

Towards the development of an electronic nose

By

Bashan Naidoo BScEng

Submitted in fulfilment of the requirements for the degree of Master of Science in Engineering, in the School of Electrical, Electronic & Computer Engineering, University of Natal, South Africa.

Durban

September 2003

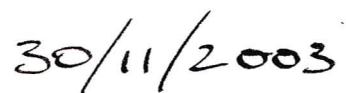
PREFACE

The experimental work in this thesis was carried out in the School of Electrical, Electronic and Computer Engineering, University of Natal, Durban during the period January 1998 to September 2003 under the supervision of Professor A.D. Broadhurst.

The author certifies that this thesis represents his own work, except where indicated in the text, and has not been submitted to any other university for degree purposes.



B. Naidoo
Author



Date

ABSTRACT

Electronic noses are targeted at determining odour character in a fashion that emulates conscious odour perception in mammals. The intention of this study was to develop an organisational framework for electronic noses and deploy a sample cheese odour discriminator within this framework.

Biological olfactory systems are reviewed with the purpose of extracting the organisational principles that result in successful olfaction. Principles of gas handling, chemoreception, and neural processing are considered in the formulation of an organisational framework. An electronic nose is then developed in accordance with the biologically inspired framework.

Gas sensing is implemented by an array of six commercially available (tin oxide) semiconductor sensors. These popular gas sensors are known to lack stability thus necessitating hardware and signal processing measures to limit or compensate for instability. An odorant auto-sampler was developed to deliver measured amounts of odorant to the sensors in a synthetic air medium. Each measurement event encodes a simulated sniff, and is captured across six sensor channels over a period of 256 seconds at a sampling rate of 1Hz. The simulated sniff captures sensor base references and responses to odorant introduction and removal.

A technique is presented for representation and processing of sensor-array data as a two-dimensional (2D) image where one dimension encodes time, and the other encodes multi-channel sensory outputs. The near optimal, computationally efficient 2D Discrete Cosine Transform (DCT) is used to represent the 2D signal in a decorrelated frequency domain. Several coefficient selection strategies are proposed and tested. A heuristic technique is developed for the selection of transform domain coefficients as inputs to a non-linear neural network based classifier. The benefits of using the selection heuristic as compared to standard variance-based selection are evident in the results. Benefits include; significant dimensionality reduction with concomitant reduction in classifier size and training time, improved generalisation by the neural network and improved classification performance. The electronic nose produced a 99.1% classification rate across a set of seven different cheeses.

ACKNOWLEDGEMENTS

I am indebted to many people for the contributions that they have made towards my efforts in this degree and my career in general:

Professor A.D. Broadhurst for his belief in me, moral support, guidance and supervision. On more than one occasion, it was he that rescued me from the all too common doldrums of postgraduate study.

Our technical support staff including, David Long, Tony Roos, Bruce Harrison, Tony Munnik, Andrew Stengel, Divesh Maharaj, Sanjay Deeraj, Greg Loubser and Basil, all of whom contributed in various ways towards the construction of the experimental apparatus.

The academic staff and postgraduate students for their interest and tea-time discussions on this curious project.

My parents and sister for their support, tolerance and acceptance of the fact that I spent all these years not earning an engineer's salary.

I also acknowledge financial support from the University Research Fund.

CONTENTS

	Page
CHAPTER 1: INTRODUCTION	
1.1 A brief perspective	1.1
1.2 The olfactory sense	1.2
1.3 The electronic nose	1.3
1.4 Applications of the electronic nose	1.4
1.5 Specific contributions	1.5
1.6 Thesis overview	1.6
CHAPTER 2: BIOLOGICALLY INSPIRED SYSTEM ORGANISATION	
2.1 Introduction	2.1
2.2 Elements of olfactory biology	2.1
2.2.1 Prereceptor events	2.2
2.2.2 Chemoreception: The olfactory epithelium	2.5
2.2.3 Low-level processing: The olfactory bulb	2.9
2.2.4 High-level processing: Cortical and limbic pathways	2.11
2.3 Summary	2.12
2.3.1 Gas handling	2.13
2.3.2 Chemoreception	2.13
2.3.3 Signal processing	2.14
2.4 Conclusion	2.14
CHAPTER 3: ODOUR RESPONSE MEASUREMENTS	
3.1 Introduction	3.1
3.2 Measurement front-end	3.2
3.2.1 Headspace sampling	3.2
3.2.2 Odorant handling	3.4
3.2.3 Sensing	3.6
3.2.4 Recovery	3.11
3.3 Measurement configuration tools	3.13
3.4 The standard measurement event	3.16
3.5 Conclusion	3.18

CHAPTER 4: A DATA PREPROCESSING STRATEGY

4.1	Introduction	4.1
4.2	Discrete cosine transformation	4.1
4.3	Transform coefficient selection	4.8
4.3.1	DC coefficient removal	4.10
4.3.2	High frequency removal	4.14
4.3.3	Simple variance-based selection	4.17
4.3.4	Coefficient selection heuristic	4.18
4.4	Preliminary classification results	4.22
4.5	Conclusion	4.23

CHAPTER 5: CLASSIFICATION BY NEURAL NETWORK

5.1	Introduction	5.1
5.2	Software environment	5.1
5.2.1	Stuttgart Neural Network Simulator	5.2
5.2.2	Neural network configuration system	5.2
5.2.3	Result post-processing	5.11
5.3	Training configuration	5.12
5.4	Results	5.14
5.4.1	Pre-processing results	5.14
5.4.2	Training results	5.17
5.4.3	Final comparative result	5.29
5.4	Conclusion	5.29

CHAPTER 6: CONCLUSION

6.1

REFERENCES

APPENDIX A: SELECTED SOFTWARE IMPLEMENTATIONS

A.1	Organisation of Appendix A	A.1
A.2	The Measurement Control Language three-pass compiler	A.1
A.2.1	First pass compiler script	A.3
A.2.2	Second pass compiler script	A.4
A.2.3	Third pass compiler script	A.6
A.2.4	Final control script	A.7
A.2.5	Compiler invocation shell script	A.9
A.2.6	Windows IDE Compiler invocation function	A.10

A.3	MCL interpreter for DOS	A.13
A.4	Pre-processing scripts	A.19
A.4.1	Fast DCT script	A.19
A.4.2	D3C coefficient selection script	A.28
A.4.3	Pattern-set generator script	A.31
A.5	Simulator utility source modification	A.37

APPENDIX B: SELECTED RESULTS

B.1	Organisation of Appendix B	B.1
B.2	Input dimensionality comparison across selected datasets	B.1
B.3	PUR dataset results	B.2
B.4	DCT dataset results	B.3
B.5	D1C dataset results	B.4
B.6	D2C dataset results	B.5
B.7	D3CA dataset results	B.6
B.8	D3CB dataset results	B.7

References

1. Gardner, J. W., and Bartlett, P. N.: 'Electronic noses: Principles and applications', Oxford Science Publications, Oxford University Press, New York, 1999.
2. Zwaardemaker, H. and Hogewind, F.: 'On spray-electricity and waterfall-electricity ' Proc. Acad. Sci. Amst., Vol. 22, pp. 429-437, 1920.
3. Hartman, J.D.: 'A possible objective method for the rapid estimation of flavours in vegetables', Proc. Am. Soc. Hort. Sci., Vol. 64, pp. 335, 1954.
4. Persaud, K. and Dodd, G.H.: 'Analysis of discrimination mechanisms of the mammalian olfactory system using a model nose', Nature, Vol. 299, pp. 352-355, 1982.
5. Nagle, H.T., Schiffman, S.S. and Gutierrez-Osuna, R.: 'The how and why of electronic noses', IEEE Spectrum, September, 1998.
6. Lehrner, J.P., Gluck, J. and Laska, M.: 'Odor identification, consistency of label use, olfactory threshold and their relationships to odor memory over the human lifespan', Chemical Senses, Vol. 24, pp., 337-346, 1999.
7. Broughan, C.: 'Odours in culture: How our olfactory world changes with history and fashion', ChemoSense, ISSN: 1442-9098, Vol. 2, No. 4, September 2000.
8. Suskind, P.: 'Perfume: The story of a murderer', Washington Square Press, Washington, 1991.
9. Naidoo, B.: 'Electronic noses turn up in Brighton: Overview of the 7th international symposium on olfaction & electronic nose (ISOEN 2000: 20-24 July 2000)', ChemoSense, ISSN: 1442-9098, Vol. 2, No. 4, September 2000.
10. Harwood, D.: 'Something in the air', IEE Review, January 2001.
11. Gardner, J. W., Bartlett, P. N., Dodd, G. H., and Shurmer, H. V.: 'The design of an artificial olfactory system', pp.131-173 in, Schild, D., (ed.): 'Chemosensory Information Processing', NATO ASI Series Vol. H 39, Springer-Verlag, Berlin, 1990.

