptions()
{
 // TODO: Add your command handler code here
 //CDialog ViewDlg(IDD_VIEW_DIALOG);

 m_Viewdlg.m_Sharp = ShowSharp;
 m_Viewdlg.m_Diff = ShowDiff;
 m_Viewdlg.m_Boxes = ShowBox;
 m_Viewdlg.m_Net = ShowNet;

 if(m_Viewdlg.DoModal()==IDOK)
 {
  ShowSharp = m_Viewdlg.m_Sharp;
  ShowDiff = m_Viewdlg.m_Diff;
  ShowBox = m_Viewdlg.m_Boxes;
  ShowNet = m_Viewdlg.m_Net;
 }
}

void CChlorideDemoDlg::OnStartBatchButton()
{
 // TODO: Add your control notification handler code here
 ShowSharp = FALSE;
 ShowDiff = FALSE;
 ShowBox = TRUE;
 ShowNet = FALSE;

 CFileFind Finder;
 CString Title;
 chdir(m_DatumDirectory);
 BOOL bWorking = Finder.FindFile("*.bmp");

 MessageBox("About to enter File Search mode",MB_OK);
 UpdateData(FALSE);
 int loop, loop1,loop2;
 while(bWorking)
 {
  bWorking = Finder.FindNextFile();
  Title = Finder.GetFileTitle();
  CString FullName = Finder.GetFilePath();
  int Pos = FullName.Find(Title);
  CString TempDir = FullName.Left(Pos);

  OpenFile(FullName);
  OnCalculationsObjectboundaries();
  GetNetInputs();
  if ((BatchAll) || (BatchPrompt))
  {
   for(loop = 1; loop<=NumBoxes; loop++)
   {
    char Text[100];
    sprintf(Text,"Would you like to save info for Box %i ?",loop);
    if (m_SaveDir=="")
    {
     m_SaveDir = TempDir;
     m_Savew = m_SaveDir;
     UpdateData(FALSE);
    }
    if(BatchPrompt)
    {
     if(MessageBox(Text,"Save Query",MB_YESNO)==IDYES)
     {
      char Ext[10];
      sprintf(Ext,"%s%s-%i.sdf",m_SaveDir,Title,loop);
      ofstream OutFile(Ext);
      for(loop1 = 0; loop1<6; loop1++)
```

```
      {
       for(loop2 = 0; loop2<4; loop2++)
       {
        OutFile << (Box[loop].SegArea[loop2][loop1]) <<"\t";

       }
       OutFile << endl;
      }
      OutFile << "Segment Area Measurements in order 4x6 grid\n";
      OutFile << "Datum File : "<< m_DatumName << endl;
      OutFile << "Camera File : "<< m_CameraName<< endl;
      OutFile << "Object No : " << loop;
      OutFile.close();
     }
    }
    if(BatchAll)
    {
     char Ext[10];
     sprintf(Ext,"%s%s-%i.sdf",m_SaveDir,Title,loop);

     ofstream OutFile(Ext);
     for(loop1 = 0; loop1<6; loop1++)
     {
      for(loop2 = 0; loop2<4; loop2++)
      {
       OutFile << Box[loop].SegArea[loop2][loop1] <<"\t";
      }
      OutFile << endl;
     }
     OutFile << "Segment Area Measurements in order 4x6 grid" << endl;
     OutFile << "Datum File : "<< m_DatumName << endl;
     OutFile << "Camera File : "<< m_CameraName << endl;
     OutFile << "Object No : " << loop;
     OutFile.close();
    }
   }
  }
  //Depending on the vBatch Svae settings
  //save vdf file or not !
  //****************************************
  //  INSERT INTERRUPT CODE HERE
  //****************************************
 }
 MessageBox("End of Search",MB_OK);
}


void CChlorideDemoDlg::OnBatchCheck()
{
 // TODO: Add your control notification handler code here

}

void CChlorideDemoDlg::OnNoneRadio()
{
 // TODO: Add your control notification handler code here
 BatchNone = TRUE;
 BatchAll = FALSE;
 BatchPrompt = FALSE;
}

void CChlorideDemoDlg::OnAllRadio()
{
 // TODO: Add your control notification handler code here
 BatchNone = FALSE;
 BatchAll = TRUE;
 BatchPrompt = FALSE;
}
```

```
void CChlorideDemoDlg::OnPromptRadio()
{
 // TODO: Add your control notification handler code here
 BatchNone = FALSE;
 BatchAll = FALSE;
 BatchPrompt = TRUE;
}

void CChlorideDemoDlg::OnSaveButton()
{
 // TODO: Add your control notification handler code here
 CFileDialog SaveDlg(FALSE,"save.sdf","save.sdf");
 if(SaveDlg.DoModal() == IDOK)
 {
  m_Savew = SaveDlg.GetPathName();
  CString Temp = SaveDlg.GetFileTitle();
  int Pos = m_Savew.Find(Temp);
  m_SaveDir = m_Savew.Left(Pos);
  m_Savew = m_SaveDir;
  UpdateData(FALSE);

 }
}

void CChlorideDemoDlg::OnHelpAbout()
{
 // TODO: Add your command handler code here
 CAboutDlg m_dlg;
 m_dlg.DoModal();
}
```

# 20.6 - Bitmap Headers

```
////////////////////////////////////////////////////////////////////////
//
//      BITMAPHEADER.H
//
// Reference Header file in C++ designed for Bitmap Interpretation
// Aiming Specifically at a 320*240 24bit Bitmap
//
// When File is read, image data is saved in grayscale to an array
// of type _int8
// All the bitmap major header infois also saved to a structure
// which must be declared in the main program
//
//
// Written: 01.03.2000 by Jean-Marc Graumann
//
////////////////////////////////////////////////////////////////////////

#include <fstream.h>


struct BitmapStruct
{
 CString Name;
 unsigned _int8 Gray[320][240];          //hold bitmap grayscale info
 unsigned long Sum;                      //keeps track of the total image intensity
                                         //divide by 76800 to get the grayscale median

 unsigned long FileSize;
 unsigned short Width, Height;           //Width and Height of the Bitmap in Pixels
 unsigned char Compression;              //Specifies the type of compression used
 unsigned long ImageOffset;              //Number of bytes from start of file to image data
 unsigned char ColDepth;                 //type of Bitmap
 unsigned long Colours;
 unsigned long ImpColours;
};



unsigned long ReadLong(fstream InFile)
{
 unsigned char a1,a2,a3,a4;
 unsigned long ReturnVal;

 a1 = InFile.get();
 a2 = InFile.get();
 a3 = InFile.get();
 a4 = InFile.get();

 ReturnVal = a1 + a2<<8 + a3<<16 + a4<<24;

 return ReturnVal;
}


unsigned short ReadShort(fstream InFile)
{
 unsigned char a1,a2;
 unsigned short ReturnVal;

 a1 = InFile.get();
 a2 = InFile.get();

 ReturnVal = a1 + a2<<8;
```

```
 return ReturnVal;
}


BOOL CheckHeader(fstream InFile)
{
 InFile.get();// return FALSE;
 InFile.get();// return FALSE;

 return TRUE;
}


BOOL ReadBitmapHeader(fstream InFile,BitmapStruct &Map)
{
 CheckHeader(InFile);//return FALSE;          //if first to bytes are not BM, file format is invalid
 Map.FileSize = ReadLong(InFile);                                    //Get the FileSize;
 InFile.seekg(4,ios::cur);                                           //skip over reserved bits
 Map.ImageOffset = ReadLong(InFile);                                 // Get the image offset
 unsigned long HeaderSize = ReadLong(InFile);
 Map.Width = (unsigned short)ReadLong(InFile);
 Map.Height = (unsigned short)ReadLong(InFile);
 if(ReadShort(InFile)!=1) return FALSE;       //Bitmap can only have 1 colour plane
 Map.ColDepth = (unsigned char)ReadShort(InFile);
 Map.Compression = (unsigned char)ReadLong(InFile);
 unsigned long ImageSize = ReadLong(InFile);
 unsigned long XPelsMeter = ReadLong(InFile);
 unsigned long YPelsMeter = ReadLong(InFile);
 Map.Colours = ReadLong(InFile);
 Map.ImpColours = ReadLong(InFile);

 return TRUE;
}


BOOL ReadImageData(fstream InFile,BitmapStruct &Map)
{
 if(Map.Compression!=0) return FALSE;        //Not designed for compressed bitmaps
 if(Map.Width!=320) return FALSE;
 if(Map.Height!=240) return FALSE;                    //only for a 320x240 bitmap!
 if(Map.ColDepth!=24) return FALSE;                   //only for 24bit bitmaps

 unsigned char PixelR, PixelG, PixelB;
 int loop, loop2;

 Map.Sum = 0;
 InFile.seekg(Map.ImageOffset,ios::beg);      //set file pointer to start of image data

 for(loop =0; loop<240; loop++)
 {
  for (loop2 = 0; loop2<320; loop2++)
  {
    PixelB = InFile.get();
    PixelG = InFile.get();
    PixelR = InFile.get();

    Map.Gray[loop2][240-loop-1] = (unsigned _int8)((255-PixelR)*0.3 + (255-PixelG)*0.59 + (255-PixelR)
*0.11);
    Map.Sum += Map.Gray[loop2][240-loop-1];
  }
 }

 return TRUE;
}
```

## 20.7 - Cheat Office

```
// Cheat OfficeDlg.cpp : implementation file
//

#include "stdafx.h"
#include "Cheat Office.h"
#include "Cheat OfficeDlg.h"
#include <fstream.h>
#include <math.h>

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

/////////////////////////////////////////////////////////////////////////////
// CAboutDlg dialog used for App About



class CAboutDlg : public CDialog
{
public:
 CAboutDlg();

// Dialog Data
 //{{AFX_DATA(CAboutDlg)
 enum { IDD = IDD_ABOUTBOX };
 double m_Mult;
 //}}AFX_DATA

 // ClassWizard generated virtual function overrides
 //{{AFX_VIRTUAL(CAboutDlg)
 protected:
 virtual void DoDataExchange(CDataExchange* pDX);    // DDX/DDV support
 //}}AFX_VIRTUAL

// Implementation
protected:
 //{{AFX_MSG(CAboutDlg)
 afx_msg void OnMultButton();
 //}}AFX_MSG
 DECLARE_MESSAGE_MAP()
};

CAboutDlg::CAboutDlg() : CDialog(CAboutDlg::IDD)
{
 //{{AFX_DATA_INIT(CAboutDlg)
 m_Mult = 0.0;
```

```
 //}}AFX_DATA_INIT

}

void CAboutDlg::DoDataExchange(CDataExchange* pDX)
{
 CDialog::DoDataExchange(pDX);
 //{{AFX_DATA_MAP(CAboutDlg)
 DDX_Text(pDX, IDC_MULT_EDIT, m_Mult);
 DDV_MinMaxDouble(pDX, m_Mult, 0., 5.);
 //}}AFX_DATA_MAP
}

BEGIN_MESSAGE_MAP(CAboutDlg, CDialog)
 //{{AFX_MSG_MAP(CAboutDlg)
 //}}AFX_MSG_MAP
END_MESSAGE_MAP()

/////////////////////////////////////////////////////////////////////////////
// CCheatOfficeDlg dialog

CCheatOfficeDlg::CCheatOfficeDlg(CWnd* pParent /*=NULL*/)
 : CDialog(CCheatOfficeDlg::IDD, pParent)
{
 //{{AFX_DATA_INIT(CCheatOfficeDlg)
 m_DatumCheck = FALSE;
 m_Mult = 0.0;
 //}}AFX_DATA_INIT
 // Note that LoadIcon does not require a subsequent DestroyIcon in Win32
 m_hIcon = AfxGetApp()->LoadIcon(IDR_MAINFRAME);
}

void CCheatOfficeDlg::DoDataExchange(CDataExchange* pDX)
{
 CDialog::DoDataExchange(pDX);
 //{{AFX_DATA_MAP(CCheatOfficeDlg)
 DDX_Check(pDX, IDC_DATUM_CHECK, m_DatumCheck);
 DDX_Text(pDX, IDC_MULT_EDIT, m_Mult);
 //}}AFX_DATA_MAP
}

BEGIN_MESSAGE_MAP(CCheatOfficeDlg, CDialog)
 //{{AFX_MSG_MAP(CCheatOfficeDlg)
 ON_WM_SYSCOMMAND()
 ON_WM_PAINT()
 ON_WM_QUERYDRAGICON()
 ON_BN_CLICKED(IDC_BACKGROUND_BUTTON, OnBackgroundButton)
 ON_BN_CLICKED(IDC_DATUM_BUTTON, OnDatumButton)
 ON_BN_CLICKED(IDC_INCOMING_BUTTON, OnIncomingButton)
 ON_BN_CLICKED(IDC_EXIT_BUTTON, OnExitButton)
 ON_BN_CLICKED(IDC_PROCESS_BUTTON, OnProcessButton)
 ON_BN_CLICKED(IDC_SET_BUTTON, OnSetButton)
 //}}AFX_MSG_MAP
END_MESSAGE_MAP()

/////////////////////////////////////////////////////////////////////////////
// CCheatOfficeDlg message handlers

BOOL CCheatOfficeDlg::OnInitDialog()
{
 CDialog::OnInitDialog();

 // Add "About..." menu item to system menu.

 // IDM_ABOUTBOX must be in the system command range.
 ASSERT((IDM_ABOUTBOX & 0xFFF0) == IDM_ABOUTBOX);
 ASSERT(IDM_ABOUTBOX < 0xF000);
```

```
CMenu* pSysMenu = GetSystemMenu(FALSE);
if (pSysMenu != NULL)
{
 CString strAboutMenu;
 strAboutMenu.LoadString(IDS_ABOUTBOX);
 if (!strAboutMenu.IsEmpty())
 {
  pSysMenu->AppendMenu(MF_SEPARATOR);
  pSysMenu->AppendMenu(MF_STRING, IDM_ABOUTBOX, strAboutMenu);
 }
}

// Set the icon for this dialog.  The framework does this automatically
//  when the application's main window is not a dialog
SetIcon(m_hIcon, TRUE);                                    // Set big icon
SetIcon(m_hIcon, FALSE);                          // Set small icon

// TODO: Add extra initialization here
for(int loop = 0; loop<320; loop++)
{
 for ( int loop2 = 0; loop2<240; loop2++)
 {
  Camera.Data[loop][loop2] = 0;
  Background.Data[loop][loop2] = 0;
  Final.Data[loop][loop2] = 0;
  Datum.Data[loop][loop2] = 0;
 }
}
m_Mult = 0.2;
UpdateData(FALSE);
return TRUE;  // return TRUE  unless you set the focus to a control
}

void CCheatOfficeDlg::OnSysCommand(UINT nID, LPARAM lParam)
{
 if ((nID & 0xFFF0) == IDM_ABOUTBOX)
 {
  CAboutDlg dlgAbout;
  dlgAbout.DoModal();
 }
 else
 {
  CDialog::OnSysCommand(nID, lParam);
 }
}

// If you add a minimize button to your dialog, you will need the code below
//  to draw the icon.  For MFC applications using the document/view model,
//  this is automatically done for you by the framework.

void CCheatOfficeDlg::OnPaint()
{
 if (IsIconic())
 {
  CPaintDC dc(this); // device context for painting

  SendMessage(WM_ICONERASEBKGND, (WPARAM) dc.GetSafeHdc(), 0);

  // Center icon in client rectangle
  int cxIcon = GetSystemMetrics(SM_CXICON);
  int cyIcon = GetSystemMetrics(SM_CYICON);
  CRect rect;
  GetClientRect(&rect);
  int x = (rect.Width() - cxIcon + 1) / 2;
  int y = (rect.Height() - cyIcon + 1) / 2;

  // Draw the icon
  dc.DrawIcon(x, y, m_hIcon);
```

```
 }
 else
 {
  CPaintDC dc(this);
  int loop, loop2;

  for(loop2 = 0; loop2 <240; loop2++)
  {
   for ( loop = 0; loop < 320; loop++)
   {
    int Value = Final.Data[loop][loop2];
    dc.SetPixel(14 + loop, 14 + loop2, RGB(Value, Value, Value));
    Value = Camera.Data[loop][loop2];
    dc.SetPixel(345 + loop/2, 14 + loop2/2,RGB(Value, Value, Value));
    Value = Background.Data[loop][loop2];
    dc.SetPixel(345 + loop/2, 148 + loop2/2,RGB(Value, Value, Value));
   }
  }