12. Sicard, G.: 'Receptor selectivity and dimensionality of odours at the stage of the olfactory receptor cells', pp.21-32 in, Schild, D., (ed.): 'Chemosensory Information Processing', NATO ASI Series Vol. H 39, Springer-Verlag, Berlin, 1990.
13. Kauer, J. S.: 'Coding in the olfactory system', ch9, pp.205-231 in, Finger, T. E., and Silver, W. L., (eds.): 'Neurobiology of taste and smell', Wiley Series in Neurobiology, John Wiley and Sons, New York, 1987.
14. Sobel, N., Khan, R. M., Hartley, C. A., Sullivan, E. V., and Gabrieli, J. D. E.: 'Sniffing longer rather than stronger to maintain olfactory detection threshold', *Chemical Senses* 25, pp.1-8, 2000.
15. Heidland, D., and Kohl, D.: 'Physical and chemical aspects of oxidic semiconductor gas sensors', pp.15-38 in, Seiyama, T. (ed.): 'Chemical sensor technology', vol.1, Elsevier Science Publishers, Amsterdam, 1988.
16. Takahata, K.: 'Tin dioxide sensors – Development and applications', pp.39-55 in, Seiyama, T. (ed.): 'Chemical sensor technology', vol.1, Elsevier Science Publishers, Amsterdam, 1988.
17. Getz, W. M., and Lutz, A.: 'A neural network model of general olfactory coding in the insect antennal lobe', *Chemical Senses* 24, pp.351-372, 1999.
18. Baek, I., Linforth, R. S. T., Blake, A., and Taylor, A. J.: 'Sensory perception is related to the rate of change of volatile concentration in-nose during eating of model gels', *Chemical Senses* 24, pp.155-160, 1999.
19. Getchell, T. V., and Getchell, M. L.: 'Peripheral mechanisms of olfaction: Biochemistry and neurophysiology', ch5 in, Finger, T. E., and Silver, W. L., (eds.): 'Neurobiology of taste and smell', Wiley series in neurobiology, Wiley-Interscience Publication, Canada, 1987.
20. Getchell, T. V., Margolis, F. L., and Getchell, M. L.: 'Perireceptor and receptor events in vertebrate olfaction', *Prog. Neurobiol.* 23, pp.317-345, 1984.
21. Mozell, M. M., Sheehe, P. R., Swieck, S. W., Kurtz, D. B., and Hornung, D. E.: 'A parametric study of the stimulation variables affecting the magnitude of the olfactory nerve response', *J. Gen. Physiol.* 83, pp.233-267, 1984.
22. Mozell, M. M.: 'Olfactory discrimination: Electrophysiological spatiotemporal basis', *Science* 143, pp.1336-1337, 1964.

47. Seiyama, T.: 'Chemical Sensors – Current State and Future Outlook', pp.1-13 in, Seiyama, T. (ed.): 'Chemical sensor technology', vol.1, Elsevier Science Publishers, Amsterdam, 1988.
48. Lee, D.D., and Lee D.S.: 'Environmental gas sensors', IEEE Sensors Journal, Vol. 1, No. 3, October 2001.
49. Barnett, D.: 'Sensors: mimicking the nose', Centre for Chemosensory Research, University of New South Wales, <http://analytical.chem.unsw.edu.au/ccr/sensors.htm>.
50. Nakahara, T., and Koda, H.: 'Tin dioxide gas sensor – a new approach to odor sensing', pp.19-32 in, Seiyama, T. (ed.): 'Chemical sensor technology', vol.3, Elsevier Science Publishers, Amsterdam, 1991.
51. 'Product Catalogue 1: Figaro Gas Sensors, 1-Series, 8-Series', <http://www.omni-components.co.uk/Figaro/Catalogue/1-8SERIES.PDF>.
52. 'Introduction: Figaro Gas Sensors' <http://www.omni-components.co.uk/Figaro/Catalogue/INTRODUCTION.PDF>
53. Nakamura, Y.: 'Stability of the sensitivity of SnO₂-based elements in the field', pp.71-82 in, Seiyama, T. (ed.): 'Chemical sensor technology', vol.2, Elsevier Science Publishers, Amsterdam, 1989.
54. 'General information for TGS sensors', June 1998 revision, <http://www.figarosensor.com/products/common.pdf>.
55. Wada, K., and Egashira, M.: 'Improvement of gas-sensing properties of SnO₂ by surface chemical modification with diethoxydimethylsilane', Sensors and Actuators B, 53, pp.147-154, 1998.
56. D'Amico, A., and Di Natale, C.: 'A contribution on some basic definitions of sensors properties', pp.183-190, IEEE Sensors Journal, Vol. 1, No. 3, October 2001.
57. Naidoo, B., and Broadhurst, A.D.: 'Sensor array data processing using a 2-d discrete cosine transform', pp.153-158 in, Gardner, J.W., and Persaud, K.C., (eds.): 'Electronic noses and olfaction 2000', Series in sensors, Institute of Physics, Bristol, 2000.
58. Ahmed, N., Natarajan, T., Rao, K.R.: 'Discrete cosine transform', pp. 90-93, IEEE Transactions on Computers, January 1974.

59. Gardner, J.W., Hines, E.L., and Tang, H.C.: 'Detection of vapours and odours from a multisensor array using pattern-recognition techniques', Part 2, Artificial neural networks, Sensors and Actuators B, 9, 1992.
60. Holmberg, M., Winquist, F., Lundstrom, I., Gardner, J.W., and Hines, E.L.: 'Identification of paper quality using a hybrid electronic nose' pp. 246-249, Sensors and Actuators B, 26-27, 1995.
61. Elliot, D.F., and Rao, K.R.: 'Fast transforms: algorithms, analyses, applications', Academic Press, New York, 1982.
62. Cvetković, Z., and Popović, M.V.: 'New fast recursive algorithms for the computation of discrete cosine and sine transforms', pp. 2083-2086, IEEE Transactions on Signal Processing, Vol. 40, No. 8, August 1992.
63. Sherlock, B.G., and Monro, D.M.: 'Algorithm 749: Fast discrete cosine transform', pp. 372-378, ACM Transactions on Mathematical Software, Vol. 21, No. 4, December 1995.
64. Strang, G.: 'The discrete cosine transform', pp. 135-147, SIAM Review, Vol. 41, No. 1, Society for Industrial and Applied Mathematics, 1999.
65. Andrews, H.C., 'Multidimensional rotations in feature selection', pp. 1045-1051, IEEE Transactions on Computers, vol. C-20, September 1971.
66. Russell, S.J., and Norvig, P.: 'Artificial intelligence: A modern approach', Prentice Hall, New Jersey, 1995.
67. Bell, G.: 'Poop or scoop? Are you a data snoop?', ChemoSense, Vol. 2, No. 4, September 2000.
68. Cybenko, G.: 'Approximation by superpositions of a sigmoidal function', Mathematics of control signals and systems, 2, pp. 303-314, 1989.
69. Rumelhart, D.E., Hinton, G.E., and Williams, R.J.: 'Learning errors by back-propagating errors', Nature, 323, pp. 533-536, 1986.
70. Stuttgart Neural Network Simulator, version 4.1, Institute for parallel and distributed high performance systems (IPVR), University of Stuttgart.

71. Sartori, M.A., and Antsaklis, P.J.: 'A simple method to derive bounds on the size and to train multiplayer neural networks', IEEE Transactions on neural networks, Vol. 2, No. 4, July 1991.
72. Hoekstra, A.: 'Generalisation in feed forward neural classifiers', PhD thesis, Technische Universiteit Delft, 1998.

Appendix A

SELECTED SOFTWARE IMPLEMENTATIONS

This study was implemented mainly in software. Hardware involvement was limited only to the capturing of raw measurements. Given that much of the software implementation is concerned with mundane issues such as the straightforward implementation of algorithms described in the text, this appendix is selective with respect to the software that is presented. A full listing of all source code may be found in RESULT_CD:\Configuration and its various subdirectories.

A.1 Organisation of Appendix A

The following aspects of the software implementation are presented in this appendix.

- The AWK implementation of the Measurement Control Language (MCL) compiler
- The MCL command set description (Note: the MCL implementation of the measurement event is listed in Chapter 3 and is not repeated here)
- A code selection from the IDE illustrating compiler invocation
- The MCL interpreter
- Selected pre-processing scripts
- Modified Simulator source code (code fragment)

A.2 The Measurement Control Language three-pass compiler

Table A.1: Measurement control language command set

<i>Command</i>	<i>Description</i>
stop	Shutdown the system
open(<i>valve</i>)	Open the specified valve or valve set
close(<i>valve</i>)	Close the specified valve or valve set
flush(<i>begin</i>)	Begin flushing the odorant delivery manifold and sensor head
flush(<i>end</i>)	Stop flushing
rate(<i>high</i>)	Activate high flow rate
rate(<i>low</i>)	Activate low flow rate (measurement flow rate)
capture	Capture the current output of the six sensors (one sample only)
delay(<i>period</i>)	Delay for the specified number of seconds
loop(<i>dest</i> , <i>iterations</i>)	Go <u>back</u> to destination no more than the specified number of iterations
log(<i>open</i>)	Create a new log file for sensor data (filename is auto-generated)
log(<i>close</i>)	Close the currently open log file
macro name(<i>paras</i>)	Begin macro definition with name and parameters specified
mend	End current macro definition

The MCL command set contains twelve directly executable commands that are understood by a command interpreter that is described in section A.3. Two non-executable commands (macro & mend) exist for the sake of macro definition. Macros are expanded during pass two of the three-pass compilation process. The compiled output is written to file in text format. This makes it possible for the programmer to easily inspect and modify the compiled output.

A simple demo script ("Test.nse") is provided below. Sections A.2.1 to A.2.3 illustrate the AWK implementation of each compiler pass and the output as each pass processes the demo script.

Listing of Test.nse, the original demo script:

```
# This is a simple demo program
# It is designed to demonstrate compilation output only

#####Example macro#####
macro sample (cell, dur, NumSamples)
{
    # air flow via sample cell number "cell"
    open( cell )
    # capture several samples: "NumSamples" + 1
    # delay for "dur" seconds between captures
    rpt:
        capture
        delay( dur )
        loop( rpt, NumSamples)
    close(cell)
}
mend
#####End of Macro#####

#####MAIN PROGRAM#####
# capture 21 samples each from sample cells 1,2 & 3
# sampling period 2 seconds
# flow rate is low
# delay 20 seconds between changing cells

# open the log file
log(open)

rate(low)

sample(1, 2, 20 )
delay(20)
sample(2, 2, 20 )
delay(20)
sample(3, 2, 20 )
log(close)

stop
```

A.2.1 First pass compiler script

Filename: Pass1

Description: The first pass is implemented as an AWK script. The script reads the MCL input file, removes all comments, brackets and punctuation marks, and prints the uncommented output to the standard output. The standard output is redirected to a file ("pass1.out") by the calling program "compile.bat" which is described in section A.2.4.

Pass1 listing:

```
BEGIN {ORS = " "}
NF > 0 {split($0, a) #split input line into array of fields
  i = 1
  #now count the number of fields from the left, that do
  #not start with the "#" character
  while (substr(a[i], 1, 1) != "#" && i <= NF) i++
  #print those fields
  for(y=1; y < i; y++) {
    out = a[y]
    gsub(/[(,){}\[\]]/, " ", out)
    print out
  }
  if (i != 1) print "\n"}
END {ORS = "\n"
  print}
```

Pass1 output for Test.nse:

```
macro sample cell dur NumSamples

open cell
rpt:
capture
delay dur
loop rpt NumSamples
close cell

mend
log open
rate low
sample 1 2 20
delay 20
sample 2 2 20
delay 20
sample 3 2 20
log close
stop
```


A.2.2 Second pass compiler script

Filename: Pass2

Description: The second pass is also implemented as an AWK script. The script reads the uncommented MCL output file ("pass1.out") from pass one. It then buffers all macro definitions and expands macro insertions in the main program. The uncommented macro expanded output is written to the standard output. The standard output is redirected to a file ("pass2.out") by the calling program "compile.bat" which is described in section A.2.4.

Pass2 listing:

```
#Macro processor for enose script
BEGIN {      mcro = "0"
            mline = 0
            domacro = "false"}

$1 == "macro" && NF >= 2 {
    mcro = $2
    macros[$2] = NF - 2
    macro[mcro,"args","0"] = 0
    macro[mcro,"lines","0"] = 0
    for (j = 3; j <= NF; j++) {
        macro[mcro,"args",j-2] = $j
        macro[mcro,"args","0"]++
    }
    next
}

mcro != "0" && $1 != "mend" { mline++
    macro[mcro,"lines",mline] = $0
    macro[mcro,"lines","0"]++ }

$1 == "mend" { mcro = 0
    mline = 0
    next }

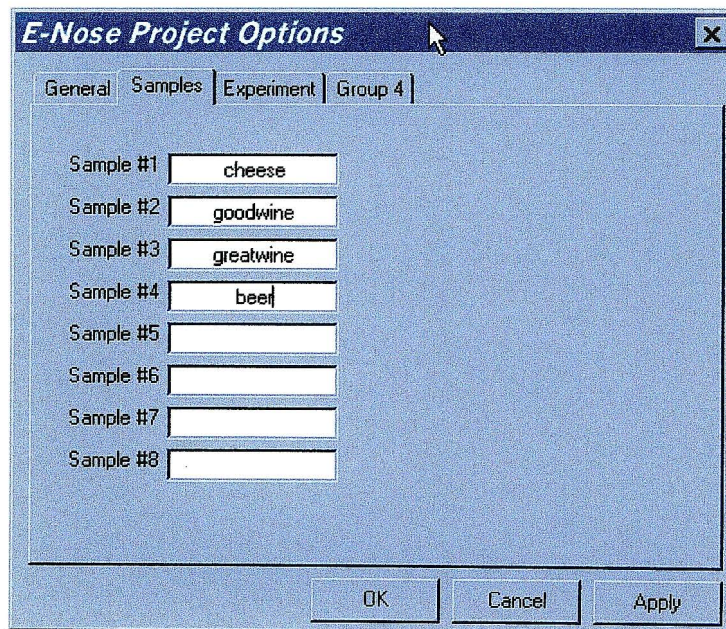
mcro == "0" {for (mac in macros) {
    if ($1 == mac && macros[mac] == NF -1 ) {
        domacro = "true"
        for (cmds = 1; cmds <= macro[mac,"lines","0"]; cmds++) {
            out = macro[mac,"lines",cmds]
```

Pass3 output for Test.nse:

```
1 23 0 0
2 10 0 0
3 1 0 0
4 20 0 0
5 21 2 0
6 22 4 20
7 11 0 0
8 21 20 0
9 2 0 0
10 20 0 0
11 21 2 0
12 22 10 20
13 12 0 0
14 21 20 0
15 3 0 0
16 20 0 0
17 21 2 0
18 22 16 20
19 13 0 0
20 24 0 0
21 0 0 0
```

A.2.4 Final control script

The Project options window is used to input the sample category names. This information is inserted into the final script header.



Final output IDE (Test.scr) that includes header:

```
[CFG]
Sample 1:
cheese
Sample 2:
goodwine
Sample 3:
greatwine
Sample 4:
beer
Sample 5:
Empty
Sample 6:
Empty
Sample 7:
Empty
Sample 8:
Empty
Experiment:
4
Date:
18/09/2003
[END]
[CODE]
1 23 0 0
2 10 0 0
3 1 0 0
4 20 0 0
5 21 2 0
6 22 4 20
7 11 0 0
8 21 20 0
9 2 0 0
10 20 0 0
11 21 2 0
12 22 10 20
13 12 0 0
14 21 20 0
15 3 0 0
16 20 0 0
17 21 2 0
18 22 16 20
19 13 0 0
20 24 0 0
21 0 0 0
```


A.2.5 Compiler invocation shell script

Filename: compile.bat

Description: The compiler is invoked via a Microsoft Windows ® command shell (“command.com”) script. The script uses the 32-bit Windows version of the GNU-AWK interpreter (“gawk.exe”) to interpret each stage of the compilation process. Data is transferred appropriately between compiler passes via text files.

Listing:

```
@echo off
gawk.exe -f Pass1 %1 > Pass1.out
gawk.exe -f Pass1 %1| gawk.exe -f Pass2 > Pass2.out
gawk.exe -f Pass1 %1| gawk.exe -f Pass2 | gawk.exe -f Pass3 > pass3.tmp
:wait
if not exist pass3.tmp goto wait
rename pass3.tmp pass3.out
```

A.2.6 Windows IDE Compiler invocation function

Procedure name: BuildIt

Description: The IDE uses the BuildIt procedure to compile the MCL script. BuildIt does the following:

- Copy the compiler files (Pass1, Pass2, Pass3, gawk.exe & compile.bat) to the working directory. We assume that the script is already in this directory.
- Create a header file that contains set-up information such as cheese sample names, experiment number and date.
- Compile the script
- Optionally print the header and compiler outputs to the IDE
- Insert header into the compiler Pass3 output.
- Cleanup the working directory

Note: The header information, which includes cheese sample names, experiment number and date are all entered into the IDE directly. This information is not in the script.

Known bug: BuildIt does not work on Windows 2000 and possibly XP. The Shell invocation fails.

Work around: Press the build button (hammer icon). The IDE will copy all the necessary files to the working directory. Open a command shell in the working directory and type “compile.bat *filename*” where filename is the name of the script to be compiled. The IDE will resume proper function and control of the process automatically when the file “pass3.out” is produced.