  CDialog::OnPaint();
 }
}

// The system calls this to obtain the cursor to display while the user drags
//  the minimized window.
HCURSOR CCheatOfficeDlg::OnQueryDragIcon()
{
 return (HCURSOR) m_hIcon;
}

void CCheatOfficeDlg::OnBackgroundButton()
{
 // TODO: Add your control notification handler code here
 CFileDialog m_BackDlg(TRUE,"bmp","*.bmp");
 CString Temp;
 if (m_BackDlg.DoModal()==IDOK)
 {
  Temp = m_BackDlg.GetPathName();
  Background = OpenFile(Temp);
  Invalidate();
 }
}

void CCheatOfficeDlg::OnDatumButton()
{
 // TODO: Add your control notification handler code here
 CFileDialog m_BackDlg(TRUE,"bmp","*.bmp");
 CString Temp;
 if (m_BackDlg.DoModal()==IDOK)
 {
  Temp = m_BackDlg.GetPathName();
  Datum = OpenFile(Temp);
  Camera = Datum;
  Invalidate();

  if(MessageBox("Do you want to use this Datum image ?","Datum Selection",MB_OKCANCEL)==IDOK)
   m_DatumCheck = TRUE;
  else m_DatumCheck = FALSE;
  UpdateData(FALSE);
 }
}

void CCheatOfficeDlg::OnIncomingButton()
{
 // TODO: Add your control notification handler code here
 CFileDialog m_BackDlg(TRUE,"bmp","*.bmp");
 CString Temp;
 if (m_BackDlg.DoModal()==IDOK)
```

```
 {
  Temp = m_BackDlg.GetPathName();
  Camera = OpenFile(Temp);
  Invalidate();
 }
}

void CCheatOfficeDlg::OnExitButton()
{
 // TODO: Add your control notification handler code here
 OnOK();
}



DWORD ReadLong(fstream InFile)
{
 BYTE a1,a2,a3,a4;
 DWORD ReturnVal;

 InFile >> a1;
 InFile >> a2;
 InFile >> a3;
 InFile >>a4;

 ReturnVal =(DWORD)( a1 + a2*256 + a3*pow(256,2) + a4*pow(256,3));

 return ReturnVal;
}



WORD ReadShort(fstream InFile)
{
 BYTE a1,a2;
 WORD ReturnVal;

 InFile >> a1;
 InFile >> a2;

 ReturnVal = (WORD)(a1 + a2*256);

 return ReturnVal;
}



ImageInfo CCheatOfficeDlg::OpenFile(CString Name)
{
 ImageInfo Temp;
 Temp.FileName = Name;
 CString m_Detail;
 BeginWaitCursor();
 fstream InFile;
 InFile.open(Name,ios::binary|ios::in);

 WORD Header;
 Header = ReadShort(InFile);
 if(Header!=(('M'<<8) + 'B')) //Expect to read BM as first two bytes
 {
  m_Detail = "File Header incorrect";
  UpdateData(FALSE);
  EndWaitCursor();
  return Temp;
 }
 DWORD FileSize = ReadLong(InFile);
```

```
WORD Res1 = ReadShort(InFile);
WORD Res2 = ReadShort(InFile);
DWORD ImageOffset = ReadLong(InFile);
DWORD HeaderSize = ReadLong(InFile);
DWORD Width = ReadLong(InFile);
DWORD Height = ReadLong(InFile);
if (ReadShort(InFile) !=1)
{
 m_Detail = "File has more than one colour plane";
 UpdateData(FALSE);
 EndWaitCursor();
 return Temp;
}
WORD ColDepth = ReadShort(InFile);
DWORD Compression = ReadLong(InFile);
DWORD ImageSize = ReadLong(InFile);
DWORD XPelsMeter = ReadLong(InFile);
DWORD YPelsMeter = ReadLong(InFile);
DWORD Colours = ReadLong(InFile);
DWORD ImpColours = ReadLong(InFile);

if(Compression!=0) //Not designed for compressed bitmaps
{
 m_Detail = "Bitmap File is Compressed";
 UpdateData(FALSE);
 EndWaitCursor();
 return Temp;
}
if(Width!=320)
{
 m_Detail = "Image Width is not 320 pixels";
 UpdateData(FALSE);
 EndWaitCursor();
 return Temp;
}
if(Height!=240)  //only for a 320x240 bitmap!
{
 m_Detail = "File's Height is not 240 pixels";
 UpdateData(FALSE);
 EndWaitCursor();
 return Temp;
}
if(ColDepth!=24)   //only for 24bit bitmaps
{
 m_Detail = "File is not a 24bit RGB image";
 UpdateData(FALSE);
 EndWaitCursor();
 return Temp;
}

unsigned char PixelR, PixelG, PixelB;
int loop, loop2;

InFile.seekg(ImageOffset,ios::beg);              //set file pointer to start of image data

int Sum = 0;

for(loop =0; loop<240; loop++)
{
 for (loop2 = 0; loop2<320; loop2++)
 {
   PixelB = InFile.get();
   PixelG = InFile.get();
   PixelR = InFile.get();

   Temp.Data[loop2][240-loop-1] = (unsigned _int8)((PixelR)*0.3 + (PixelG)*0.59 + (PixelR)*0.11);
   Sum += Temp.Data[loop2][240-loop-1];
 }
```

```
 }
 InFile.close();


 //Finished reading the bitmap file, now calculate threshold
 //and threshold version of image map
 Temp.Median = (unsigned _int8)(Sum/76800);
 EndWaitCursor();
 return Temp;
}

void CCheatOfficeDlg::OnProcessButton()
{
 // TODO: Add your control notification handler code here
 if (!m_DatumCheck) return;
 DoJitter();
 DoSharp();
 DoObjects();
 int loop, loop2;
 Final = Background;
 for(loop = 0; loop<320; loop++)
 {
  for(loop2 = 0; loop2<240; loop2++)
  {
   //if((Sharp[loop/4][loop2/4])&&(Difference[loop][loop2]))
   if (Difference[loop][loop2])
    Final.Data[loop][loop2] = Camera.Data[loop][loop2];
  }
 }
 Invalidate();
}


void CCheatOfficeDlg::DoJitter()
{
 //*************************************************************************
 //Following Code generates the thresholded array of the incomming image //
 //first sequence, image as is
 int loop, loop2, Space, Diff,count = 0;


 Space = (int)sqrt(pow(Camera.Median - Datum.Median,2));
 for (loop = 0; loop <320; loop ++)
 {
  for (loop2 = 0; loop2 < 240; loop2 ++)
  {
   //image jitter correction code
   //can compensate by 1 pixel in a certain direction
   //Normal image
   Diff = (int)sqrt(pow((Camera.Data[loop][loop2] - Datum.Data[loop][loop2]),2));
   if (Diff> (m_Mult*Space))  //m_Mult adjusts the sensitivity
   {
    // If a difference is detected, the pixel is shifted
    // around by a distance of one pixel to evaluate if
    // this is due to a small wind movement. The 9 pixels
    // surrounding the current pixel will be evaluated
    // Every image pixel can be shifted in a different direction

    //Image shifted up and left
    if ((loop<319)&&(loop2<239))
    {
     Diff = (int)sqrt(pow((Camera.Data[loop+1][loop2+1] - Datum.Data[loop][loop2]),2));
     if (Diff > (m_Mult*Space))  //m_Mult adjusts the sensitivity

     {
      //Image shifted up
      if (loop2<239)
      {
```

```
Diff = (int)sqrt(pow((Camera.Data[loop][loop2+1] - Datum.Data[loop][loop2]),2));
if (Diff > (m_Mult*Space)) //m_Mult adjusts the sensitivity
{
 //Image shifted up and right
 if ((loop>0)&&(loop2<239))
 {
  Diff = (int)sqrt(pow((Camera.Data[loop-1][loop2+1] - Datum.Data[loop][loop2]),2));
  if (Diff > (m_Mult*Space)) //m_Mult adjusts the sensitivity
  {
   //Image shifted left
   if (loop<319)
   {
    Diff = (int)sqrt(pow((Camera.Data[loop+1][loop2] - Datum.Data[loop][loop2]),2));
    if (Diff > (m_Mult*Space)) //m_Mult adjusts the sensitivity
    {
     //Image shifted right
     if (loop>0)
     {
      Diff = (int)sqrt(pow((Camera.Data[loop-1][loop2] - Datum.Data[loop][loop2]),2));
      if (Diff > (m_Mult*Space)) //m_Mult adjusts the sensitivity
      {
       //Image shifted down and left
       if ((loop<319)&&(loop2>0))
       {
        Diff = (int)sqrt(pow((Camera.Data[loop+1][loop2-1] - Datum.Data[loop][loop2]),2));
        if (Diff > (m_Mult*Space)) //m_Mult adjusts the sensitivity
        {
         //Image shifted down
         if (loop2>0)
         {
          Diff = (int)sqrt(pow((Camera.Data[loop][loop2+1] - Datum.Data[loop][loop2]),2));
          if (Diff > (m_Mult*Space)) //m_Mult adjusts the sensitivity
          {


           //Image shifted down and right
           if ((loop>0)&&(loop2>0))
           {
            Diff = (int)sqrt(pow((Camera.Data[loop-1][loop2-1] - Datum.Data[loop][loop2]),2));
            if (Diff > (m_Mult*Space)) //m_Mult adjusts the sensitivity
            {
             //If none of the pixel shifts Difference in a difference exclusion
             //the pixel is marked as a set movement difference
             Difference[loop][loop2] = TRUE;
             count++;
            }

           }
          }
         }
        }
       }
      }
     }
    }
   }
  }
 }
}
// If any one of the pixel shifts Differences in a difference
//exclusion, the pixel is marked as non altered, ie only
//a small movement due to wind or BitmapCamera.Gray vibration

else Difference[loop][loop2] = FALSE;
```

```
  }
 }
 NetworkThreshold = 6 + (int)(count/5000);
}



void CCheatOfficeDlg::DoSharp()
{
 int loop,loop2, loop3, loop4;
 int count = 0;

 for (loop2=0; loop2<60; loop2++)
 {
  for ( loop=0; loop<80; loop++)
  {
    Sharp[loop][loop2] = FALSE;
    count = 0;
    for (loop3=0; loop3<4; loop3++)
    {
     for ( loop4 = 0; loop4<4; loop4++)
     {
      if (Difference[4*loop+loop3][4*loop2+loop4])
        count++;
     }
    }
    if (count >NetworkThreshold)
    {
     Sharp[loop][loop2] = TRUE;
    }
  }
 }
}



void CCheatOfficeDlg::DoObjects()
{
 int counter = 0;
 int loop, loop1, loop2, loop3;
 int Mem, Set, gap;

 for (loop = 0; loop<100; loop++)
 {
  Box[loop].Xmin = 80;
  Box[loop].Ymin = 60;
  Box[loop].Xmax = 0;
  Box[loop].Ymax = 0;
 }

 //Check every line in the image
 loop=0;
 for(loop2=0; loop2<60; loop2++)
 {
  Set = 0;
  Mem = 0;
  gap = 0;

  for (loop=0; loop<80; loop++)
  {
    if (Sharp[loop][loop2]==TRUE)
    {
     counter++;
     Box[counter].Xmin = loop;
```

```
    Box[counter].Ymin = loop2;
    Box[counter].Ymax = loop2;
    Box[counter].Xmax = loop;
    Box[counter].Xmini[loop2] = loop;
    Box[counter].Xmaxi[loop2] = loop;

    //Previous line check for position matching
    for(loop3 = 0; loop3<counter; loop3++)
    {
     if((loop>=(Box[loop3].Xmin-2))&&(loop<=(Box[loop3].Xmax+2))&&
       ((loop2-Box[loop3].Ymax)<3)&&(loop3!=counter)&&
       (loop2>0))
     {
       Set = 1;
       Mem = loop3;
     }
    }
    //End of line check

    while((gap<3)&&(loop<80))
    {
     loop++;
     if(Sharp[loop][loop2]==1)
     {
       gap = 0;
       Box[counter].Xmax = loop;
       Box[counter].Xmaxi[loop2] = loop;

       //Previous line check for position matching
       for(loop3 = 0; loop3<counter; loop3++)
       {
        if((loop>=Box[loop3].Xmin-2)&&(loop<=Box[loop3].Xmax+2)
          &&(loop2-Box[loop3].Ymax<3)&&(loop3!=counter)&&(counter>0))
         {
          Set = 1;
          Mem = loop3;
         }
       }
       //End of line check
     }
     else gap++;
    }
    gap = 0;

    //Matching correction code, updates matched object
    //and deleted new object created
    if (Set==1)
    {
     if(Box[counter].Xmin<Box[Mem].Xmin) Box[Mem].Xmin = Box[counter].Xmin;
     if(Box[counter].Xmax>Box[Mem].Xmax) Box[Mem].Xmax = Box[counter].Xmax;
     Box[Mem].Ymax = loop2;
     Box[counter].Xmin = 80;
     Box[counter].Xmax = 0;
     Box[counter].Ymin = 60;
     Box[counter].Ymax = 0;
     Box[Mem].Xmaxi[loop2] = Box[counter].Xmaxi[loop2];
     Box[Mem].Xmini[loop2] = Box[counter].Xmini[loop2];
     counter--;
     Set = 0;
     Mem = 0;
    }
   }
  }
  loop=0;
 }
 //Box size filtering code, checking for minimum target size

 int IsZero = 0;
```

```
 for ( loop = 1; loop<=counter; loop++)
 {
  if (IsZero==1)
  {
    loop--;
    IsZero = 0;
  }
  int TArea = ((Box[loop].Xmax - Box[loop].Xmin)*(Box[loop].Ymax-Box[loop].Ymin));
  if (TArea <74)
  {
    for (loop1 = loop; loop1<=counter; loop1++)
    {
      Box[loop1] = Box[loop1+1];
      if (loop==1) IsZero = 1;
      else loop--;
    }
    counter--;
  }
 }
}


void CCheatOfficeDlg::OnSetButton()
{
 // TODO: Add your control notification handler code here
 UpdateData(TRUE);
}
```

```
// Cheat OfficeDlg.h : header file
//

#if !defined(AFX_CHEATOFFICEDLG_H__C9352226_1C58_11D5_B1DA_F413A2AD906F__INCLUDED_)
#define AFX_CHEATOFFICEDLG_H__C9352226_1C58_11D5_B1DA_F413A2AD906F__INCLUDED_

#if _MSC_VER >= 1000
#pragma once
#endif // _MSC_VER >= 1000

/////////////////////////////////////////////////////////////////////////////
// CCheatOfficeDlg dialog


struct ImageInfo
{
 CString FileName;
 unsigned _int8 Data[320][240];
 int Median;
};
```

```
struct BoxInfo
{
 unsigned _int8 Xmin;
 unsigned _int8 Xmax;
 unsigned _int8 Ymin;
 unsigned _int8 Ymax;
 unsigned _int8 Xmini[60];
 unsigned _int8 Xmaxi[60];
};



class CCheatOfficeDlg : public CDialog
{
// Construction
public:
 void DoObjects();
 void DoSharp();
 void DoJitter();
 ImageInfo OpenFile(CString Name);
 CCheatOfficeDlg(CWnd* pParent = NULL);        // standard constructor

// Dialog Data
 //{{AFX_DATA(CCheatOfficeDlg)
 enum { IDD = IDD_CHEATOFFICE_DIALOG };
 BOOL m_DatumCheck;
 double m_Mult;
 //}}AFX_DATA

 // ClassWizard generated virtual function overrides
 //{{AFX_VIRTUAL(CCheatOfficeDlg)
 protected:
 virtual void DoDataExchange(CDataExchange* pDX);              // DDX/DDV support
 //}}AFX_VIRTUAL

// Implementation
protected:
 HICON m_hIcon;

 // Generated message map functions
 //{{AFX_MSG(CCheatOfficeDlg)
 virtual BOOL OnInitDialog();
 afx_msg void OnSysCommand(UINT nID, LPARAM lParam);
 afx_msg void OnPaint();
 afx_msg HCURSOR OnQueryDragIcon();
 afx_msg void OnBackgroundButton();
 afx_msg void OnDatumButton();
 afx_msg void OnIncomingButton();
 afx_msg void OnExitButton();
 afx_msg void OnProcessButton();
 afx_msg void OnSetButton();
 //}}AFX_MSG
 DECLARE_MESSAGE_MAP()

private:
 BoxInfo Box[100];
 int NetworkThreshold;
 BOOL Sharp[80][60];
 BOOL Difference[320][240];
 ImageInfo Final;
 ImageInfo Background;
 ImageInfo Datum;
 ImageInfo Camera;
};
```

```
//{{AFX_INSERT_LOCATION}}
// Microsoft Developer Studio will insert additional declarations immediately before the previous line.

#endif // !defined
(AFX_CHEATOFFICEDLG_H__C9352226_1C58_11D5_B1DA_F413A2AD906F__INCLUDED_)
```

# 20.8 - Vos Reader

```
// VosReaderDlg.cpp : implementation file
//

#include "stdafx.h"
#include "VosReader.h"
#include "VosReaderDlg.h"
#include <windowsx.h>
#include <string.h>
#include <fstream.h>
#include <iostream.h>
#include <direct.h>


#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

/////////////////////////////////////////////////////////////////////////////
// CAboutDlg dialog used for App About

class CAboutDlg : public CDialog
{
public:
 CAboutDlg();

// Dialog Data
 //{{AFX_DATA(CAboutDlg)
 enum { IDD = IDD_ABOUTBOX };
 //}}AFX_DATA

 // ClassWizard generated virtual function overrides
 //{{AFX_VIRTUAL(CAboutDlg)
 protected:
 virtual void DoDataExchange(CDataExchange* pDX);    // DDX/DDV support
 //}}AFX_VIRTUAL

// Implementation
protected:
 //{{AFX_MSG(CAboutDlg)
 //}}AFX_MSG
 DECLARE_MESSAGE_MAP()
};

CAboutDlg::CAboutDlg() : CDialog(CAboutDlg::IDD)
{
 //{{AFX_DATA_INIT(CAboutDlg)
 //}}AFX_DATA_INIT
}

void CAboutDlg::DoDataExchange(CDataExchange* pDX)
{
 CDialog::DoDataExchange(pDX);
 //{{AFX_DATA_MAP(CAboutDlg)
 //}}AFX_DATA_MAP
}

BEGIN_MESSAGE_MAP(CAboutDlg, CDialog)
 //{{AFX_MSG_MAP(CAboutDlg)
  // No message handlers
 //}}AFX_MSG_MAP
END_MESSAGE_MAP()
```

```
/////////////////////////////////////////////////////////////////////////////
// CVosReaderDlg dialog

CVosReaderDlg::CVosReaderDlg(CWnd* pParent /*=NULL*/)
 : CDialog(CVosReaderDlg::IDD, pParent)
{
 //{{AFX_DATA_INIT(CVosReaderDlg)
 m_ComboResult = _T("");
 //}}AFX_DATA_INIT
 // Note that LoadIcon does not require a subsequent DestroyIcon in Win32
 m_hIcon = AfxGetApp()->LoadIcon(IDR_MAINFRAME);
}

void CVosReaderDlg::DoDataExchange(CDataExchange* pDX)
{
 CDialog::DoDataExchange(pDX);
 //{{AFX_DATA_MAP(CVosReaderDlg)
 DDX_CBString(pDX, IDC_COMBO1, m_ComboResult);
 //}}AFX_DATA_MAP
}

BEGIN_MESSAGE_MAP(CVosReaderDlg, CDialog)
 //{{AFX_MSG_MAP(CVosReaderDlg)
 ON_WM_SYSCOMMAND()
 ON_WM_PAINT()
 ON_WM_QUERYDRAGICON()
 ON_BN_CLICKED(IDC_EXIT_BUTTON, OnExitButton)
 ON_BN_CLICKED(IDC_OPEN_BUTTON, OnOpenButton)
 ON_BN_CLICKED(IDC_RESTORE_BUTTON, OnRestoreButton)
 ON_BN_CLICKED(IDC_CREATE_BUTTON, OnCreateButton)
 ON_BN_CLICKED(IDC_APPLY_BUTTON, OnApplyButton)
 ON_BN_CLICKED(IDC_SAVE_BUTTON, OnSaveButton)
 ON_WM_DESTROY()
 ON_BN_CLICKED(IDC_VIEW_BUTTON, OnViewButton)
 ON_BN_CLICKED(IDC_DELETE_BUTTON, OnDeleteButton)
 //}}AFX_MSG_MAP
END_MESSAGE_MAP()

/////////////////////////////////////////////////////////////////////////////
// CVosReaderDlg message handlers


BOOL CVosReaderDlg::OnInitDialog()
{
 CDialog::OnInitDialog();

 // Add "About..." menu item to system menu.

 // IDM_ABOUTBOX must be in the system command range.
 ASSERT((IDM_ABOUTBOX & 0xFFF0) == IDM_ABOUTBOX);
 ASSERT(IDM_ABOUTBOX < 0xF000);

 CMenu* pSysMenu = GetSystemMenu(FALSE);
 if (pSysMenu != NULL)
 {
  CString strAboutMenu;
  strAboutMenu.LoadString(IDS_ABOUTBOX);
  if (!strAboutMenu.IsEmpty())
  {
    pSysMenu->AppendMenu(MF_SEPARATOR);
    pSysMenu->AppendMenu(MF_STRING, IDM_ABOUTBOX, strAboutMenu);
  }
 }

 // Set the icon for this dialog.  The framework does this automatically
 //  when the application's main window is not a dialog
 SetIcon(m_hIcon, TRUE);                                    // Set big icon
```

```
 SetIcon(m_hIcon, FALSE);                    // Set small icon

 // TODO: Add extra initialization here
 Set1 = 0;
 Set2 = 0;
 Set3 = 0;

 char* buffer= "";
 getcwd(buffer, 500);
 m_Working = buffer;
 CFileFind finder;
 CString FilterDir = buffer;
 FilterDir += "\\Filters";
 chdir(FilterDir);
 GetDlgItem(IDC_VIEW_BUTTON)->EnableWindow(FALSE);

 CComboBox* FilterList = (CComboBox*)GetDlgItem(IDC_COMBO1);

 BOOL bWorking = finder.FindFile("*.VRF");
 while (bWorking)
 {
  bWorking = finder.FindNextFile();
  FilterList->AddString(finder.GetFileTitle());
 }
 chdir(buffer);


 return TRUE;  // return TRUE  unless you set the focus to a control
}

void CVosReaderDlg::OnSysCommand(UINT nID, LPARAM lParam)
{
 if ((nID & 0xFFF0) == IDM_ABOUTBOX)
 {
  CAboutDlg dlgAbout;
  dlgAbout.DoModal();
 }
 else
 {
  CDialog::OnSysCommand(nID, lParam);
 }
}



//************************ GLOBALS *************************

int Stored[324][244];
int Screen[324][244];
int Change[324][244];
float Filter[25];
CString Used = "";


// If you add a minimize button to your dialog, you will need the code below
//  to draw the icon.  For MFC applications using the document/view model,
//  this is automatically done for you by the framework.

void CVosReaderDlg::OnPaint()
{
 if (IsIconic())
 {
  CPaintDC dc(this); // device context for painting

  SendMessage(WM_ICONERASEBKGND, (WPARAM) dc.GetSafeHdc(), 0);

  // Center icon in client rectangle
```

```
int cxIcon = GetSystemMetrics(SM_CXICON);
int cyIcon = GetSystemMetrics(SM_CYICON);
CRect rect;
GetClientRect(&rect);
int x = (rect.Width() - cxIcon + 1) / 2;
int y = (rect.Height() - cyIcon + 1) / 2;

// Draw the icon
dc.DrawIcon(x, y, m_hIcon);
}
else
{
CPaintDC dc(this);
if (Set3 == 0)
{
 HBITMAP hbitmap = ::LoadBitmap(::AfxGetInstanceHandle(),MAKEINTRESOURCE(IDB_BITMAP2));
 HDC hMemDC = ::CreateCompatibleDC(NULL);
 SelectObject(hMemDC,hbitmap);
 ::StretchBlt(dc.m_hDC, 12, 18, 320, 240,hMemDC, 0,0,320, 240, SRCCOPY);
 ::DeleteDC(hMemDC);
 ::DeleteObject(hbitmap);
}
else
{
 for (int loop = 0; loop <320; loop ++)
 {
  for (int loop2 = 0; loop2<240; loop2 ++)
  {
   if (Screen[loop+2][loop2+2] == 1)
    SetPixel(dc, loop+12, loop2+18,RGB(255,255,255));
   else
    SetPixel(dc, loop+12, loop2+18, RGB(0,0,0));
  }
 }
}

if (Set1 != 0)
{
 ::StretchDIBits(dc.m_hDC,
      347,
      18,
      160,
      120,
      0,
      0,
      DibWidth(Image1),
      DibHeight(Image1),
      DibPtr(Image1),
      DibInfo(Image1),
      DIB_RGB_COLORS,
      SRCCOPY);

}
else
{
 HBITMAP hbitmap = ::LoadBitmap(::AfxGetInstanceHandle(),MAKEINTRESOURCE(IDB_BITMAP1));
 HDC hMemDC = ::CreateCompatibleDC(NULL);
 SelectObject(hMemDC,hbitmap);
 ::StretchBlt(dc.m_hDC, 347, 18, 160, 120,hMemDC, 0,0,160, 120, SRCCOPY);
 ::DeleteDC(hMemDC);
 ::DeleteObject(hbitmap);
}

if (Set2 != 0)
{

  ::StretchDIBits(dc.m_hDC,
      347,
```

```
        141,
        160,
        120,
        0,
        0,
        DibWidth(Image2),
        DibHeight(Image2),
        DibPtr(Image2),
        DibInfo(Image2),
        DIB_RGB_COLORS,
        SRCCOPY);


  }
  else
  {
    HBITMAP hbitmap = ::LoadBitmap(::AfxGetInstanceHandle(),MAKEINTRESOURCE(IDB_BITMAP1));
    HDC hMemDC = ::CreateCompatibleDC(NULL);
    SelectObject(hMemDC,hbitmap);
    ::StretchBlt(dc.m_hDC, 347, 141, 160, 120,hMemDC, 0,0,160, 120, SRCCOPY);
    ::DeleteDC(hMemDC);
    ::DeleteObject(hbitmap);
  }



  CDialog::OnPaint();
 }
}

// The system calls this to obtain the cursor to display while the user drags
//  the minimized window.
HCURSOR CVosReaderDlg::OnQueryDragIcon()
{
 return (HCURSOR) m_hIcon;
}

void CVosReaderDlg::OnExitButton()
{
 // TODO: Add your control notification handler code here
 OnOK();
}



//***************** OWN FUNCTION *********************



PDIB CVosReaderDlg::DibOpenFile(LPSTR szFile)
{
 HFILE fh;
 DWORD dwLen;
 DWORD dwBits;
 PDIB pdib;
 LPVOID p;
 OFSTRUCT of;

#if defined(WIN32) || defined (_WIN32)
#define GetCurrentInstance() GetModuleHandle(NULL)
#else
#define GetCurrentInstance() (HINSTANCE)SELECTOROF((LPVOID)&of)
#endif

 fh = OpenFile(szFile, &of, OF_READ);
```

```
 if (fh == -1)
 {
  HRSRC h;
  h = FindResource(GetCurrentInstance(), szFile, RT_BITMAP);

#if defined(WIN32) || defined(_WIN32)
  if (h)
    return (PDIB)LockResource(LoadResource(GetCurrentInstance(), h));
#else
  if (h)
    fh = AccessResource(GetCurrentInstance(),h);
  #endif
 }


 if (fh == -1)
  return NULL;

 pdib = DibReadBitmapInfo(fh);

 if (!pdib)
  return NULL;

 dwBits = pdib->biSizeImage;
 dwLen = pdib->biSize + DibPaletteSize(pdib) + dwBits;

 p = GlobalReAllocPtr(pdib,dwLen,0);

 if (!p)
 {
  GlobalFreePtr(pdib);
  pdib = NULL;
 }
 else
 {
  pdib = (PDIB)p;
 }

 if (pdib)
 {
  _hread(fh, (LPBYTE)pdib + (UINT)pdib->biSize + DibPaletteSize(pdib), dwBits);
 }

 _lclose(fh);

 return pdib;

}

PDIB CVosReaderDlg::DibReadBitmapInfo(HFILE fh)
{
 DWORD off;
 HANDLE hbi = NULL;
 int size;
 int i;
 int nNumColors;

 RGBQUAD FAR *pRgb;
 BITMAPINFOHEADER bi;
 BITMAPCOREHEADER bc;
 BITMAPFILEHEADER bf;
 PDIB pdib;

 if (fh == -1)
  return NULL;

 off = _llseek(fh, 0L, SEEK_CUR);
```

```
if (sizeof(bf) != _lread(fh,(LPSTR)&bf, sizeof(bf)))
 return FALSE;

if ( bf.bfType != BFT_BITMAP)
{
 bf.bfOffBits = 0L;
 _llseek(fh, off, SEEK_SET);
}

if (sizeof(bi) != _lread(fh,(LPSTR)&bi, sizeof(bi)))
 return FALSE;

switch (size = (int)bi.biSize)
{

default :
case sizeof(BITMAPINFOHEADER):break;

case sizeof(BITMAPCOREHEADER):
     bc = *(BITMAPCOREHEADER*)&bi;
     bi.biSize = sizeof(BITMAPINFOHEADER);
     bi.biWidth = (DWORD)bc.bcWidth;
     bi.biHeight = (DWORD)bc.bcHeight;
     bi.biPlanes = (UINT)bc.bcPlanes;
     bi.biBitCount = (UINT)bc.bcBitCount;
     bi.biCompression = BI_RGB;
     bi.biSizeImage = 0;
     bi.biXPelsPerMeter = 0;
     bi.biYPelsPerMeter = 0;
     bi.biClrUsed = 0;
     bi.biClrImportant = 0;

     _llseek(fh, (LONG)sizeof(BITMAPCOREHEADER)-sizeof(BITMAPINFOHEADER),SEEK_CUR);
     break;

}

nNumColors = DibNumColors(&bi);

#if 0
 if (bi.biSizeImage == 0)
  bi.biSizeImage = DibSizeImage(&bi);

 if (bi.biClrUsed == 0)
  bi.biClrUsed = DibNumColors(&bi);
#else
 FixBitmapInfo(&bi);
#endif

pdib = (PDIB)GlobalAllocPtr(GMEM_MOVEABLE, (LONG)bi.biSize + nNumColors * sizeof(RGBQUAD));

if ( !pdib)
 return NULL;

*pdib = bi;
pRgb = DibColors(pdib);

if ( nNumColors)
{
 if ( size == sizeof(BITMAPCOREHEADER))
 {
  _lread(fh, (LPVOID)pRgb, nNumColors * sizeof (RGBTRIPLE));
  for (i = nNumColors -1; i >=0; i--)
  {
   RGBQUAD rgb;
   rgb.rgbRed = ((RGBTRIPLE FAR *)pRgb)[i].rgbtRed;
   rgb.rgbBlue = ((RGBTRIPLE FAR *)pRgb)[i].rgbtBlue;
```

```
      rgb.rgbGreen = ((RGBTRIPLE FAR *)pRgb)[i].rgbtGreen;
      rgb.rgbReserved = (BYTE)0;

      pRgb[i] = rgb;
     }
    }
    else
    {
     _lread(fh,(LPVOID)pRgb, nNumColors * sizeof(RGBQUAD));
    }
   }

   if (bf.bfOffBits != 0L)
    _llseek(fh, off + bf.bfOffBits, SEEK_SET);

   return pdib;

}

void CVosReaderDlg::OnOpenButton()
{
   // TODO: Add your control notification handler code here
   char FileTitle[100];
   char FileName[500];
   Set1 = 0;
   Set2 = 0;
   Set3 = 0;