Listing:

```
Private Sub BuildIt()  
Dim i As Integer, srcfl As Integer  
Dim shstr As String, hfile As String  
Dim sourcefound As Boolean  
  
'Create header file and write header to it  
hfile = setup.WorkingDir & "\" & setup.HeaderFileName  
Open hfile For Output As #1  
Print #1, "[CFG]"  
For i = 1 To 8  
    Print #1, "Sample " & i & ":"  
    Print #1, setup.Sample(i)  
Next i
```

```

Print #1, "Experiment:"
Print #1, setup.ExpNum
Print #1, "Date:"
Print #1, setup.ExpDate
Print #1, "[END]"
Print #1, "[CODE]"
Close (1)

'Optionally view the header file in the IDE
If setup.ViewHeader = True Then
    LoadNewDoc hfile
    ActiveForm.LoadAFile hfile, 1
End If

'find the source file
sourcefound = False
For srcfl = 1 To frmDocs.Count
    If (Mid(frmDocs(srcfl).Caption, InStrRev(frmDocs(srcfl).Caption, ".") _
+ 1)) = "nse" Then
        sourcefound = True
        Exit For
    End If
Next srcfl

If sourcefound = False Then
    MsgBox "Compile error: Missing source file."
    Exit Sub
Else
    frmDocs(srcfl).SetFocus
    mnuFileSave_Click
    If Len(Dir(setup.WorkingDir & "\pass1.out")) <> 0 Then _
        Kill setup.WorkingDir & "\pass1.out"
    If Len(Dir(setup.WorkingDir & "\pass2.out")) <> 0 Then _
        Kill setup.WorkingDir & "\pass2.out"
    If Len(Dir(setup.WorkingDir & "\pass3.out")) <> 0 Then _
        Kill setup.WorkingDir & "\pass3.out"

    'copy files to working directory
    fcopy
    While Len(Dir(setup.WorkingDir & "\compile.bat")) = 0
        DoEvents
    Wend

    'Execute the compile.bat command shell script
    retval = Shell(setup.WorkingDir & "\compile.bat " & _

```

```

(Mid(ActiveForm.Caption, InStrRev(ActiveForm.Caption, "\" + 1)))

While Len(Dir(setup.WorkingDir & "\pass3.out")) = 0
    DoEvents
Wend

'Optionally print pass1 output to the IDE
If setup.ViewPOut(1) = True Then
    LoadNewDoc setup.WorkingDir & "\pass1.out"
    ActiveForm.LoadAFile setup.WorkingDir & "\pass1.out", 1
End If

'Optionally print pass2 output to the IDE
If setup.ViewPOut(2) = True Then
    LoadNewDoc setup.WorkingDir & "\pass2.out"
    ActiveForm.LoadAFile setup.WorkingDir & "\pass2.out", 1
End If

'Optionally print pass3 output to the IDE
If setup.ViewPOut(3) = True Then
    LoadNewDoc setup.WorkingDir & "\pass3.out"
    ActiveForm.LoadAFile setup.WorkingDir & "\pass3.out", 1
End If

'Write header to compiler output
Open hfile For Input As #1
Open setup.WorkingDir & "\Pass3.out" For Input As #2
LoadNewDoc setup.OutputFileName
ActiveForm.rtfText.Text = StrConv(InputB$(LOF(1), 1), vbUnicode) _
    & StrConv(InputB$(LOF(2), 2), vbUnicode)

'Cleanup
Close (1)
Close (2)
Kill setup.WorkingDir & "\PASS1"
Kill setup.WorkingDir & "\PASS2"
Kill setup.WorkingDir & "\PASS3"
Kill setup.WorkingDir & "\gawk.exe"
Kill setup.WorkingDir & "\compile.bat"

End If

End Sub

```

A.3 MCL interpreter for DOS

The MCL interpreter was written specifically for MS-DOS because the legacy data capture card that was used in this study required an IBM XT with an 8MHz backplane bus and MS-DOS. The interpreter is written in C and can easily be ported to newer operating systems with newer capture cards.

```
#define tcc
#define badd 0x700
//#define data_30s
#include <stdio.h>
#include <stdlib.h>
#include <dos.h>
#include "displib.h"
#include "pc30.h"

void batchman(void);
long filesize(FILE *stream);

unsigned int BASE;
static int AD[6];

void main(void)
{
    printf("\nFinding parallel ports");
    BASE = findLPT(1); /* find LPT1 */
    if (BASE == 0) BASE = findLPT(2); /*find LPT2 */
    if (BASE == 0) {
        printf("\n\nError: cannot find LPT");
        exit (0);
    }

    printf("\n All valves off");
    Valve(0x00,BASE); /* Clear all outputs */
    Auxil(0x00,BASE);
    printf("\ninitialising PC26");
    if ( pc26init() == -1 ) printf("\npc26init failure"), exit(0);
    printf("\nrunning batchman");
    batchman();
}
```

```

        return;
    }

void batchman(void)
{
    int line, cmd, dat1 = 0, dat2 = 0, *bat, *iptr, mode = 1;
    char batname[40], str[80];
    FILE *batptr;

    printf("\nWhat is the name of your batch file? ");
    scanf("%s", &batname);

    if ( ( batptr = fopen( batname,"r" ) ) == NULL ) {
        printf( "Could not open input file.\n" );
        printf( "Exiting program.\n" );
        exit( 0 );
    }

    /* allocate memory for command array - bat will point to start of array */
    bat = (int *) calloc(((int)filesize(batptr) / 8) + 32, sizeof(int));
    /* iptr will point to current instruction in array */
    iptr = bat;
    /* Command array format...
    Each command uses 4 integers. Space is allocated for 256 commands.
    int 1: line number    <- not really required :-)
    int 2: command opcode
    int 3: data1
    int 4: data2
    Now read in each line from the batch file, strip the 4 ints and
    store in the command array. The command array will be used to
    sequence the data capturing process.
    */
    while (!(feof(batptr))) {
        fgets(str, 80, batptr);
        line = 0, cmd = 0, dat1 = 0, dat2 = 0;
        sscanf(str,"%d %d %d %d", &line, &cmd, &dat1, &dat2);
        if (cmd == 22) line = dat2;
        iptr[0] = line, iptr[1] = cmd, iptr[2] = dat1, iptr[3] = dat2;
        printf("\n%d %d %d %d", iptr[0], iptr[1], iptr[2], iptr[3]);
        iptr += 4;
    }

    iptr = bat; /* look at the first command */
    /* Decode and execute the command. If the current command is 0, then stop;
    if the current mode is 0 (i.e. error), then stop. */

```

```

while(iptr[1] != 0 && mode == 1) {
switch (iptr[1]) {
case 1: printf("\n1 -- on"); Valve(128, BASE); break;
case 2: printf("\n2 -- on"); Valve(64, BASE); break;
case 3: printf("\n3 -- on"); Valve(32, BASE); break;
case 4: printf("\n4 -- on"); Valve(16, BASE); break;
case 5: printf("\n5 -- on"); Valve(8, BASE); break;
case 6: printf("\n6 -- on"); Valve(4, BASE); break;
case 7: printf("\n7 -- on"); Valve(2, BASE); break;
case 8: printf("\n8 -- on"); Valve(1, BASE); break;
case 9: printf("\n9 -- on"); Auxil(4, BASE); break;
case 11: printf("\nall -- off"); Valve(0, BASE); break;
case 12: printf("\nall -- off"); Valve(0, BASE); break;
case 13: printf("\nall -- off"); Valve(0, BASE); break;
case 14: printf("\nall -- off"); Valve(0, BASE); break;
case 15: printf("\nall -- off"); Valve(0, BASE); break;
case 16: printf("\nall -- off"); Valve(0, BASE); break;
case 17: printf("\nall -- off"); Valve(0, BASE); break;
case 18: printf("\nall -- off"); Valve(0, BASE); break;
case 19: printf("\nall -- off"); Valve(0, BASE); break;
case 20: capture(AD);
        printf("\n%d %d %d %d %d %d",AD[0],AD[1],AD[2],AD[3],AD[4],AD[5]);
        break;
case 21: printf("\ndelay %d seconds", iptr[2]); delay(iptr[2] * 1000);
break;
case 22: /* branch to another instruction */
        if( iptr[3] > 0) { /* test loop counter */
            iptr[3] -= 1; /*decrement loop counter*/
            iptr = ((iptr[2] - 2) * 4) + bat; /*implement the jump*/
        }
        else {
            iptr[3] = iptr[0]; /* restore loop counter */
        }
        break;
default: Valve(0,BASE), Auxil(0,BASE), mode = 0;
        printf("\nError-all off"); break;
}
iptr += 4; /* Move to next instruction */
}

return;
}

long filesize(FILE *stream)
{

```



```

long curpos, length;

curpos = ftell(stream);
fseek(stream, 0L, SEEK_END);
length = ftell(stream);
fseek(stream, curpos, SEEK_SET);
return length;
}

static int          gl[7];

int pc26init(void)
{
    int i;
    printf("\n\nPC26 initialisation:");
    printf("\n  Setting base address");
    set_base(badd);

    printf("\n  Fault diagnosis");
    if (diag()){
        printf("\n PC-30 fault.");
        return -1;
    }
    printf("\n  Hardware initialisation");
    init();
    printf("\n  Setting clock prescaler");
    ad_prescaler(10);
    printf("\n  Setting clock divider");
    ad_clock(200);
    printf("\n  Setting Channel gains");
    for (i = 0; i < 16; i++) set_gain(i, 0);
    printf("\n  Selecting channels");
    for (i = 0; i < 6; i++) gl[i] = ( i + 2 );
    gl[i] = 16;
    printf("\nPC26 initialised");
    return 0;
}

/* Must be preceded by a call to PC26init */
/* requires pointer to array of 6 integers to store values*/
/* returns -1 for error o for success */
int capture(int *d_a)
{
    if (m_chan(gl, 6, d_a) == ok_30) return 0;
    return -1;
}