   OPENFILENAME ofn;
   _fmemset(&ofn, 0, sizeof(ofn));
   ofn.lStructSize = sizeof(OPENFILENAME);

   ofn.hwndOwner = m_hWnd;;
   ofn.hInstance = NULL;

   ofn.lpstrFilter = TEXT("VosDemo files *.vos\0*.vos\0\0");

   ofn.lpstrCustomFilter = NULL;
   ofn.nMaxCustFilter = 0;
   ofn.nFilterIndex = 1;
   ofn.lpstrFile = FileName;
   ofn.nMaxFile = 500;
   ofn.lpstrFileTitle = FileTitle;
   ofn.nMaxFileTitle = 99;
   ofn.lpstrInitialDir = NULL;
   ofn.lpstrTitle = "Open BMP file";
   ofn.Flags = OFN_FILEMUSTEXIST;
   ofn.lpstrDefExt = "BMP";
   ofn.lCustData = NULL;
   ofn.lpfnHook = NULL;
   ofn.lpTemplateName = NULL;

   FileName[0] = '\0';

   GetOpenFileName(&ofn);


   if (FileName[0] != '\0')
   {
    CString FullName;
    CString PartName;

    FullName = FileName;
    PartName = FileTitle;

    char Name1[500];
    char Name2[500];
```

```
for (int loop = 0; loop<500; loop ++)
{
 Name1[loop] = NULL;
 Name2[loop] = NULL;
}

for (loop = 0; loop < 324; loop ++)
{
 for (int loop2 = 0; loop2 < 244; loop2 ++)
 {
  Change[loop][loop2] = 0;
  Stored[loop][loop2] = 0;
  Screen[loop][loop2] = 0;
 }
}

int Count;
Count = FullName.Find(PartName);

for (loop = 0; loop < Count; loop ++)
{
 Name1[loop] = FileName[loop];
 Name2[loop] = FileName[loop];
}

strcat(Name2, "Camera.bmp");
strcat(Name1, "Datum.bmp");

ifstream file_in(FileName);
int value1;
int value2;
do
{
 file_in >> value1;
 file_in >> value2;

 Change[value1+2][value2+2] = 1;
 Screen[value1+2][value2+2] = 1;
 Stored[value1+2][value2+2] = 1;
}
while ( file_in.eof() == 0);

for (loop = 2; loop <322; loop++)
{
 for (int loop2 = 2; loop2 <242; loop2++)
 {
  if (Stored[loop][loop2] != 1)
  {
   Stored[loop][loop2] = -1;
   Screen[loop][loop2] = -1;
   Change[loop][loop2] = -1;

  }

 }
}


Set3 = 1;

InvalidateRect(CRect(12,18,332,258), FALSE);

if(Image1 = DibOpenFile(Name1))
{
 Set1 = 1;
}
if(Image2 = DibOpenFile(Name2))
```

```
  {
   Set2 = 1;
  }
 }

 Invalidate();
}



void CVosReaderDlg::OnRestoreButton()
{
 // TODO: Add your control notification handler code here
 Used = "";
 for (int loop=0; loop<324; loop++)
 {
  for (int loop2 = 0; loop2<244; loop2++)
  {
   Screen[loop][loop2] = Stored[loop][loop2];
   Change[loop][loop2] = Stored[loop][loop2];
  }
 }
 InvalidateRect(CRect(12,18,336,262));
}



void CVosReaderDlg::OnCreateButton()
{
 // TODO: Add your control notification handler code here
 CString FilterDir = m_Working + "\\Filters" ;
 chdir(FilterDir);

 m_dlg.DoModal();

 CFileFind finder;
 CComboBox* FilterList = (CComboBox*)GetDlgItem(IDC_COMBO1);
 FilterList->ResetContent();
 BOOL bWorking = finder.FindFile("*.VRF");
 while (bWorking)
 {
  bWorking = finder.FindNextFile();
  FilterList->AddString(finder.GetFileTitle());
 }
 chdir(m_Working);


}



void CVosReaderDlg::OnApplyButton()
{
 // TODO: Add your control notification handler code here
 UpdateData(TRUE);
 m_FilterName = m_Working + "\\Filters\\" + m_ComboResult + ".vrf";
 Used = m_ComboResult;
 ifstream file_in(m_FilterName);
 char Validation1;
 char Validation2;
 char Validation3;

 file_in >> Validation1;
 file_in >> Validation2;
 file_in >> Validation3;
```

```
 if ((Validation1 =='V') && (Validation2 =='R') && ((Validation3 =='3')||(Validation3 =='5')))
 {

  for ( int loop = 0; loop<5; loop ++)
  {
    Filter[0+loop] = 0;
    Filter[1+loop] = 0;
    Filter[2+loop] = 0;
    Filter[3+loop] = 0;
    Filter[4+loop] = 0;

    file_in >> Filter[0+loop];
    file_in >> Filter[1+loop];
    file_in >> Filter[2+loop];
    file_in >> Filter[4+loop];

  }
  float Temp = 0;
  for (loop = 0; loop <320; loop ++)
  {
    for (int loop2 = 0; loop2 <240; loop2 ++)
    {
      for (int loop3 = 0; loop3<5; loop3 ++)
      {
       Temp +=(Change[loop+loop3][loop2]) * (Filter[loop3]);
       Temp +=(Change[loop+loop3][loop2+1]) * (Filter[loop3+5]);
       Temp +=(Change[loop+loop3][loop2+2]) * (Filter[loop3+10]);
       Temp +=(Change[loop+loop3][loop2+3]) * (Filter[loop3+15]);
       Temp +=(Change[loop+loop3][loop2+4]) * (Filter[loop3+20]);
      }
      if (Temp >=0)
      Screen[loop+2][loop2+2]=1;
      else Screen[loop+2][loop2+2]=-1;
      Temp = 0;
    }
  }

  for (loop = 2;loop<322; loop++);
  {
    for (int loop2 = 2; loop2<242; loop2++)
    {
     Change[loop][loop2] = Screen[loop][loop2];
    // Screen[loop][loop2] = Change[loop][loop2];
    }
  }
  InvalidateRect(CRect(12,18,336,262));
 }
 else MessageBox("Invalid Filter File");
}


void CVosReaderDlg::OnSaveButton()
{
 // TODO: Add your control notification handler code here
 CString SaveDir = "";
 SaveDir = m_Working + "\\Saved Data";
 chdir(SaveDir);

 CFileFind finder;
 BOOL bWorking = finder.FindFile("*.VRD");
 int Trial = 1;
 while (bWorking)
 {
  bWorking = finder.FindNextFile();
  if (Trial == atoi(finder.GetFileTitle()))
  {
    Trial ++;
```

```
 }
 }
 char* Temp = "";
 itoa(Trial, Temp,10);
 CString NewName = Temp;
 NewName += ".vrd";

 UpdateData(TRUE);

 ofstream SaveFile(NewName);

 SaveFile << "VRD\n";
 SaveFile << "Filter used : ";
 SaveFile << Used <<"\n";
 SaveFile << "Original Data - Filtered Data :\n";
 for (int loop = 2; loop <322; loop++)
 {
  for (int loop2 = 2; loop2 <242; loop2++)
  {
    SaveFile << Stored[loop][loop2]<<"\t"<<Screen[loop][loop2]<<"\n";
  }
 }
 SaveFile.close();
 chdir(m_Working);

}



void CVosReaderDlg::OnDestroy()
{
 CDialog::OnDestroy();

 // TODO: Add your message handler code here
  GlobalFreePtr(Image1);
  GlobalFreePtr(Image2);
}

void CVosReaderDlg::OnViewButton()
{
 // TODO: Add your control notification handler code here
// chdir(m_Working);
// m_dlg.DoModal();




}

void CVosReaderDlg::OnDeleteButton()
{
 // TODO: Add your control notification handler code here
 CString FilterDir = m_Working + "\\Filters" ;
 chdir(FilterDir);

 UpdateData(TRUE);

 CString Temp = m_ComboResult;
 m_FilterName = m_ComboResult + ".vrf";

 if ((Temp == "") || (DeleteFile(m_FilterName) == 0))
 {
  MessageBox("File could not be found");
 }
 else
 {
  CFileFind finder;
```

```
  CComboBox* FilterList = (CComboBox*)GetDlgItem(IDC_COMBO1);
  FilterList->ResetContent();
  BOOL bWorking = finder.FindFile("*.VRF");
  while (bWorking)
  {
    bWorking = finder.FindNextFile();
    FilterList->AddString(finder.GetFileTitle());
  }
 }

 chdir(m_Working);

}

// VosReaderDlg.h : header file

#include "Editor.h"
//

#if !defined(AFX_VOSREADERDLG_H__CA31C186_4268_11D2_BA0C_0060084F84CD__INCLUDED_)
#define AFX_VOSREADERDLG_H__CA31C186_4268_11D2_BA0C_0060084F84CD__INCLUDED_

#if _MSC_VER >= 1000
#pragma once
#endif // _MSC_VER >= 1000

/////////////////////////////////////////////////////////////////////////////
// CVosReaderDlg dialog
//Overhead declarations for DIB manipulation
typedef LPBITMAPINFOHEADER PDIB;

#define DibWidth(lpbi) \
 (UINT)(((LPBITMAPINFOHEADER)(lpbi))->biWidth)

#define DibHeight(lpbi) \
 (UINT)(((LPBITMAPINFOHEADER)(lpbi))->biHeight)

#define DibColors(lpbi) \
 ((RGBQUAD FAR *)((LPBYTE)(lpbi) + (int)(lpbi)->biSize))

#ifdef WIN32
#define DibPtr(lpbi) \
 ((lpbi)->biCompression == BI_BITFIELDS \
 ? (LPVOID)(DibColors(lpbi) + 3)\
 : (LPVOID)(DibColors(lpbi) + (UINT)(lpbi)->biClrUsed))
#else
#define DibPtr(lpbi) \
 (LPVOID)(DibColors(lpbi) + (UINT)(lpbi)->biClrUsed)
#endif

#define DibInfo(pDIB) \
 ((BITMAPINFO FAR *)(pDIB))

#define DibNumColors(lpbi) \
 ((lpbi)->biClrUsed == 0 && (lpbi)->biBitCount <= 8 \
 ? (int)(1 << (int)(lpbi)->biBitCount) \
 : (int)(lpbi)->biClrUsed)

#define DibPaletteSize(lpbi) \
 (DibNumColors(lpbi) * sizeof(RGBQUAD))

#define BFT_BITMAP 0x4d42

#define WIDTHBYTES(i) \
 ((unsigned)((i+31)&(~31))/8)

#define DibWidthBytesN(lpbi, n) \
 (UINT)WIDTHBYTES((UINT)(lpbi)->biWidth * (UINT)(n))
```

```
#define DibWidthBytes(lpbi) \
 DibWidthBytesN(lpbi, (lpbi)->biBitCount)

#define DibSizeImage(lpbi) \
 ((lpbi)->biSizeImage == 0 \
 ? ((DWORD)(UINT)DibWidthBytes(lpbi) * (DWORD)(UINT)(lpbi)->biHeight) \
 : (lpbi)->biSizeImage)
#ifndef BI_BITFIELDS
#define BI_BITFIELDS 3
#endif
#define FixBitmapInfo(lpbi) \
 if ((lpbi)->biSizeImage == 0) \
 (lpbi)->biSizeImage = DibSizeImage(lpbi); \
 if ((lpbi)->biClrUsed == 0) \
 (lpbi)->biClrUsed = DibNumColors(lpbi); \
 if ((lpbi)->biCompression == BI_BITFIELDS && (lpbi)->biClrUsed == 0)


class CVosReaderDlg : public CDialog
{
// Construction
public:

 int Set1;
 int Set2;
 int Set3;
 PDIB Image1;
 PDIB Image2;
 CString m_FilterName;
 CString m_Working;



 CVosReaderDlg(CWnd* pParent = NULL);        // standard constructor

 CEditor m_dlg;

// Dialog Data
 //{{AFX_DATA(CVosReaderDlg)
 enum { IDD = IDD_VOSREADER_DIALOG };
 CString m_ComboResult;
 //}}AFX_DATA

 // ClassWizard generated virtual function overrides
 //{{AFX_VIRTUAL(CVosReaderDlg)
 protected:
 virtual void DoDataExchange(CDataExchange* pDX);              // DDX/DDV support
 //}}AFX_VIRTUAL

// Implementation
protected:
 HICON m_hIcon;
 PDIB DibOpenFile(LPSTR szFile);
 PDIB DibReadBitmapInfo(HFILE fh);

 // Generated message map functions
 //{{AFX_MSG(CVosReaderDlg)
 virtual BOOL OnInitDialog();
 afx_msg void OnSysCommand(UINT nID, LPARAM lParam);
 afx_msg void OnPaint();
 afx_msg HCURSOR OnQueryDragIcon();
 afx_msg void OnExitButton();
 afx_msg void OnOpenButton();
 afx_msg void OnRestoreButton();
 afx_msg void OnCreateButton();
 afx_msg void OnApplyButton();
 afx_msg void OnSaveButton();
```

```
 afx_msg void OnDestroy();
 afx_msg void OnViewButton();
 afx_msg void OnDeleteButton();
 //}}AFX_MSG
 DECLARE_MESSAGE_MAP()
};

//{{AFX_INSERT_LOCATION}}
// Microsoft Developer Studio will insert additional declarations immediately before the previous line.

#endif // !defined(AFX_VOSREADERDLG_H__CA31C186_4268_11D2_BA0C_0060084F84CD__INCLUDED_)
```

```
// Editor.cpp : implementation file
//

#include "stdafx.h"
#include "VosReader.h"
#include "Editor.h"
#include <fstream.h>
#include <direct.h>

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

/////////////////////////////////////////////////////////////////////////////
// CEditor dialog


CEditor::CEditor(CWnd* pParent /*=NULL*/)
 : CDialog(CEditor::IDD, pParent)
{
 //{{AFX_DATA_INIT(CEditor)
 m_Input0 = 0.0f;
 m_Input1 = 0.0f;
 m_Input10 = 0.0f;
 m_Input11 = 0.0f;
 m_Input12 = 0.0f;
 m_Input13 = 0.0f;
 m_Input14 = 0.0f;
 m_Input15 = 0.0f;
 m_Input16 = 0.0f;
 m_Input17 = 0.0f;
 m_Input18 = 0.0f;
 m_Input19 = 0.0f;
 m_Input2 = 0.0f;
 m_Input20 = 0.0f;
```

```
  m_Input21 = 0.0f;
  m_Input22 = 0.0f;
  m_Input23 = 0.0f;
  m_Input24 = 0.0f;
  m_Input3 = 0.0f;
  m_Input4 = 0.0f;
  m_Input5 = 0.0f;
  m_Input6 = 0.0f;
  m_Input7 = 0.0f;
  m_Input8 = 0.0f;
  m_Input9 = 0.0f;
  m_FilterName = _T("");
  //}}AFX_DATA_INIT
 }


 void CEditor::DoDataExchange(CDataExchange* pDX)
 {
  CDialog::DoDataExchange(pDX);
  //{{AFX_DATA_MAP(CEditor)
  DDX_Text(pDX, IDC_EDIT0, m_Input0);
  DDX_Text(pDX, IDC_EDIT1, m_Input1);
  DDX_Text(pDX, IDC_EDIT10, m_Input10);
  DDX_Text(pDX, IDC_EDIT11, m_Input11);
  DDX_Text(pDX, IDC_EDIT12, m_Input12);
  DDX_Text(pDX, IDC_EDIT13, m_Input13);
  DDX_Text(pDX, IDC_EDIT14, m_Input14);
  DDX_Text(pDX, IDC_EDIT15, m_Input15);
  DDX_Text(pDX, IDC_EDIT16, m_Input16);
  DDX_Text(pDX, IDC_EDIT17, m_Input17);
  DDX_Text(pDX, IDC_EDIT18, m_Input18);
  DDX_Text(pDX, IDC_EDIT19, m_Input19);
  DDX_Text(pDX, IDC_EDIT2, m_Input2);
  DDX_Text(pDX, IDC_EDIT20, m_Input20);
  DDX_Text(pDX, IDC_EDIT21, m_Input21);
  DDX_Text(pDX, IDC_EDIT22, m_Input22);
  DDX_Text(pDX, IDC_EDIT23, m_Input23);
  DDX_Text(pDX, IDC_EDIT24, m_Input24);
  DDX_Text(pDX, IDC_EDIT3, m_Input3);
  DDX_Text(pDX, IDC_EDIT4, m_Input4);
  DDX_Text(pDX, IDC_EDIT5, m_Input5);
  DDX_Text(pDX, IDC_EDIT6, m_Input6);
  DDX_Text(pDX, IDC_EDIT7, m_Input7);
  DDX_Text(pDX, IDC_EDIT8, m_Input8);
  DDX_Text(pDX, IDC_EDIT9, m_Input9);
  DDX_Text(pDX, IDC_FILTERNAME_EDIT, m_FilterName);
  //}}AFX_DATA_MAP
 }


BEGIN_MESSAGE_MAP(CEditor, CDialog)
 //{{AFX_MSG_MAP(CEditor)
 ON_BN_CLICKED(IDC_CANCEL_BUTTON, OnCancelButton)
 ON_BN_CLICKED(IDC_SAVE2_BUTTON, OnSave2Button)
 //}}AFX_MSG_MAP
END_MESSAGE_MAP()

////////////////////////////////////////////////////////////////////////////
// CEditor message handlers

void CEditor::OnCancelButton()
{
 // TODO: Add your control notification handler code here
 OnOK();
}
```

```
void CEditor::OnSave2Button()
{
 // TODO: Add your control notification handler code here
 CString Name = "";
 m_FilterName = "";
 UpdateData(TRUE);
 CFileFind finder;
 int Error = 0;
 chdir("Filters");

 BOOL bWorking = finder.FindFile("*.VRF");
 while (bWorking)
 {
  bWorking = finder.FindNextFile();
  if (finder.GetFileTitle() == m_FilterName)
  {
    Error = 1;
  }

 }
 if (m_FilterName =="")
 {
  Error = 2;
 }


 switch (Error)
 {
 case 0:{
    Name = m_FilterName + ".vrf";
    ofstream file_out(Name);
    file_out << "VR5\n";
    file_out << m_Input0 << "\t"
<<m_Input1<<"\t"<<m_Input2<<"\t"<<m_Input3<<"\t"<<m_Input4<<"\n";
    file_out << m_Input5 << "\t"
<<m_Input6<<"\t"<<m_Input7<<"\t"<<m_Input8<<"\t"<<m_Input9<<"\n";
    file_out << m_Input10 << "\t"
<<m_Input11<<"\t"<<m_Input12<<"\t"<<m_Input13<<"\t"<<m_Input14<<"\n";
    file_out << m_Input15<< "\t"
<<m_Input16<<"\t"<<m_Input17<<"\t"<<m_Input18<<"\t"<<m_Input19<<"\n";
    file_out << m_Input20 << "\t"
<<m_Input21<<"\t"<<m_Input22<<"\t"<<m_Input23<<"\t"<<m_Input24<<"\n";
    file_out.close();

    m_Input0 = 0;
    m_Input1 = 0;
    m_Input2 = 0;
    m_Input3 = 0;
    m_Input4 = 0;
    m_Input5 = 0;
    m_Input6 = 0;
    m_Input7 = 0;
    m_Input8 = 0;
    m_Input9 = 0;
    m_Input10 = 0;
    m_Input11 = 0;
    m_Input12 = 0;
    m_Input13 = 0;
    m_Input14 = 0;
    m_Input15 = 0;
    m_Input16 = 0;
    m_Input17 = 0;
    m_Input18 = 0;
    m_Input19 = 0;
    m_Input20 = 0;
    m_Input21 = 0;
    m_Input22 = 0;
    m_Input23 = 0;
```

```
    m_Input24 = 0;

    UpdateData(FALSE);
    OnOK();
    }
    break;

  case 1:MessageBox("File already exists");
    break;

  case 2:MessageBox("Enter a Filter Name");
    break;
 }

 m_FilterName = "";


}
```

```
#if !defined(AFX_EDITOR_H__60F2F6E0_4B4C_11D2_BA11_0060084F84CD__INCLUDED_)
#define AFX_EDITOR_H__60F2F6E0_4B4C_11D2_BA11_0060084F84CD__INCLUDED_

#if _MSC_VER >= 1000
#pragma once
#endif // _MSC_VER >= 1000
// Editor.h : header file
//

/////////////////////////////////////////////////////////////////////////////
// CEditor dialog

class CEditor : public CDialog
{
```

```
// Construction
public:
 CEditor(CWnd* pParent = NULL);   // standard constructor


// Dialog Data
 //{{AFX_DATA(CEditor)
 enum { IDD = IDD_EDIT_DIALOG };
 float m_Input0;
 float m_Input1;
 float m_Input10;
 float m_Input11;
 float m_Input12;
 float m_Input13;
 float m_Input14;
 float m_Input15;
 float m_Input16;
 float m_Input17;
 float m_Input18;
 float m_Input19;
 float m_Input2;
 float m_Input20;
 float m_Input21;
 float m_Input22;
 float m_Input23;
 float m_Input24;
 float m_Input3;
 float m_Input4;
 float m_Input5;
 float m_Input6;
 float m_Input7;
 float m_Input8;
 float m_Input9;
 CString m_FilterName;
 //}}AFX_DATA


// Overrides
 // ClassWizard generated virtual function overrides
 //{{AFX_VIRTUAL(CEditor)
 protected:
 virtual void DoDataExchange(CDataExchange* pDX);    // DDX/DDV support
 //}}AFX_VIRTUAL

// Implementation
protected:

 // Generated message map functions
 //{{AFX_MSG(CEditor)
 afx_msg void OnCancelButton();
 afx_msg void OnSave2Button();
 //}}AFX_MSG
 DECLARE_MESSAGE_MAP()
};

//{{AFX_INSERT_LOCATION}}
// Microsoft Developer Studio will insert additional declarations immediately before the previous line.

#endif // !defined(AFX_EDITOR_H__60F2F6E0_4B4C_11D2_BA11_0060084F84CD__INCLUDED_)
```

# 20.9 - VosViewer

```
// vos viewerDlg.cpp : implementation file
//

#include "stdafx.h"
#include "vos viewer.h"
#include "vos viewerDlg.h"
#include <fstream.h>
#include <direct.h>

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

/////////////////////////////////////////////////////////////////////////////
// CAboutDlg dialog used for App About

class CAboutDlg : public CDialog
{
public:
 CAboutDlg();

// Dialog Data
 //{{AFX_DATA(CAboutDlg)
 enum { IDD = IDD_ABOUTBOX };
 //}}AFX_DATA

 // ClassWizard generated virtual function overrides
 //{{AFX_VIRTUAL(CAboutDlg)
 protected:
 virtual void DoDataExchange(CDataExchange* pDX);    // DDX/DDV support
 //}}AFX_VIRTUAL

// Implementation
protected:
 //{{AFX_MSG(CAboutDlg)
 //}}AFX_MSG
 DECLARE_MESSAGE_MAP()
};

CAboutDlg::CAboutDlg() : CDialog(CAboutDlg::IDD)
{
 //{{AFX_DATA_INIT(CAboutDlg)
 //}}AFX_DATA_INIT
}

void CAboutDlg::DoDataExchange(CDataExchange* pDX)
{
 CDialog::DoDataExchange(pDX);
 //{{AFX_DATA_MAP(CAboutDlg)
 //}}AFX_DATA_MAP
}

BEGIN_MESSAGE_MAP(CAboutDlg, CDialog)
 //{{AFX_MSG_MAP(CAboutDlg)
  // No message handlers
 //}}AFX_MSG_MAP
END_MESSAGE_MAP()

/////////////////////////////////////////////////////////////////////////////
// CVosviewerDlg dialog
```

```
CVosviewerDlg::CVosviewerDlg(CWnd* pParent /*=NULL*/)
 : CDialog(CVosviewerDlg::IDD, pParent)
{
 //{{AFX_DATA_INIT(CVosviewerDlg)
 m_Name = _T("");
 //}}AFX_DATA_INIT
 // Note that LoadIcon does not require a subsequent DestroyIcon in Win32
 m_hIcon = AfxGetApp()->LoadIcon(IDR_MAINFRAME);
}

void CVosviewerDlg::DoDataExchange(CDataExchange* pDX)
{
 CDialog::DoDataExchange(pDX);
 //{{AFX_DATA_MAP(CVosviewerDlg)
 DDX_Text(pDX, IDC_NAME_EDIT, m_Name);
 //}}AFX_DATA_MAP
}

BEGIN_MESSAGE_MAP(CVosviewerDlg, CDialog)
 //{{AFX_MSG_MAP(CVosviewerDlg)
 ON_WM_SYSCOMMAND()
 ON_WM_PAINT()
 ON_WM_QUERYDRAGICON()
 ON_BN_CLICKED(IDC_COPY_BUTTON, OnCopyButton)
 ON_BN_CLICKED(IDC_DEL_BUTTON, OnDelButton)
 ON_BN_CLICKED(IDC_NEXT_BUTTON, OnNextButton)
 ON_BN_CLICKED(IDC_OPEN_BUTTON, OnOpenButton)
 ON_BN_CLICKED(IDC_PREV_BUTTON, OnPrevButton)
 ON_BN_CLICKED(IDC_QUIT_BUTTON, OnQuitButton)
 ON_BN_CLICKED(IDC_COL1_BUTTON, OnCol1Button)
 ON_BN_CLICKED(IDC_COL2_BUTTON, OnCol2Button)
 ON_BN_CLICKED(IDC_HOME_BUTTON, OnHomeButton)
 ON_BN_CLICKED(IDC_LAST_BUTTON, OnLastButton)
 //}}AFX_MSG_MAP
END_MESSAGE_MAP()

/////////////////////////////////////////////////////////////////////////////
// CVosviewerDlg message handlers

BOOL CVosviewerDlg::OnInitDialog()
{
 CDialog::OnInitDialog();

 // Add "About..." menu item to system menu.

 // IDM_ABOUTBOX must be in the system command range.
 ASSERT((IDM_ABOUTBOX & 0xFFF0) == IDM_ABOUTBOX);
 ASSERT(IDM_ABOUTBOX < 0xF000);

 CMenu* pSysMenu = GetSystemMenu(FALSE);
 if (pSysMenu != NULL)
 {
  CString strAboutMenu;
  strAboutMenu.LoadString(IDS_ABOUTBOX);
  if (!strAboutMenu.IsEmpty())
  {
   pSysMenu->AppendMenu(MF_SEPARATOR);
   pSysMenu->AppendMenu(MF_STRING, IDM_ABOUTBOX, strAboutMenu);
  }
 }

 // Set the icon for this dialog.  The framework does this automatically
 //  when the application's main window is not a dialog
 SetIcon(m_hIcon, TRUE);                                    // Set big icon
 SetIcon(m_hIcon, FALSE);                    // Set small icon

 // TODO: Add extra initialization here
```

```
 c_BackColor = RGB(0,0,0);
 c_ForeColor = RGB(255,255,0);
 GetDlgItem(IDC_COPY_BUTTON)->EnableWindow(FALSE);
 GetDlgItem(IDC_DEL_BUTTON)->EnableWindow(FALSE);
 GetDlgItem(IDC_PREV_BUTTON)->EnableWindow(FALSE);
 GetDlgItem(IDC_NEXT_BUTTON)->EnableWindow(FALSE);
 GetDlgItem(IDC_HOME_BUTTON)->EnableWindow(FALSE);
 GetDlgItem(IDC_LAST_BUTTON)->EnableWindow(FALSE);
 FileOpen = FALSE;
 return TRUE;  // return TRUE  unless you set the focus to a control
}

void CVosviewerDlg::OnSysCommand(UINT nID, LPARAM lParam)
{
 if ((nID & 0xFFF0) == IDM_ABOUTBOX)
 {
  CAboutDlg dlgAbout;
  dlgAbout.DoModal();
 }
 else
 {
  CDialog::OnSysCommand(nID, lParam);
 }
}

// If you add a minimize button to your dialog, you will need the code below
//  to draw the icon.  For MFC applications using the document/view model,
//  this is automatically done for you by the framework.

void CVosviewerDlg::OnPaint()
{
 if (IsIconic())
 {
  CPaintDC dc(this); // device context for painting

  SendMessage(WM_ICONERASEBKGND, (WPARAM) dc.GetSafeHdc(), 0);

  // Center icon in client rectangle
  int cxIcon = GetSystemMetrics(SM_CXICON);
  int cyIcon = GetSystemMetrics(SM_CYICON);
  CRect rect;
  GetClientRect(&rect);
  int x = (rect.Width() - cxIcon + 1) / 2;
  int y = (rect.Height() - cyIcon + 1) / 2;

  // Draw the icon
  dc.DrawIcon(x, y, m_hIcon);
 }
 else
 {
  if (FileOpen==TRUE) DisplayFile();
  CDialog::OnPaint();
 }
}

// The system calls this to obtain the cursor to display while the user drags
//  the minimized window.
HCURSOR CVosviewerDlg::OnQueryDragIcon()
{
 return (HCURSOR) m_hIcon;
}

void CVosviewerDlg::OnCopyButton()
{
 // TODO: Add your control notification handler code here
 CFileDialog m_FileDlg(FALSE,"vos",m_FileName);
 if(m_FileDlg.DoModal()==IDOK)
 {
```

```
  CString Path = m_FileDlg.GetPathName();
  CopyFile(m_PathName, Path,TRUE);
 }
}

void CVosviewerDlg::OnDelButton()
{
 // TODO: Add your control notification handler code here
 if(MessageBox("Do you really want to delete this File ?","FileDelete", MB_YESNO)==IDYES)
  if(DeleteFile(m_FileName))
  {
   MessageBox("File Deleted");
   m_Name = "";
   GetDlgItem(IDC_DEL_BUTTON)->EnableWindow(FALSE);
   GetDlgItem(IDC_COPY_BUTTON)->EnableWindow(FALSE);
   FileOpen = FALSE;
   UpdateData(FALSE);
   Invalidate();
  }
  else MessageBox("Cannot delete File");
}

void CVosviewerDlg::OnNextButton()
{
 // TODO: Add your control notification handler code here
 CFileFind Finder;
 BOOL bWorking = Finder.FindFile("*.vos");
 int iResult = 0;
 CString m_Temp;
 while (bWorking)
 {
  bWorking = Finder.FindNextFile();          //Try to find the current file
  m_Temp = Finder.GetFileName();
  if ((m_Temp == m_FileName) && (bWorking))
  {
   bWorking = Finder.FindNextFile();         //When found, find the next file
   m_NextFile = Finder.GetFileName();
   m_PathName = Finder.GetFilePath();
   m_FileName = m_NextFile;
   iResult = DisplayFile();
   bWorking = FALSE;
  }
  else if(!bWorking) MessageBox("End of File List");
 }
}

void CVosviewerDlg::OnOpenButton()
{
 // TODO: Add your control notification handler code here
 CFileDialog m_Filedlg(TRUE,"VOS data file","*.vos");
 if(m_Filedlg.DoModal()==IDOK)
 {
  m_FileName = m_Filedlg.GetFileName();
  m_PathName = m_Filedlg.GetPathName();
  int iResult = DisplayFile();
 }
}

void CVosviewerDlg::OnPrevButton()
{
 // TODO: Add your control notification handler code here
 CFileFind Finder;
 BOOL bWorking = Finder.FindFile("*.vos");
 int iCount = 0;
 int iResult = 0;
 CString m_Temp;

 bWorking = Finder.FindNextFile();
```

```
 while(bWorking)
 {
  m_PrevFile = Finder.GetFileName();
  m_PathName = Finder.GetFilePath();

  bWorking = Finder.FindNextFile();
  if (m_FileName==Finder.GetFileName())
  {
    bWorking=FALSE;
    m_FileName = m_PrevFile;
    iCount = 1;
  }
 }
 if (iCount==0) MessageBox("No Previous Files");
 else iResult = DisplayFile();
}

void CVosviewerDlg::OnQuitButton()
{
 // TODO: Add your control notification handler code here
 OnOK();
}

BOOL CVosviewerDlg::DisplayFile()
{

 m_Name = m_PathName;
 UpdateData(FALSE);