```



```

}

/*****
Valve: writes data to the valve drivers
arguments:      unsigned int a
                only lower byte is considered.
                a 1 turns on the corresponding valve
                a 0 turns off the corresponding valve
                masking is implemented

                unsigned int base
                base address of the centronics interface
*****/
int Valve(unsigned int a, unsigned int base)
{
    outportb (base, ~a );
    return 0;
}

/*****
Auxil: writes data to the auxilliary drivers
arguments:      unsigned int a
                only lowest nibble is considered.
                masking is implemented
                a 1 turns on the corresponding output
                a 0 turns off the corresponding output
                all hardware inversions are compensated
                for in code.

                unsigned int base
                base address of the centronics interface
*****/
int Auxil(unsigned int a, unsigned int base)
{
    outportb (base + 2, ~( a ^ 0x0b));
    return 0;
}

/*****
Find the base address of the computer's parallel port.
arguments:      1 - find LPT1
                2 - find LPT2
                3 - find LPT3
return value:   unsigned int - port adress

```

```
0 - no port found
*****/
unsigned int findLPT (int a)
{
    unsigned int far *addrptr; /* pointer to location of LPT address */

    switch (a) {
    case 1:
        addrptr = (unsigned int far *)0x00000408;
        return *addrptr;
    case 2:
        addrptr = (unsigned int far *)0x0000040A;
        return *addrptr;
    case 3:
        addrptr = (unsigned int far *)0x0000040C;
        return *addrptr;
    default:
        return 0;
    }
}
```

A.4 Preprocessing scripts

A.4.1 Fast DCT script

The code indicated below is a modification of the freeware implementation of the fast DCT based on the algorithm proposed by, Z. Cvetkovic & M. V. Popovic in IEEE signal processing, 1992 no. 8.

This program produces the following data types: PUR, DCT, D1C, D2C, ID1 & ID2.

Fdct.c listing:

```
#include <stdio.h>
#include <math.h>
#include <stdlib.h>
#include <string.h>

#define invroot2 0.7071067814
#define M_PI 3.14159265358979

static void rarrwrt(double f[],int n)
{
    int i;

    for (i=0; i<=n-1; i++){
        printf("%4d : %f\n",i,f[i]);
    }
}

/* fast DCT based on IEEE signal proc, 1992 #8, Z Cvetkovic & M V Popovic */

static int N=0;
static int m=0;
static double two_over_N=0;
static double root2_over_rootN=0;
static double *C=NULL;

static void bitrev(double *f, int len)
{
    int i,j,m,halflen;
    double temp;

    if (len<=2) return; /* No action necessary if n=1 or n=2 */
    halflen = len>>1;
    j=1;
    for(i=1; i<=len; i++){
        if(i<j){
            temp=f[j-1];
            f[j-1]=f[i-1];
            f[i-1]=temp;
        }
        m = halflen;
        while(j>m){
            j=j-m;
            m=(m+1)>>1;
        }
        j=j+m;
    }
}

static void inv_sums(double *f)
{
    int ii,stepsize,stage,curptr,nthreads,thread,step,nsteps;
```

```

for(stage=1; stage <=m-1; stage++){
    nthreads = 1<<(stage-1);
    stepsize = nthreads<<1;
    nsteps = (1<<(m-stage)) - 1;
    for(thread=1; thread<=nthreads; thread++){
        curptr=N-thread;
        for(step=1; step<=nsteps; step++){
            f[curptr] += f[curptr-stepsize];
            curptr -= stepsize;
        }
    }
}

static void fwd_sums(double *f)
{
    int ii,stepsize,stage,curptr,nthreads,thread,step,nsteps;

    for(stage=m-1; stage >=1; stage--){
        nthreads = 1<<(stage-1);
        stepsize = nthreads<<1;
        nsteps = (1<<(m-stage)) - 1;
        for(thread=1; thread<=nthreads; thread++){
            curptr=nthreads +thread-1;
            for(step=1; step<=nsteps; step++){
                f[curptr] += f[curptr+stepsize];
                curptr += stepsize;
            }
        }
    }
}

static void scramble(double *f,int len){
    double temp;
    int i,ii1,ii2,halflen,qtrlen;

    halflen = len >> 1;
    qtrlen = halflen >> 1;
    bitrev(f,len);
    bitrev(&f[0], halflen);
    bitrev(&f[halflen], halflen);
    ii1=len-1;
    ii2=halflen;
    for(i=0; i<=qtrlen-1; i++){
        temp = f[ii1];
        f[ii1] = f[ii2];
        f[ii2] = temp;
        ii1--;
        ii2++;
    }
}

static void unscramble(double *f,int len)
{
    double temp;
    int i,ii1,ii2,halflen,qtrlen;

    halflen = len >> 1;
    qtrlen = halflen >> 1;
    ii1 = len-1;
    ii2 = halflen;
    for(i=0; i<=qtrlen-1; i++){
        temp = f[ii1];
        f[ii1] = f[ii2];
        f[ii2] = temp;
        ii1--;
        ii2++;
    }
    bitrev(&f[0], halflen);
    bitrev(&f[halflen], halflen);
    bitrev(f,len);
}

static void initcosarray(int length)
{

```

```

int i,group,base,item,nitems,halfN;
double factor;

//printf("FCT-- new N=%d\n",length);
m = -1;
do{
    m++;
    N = 1<<m;
    if (N>length){
        printf("ERROR in FCT-- length %d not a power of 2\n",length);
        exit(1);
    }
}while(N<length);
if(C != NULL) free(C);
C = (double *)calloc(N,sizeof(double));
if(C == NULL){
    printf("Unable to allocate C array\n");
    exit(1);
}
halfN=N/2;
two_over_N = 2.0/(double)N;
root2_over_rootN = sqrt(2.0/(double)N);
for(i=0;i<=halfN-1;i++) C[halfN+i]=4*i+1;
for(group=1;group<=m-1;group++){
    base= 1<<(group-1);
    nitems=base;
    factor = 1.0*(1<<(m-group));
    for(item=1; item<=nitems;item++) C[base+item-1]=factor*C[halfN+item-1];
}

//printf("before taking cos, C array =\n"); rarrwrt(C,N);
for(i=1;i<=N-1;i++) C[i] = 1.0/(2.0*cos(C[i]*M_PI/(2.0*N)));
//printf("After taking cos, C array = \n"); rarrwrt(C,N);
}

static void inv_butterflies(double *f)
{
    int stage,ii1,ii2,butterfly,ngroups,group,wingspan,increment,baseptr;
    double Cfac,T;

    for(stage=1; stage<=m;stage++){
        ngroups=1<<(m-stage);
        wingspan=1<<(stage-1);
        increment=wingspan<<1;
        for(butterfly=1; butterfly<=wingspan; butterfly++){
            Cfac = C[wingspan+butterfly-1];
            baseptr=0;
            for(group=1; group<=ngroups; group++){
                ii1=baseptr+butterfly-1;
                ii2=ii1+wingspan;
                T=Cfac * f[ii2];
                f[ii2]=f[ii1]-T;
                f[ii1]=f[ii1]+T;
                baseptr += increment;
            }
        }
    }
}

static void fwd_butterflies(double *f)
{
    int stage,ii1,ii2,butterfly,ngroups,group,wingspan,increment,baseptr;
    double Cfac,T;

    for(stage=m; stage>=1;stage--){
        ngroups=1<<(m-stage);
        wingspan=1<<(stage-1);
        increment=wingspan<<1;
        for(butterfly=1; butterfly<=wingspan; butterfly++){
            Cfac = C[wingspan+butterfly-1];
            baseptr=0;
            for(group=1; group<=ngroups; group++){
                ii1=baseptr+butterfly-1;
                ii2=ii1+wingspan;
                T= f[ii2];
                f[ii2]=Cfac *(f[ii1]-T);
            }
        }
    }
}

```

```

        f[i11]=f[i11]+T;
        baseptr += increment;
    }
}
}

static void ifct_noscale(double *f, int length)
{
    if (length != N) initcosarray(length);
    f[0] *= invroot2;
    inv_sums(f);
    bitrev(f,N);
    inv_butterflies(f);
    unscramble(f,N);
}

static void fct_noscale(double *f, int length)
{
    if (length != N) initcosarray(length);
    scramble(f,N);
    fwd_butterflies(f);
    bitrev(f,N);
    fwd_sums(f);
    f[0] *= invroot2;
}

static void ifct_defn_scaling(double *f, int length){
    ifct_noscale(f,length);
}

static void fct_defn_scaling(double *f, int length){
    int i;

    fct_noscale(f,length);
    for(i=0;i<=N-1;i++) f[i] *= two_over_N;
}

void ifct(double *f, int length){
/* CALL THIS FOR INVERSE 1D DCT DON-MONRO PREFERRED SCALING */
    int i;

    if (length != N) initcosarray(length); /* BGS patch June 1997 */
    for(i=0;i<=N-1;i++) f[i] *= root2_over_rootN;
    ifct_noscale(f,length);
}

void fct(double *f, int length){
/* CALL THIS FOR FORWARD 1D DCT DON-MONRO PREFERRED SCALING */
    int i;

    fct_noscale(f,length);
    for(i=0;i<=N-1;i++) f[i] *= root2_over_rootN;
}

/*****
    2D FAST DCT SECTION
*****/

#define VERBOSE 0

static double *g = NULL;
static double two_over_sqrtncolsnrows = 0.0;
static int ncolsvalue = 0;
static int nrowsvalue = 0;

static void initfct2d(int nrows, int ncols){
    if(VERBOSE) printf("FCT2D -- Initialising for new nrows=%d\n",nrows);
    if ((nrows<=0)|| (ncols<0)){
        printf("FCT2D -- ncols=%d or nrows=%d is <=0\n",nrows,ncols);
        exit(1);
    }
    if(g != NULL) free(g);
    g = (double *)calloc(nrows,sizeof(double));
    if(g == NULL){
        printf("FCT2D -- Unable to allocate g array\n");
    }
}