 GetDlgItem(IDC_DEL_BUTTON)->EnableWindow(TRUE);
 GetDlgItem(IDC_COPY_BUTTON)->EnableWindow(TRUE);
 GetDlgItem(IDC_PREV_BUTTON)->EnableWindow(TRUE);
 GetDlgItem(IDC_NEXT_BUTTON)->EnableWindow(TRUE);
 GetDlgItem(IDC_HOME_BUTTON)->EnableWindow(TRUE);
 GetDlgItem(IDC_LAST_BUTTON)->EnableWindow(TRUE);

 ifstream InFile(m_FileName);
 DWORD Attribs = GetFileAttributes(m_FileName);
 if (Attribs == FILE_ATTRIBUTE_READONLY)
 {
  MessageBox("File is Read Only");
  GetDlgItem(IDC_DEL_BUTTON)->EnableWindow(FALSE);
 }
// if(GetFileSize(InFile,NULL)==0xFFFFFFFF)
 if (sizeof(InFile)==0)
 {
  MessageBox("File is Void");
  GetDlgItem(IDC_COPY_BUTTON)->EnableWindow(FALSE);
  return FALSE;
 }

 FileOpen = TRUE;
 CClientDC dc(this);

 int value1;
 int value2;

 for(int loop = 0; loop<320; loop++)
 {
  for ( int loop2 = 0; loop2 <240; loop2++)
  {
    SetPixel(dc, loop+15, loop2+15,c_BackColor);
  }
 }

 do
 {
  InFile >> value1;
```

```
  InFile >> value2;
  SetPixel(dc,value1+15,value2+15,c_ForeColor);
 }
 while ( InFile.eof() == 0);

 InFile.close();

 return TRUE;
}

void CVosviewerDlg::OnCol1Button()
{
 // TODO: Add your control notification handler code here
 CColorDialog m_Coldlg(TRUE);
 m_Coldlg.DoModal();
 c_BackColor = m_Coldlg.GetColor();
}

void CVosviewerDlg::OnCol2Button()
{
 // TODO: Add your control notification handler code here
 CColorDialog m_Coldlg(TRUE);
 m_Coldlg.DoModal();
 c_ForeColor = m_Coldlg.GetColor();
}

void CVosviewerDlg::OnHomeButton()
{
 // TODO: Add your control notification handler code here
  // TODO: Add your control notification handler code here
 CFileFind Finder;
 BOOL bWorking = Finder.FindFile("*.vos");
 CString m_Temp;
 int iResult;

 bWorking = Finder.FindNextFile();
 m_FileName = Finder.GetFileName();
 m_PathName = Finder.GetFilePath();
 iResult = DisplayFile();

}

void CVosviewerDlg::OnLastButton()
{
 // TODO: Add your control notification handler code here
  CFileFind Finder;
 BOOL bWorking = Finder.FindFile("*.vos");
 CString m_Temp;
 int iResult;
 while(bWorking)
 {
  bWorking = Finder.FindNextFile();
 }
 m_FileName = Finder.GetFileName();
 m_PathName = Finder.GetFilePath();
 iResult = DisplayFile();
}
```

# 20.10 - Weyrad Demo

```
// Demo 2Dlg.cpp : implementation file
//

#include "stdafx.h"
#include "Demo 2.h"
#include "Demo 2Dlg.h"
#include <string.h>
#include <windowsx.h>
#include <direct.h>
#include <fstream.h>
#include <math.h>
#include <afx.h>

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

/////////////////////////////////////////////////////////////////////////////
// CAboutDlg dialog used for App About

class CAboutDlg : public CDialog
{
public:
 CAboutDlg();

// Dialog Data
 //{{AFX_DATA(CAboutDlg)
 enum { IDD = IDD_ABOUTBOX };
 //}}AFX_DATA

 // ClassWizard generated virtual function overrides
 //{{AFX_VIRTUAL(CAboutDlg)
 protected:
 virtual void DoDataExchange(CDataExchange* pDX);    // DDX/DDV support
 //}}AFX_VIRTUAL

// Implementation
protected:
 //{{AFX_MSG(CAboutDlg)
 //}}AFX_MSG
 DECLARE_MESSAGE_MAP()
};

CAboutDlg::CAboutDlg() : CDialog(CAboutDlg::IDD)
{
 //{{AFX_DATA_INIT(CAboutDlg)
 //}}AFX_DATA_INIT
}

void CAboutDlg::DoDataExchange(CDataExchange* pDX)
{
 CDialog::DoDataExchange(pDX);
 //{{AFX_DATA_MAP(CAboutDlg)
 //}}AFX_DATA_MAP
}

BEGIN_MESSAGE_MAP(CAboutDlg, CDialog)
 //{{AFX_MSG_MAP(CAboutDlg)
  // No message handlers
 //}}AFX_MSG_MAP
END_MESSAGE_MAP()

/////////////////////////////////////////////////////////////////////////////
```

```
// CDemo2Dlg dialog

CDemo2Dlg::CDemo2Dlg(CWnd* pParent /*=NULL*/)
 : CDialog(CDemo2Dlg::IDD, pParent)
{
 //{{AFX_DATA_INIT(CDemo2Dlg)
 m_Source = _T("");
 m_SaveEdit = _T("");
 m_Batch = FALSE;
 m_Mult = 0;
 //}}AFX_DATA_INIT
 // Note that LoadIcon does not require a subsequent DestroyIcon in Win32
 m_hIcon = AfxGetApp()->LoadIcon(IDR_MAINFRAME);
}

void CDemo2Dlg::DoDataExchange(CDataExchange* pDX)
{
 CDialog::DoDataExchange(pDX);
 //{{AFX_DATA_MAP(CDemo2Dlg)
 DDX_Control(pDX, IDC_START_BUTTON, m_Start);
 DDX_Text(pDX, IDC_SOURCE_EDIT, m_Source);
 DDX_Text(pDX, IDC_SAVE_EDIT, m_SaveEdit);
 DDX_Check(pDX, IDC_BATCH_CHECK, m_Batch);
 DDX_Text(pDX, IDC_UPDATE_EDIT, m_Mult);
 //}}AFX_DATA_MAP
}

BEGIN_MESSAGE_MAP(CDemo2Dlg, CDialog)
 //{{AFX_MSG_MAP(CDemo2Dlg)
 ON_WM_SYSCOMMAND()
 ON_WM_PAINT()
 ON_WM_QUERYDRAGICON()
 ON_BN_CLICKED(IDC_EXIT_BUTTON, OnExitButton)
 ON_BN_CLICKED(IDC_GRAB1_BUTTON, OnGrab1Button)
 ON_BN_CLICKED(IDC_GRAB2_BUTTON, OnGrab2Button)
 ON_BN_CLICKED(IDC_START_BUTTON, OnStartButton)
 ON_BN_CLICKED(IDC_SELECT_BUTTON, OnSelectButton)
 ON_WM_DESTROY()
 ON_BN_CLICKED(IDC_SAVESELECT_BUTTON, OnSaveselectButton)
 ON_BN_CLICKED(IDC_BATCH_CHECK, OnBatchCheck)
 ON_BN_CLICKED(IDC_UPDATE_BUTTON, OnUpdateButton)
 //}}AFX_MSG_MAP
END_MESSAGE_MAP()

/////////////////////////////////////////////////////////////////////////////
// CDemo2Dlg message handlers

BOOL CDemo2Dlg::OnInitDialog()
{
 CDialog::OnInitDialog();

 // Add "About..." menu item to system menu.

 // IDM_ABOUTBOX must be in the system command range.
 ASSERT((IDM_ABOUTBOX & 0xFFF0) == IDM_ABOUTBOX);
 ASSERT(IDM_ABOUTBOX < 0xF000);

 CMenu* pSysMenu = GetSystemMenu(FALSE);
 if (pSysMenu != NULL)
 {
  CString strAboutMenu;
  strAboutMenu.LoadString(IDS_ABOUTBOX);
  if (!strAboutMenu.IsEmpty())
  {
   pSysMenu->AppendMenu(MF_SEPARATOR);
   pSysMenu->AppendMenu(MF_STRING, IDM_ABOUTBOX, strAboutMenu);
  }
 }
```

```
// Set the icon for this dialog.  The framework does this automatically
//  when the application's main window is not a dialog
SetIcon(m_hIcon, TRUE);                                    // Set big icon
SetIcon(m_hIcon, FALSE);                        // Set small icon

// TODO: Add extra initialization here
ButtonText="&START";
UpdateData(FALSE);
Selected = 0;
Selected2 = 0;
grabbed = 0;
m_Mult = 5;


UpdateData(FALSE);
GetDlgItem(IDC_START_BUTTON)->EnableWindow(FALSE);
GetDlgItem(IDC_GRAB1_BUTTON)->EnableWindow(FALSE);
GetDlgItem(IDC_GRAB2_BUTTON)->EnableWindow(FALSE);



 return TRUE;  // return TRUE  unless you set the focus to a control
}

void CDemo2Dlg::OnSysCommand(UINT nID, LPARAM lParam)
{
 if ((nID & 0xFFF0) == IDM_ABOUTBOX)
 {
  CAboutDlg dlgAbout;
  dlgAbout.DoModal();
 }
 else
 {
  CDialog::OnSysCommand(nID, lParam);
 }
}

// If you add a minimize button to your dialog, you will need the code below
//  to draw the icon.  For MFC applications using the document/view model,
//  this is automatically done for you by the framework.


//********************** GLOBAL VARIABLES DECLARATIONS ***************************

int Camera[320][240];
int Reference[320][240];
int Result[320][240];
PDIB m_datum;
int Iterations = 10;
int Count = 0;
int DatumThreshold = 0;
int CameraThreshold = 0;
int Len = 0;
CString m_SourceDir;
//*************************************************************************************



void CDemo2Dlg::OnPaint()
{
 if (IsIconic())
 {
  CPaintDC dc(this); // device context for painting

  SendMessage(WM_ICONERASEBKGND, (WPARAM) dc.GetSafeHdc(), 0);

  // Center icon in client rectangle
```

```
int cxIcon = GetSystemMetrics(SM_CXICON);
int cyIcon = GetSystemMetrics(SM_CYICON);
CRect rect;
GetClientRect(&rect);
int x = (rect.Width() - cxIcon + 1) / 2;
int y = (rect.Height() - cyIcon + 1) / 2;

// Draw the icon
dc.DrawIcon(x, y, m_hIcon);
}
else
{
CPaintDC dc(this);
if (grabbed2 ==1 )
{
::StretchDIBits(dc.m_hDC,
      354,
      21,
      320,
      240,
      0,
      0,
      DibWidth(m_pdibPicture),
      DibHeight(m_pdibPicture),
      DibPtr(m_pdibPicture),
      DibInfo(m_pdibPicture),
      DIB_RGB_COLORS,
      SRCCOPY);
}
if (grabbed == 1)
{
::StretchDIBits(dc.m_hDC,
      15,
      21,
      320,
      240,
      0,
      0,
      DibWidth(m_datum),
      DibHeight(m_datum),
      DibPtr(m_datum),
      DibInfo(m_datum),
      DIB_RGB_COLORS,
      SRCCOPY);

}


//*************************************************************************
//Following Code grabs the image displayed on screen into an array(Incomming)
//Proceeds to invert the image ( Invert), stores the inverted grayscale image
//( Grayed ) and calculates the threshold //

int Red, Green, Blue;
int Sum = 0;
int loop, loop2;
int ColMax=0,ColMin=255;


for (loop=0; loop < 320;loop++)
{
 for (loop2=0; loop2<240; loop2++)
 {
  Red = GetRValue(GetPixel(dc, 354+loop,21+loop2));
  Green = GetGValue(GetPixel(dc, 354+loop,21+loop2));
  Blue = GetBValue(GetPixel(dc, 354+loop,21+loop2));
```

```
    Camera[loop][loop2] = abs((int)((255-Red)*0.3) + (int)((255-Green)*0.58) + (int)((255-Blue)*0.12));

    ColMin = (ColMin>Camera[loop][loop2])?Camera[loop][loop2]:ColMin;
    ColMax = (ColMax<Camera[loop][loop2])?Camera[loop][loop2]:ColMax;

    Sum += Camera[loop][loop2];

   }
  }
  ofstream Check("c:\\temp\\check.txt");
  Check << ColMax<<endl;
  Check << ColMin<<endl;

  int Range = ColMax-ColMin;

  Check <<Range;

  CameraThreshold = abs(Sum/76000);


  m_Mult = (int)((Range*3/255)+2);

  UpdateData(FALSE);

  Check << m_Mult;
  Check.close();

  //*************************************************************************
  //Following Code generates the thresholded array of the incomming image //


//first sequence, image as is

  if ((grabbed != 0) && (ButtonText =="&STOP"))
  {
   int Space = (int)sqrt(pow(CameraThreshold - DatumThreshold,2));
   for (loop = 0; loop <320; loop ++)
   {
    for (loop2 = 0; loop2 < 240; loop2 ++)
    {
     //image jitter correction code
     //can compensate by 1 pixel in a certain direction

     //Normal image
     int Diff = (int)sqrt(pow((Camera[loop][loop2] - Reference[loop][loop2]),2));
     if (Diff > (m_Mult*Space))  //m_Mult adjusts the sensitivity
     {
      // If a difference is detected, the pixel is shifted
      // around by a distance of one pixel to evaluate if
      // this is due to a small wind movement. The 9 pixels
      // surrounding the current pixel will be evaluated
      // Every image pixel can be shifted in a different direction

      //Image shifted up and left
      if ((loop<319)&&(loop2<239))
      {
       Diff = (int)sqrt(pow((Camera[loop+1][loop2+1] - Reference[loop][loop2]),2));
       if (Diff > (m_Mult*Space))  //m_Mult adjusts the sensitivity

       {
        //Image shifted up
        if (loop2<239)
        {
         Diff = (int)sqrt(pow((Camera[loop][loop2+1] - Reference[loop][loop2]),2));
         if (Diff > (m_Mult*Space))  //m_Mult adjusts the sensitivity
         {
          //Image shifted up and right
          if ((loop>0)&&(loop2<239))
```

```
{
 Diff = (int)sqrt(pow((Camera[loop-1][loop2+1] - Reference[loop][loop2]),2));
 if (Diff > (m_Mult*Space)) //m_Mult adjusts the sensitivity
 {
  //Image shifted left
  if (loop<319)
  {
   Diff = (int)sqrt(pow((Camera[loop+1][loop2] - Reference[loop][loop2]),2));
   if (Diff > (m_Mult*Space)) //m_Mult adjusts the sensitivity
   {
    //Image shifted right
    if (loop>0)
    {
     Diff = (int)sqrt(pow((Camera[loop-1][loop2] - Reference[loop][loop2]),2));
     if (Diff > (m_Mult*Space)) //m_Mult adjusts the sensitivity
     {
      //Image shifted down and left
      if ((loop<319)&&(loop2>0))
      {
       Diff = (int)sqrt(pow((Camera[loop+1][loop2-1] - Reference[loop][loop2]),2));
       if (Diff > (m_Mult*Space)) //m_Mult adjusts the sensitivity
       {
        //Image shifted down
        if (loop2>0)
        {
         Diff = (int)sqrt(pow((Camera[loop][loop2+1] - Reference[loop][loop2]),2));
         if (Diff > (m_Mult*Space)) //m_Mult adjusts the sensitivity
         {


          //Image shifted down and right
          if ((loop>0)&&(loop2>0))
          {
                    Diff = (int)sqrt(pow((Camera[loop-1][loop2-1] - Reference[loop][loop2]),
2));
                    if (Diff > (m_Mult*Space)) //m_Mult adjusts the sensitivity
                    {
                              //If none of the pixel shifts result in a difference exclusion
                              //the pixel is marked as a set movement difference
                              Result[loop][loop2] = 1;
                              SetPixel(dc,354+loop,21+loop2,RGB(255,255,0));
                    }

          }
         }
        }
       }
      }
     }
    }
   }
  }
 }
// If any one of the pixel shifts results in a difference
//exclusion, the pixel is marked as non altered, ie only
//a small movement due to wind or camera vibration

else Result[loop][loop2] = 0;
}
}
}
CDialog::OnPaint();
```

```
 }
}


// The system calls this to obtain the cursor to display while the user drags
//  the minimized window.
HCURSOR CDemo2Dlg::OnQueryDragIcon()
{
 return (HCURSOR) m_hIcon;
}

void CDemo2Dlg::OnExitButton()
{
 // TODO: Add your control notification handler code here
 OnOK();
}


void CDemo2Dlg::OnGrab1Button()
{
 // TODO: Add your control notification handler code here

 CopyFile(FileName, "Datum.tmp", FALSE);

 m_datum = m_pdibPicture;
 grabbed = 1;
 int loop, loop2;

 for (loop = 0; loop < 320; loop++)
 {
  for (loop2 = 0; loop2 < 240; loop2++)
  {
   Reference[loop][loop2] = Camera[loop][loop2];
  }
 }
 DatumThreshold = CameraThreshold;

 InvalidateRect(CRect(15,21,335,261),FALSE);
}


void CDemo2Dlg::OnGrab2Button()
{
 // TODO: Add your control notification handler code here
 m_pdibPicture = DibOpenFile(FileName);
 GetDlgItem(IDC_GRAB1_BUTTON)->EnableWindow(TRUE);
 InvalidateRect(CRect(354,21,674,261),FALSE);
}

void CDemo2Dlg::SaveFile()
{
//File Details being Saved here

  CString NewName;
  CString OldExt;
  OldExt = "bmp";
  CString OldExt1;
  OldExt1 = "BMP";
  CString NewExt;
  NewExt = "vos";

  CString Temp;

  int Pos = m_Source.Find(OldExt);
  if (Pos==-1)
  {
```

```
    Pos = m_Source.Find(OldExt1);
   }
   Temp = m_Source.Left(Pos);
   Temp = Temp + NewExt;


   Pos = m_SaveEdit.Find("image.vos");
   NewName = m_SaveEdit.Left(Pos);

   NewName = NewName + Temp;

   ofstream file;
   file.open(NewName);
   for ( int loop = 0; loop< 320; loop++)
   {
    for (int loop2 = 0; loop2<240; loop2++)
    {
     if (Result[loop][loop2] == 1)
      file << loop<<"\t"<<loop2<<"\n";
    }
   }
   file.close();

   //End of Save procedure
}

void CDemo2Dlg::OnStartButton()
{
 // TODO: Add your control notification handler code here
 if ((ButtonText == "&START") && (Selected != 0))
 {
  ButtonText = "&STOP";
  GetDlgItem(IDC_SELECT_BUTTON)->EnableWindow(FALSE);
  GetDlgItem(IDC_SAVESELECT_BUTTON)->EnableWindow(FALSE);
  GetDlgItem(IDC_GRAB1_BUTTON)->EnableWindow(FALSE);
  GetDlgItem(IDC_EXIT_BUTTON)->EnableWindow(FALSE);


  if ( m_Batch==TRUE)
  {
   CFileFind Finder;
   CString Title;
   GetDlgItem(IDC_START_BUTTON)->EnableWindow(FALSE);
  // chdir(m_SourceDir);

   BOOL bWorking = Finder.FindFile("*.bmp");

   while(bWorking)
   {
   // chdir(m_SourceDir);
    bWorking = Finder.FindNextFile();
    Title = Finder.GetFileTitle();

    char *title;
    title = "";
    strcat(title,Title);
    m_pdibPicture = DibOpenFile(title);
    Invalidate();
    SaveFile();
   }
  }
  else
  {
   Invalidate();
   SaveFile();
  }

 }
```

```
 else
 {
  ButtonText = "&START";
  GetDlgItem(IDC_SELECT_BUTTON)->EnableWindow(TRUE);
  GetDlgItem(IDC_EXIT_BUTTON)->EnableWindow(TRUE);
  GetDlgItem(IDC_GRAB1_BUTTON)->EnableWindow(TRUE);
  GetDlgItem(IDC_SAVESELECT_BUTTON)->EnableWindow(TRUE);
 }

 SetDlgItemText(IDC_START_BUTTON,ButtonText);

}


void CDemo2Dlg::OnSelectButton()
{
 // TODO: Add your control notification handler code here
 Selected = 1;
 ButtonText = "&START";
 SetDlgItemText(IDC_START_BUTTON,ButtonText);


 char FileTitle[100];

 OPENFILENAME ofn;
 memset(&ofn, 0, sizeof(ofn));
 ofn.lStructSize = sizeof(OPENFILENAME);
 ofn.hwndOwner = NULL;
 ofn.hInstance = NULL;

 ofn.lpstrFilter = TEXT("Bitmap Picture Files *.bmp\0*.bmp\0\0");

 ofn.lpstrCustomFilter = NULL;
 ofn.nMaxCustFilter = 0;
 ofn.nFilterIndex = 1;
 ofn.lpstrFile = FileName;
 ofn.nMaxFile = 500;
 ofn.lpstrFileTitle = FileTitle;
 ofn.nMaxFileTitle = 99;
 ofn.lpstrInitialDir = NULL;
 ofn.lpstrTitle = "Open Bmp File";
 ofn.Flags = OFN_FILEMUSTEXIST;
 ofn.lpstrDefExt = "BMP";
 ofn.lCustData = NULL;
 ofn.lpfnHook = NULL;
 ofn.lpTemplateName = NULL;

 FileName[0] = '\0';
 GetOpenFileName(&ofn);

 if (FileName[0] != '\0')
 {
  grabbed2 = 1;
  m_Source = FileTitle;
  CString Temp;
  Temp = FileName;
  int Pos;
  Pos = Temp.Find(m_Source);
  m_SourceDir = Temp.Left(Pos);

  UpdateData(FALSE);
  GetDlgItem(IDC_START_BUTTON)->EnableWindow(TRUE);
  GetDlgItem(IDC_GRAB2_BUTTON)->EnableWindow(TRUE);

  CopyFile(FileName, "Camera.tmp", FALSE);
  OnGrab2Button();
 }
```

```
}


void CDemo2Dlg::OnSaveselectButton()
{
 // TODO: Add your control notification handler code here
 CFileDialog dlg(FALSE,"vos file","image.vos");
 dlg.DoModal();
 m_SaveEdit = dlg.GetPathName();
 UpdateData(FALSE);
}

void CDemo2Dlg::OnBatchCheck()
{
 // TODO: Add your control notification handler code here
 if (m_Batch==TRUE) m_Batch = FALSE;
 else m_Batch = TRUE;
}


//****************** OWN FUNCTION *********************




PDIB CDemo2Dlg::DibOpenFile(LPSTR szFile)
{
 HFILE fh;
 DWORD dwLen;
 DWORD dwBits;
 PDIB pdib;
 LPVOID p;
 OFSTRUCT of;

#if defined(WIN32) || defined (_WIN32)
#define GetCurrentInstance() GetModuleHandle(NULL)
#else
#define GetCurrentInstance() (HINSTANCE)SELECTOROF((LPVOID)&of)
#endif

 fh = OpenFile(szFile, &of, OF_READ);

 if (fh == -1)
 {
  HRSRC h;
  h = FindResource(GetCurrentInstance(), szFile, RT_BITMAP);

#if defined(WIN32) || defined(_WIN32)
  if (h)
    return (PDIB)LockResource(LoadResource(GetCurrentInstance(), h));
#else
  if (h)
    fh = AccessResource(GetCurrentInstance(),h);
  #endif
 }


 if (fh == -1)
  return NULL;

 pdib = DibReadBitmapInfo(fh);

 if (!pdib)
  return NULL;
```

```
 dwBits = pdib->biSizeImage;
 dwLen = pdib->biSize + DibPaletteSize(pdib) + dwBits;

 p = GlobalReAllocPtr(pdib,dwLen,0);

 if (!p)
 {
  GlobalFreePtr(pdib);
  pdib = NULL;
 }
 else
 {
  pdib = (PDIB)p;
 }

 if (pdib)
 {
  _hread(fh, (LPBYTE)pdib + (UINT)pdib->biSize + DibPaletteSize(pdib), dwBits);
 }

 _lclose(fh);

 return pdib;

}




PDIB CDemo2Dlg::DibReadBitmapInfo(HFILE fh)
{
 DWORD off;
 HANDLE hbi = NULL;
 int size;
 int i;
 int nNumColors;

 RGBQUAD FAR *pRgb;
 BITMAPINFOHEADER bi;
 BITMAPCOREHEADER bc;
 BITMAPFILEHEADER bf;
 PDIB pdib;

 if (fh == -1)
  return NULL;

 off = _llseek(fh, 0L, SEEK_CUR);

 if (sizeof(bf) != _lread(fh,(LPSTR)&bf, sizeof(bf)))
  return FALSE;

 if ( bf.bfType != BFT_BITMAP)
 {
  bf.bfOffBits = 0L;
  _llseek(fh, off, SEEK_SET);
 }

 if (sizeof(bi) != _lread(fh,(LPSTR)&bi, sizeof(bi)))
  return FALSE;

 switch (size = (int)bi.biSize)
 {

 default :
 case sizeof(BITMAPINFOHEADER):break;
```

```
 case sizeof(BITMAPCOREHEADER):
      bc = *(BITMAPCOREHEADER*)&bi;
      bi.biSize = sizeof(BITMAPINFOHEADER);
      bi.biWidth = (DWORD)bc.bcWidth;
      bi.biHeight = (DWORD)bc.bcHeight;
      bi.biPlanes = (UINT)bc.bcPlanes;
      bi.biBitCount = (UINT)bc.bcBitCount;
      bi.biCompression = BI_RGB;
      bi.biSizeImage = 0;
      bi.biXPelsPerMeter = 0;
      bi.biYPelsPerMeter = 0;
      bi.biClrUsed = 0;
      bi.biClrImportant = 0;

      _llseek(fh, (LONG)sizeof(BITMAPCOREHEADER)-sizeof(BITMAPINFOHEADER),SEEK_CUR);
      break;

 }

 nNumColors = DibNumColors(&bi);

#if 0
 if (bi.biSizeImage == 0)
  bi.biSizeImage = DibSizeImage(&bi);

 if (bi.biClrUsed == 0)
  bi.biClrUsed = DibNumColors(&bi);
#else
 FixBitmapInfo(&bi);
#endif

 pdib = (PDIB)GlobalAllocPtr(GMEM_MOVEABLE, (LONG)bi.biSize + nNumColors * sizeof(RGBQUAD));

 if ( !pdib)
  return NULL;

 *pdib = bi;
 pRgb = DibColors(pdib);

 if ( nNumColors)
 {
  if ( size == sizeof(BITMAPCOREHEADER))
  {
   _lread(fh, (LPVOID)pRgb, nNumColors * sizeof (RGBTRIPLE));
   for (i = nNumColors -1; i >=0; i--)
   {
    RGBQUAD rgb;
    rgb.rgbRed = ((RGBTRIPLE FAR *)pRgb)[i].rgbtRed;
    rgb.rgbBlue = ((RGBTRIPLE FAR *)pRgb)[i].rgbtBlue;
    rgb.rgbGreen = ((RGBTRIPLE FAR *)pRgb)[i].rgbtGreen;
    rgb.rgbReserved = (BYTE)0;

    pRgb[i] = rgb;
   }
  }
  else
  {
   _lread(fh,(LPVOID)pRgb, nNumColors * sizeof(RGBQUAD));
  }
 }

 if (bf.bfOffBits != 0L)
  _llseek(fh, off + bf.bfOffBits, SEEK_SET);

 return pdib;

}
```

```
void CDemo2Dlg::OnDestroy()
{
 CDialog::OnDestroy();

 // TODO: Add your message handler code here
 GlobalFreePtr(m_pdibPicture);
 GlobalFreePtr(m_datum);
}


void CDemo2Dlg::OnUpdateButton()
{
 // TODO: Add your control notification handler code here
 UpdateData(TRUE);
}
```

# 20.11 - Multiple Data Extractor

```
// Data ExtractorDlg.cpp : implementation file
//

#include "stdafx.h"
#include "Data Extractor.h"
#include "Data ExtractorDlg.h"

#include <windowsx.h>
#include <string.h>
#include <fstream.h>
#include <iostream.h>
#include <direct.h>
#include <math.h>
#include <iomanip.h>

#include <fstream.h>

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

/////////////////////////////////////////////////////////////////////////////
// CAboutDlg dialog used for App About

class CAboutDlg : public CDialog
{
public:
 CAboutDlg();

// Dialog Data
 //{{AFX_DATA(CAboutDlg)
 enum { IDD = IDD_ABOUTBOX };
 //}}AFX_DATA

 // ClassWizard generated virtual function overrides
 //{{AFX_VIRTUAL(CAboutDlg)
 protected:
 virtual void DoDataExchange(CDataExchange* pDX);    // DDX/DDV support
 //}}AFX_VIRTUAL

// Implementation
protected:
 //{{AFX_MSG(CAboutDlg)
 //}}AFX_MSG
 DECLARE_MESSAGE_MAP()
};

CAboutDlg::CAboutDlg() : CDialog(CAboutDlg::IDD)
{
 //{{AFX_DATA_INIT(CAboutDlg)
 //}}AFX_DATA_INIT
}

void CAboutDlg::DoDataExchange(CDataExchange* pDX)
{
 CDialog::DoDataExchange(pDX);
 //{{AFX_DATA_MAP(CAboutDlg)
 //}}AFX_DATA_MAP
}
```

```
BEGIN_MESSAGE_MAP(CAboutDlg, CDialog)
 //{{AFX_MSG_MAP(CAboutDlg)
  // No message handlers
 //}}AFX_MSG_MAP
END_MESSAGE_MAP()

/////////////////////////////////////////////////////////////////////////////
// CDataExtractorDlg dialog

CDataExtractorDlg::CDataExtractorDlg(CWnd* pParent /*=NULL*/)
 : CDialog(CDataExtractorDlg::IDD, pParent)
{
 //{{AFX_DATA_INIT(CDataExtractorDlg)
 m_Target = _T("");
 m_Objects = 0;
 m_Percent = 0;
 m_FullName = _T("");
 //}}AFX_DATA_INIT
 // Note that LoadIcon does not require a subsequent DestroyIcon in Win32
 m_hIcon = AfxGetApp()->LoadIcon(IDR_MAINFRAME);
}

void CDataExtractorDlg::DoDataExchange(CDataExchange* pDX)
{
 CDialog::DoDataExchange(pDX);
 //{{AFX_DATA_MAP(CDataExtractorDlg)
 DDX_Control(pDX, IDC_SET_SLIDER, m_SetSlider);
 DDX_Text(pDX, IDC_TRAGET_EDIT, m_Target);
 DDX_Text(pDX, IDC_OBJECT_EDIT, m_Objects);
 DDX_Text(pDX, IDC_PERCENT_EDIT, m_Percent);
 DDX_Text(pDX, IDC_NAME_STATIC, m_FullName);
 //}}AFX_DATA_MAP
}

BEGIN_MESSAGE_MAP(CDataExtractorDlg, CDialog)
 //{{AFX_MSG_MAP(CDataExtractorDlg)
 ON_WM_SYSCOMMAND()
 ON_WM_PAINT()
 ON_WM_QUERYDRAGICON()
 ON_BN_CLICKED(IDC_EXIT_BUTTON, OnExitButton)
 ON_BN_CLICKED(IDC_OPEN_BUTTON, OnOpenButton)
 ON_BN_CLICKED(IDC_CLAC_BUTTON, OnClacButton)
 ON_BN_CLICKED(IDC_TARGET_BUTTON, OnTargetButton)
 ON_BN_CLICKED(IDC_VALID_CHECK, OnValidCheck)
 ON_BN_CLICKED(IDC_BOUND_BUTTON, OnBoundButton)
 ON_BN_CLICKED(IDC_CLEAR_BUTTON, OnClearButton)
 ON_WM_HSCROLL()
 ON_BN_CLICKED(IDC_SAVE_BUTTON, OnSaveButton)
 //}}AFX_MSG_MAP
END_MESSAGE_MAP()

/////////////////////////////////////////////////////////////////////////////
// CDataExtractorDlg message handlers



int Stored[320][240];
int Check1=0, Check2=0;
CString Title;
int sharp[80][60];
int Xmin[100],Xmax[100],Ymin[100],Ymax[100];
int counter;
int Pos = 7;


BOOL CDataExtractorDlg::OnInitDialog()
{
```

```
CDialog::OnInitDialog();

// Add "About..." menu item to system menu.