```

```

    exit(1);
}
ncolsvalue = ncols;
nrowsvalue = nrows;
two_over_sqrtncolsnrows = 2.0/sqrt(ncols*1.0*nrows);
}

void fct2d(double f[], int nrows, int ncols)
/* CALL THIS FOR FORWARD 2d DCT DON-MONRO PREFERRED SCALING */
{
    int u,v;

    if ((ncols!=ncolsvalue)|| (nrows!=nrowsvalue)){
        initfct2d(nrows,ncols);
    }
    for (u=0; u<=nrows-1; u++){
        fct_noscale(&f[u*ncols],ncols);
    }
    for (v=0; v<=ncols-1; v++){
        for (u=0; u<=nrows-1; u++){
            g[u] = f[u*ncols+v];
        }
        fct_noscale(g,nrows);
        for (u=0; u<=nrows-1; u++){
            f[u*ncols+v] = g[u]*two_over_sqrtncolsnrows;
        }
    }
}

void ifct2d(double f[], int nrows, int ncols)
/* CALL THIS FOR INVERSE 2d DCT DON-MONRO PREFERRED SCALING */
{
    int u,v;

    if ((ncols!=ncolsvalue)|| (nrows!=nrowsvalue)){
        initfct2d(nrows,ncols);
    }
    for (u=0; u<=nrows-1; u++){
        ifct_noscale(&f[u*ncols],ncols);
    }
    for (v=0; v<=ncols-1; v++){
        for (u=0; u<=nrows-1; u++){
            g[u] = f[u*ncols+v];
        }
        ifct_noscale(g,nrows);
        for (u=0; u<=nrows-1; u++){
            f[u*ncols+v] = g[u]*two_over_sqrtncolsnrows;
        }
    }
}

/*****
    UNCOMMENT THIS SECTION TO TEST 1D FAST DCT
*****/

/*
int main(void)
{
    double *f=NULL;
    int i,nsiz;

    do{
        printf("Enter nsiz:");
        scanf("%d", &nsiz);
        printf("nsiz=%d\n",nsiz);
        if(nsiz==0)break;
        f = (double *)calloc(nsiz,sizeof(double));
        if(f == NULL){
            printf("Unable to allocate f array\n");
            exit(1);
        }
        for (i=0; i<=nsiz-1; i++){
            f[i]= (i+1)*1.0;
        }
    }
}

```



```

    f[2] = 42.0;

    printf("Before fct f[] is:\n");
    rarrwrt(f,nsiz);

    fct(f,nsiz);

    printf("After fct f[] is:\n");
    rarrwrt(f,nsiz);

    ifct(f,nsiz);

    printf("After ifct f[] is:\n"); rarrwrt(f,nsiz);
    free(f);
}while(l==1);
return(0);
}

*/

float tukey(int n, int N,float alpha){
/*
%Tukey window generator - By Bashan Naidoo
%... tukey(N, alpha, M, cols)
%n      -sample number
%N      -window length
%alpha  -fraction of window taken up by the rolloff
%
%The calculated window is returned, and a new graph of the window is plotted
%
%This window function generates a right half (positive) tukey window. The window
%commences at the first sample in the output stream. The first N samples belong to
%the window. The rest are set to zero. The last alpha percent of the first N samples
%are in the form of a cosine roll-off from the max value of 1 to 0.

%output = 1 for 0 <= n <= alpha*N/2
%output = 0.5[1.0 + cos( pi*(n - alpha*N/2) / 2*(1 - alpha)*N/2)] for alpha*N/2 < n <= N/2
%alpha must be between 0 and 1
*/

    if( n > N ) return 0.0;
    if( n <= (int)(alpha * N ) ) return 1.0;

    N *= 2;
    return (float)(0.5 * (1.0 + cos( 2*M_PI*(n - (alpha*N/2)) / (2 * (1-alpha) * N/2))));
}

/*****
UNCOMMENT THIS SECTION TO TEST 2D FAST DCT
*****/

static void rarrwrt2d(double f[],int nrows,int ncols)
{
    int row;

    for (row=0;row<=nrows-1; row++){
        printf("Row %4d\n",row);
        rarrwrt(&f[row*ncols],ncols);
    }
}

main(int argc, char *argv[])
{
    double *f=NULL;
    float fp, ALPHA;
    int i,j,nrows = 0,ncols = 0, r, c, dotdisp, WL;
    FILE *infile, *ofile;
    char dline[200], ifilename[100], dlfilename[100], d2filename[100], dctfilename[100];

```



```

char id1filename[100], id2filename[100], direc[100] = {"\0"}, ifile[100] = {"\0"},
ifileshort[100] = {"\0"};
char *slashloc, *dotposition;

if ( argc > 1 ) {
    for ( i = 0; i < argc - 1; i++ ) {
        if ( strcmp(argv[i], "-f") == 0 ) strcpy (ifile, argv[i+1]);
        if ( strcmp(argv[i], "-d") == 0 ) strcpy (direc, argv[i+1]);
        if ( strcmp(argv[i], "-rows") == 0 ) sscanf(argv[i+1], "%d", &nrows);
        if ( strcmp(argv[i], "-cols") == 0 ) sscanf(argv[i+1], "%d", &ncols);
    }
    if (ifile == "\0" || direc == "\0" || nrows == 0 || ncols == 0) {
        for ( i = 0; i < argc; i++) printf("\n argv[%d] = %s", i, argv[i]);
        printf(" \n%s\n%s\n%d\n%d", ifile, direc, nrows, ncols);
        printf("\nProper usage:");
        printf("\nfdct -d workingdir -f datafilename -rows #rows -cols #cols");
        exit(1);
    }
} else {
    printf("Enter name of working directory: ");
    scanf("%s", &direc);
    printf("\nEnter input file name: ");
    scanf("%s", &ifile);
    printf("\n\nEnter nrows:");
    scanf("%d", &nrows);
    printf("\nEnter ncols:");
    scanf("%d", &ncols);
}

//see if the directory terminates in a slash...
slashloc = strrchr( direc, '\\');
i = strlen(direc);
if (slashloc == NULL || ((slashloc - direc + 1) < i)) strcat( direc, "\\");

/* slashloc = strchr( direc, (int)"\\");
//if (slashloc == NULL || (slashloc - direc < sizeof(direc))) slashloc = strchr( direc,
(int)"/");
if (slashloc == NULL || (slashloc - direc < sizeof(direc))) strcat( direc, "\\");
*/
printf("\nWorking directory... %s", direc);

//generate input filename

dotposition = strrchr( ifile, (int)'.');
dotdisp = dotposition - ifile + 1;
strncpy(ifileshort, ifile, dotdisp - 1);
strcat(ifileshort, "\0");
strcpy(id1filename, direc);
strcat(id1filename, ifileshort);
printf("\n\nInput filename... %s.reg", id1filename);

//generate DCT output filename
strcpy(dctfilename, id1filename);
strcat(dctfilename, ".dct");
printf("\nDCT output filename... %s", dctfilename);

//generate D1 (DCT with no DC terms) output filename
strcpy(dlfilename, id1filename);
strcat(dlfilename, ".d1");
printf("\nDCT with no DC terms... %s", dlfilename);

//generate D2 (DCT with no DC & HF terms) output filename
strcpy(d2filename, id1filename);
strcat(d2filename, ".d2");
printf("\nDCT with no DC & HF terms... %s", d2filename);

//generate inverse D1 (DCT with no DC terms) output filename
strcpy(id1filename, id1filename);
strcat(id1filename, ".id1");
printf("\ninverse DCT with no DC terms... %s", id1filename);

//generate inverse D2 (DCT with no DC & HF terms) output filename
strcpy(id2filename, id1filename);
strcat(id2filename, ".id2");

```

```

printf("\ninverse DCT with no DC & HF terms...  %s", id2filename);

//sprintf(ifilename,"c:\\My Documents\\FDCT\\Debug\\data1i");
//sprintf(ofilename,"c:\\My Documents\\FDCT\\Debug\\data1o");
//sprintf(invofilename,"c:\\My Documents\\FDCT\\Debug\\data1oi");

f = (double *)calloc(nrows*ncols,sizeof(double));
if(f == NULL){
    printf("Unable to allocate f array\n");
    exit(1);
}

//scan input file into array
fp = 0.0;
strcpy(ifile, ifileshort);
strcat(ifile, ".reg");
if((infile = fopen(ifile, "rt")) == NULL ) {
    printf("\nUnable to open input file");
    exit(1);
}
    for( r = 0; r <= nrows - 1 ; r++){
        for( c = 0; c <= ncols - 1; c++ ){
            fscanf(infile, "%f", &fp);
            f[(r * ncols) + c] = fp;
            // printf("\nf[%d] = %f", (r*ncols )+c, f[(r * ncols) + c]);
        }
    }
fclose(infile);

//printf("Before fct2d f[] is:\n");
//rarrwrt2d(f,nrows,ncols);

//calculate 2D DCT
fct2d(f,nrows,ncols);

//printf("After fct2d f[] is:\n");
//rarrwrt2d(f,nrows,ncols);

//Output DCT data to .dct file
ofile = fopen(dctfilename, "wt");
    for( r = 0; r <= nrows - 1 ; r++){
        for( c = 0; c <= ncols - 1; c++ ){
            fprintf(ofile,"%f ", f[(r * ncols) + c]);
        }
        fprintf(ofile, "\n");
    }
fclose(ofile);

//clear all dc terms
for( r = 0; r < ncols; r++) f[r] = 0;
//f[0] = 0.0;

//Output DCT data without DC terms to .dl file
ofile = fopen(dlfilename, "wt");
    for( r = 0; r <= nrows - 1 ; r++){
        for( c = 0; c <= ncols - 1; c++ ){
            fprintf(ofile,"%f ", f[(r * ncols) + c]);
        }
        fprintf(ofile, "\n");
    }
}
fclose(ofile);

//compute inverse DCT of .dl data (will destroy .dl data)
ifct2d(f,nrows,ncols);

//Output inverse DCT without DC terms to .idl file
ofile = fopen(idlfilename, "wt");
    for( r = 0; r <= nrows - 1 ; r++){
        for( c = 0; c <= ncols - 1; c++ ){
            fprintf(ofile,"%f ", f[(r * ncols) + c]);
        }
    }
}

```

```

        fprintf(ofile, "\n");
    }
fclose(ofile);

//RESCAN .d1 data file into array
fp = 0.0;
infile = fopen(dlfilename, "rt");
for( r = 0; r <= nrows - 1 ; r++){
    for( c = 0; c <= ncols - 1; c++ ){
        fscanf(infile, "%f", &fp);
        f[(r * ncols) + c] = fp;
        // printf("\nf[%d] = %f", (r*ncols )+c, f[(r * ncols) + c]);
    }
}
fclose(infile);

//Set HF terms = 0.0 (i.e. from row 20 onwards)
WL = 50; //window length
ALPHA = 0.75; //roll-off starts at 0.75 window length
for( r = 0; r <= nrows - 1 ; r++){
    for( c = 0; c <= ncols - 1; c++ ){
        f[(r * ncols) + c] *= tukey(r,WL,ALPHA);
    }
}

//Output DCT data without DC & HF terms to .d2 file
ofile = fopen(d2filename, "wt");
for( r = 0; r <= nrows - 1 ; r++){
    for( c = 0; c <= ncols - 1; c++ ){
        fprintf(ofile,"%f ", f[(r * ncols) + c]);
    }
    fprintf(ofile, "\n");
}
fclose(ofile);

//compute inverse DCT of .d2 data (will destroy .d2 data)
ifct2d(f,nrows,ncols);

//Output inverse DCT without DC & HF terms to .id2 file
ofile = fopen(id2filename, "wt");
for( r = 0; r <= nrows - 1 ; r++){
    for( c = 0; c <= ncols - 1; c++ ){
        fprintf(ofile,"%f ", f[(r * ncols) + c]);
    }
    fprintf(ofile, "\n");
}
fclose(ofile);

//printf("After ifct2d f[] is:\n");
//rarrwrt2d(f,nrows,ncols);
}

```

A.4.2 D3C coefficient selection script

This program takes D2C files as input and produces DC3 files. The program must be informed which coefficients to select. Coefficient ranking is performed in Matlab. A configuration file is produced informing d3clean.c what coefficients should be selected. Depending on the selection, d3clean.c will produce the following dat types: D3C2 to D3C8, D3CA & D3CB.

d3clean.c listing:

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <math.h>
/*
This program takes D2C data and converts it to D3C data.
D3C data is a selection of up to 20 coefficients from the D2C pattern vector type.
The program scans the current directory and generates a file the lists all the D2C input
files to be processed. The corresponding D3C output files will be generated if they dont
already
exist. The program also requires a selection file that indicates
the coefficients for selection. The selection file has the following format:
Line 1: Row   Col
Line 2: 2     3
Line 3: 48    5
etc...
There can be a maximum of 20 selected coeffs in the file. The Col figure must be in the range
1...8
*/
int main(int argc, char *argv[]){

    FILE *d2clst, *d3clst, *d2cfile, *outfile, *sfile;
    char direc[100] = {"\0"}, cmdstr[100], d3cname[20], d2cname[20], outname[20], line[100],
selectfile[100];
    int nxtname = 0, i, dotdisp, slash = '\\', lines = 0, sel[20][2];
    char *dotposition, *slashloc;
    float f1, f2, f3, f4, f5, f6, f7, f8;

    if ( argc > 1 ) {
        for ( i = 0; i < argc - 1; i++ ) {
            if ( strcmp(argv[i], "-d") == 0 ) strcpy (direc, argv[i+1]);
            if ( strcmp(argv[i], "-f") == 0 ) strcpy (selectfile, argv[i+1]);
        }
        if (direc == "\0" ) {
            //for ( i = 0; i < argc; i++) printf("\n argv[%d] = %s", i, argv[i]);
            //printf(" \n%s", direc);
            printf("\nProper usage:");
            printf("\nlstgen -d workingdir -f selectionfile");
            exit(1);
        }
    } else {
        printf("Enter name of working directory: ");
        scanf("%s", &direc);
        printf("Enter name of selection file: ");
        scanf("%s", &selectfile);
    }

    //see if the directory terminates in a slash...
    slashloc = strrchr( direc, slash);
    i = strlen(direc);
    if (slashloc == NULL || ((slashloc - direc + 1) < i))    strcat( direc, "\\");

    fflush();
    strcpy(cmdstr, "dir /B ");
```

```

strcat(cmdstr, direc);
strcat(cmdstr, "*.d2c > ");
strcat(cmdstr, direc);
strcat(cmdstr, "listd2c.txt \n");
printf("\n%s",cmdstr);
system(cmdstr);

flushall();
strcpy(cmdstr, "dir /B ");
strcat(cmdstr, direc);
strcat(cmdstr, "*.d3c > ");
strcat(cmdstr, direc);
strcat(cmdstr, "listd3c.txt \n");
printf("\n%s",cmdstr);
system(cmdstr);

strcpy( cmdstr, direc);
strcat( cmdstr, selectfile);
if( (sfile = fopen( cmdstr, "rt")) == NULL) {
    printf("\nCould not open coefficient selection file.");
    exit(1);
}

while (!feof(sfile)){
    fgets(line, 100, sfile);
    lines++;
}
rewind(sfile);
if (lines > 21) printf("\nMaximum of 20 coefficients allowed in selection list. Using first
20.");
fgets(line,100,sfile); //remove first line - header: row col
for( i = 1; i < lines; i++ ){
    fscanf(sfile, "%*d %d %d\n", &sel[i][1], &sel[i][2]);
}
printf("\nSelected coefficients:");
printf("\nRow Col");
for( i = 1; i < 21; i++ ){
    printf("\n%d %d", sel[i][1], sel[i][2]);
}
fclose(sfile);

strcpy( cmdstr, direc);
strcat( cmdstr, "listd2c.txt");
if( (d2clst = fopen( cmdstr, "rt")) == NULL) {
    printf("\nCould not open listd2c.txt.");
    exit(1);
}

strcpy( cmdstr, direc);
strcat( cmdstr, "listd3c.txt");
if( (d3clst = fopen( cmdstr, "rt")) == NULL) {
    printf("\nCould not open listd3c.txt.");
    exit(1);
}

while (!feof(d2clst)){
    fscanf( d2clst, "%s\n", &d2cname);
    printf("\n\n****Processing %s****", d2cname);
    dotposition = strrchr( d2cname, (int)'.');
    dotdisp = dotposition - d2cname +1;
    fseek(d3clst, 0, SEEK_SET);
    nxtname = 0;
    while(!feof(d3clst)){
        fscanf( d3clst, "%s\n", &d3cname);
        if (strncmp( d2cname, d3cname, dotdisp) == 0) {
            nxtname = 1;
            printf(" ...cancelled");
            break;
        }
    }
    if( nxtname == 1) continue;

    strncpy(d3cname, d2cname, dotdisp );
    strncpy(d3cname + dotdisp, "\0", 1);
    printf(" %s ...wait", d3cname);
}

```



```

strcpy( cmdstr, direc);
strcat( cmdstr, d2cname);
if( d2cfile = fopen( cmdstr, "rt")) == NULL) {
    printf("\nCould not open %s.", cmdstr);
    exit(1);
}

//Here we remove all header info and blank lines - store results in *.out
strcpy( outname, direc);
strcat( outname, d3cname);
strcat( outname, "d3c");
if( outfile = fopen( outname, "w+t")) == NULL) {
    printf("\nCould not open %s.", outname);
    exit(1);
}

```

/* This is the old code. It works but store coeffs in order of their line number.
The modified code below stores them in order of there apperance in the selection file.