// IDM_ABOUTBOX must be in the system command range.
ASSERT((IDM_ABOUTBOX & 0xFFF0) == IDM_ABOUTBOX);
ASSERT(IDM_ABOUTBOX < 0xF000);

CMenu* pSysMenu = GetSystemMenu(FALSE);
if (pSysMenu != NULL)
{
 CString strAboutMenu;
 strAboutMenu.LoadString(IDS_ABOUTBOX);
 if (!strAboutMenu.IsEmpty())
 {
   pSysMenu->AppendMenu(MF_SEPARATOR);
   pSysMenu->AppendMenu(MF_STRING, IDM_ABOUTBOX, strAboutMenu);
 }
}

// Set the icon for this dialog.  The framework does this automatically
//  when the application's main window is not a dialog
SetIcon(m_hIcon, TRUE);                                    // Set big icon
SetIcon(m_hIcon, FALSE);                        // Set small icon

// TODO: Add extra initialization here
m_Objects = 0;
CSliderCtrl* SliderOne = (CSliderCtrl*)GetDlgItem(IDC_SET_SLIDER);
SliderOne->SetRange(0,15);
SliderOne->SetPos(7);
SliderOne->SetTicFreq(2);

char Temp[1024];
_getcwd(Temp,1024);
m_ProgDir = Temp;

return TRUE;  // return TRUE  unless you set the focus to a control
}

void CDataExtractorDlg::OnSysCommand(UINT nID, LPARAM lParam)
{
 if ((nID & 0xFFF0) == IDM_ABOUTBOX)
 {
  CAboutDlg dlgAbout;
  dlgAbout.DoModal();
 }
 else
 {
  CDialog::OnSysCommand(nID, lParam);
 }
}

// If you add a minimize button to your dialog, you will need the code below
//  to draw the icon.  For MFC applications using the document/view model,
//  this is automatically done for you by the framework.

void CDataExtractorDlg::OnPaint()
{
 if (IsIconic())
 {
  CPaintDC dc(this); // device context for painting

  SendMessage(WM_ICONERASEBKGND, (WPARAM) dc.GetSafeHdc(), 0);

  // Center icon in client rectangle
  int cxIcon = GetSystemMetrics(SM_CXICON);
  int cyIcon = GetSystemMetrics(SM_CYICON);
  CRect rect;
```

```
  GetClientRect(&rect);
  int x = (rect.Width() - cxIcon + 1) / 2;
  int y = (rect.Height() - cyIcon + 1) / 2;

  // Draw the icon
  dc.DrawIcon(x, y, m_hIcon);
 }
 else
 {
  CPaintDC dc(this);
  CPen ShapePen;
  ShapePen.CreatePen(PS_SOLID,1,RGB(255,0, 255));
  CPen* pOriginalPen;
  pOriginalPen = dc.SelectObject(&ShapePen);

  for(int loop = 0; loop < 320; loop++)
  {
   for(int loop2 = 0; loop2<240; loop2++)
   {
    if(Stored[loop][loop2]) SetPixel(dc,loop+12,loop2+18,RGB(0,100,200));
   }
  }


  for (loop = 0; loop<80; loop++)
  {
   for (int loop2 = 0; loop2<60; loop2++)
   {
    if (sharp[loop][loop2] == 1)
     Rectangle(dc,(12+4*loop),(18+4*loop2),(16+4*loop),(22+4*loop2));
   }
  }


  for (loop = 1; loop<=counter; loop++)
  {
   CPen BoxPen;
   BoxPen.CreatePen(PS_SOLID,1,RGB(0,255,0));
   CPen* pOriginalPen;
   pOriginalPen = dc.SelectObject(&BoxPen);
   MoveToEx(dc,4*Xmin[loop]+12, 4*Ymin[loop]+18,NULL);
   LineTo(dc, 4*Xmax[loop]+16, 4*Ymin[loop]+18);
   LineTo(dc, 4*Xmax[loop]+16, 4*Ymax[loop]+22);
   LineTo(dc, 4*Xmin[loop]+12, 4*Ymax[loop]+22);
   LineTo(dc, 4*Xmin[loop]+12, 4*Ymin[loop]+18);
  }

  CDialog::OnPaint();
 }
}

// The system calls this to obtain the cursor to display while the user drags
//  the minimized window.
HCURSOR CDataExtractorDlg::OnQueryDragIcon()
{
 return (HCURSOR) m_hIcon;
}

void CDataExtractorDlg::OnExitButton()
{
 // TODO: Add your control notification handler code here
 OnOK();
}

void CDataExtractorDlg::OnOpenButton()
{
 // TODO: Add your control notification handler code here
 char FileName[500];
```

```
char FileTitle[100];

OnClearButton();

int loop, loop2;

for ( loop=0; loop < 320; loop++)
{
 for ( loop2 = 0; loop2 < 240 ; loop2++)
 {
  Stored[loop][loop2]=0;
 }
}

UpdateData(FALSE);
InvalidateRect(CRect(12,18,332,258), FALSE);


OPENFILENAME ofn;
_fmemset(&ofn, 0, sizeof(ofn));
ofn.lStructSize = sizeof(OPENFILENAME);

ofn.hwndOwner = m_hWnd;;
ofn.hInstance = NULL;

ofn.lpstrFilter = TEXT("VosDemo files *.vos\0*.vos\0\0");

ofn.lpstrCustomFilter = NULL;
ofn.nMaxCustFilter = 0;
ofn.nFilterIndex = 1;
ofn.lpstrFile = FileName;
ofn.nMaxFile = 500;
ofn.lpstrFileTitle = FileTitle;
ofn.nMaxFileTitle = 99;
ofn.lpstrInitialDir = NULL;
ofn.lpstrTitle = "Open VOS file";
ofn.Flags = OFN_FILEMUSTEXIST;
ofn.lpstrDefExt = "BMP";
ofn.lCustData = NULL;
ofn.lpfnHook = NULL;
ofn.lpTemplateName = NULL;

FileName[0] = '\0';

if (GetOpenFileName(&ofn))
{
 int count;

 Check2 = 1;
 CString FullTitle = FileTitle;
 count = FullTitle.Find(".vos");
 Title = FullTitle.Left(count);

 ifstream file_in(FileName);
 m_FileName = FileName;
 m_FullName = m_FileName;
 int Pos = m_FullName.ReverseFind('\\');
 m_FullName = m_FullName.Right(m_FullName.GetLength()-Pos-1);
 int value1;
 int value2;

 CPaintDC dc(this);

 m_Percent = 0;

 do
 {
  file_in >> value1;
```

```
    file_in >> value2;

    Stored[value1][value2] = 1;
//    SetPixel(dc,value1+12,value2+18,RGB(0,100,200));
   }
   while ( file_in.eof() == 0);
   file_in.close();

   for (loop = 0; loop <320; loop++)
   {
    for (loop2 = 0; loop2 <240; loop2++)
    {
     if (Stored[loop][loop2] != 1)
     {
      Stored[loop][loop2] = 0;

     }
     else m_Percent++;

    }
   }

   m_Percent = (int)(((float)m_Percent*100)/(320*240));
   UpdateData(FALSE);


   int Correction = (int)(0.15*m_Percent + ((100-m_Percent)/15)*sin( 0.01*3.141592654*m_Percent ));
   CSliderCtrl* SliderOne = (CSliderCtrl*)GetDlgItem(IDC_SET_SLIDER);
   SliderOne->SetPos(Correction);
   CString Texter;
   Texter.Format("%d",Correction);
   SetDlgItemText(IDC_SET_STATIC,Texter);


   InvalidateRect(CRect(12,18,332,258), FALSE);
  // InvalidateRect(CRect(12,18,332,258), FALSE);
  }

}




void CDataExtractorDlg::OnClacButton()
{
 // TODO: Add your control notification handler code here
 int loop, loop2,loop3,loop4;
 int count = 0;
 OnClearButton();

 for (loop2=0; loop2<60; loop2++)
 {
  for ( loop=0; loop<80; loop++)
  {
   sharp[loop][loop2] = 0;
   count = 0;
   for (loop3=0; loop3<4; loop3++)
   {
    for ( loop4 = 0; loop4<4; loop4++)
    {
     if (Stored[4*loop+loop3][4*loop2+loop4]==1)
       count++;
    }
   }
   if (count >Pos) //(Pos = Slider value )
   {
    sharp[loop][loop2] = 1;
```

```
    }

    InvalidateRect(CRect(12,18,332,258), FALSE);
   }
  }
}
```

```
void CDataExtractorDlg::OnTargetButton()
{
 // TODO: Add your control notification handler code here
 UpdateData(TRUE);
 if (chdir(m_Target))
 {
  MessageBox("Couldn't change to drive");
  Check1=0;
 }
 else Check1=1;
}
```

```
void CDataExtractorDlg::OnValidCheck()
{
 // TODO: Add your control notification handler code here
 UpdateData(TRUE);
}
```

```
void CDataExtractorDlg::OnBoundButton()
{
 // TODO: Add your control notification handler code here
 int loop, loop2, loop3, Mem = 0;
 int gap = 0, Set = 0;
 counter = 0;


 for (loop = 0; loop<100; loop++)
 {
  Xmin[loop] = 80;
  Ymin[loop] = 60;
  Xmax[loop] = 0;
  Ymax[loop] = 0;
 }


 //Check every second line in the image
 loop=0;
 for(loop2=0; loop2<60; loop2++)
 {
  Set = 0;
  Mem = 0;
  gap = 0;
  //while(loop2<80)
  for (loop=0; loop<80; loop++)
  {
   if (sharp[loop][loop2]==1)
   {
    counter++;
```

```
    Xmin[counter] = loop;
    Ymin[counter] = loop2;
    Ymax[counter] = loop2;
    Xmax[counter] = loop;

    //Previous line check for position matching
    for(loop3 = 0; loop3<counter; loop3++)
    {
     if((loop>=Xmin[loop3]-2)&&(loop<=Xmax[loop3]+2)&&
       (loop2-Ymax[loop3]<3)&&(loop3!=counter)&&
       (loop2>0))
     {
       Set = 1;
       Mem = loop3;
     }
    }
    //End of line check

    while((gap<1)&&(loop<80))
    {
     loop++;
     if(sharp[loop][loop2]==1)
     {
       gap = 0;
       Xmax[counter] = loop;

       //Previous line check for position matching
       for(loop3 = 0; loop3<counter; loop3++)
       {
        if((loop>=Xmin[loop3]-2)&&(loop<=Xmax[loop3]+2)
          &&(loop2-Ymax[loop3]<3)&&(loop3!=counter)&&(counter>0))
        {
          Set = 1;
          Mem = loop3;
        }
       }
       //End of line check

     }
     else gap++;

    }
    gap = 0;

    //Matching correction code, updates matched object
    //and deleted new object created
    if (Set==1)
    {
     if(Xmin[counter]<Xmin[Mem]) Xmin[Mem] = Xmin[counter];
     if(Xmax[counter]>Xmax[Mem]) Xmax[Mem] = Xmax[counter];
     Ymax[Mem] = loop2;
     Xmin[counter] = 80;
     Xmax[counter] = 0;
     Ymin[counter] = 60;
     Ymax[counter] = 0;
     counter--;
     Set = 0;
     Mem = 0;
    }

   }

  }

  loop=0;
 }
 m_Objects = counter;
 UpdateData(FALSE);
```

```
 InvalidateRect(CRect(12,18,333,259), FALSE);

}

void CDataExtractorDlg::OnClearButton()
{
 // TODO: Add your control notification handler code here
 int loop,loop2;
 for ( loop = 0; loop<80; loop++)
 {
  for (loop2 = 0; loop2<60; loop2++)
  {
    sharp[loop][loop2] = 0;
  }
 }

 counter = 0;
 m_Objects = 0;

 for (loop = 0; loop<11; loop++)
 {
  Xmin[loop] = 80;
  Xmax[loop] = 0;
  Ymin[loop] = 60;
  Ymax[loop] = 0;
 }
 UpdateData(FALSE);
 Invalidate();

 CPaintDC dc(this);
 for (loop = 0; loop<320; loop++)
 {
  for ( loop2 = 0;loop2<240; loop2++)
  {
    if(Stored[loop][loop2])
      SetPixel(dc,loop+12,loop2+18,RGB(0,100,200));
  }
 }
}

void CDataExtractorDlg::OnHScroll(UINT nSBCode, UINT nPos, CScrollBar* pScrollBar)
{
 // TODO: Add your message handler code here and/or call default
 CSliderCtrl* SliderOne = (CSliderCtrl*)GetDlgItem(IDC_SET_SLIDER);
 Pos = SliderOne->GetPos();
 CString Texter;
 Texter.Format("%d",SliderOne->GetPos());
 SetDlgItemText(IDC_SET_STATIC,Texter);

 OnClacButton();
 InvalidateRect(CRect(12,18,333,259), FALSE);


 CDialog::OnHScroll(nSBCode, nPos, pScrollBar);
}

void CDataExtractorDlg::OnSaveButton()
{
 // TODO: Add your control notification handler code here
 CSliderCtrl* SliderOne = (CSliderCtrl*)GetDlgItem(IDC_SET_SLIDER);
 Pos = SliderOne->GetPos();

 char Temp[1024];
 _getcwd(Temp,1024);

 _chdir(m_ProgDir);
 ofstream OutFile("Results.log", ios::app);
```

```
 OutFile << m_FullName << "\t" << m_Percent << "\t" << Pos << endl;
 OutFile.close();

 _chdir(Temp);
}



// Data ExtractorDlg.h : header file
//

#if !defined(AFX_DATAEXTRACTORDLG_H__0690B747_B844_11D2_9F64_9FF1E749723B__INCLUDED_)
#define AFX_DATAEXTRACTORDLG_H__0690B747_B844_11D2_9F64_9FF1E749723B__INCLUDED_

#if _MSC_VER >= 1000
#pragma once
#endif // _MSC_VER >= 1000

/////////////////////////////////////////////////////////////////////////////
// CDataExtractorDlg dialog

class CDataExtractorDlg : public CDialog
{
// Construction
public:
 CString m_FileName;
 CString m_ProgDir;
 CDataExtractorDlg(CWnd* pParent = NULL);    // standard constructor

// Dialog Data
 //{{AFX_DATA(CDataExtractorDlg)
 enum { IDD = IDD_DATAEXTRACTOR_DIALOG };
 CSliderCtrl m_SetSlider;
 int  m_Xcoord;
 int  m_Xmax;
 int  m_Xmin;
 int  m_Ymax;
 int  m_Ymin;
 CString m_Target;
 int  m_X;
 int  m_Y;
 int  m_Area;
 int  m_SegArea;
 int  m_SegHeight;
 int  m_SegWidth;
 BOOL m_ValidCheck;
 int  m_Objects;
 int  m_Percent;
 CString m_FullName;
 //}}AFX_DATA

 // ClassWizard generated virtual function overrides
 //{{AFX_VIRTUAL(CDataExtractorDlg)
 protected:
 virtual void DoDataExchange(CDataExchange* pDX);          // DDX/DDV support
 //}}AFX_VIRTUAL

// Implementation
protected:
 HICON m_hIcon;

 // Generated message map functions
 //{{AFX_MSG(CDataExtractorDlg)
 virtual BOOL OnInitDialog();
 afx_msg void OnSysCommand(UINT nID, LPARAM lParam);
 afx_msg void OnPaint();
 afx_msg HCURSOR OnQueryDragIcon();
```

```
 afx_msg void OnExitButton();
 afx_msg void OnOpenButton();
 afx_msg void OnClacButton();
 afx_msg void OnTargetButton();
 afx_msg void OnValidCheck();
 afx_msg void OnBoundButton();
 afx_msg void OnClearButton();
 afx_msg void OnHScroll(UINT nSBCode, UINT nPos, CScrollBar* pScrollBar);
 afx_msg void OnSaveButton();
 //}}AFX_MSG
 DECLARE_MESSAGE_MAP()
};

//{{AFX_INSERT_LOCATION}}
// Microsoft Developer Studio will insert additional declarations immediately before the previous line.

#endif // !defined
(AFX_DATAEXTRACTORDLG_H__0690B747_B844_11D2_9F64_9FF1E749723B__INCLUDED_)
```