```

lines = 0;
while(!feof(d2cfile)){
    if (fgets(line, 100, d2cfile) == NULL) continue;
    if(sscanf(line, "%f %f %f %f %f %f %f %f\n", &f1, &f2, &f3, &f4, &f5,
&f6, &f7, &f8) == 8) {
        lines++;
        for(i = 0; i < 20; i++){
            if(lines == sel[i][1]){
                switch( sel[i][2] ){
                    case 1: fprintf(outfile, "%f ", f1); break;
                    case 2: fprintf(outfile, "%f ", f2); break;
                    case 3: fprintf(outfile, "%f ", f3); break;
                    case 4: fprintf(outfile, "%f ", f4); break;
                    case 5: fprintf(outfile, "%f ", f5); break;
                    case 6: fprintf(outfile, "%f ", f6); break;
                    case 7: fprintf(outfile, "%f ", f7); break;
                    case 8: fprintf(outfile, "%f ", f8); break;
                }
            }
        }
    }
}

*/

for(i = 0; i < 20; i++){
    lines = 0;
    while(!feof(d2cfile)){
        if (fgets(line, 100, d2cfile) == NULL) continue;
        lines++;
        if(lines == sel[i][1]){
            if(sscanf(line, "%f %f %f %f %f %f %f %f\n", &f1, &f2,
&f3, &f4, &f5, &f6, &f7, &f8) == 8){
                switch( sel[i][2] ){
                    case 1: fprintf(outfile, "%f ", f1);
break;
                    case 2: fprintf(outfile, "%f ", f2);
break;
                    case 3: fprintf(outfile, "%f ", f3);
break;
                    case 4: fprintf(outfile, "%f ", f4);
break;
                    case 5: fprintf(outfile, "%f ", f5);
break;
                    case 6: fprintf(outfile, "%f ", f6);
break;
                    case 7: fprintf(outfile, "%f ", f7);
break;
                    case 8: fprintf(outfile, "%f ", f8);
break;
                }
            }
        }
    }
    rewind(d2cfile);
}

```



```

        fclose(d2cfile);
        fclose(outfile);

        //sprintf(cmdstr, "%sfdct -d %s -f %s -rows %d -cols 8", direc, direc, d3cname,
n);
        //printf("\n%s\n",cmdstr);
        //system(cmdstr);
    }

    fclose(d2clst);
    fclose(d3clst);
    return 0;
}

```

A.4.3 Pattern-set generator script

The programs `fdct.c` and `d3clean.c` process individual measurement files in a batch process. These files are assembled into datasets by `patgen.c`.

Listin of `Patgen.c`:

```

#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <time.h>

void putpattern( FILE *outfile, char *infilename, char *catname, int catcount, int catnum, int
patnum, int setnum);

void putpattern( FILE *outfile, char *infilename, char *catname, int catcount, int catnum, int
patnum, int setnum){

    FILE *infile;
    char data[100] = {"\0"};
    int n;

    if( (infile = fopen( infilename, "rt" )) == NULL ) {
        printf("\n could not open %s", infilename );
        exit(1);
    }

    //read data and write to pattern file
    fprintf(outfile, "\n#***** pattern %d *****", setnum);
    fprintf(outfile, "\n#input vector %d", patnum);
    fprintf(outfile, "\n#File: %s", infilename);
    fprintf(outfile, "\n#Category: %s\n", catname);
    while(!feof(infile)){
        sprintf(data, "\0\0\0");
        fgets(data, 100, infile);
        if( strlen(data) > 1) fputs(data, outfile);
    }
    fclose(infile);

    //print output vector
    fprintf(outfile, "\n#output vector %d\n", patnum);
    for(n = 0; n < catcount; n++){
        if(n == catnum) fprintf(outfile,"1 ");
        else fprintf(outfile,"0 ");
    }
    fprintf(outfile, "\n\n");
    return;
}

```

```

int main(int argc, char *argv[]){

    FILE *lst, *dfile, *pfile, *trfile, *vafile, *tsfile;
    char direc[100] = {"\0"}, lstfilename[100] = {"\0"}, cmdstr[100], line[100], dfilename[100],
    data[100];
    int exp, incr, i, cat = 0, fields = 0, datacat = 0, tmp = 0, patsize = 0, catcount[25] =
    {0}, *distrib[25] = {NULL}, distribcnt[25] = {0};
    int TS = 40, TR = 30, VA = 30, val, x, y, z, trpats, vapats, tspats, l, m;
    long n = 1, lines, vecs;
    char *slashloc, categories[25][20] = {"\0"}, ext[5] = {"\0"}, samplecat[20] = {"\0"};
    time_t t;
    unsigned int rndseed = 0;
    int pats[4] = {0};
    float datum2;

    //scan command line args
    if ( argc > 1 ) {
        for ( i = 0; i < argc - 1; i++ ) {
            if ( strcmp(argv[i], "-d") == 0 ) strcpy (direc, argv[i+1]);
            if ( strcmp(argv[i], "-l") == 0 ) strcpy (lstfilename, argv[i+1]);
            if ( strcmp(argv[i], "-tr") == 0 ) sscanf (argv[i+1], "%d", &TR);
            if ( strcmp(argv[i], "-ts") == 0 ) sscanf (argv[i+1], "%d", &TS);
            if ( strcmp(argv[i], "-va") == 0 ) sscanf (argv[i+1], "%d", &VA);
            if ( strcmp(argv[i], "-seed") == 0 ) sscanf (argv[i+1], "%u", &rndseed);
        }
        if (direc == "\0" || lstfilename == "\0" ) {
            printf("\nProper usage:");
            printf("\nlstgen -d datadir -l listfile(*.lst) -tr trsetpercentage -ts
tssetpercentage -va vasetpercentage -seed randomseed ");
            exit(1);
        }
    } else { //no args? then ask for input
        printf("Enter name of data directory: ");
        scanf("%s", &direc);
        printf("Which list must be analysed? (list filename): "); //all files in the list
        must have the same size.
        scanf("%s", &lstfilename);
        // printf("Enter name of result file: ");
        // scanf("%s", &resfilename);
    }

    //see if the directory terminates in a slash...
    slashloc = strchr( direc, '\\');
    i = strlen(direc);
    if (slashloc == NULL || ((slashloc - direc + 1) < i))   strcat( direc, "\\");

    //open the file list - files in the list are RAW files with header info
    strcpy( cmdstr, direc);
    strcat( cmdstr, lstfilename);
    if( (lst = fopen( cmdstr, "rt")) == NULL) {
        printf("\nCould not open %s.", cmdstr);
        exit(1);
    }

    //Scan through all the files in the lst file. Get their categories and pattern sizes
    //each category MUST be equally represented in the lst file
    while( !feof(lst)){
        //open a rawfile companion for the listed data file - raw file contains header for
        the data file
        sprintf(line, "\0\0\0");
        fgets(line, 100, lst);
        if (strlen(line)<=4) continue;
        sscanf(line, "%3d %4d.%s", &exp, &incr, &ext);
        sprintf(dfilename, "%s%03d %04d.raw", direc, exp, incr);
        if( (dfile = fopen( dfilename, "rt" )) == NULL ) {
            printf("\n could not open %s", dfilename );
            continue;
        }
        //scan header info for category name
        while(!feof(dfile)){
            fgets(data, 100, dfile);
            if(sscanf(data, "Sample: %s\n", &samplecat) != 1) continue;
            if( strcmp(samplecat, "Air") == 0 || strcmp(samplecat, "blank") == 0 ||
            strcmp(samplecat, "air") == 0 ) continue;

```

```

        break;
    }
    fclose(dfile);
    if (strcmp(samplecat, "\0")== 0) {
        printf("\n Could not find sample category in file %s", dfilename);
        continue;
    }

    //cat points to the first empty slot in the category name array "categories"
    // If a new category is found, allocate memory space for the data
    // open data file to see how much data it contains...
    sprintf( dfilename, "%s%03d_%04d.%s", direc, exp, incr, ext);
    if( (dfile = fopen( dfilename, "rt" )) == NULL ) {
        printf("\n could not open %s", dfilename );
        continue;
    }

    //find the number of lines in the file
    lines = 0;
    while(!feof(dfile)){
        sprintf(data, "\0\0\0");
        fgets(data, 100, dfile);
        if(strlen(data) >= 4 ) lines++; //count only non-empty lines
    }
    rewind(dfile);

    //figure out number of fields in a file from the file extension
    if (strcmp(ext, "pur") == 0) fields = 6;
    if (strcmp(ext, "d3c") == 0) {
        for(fields = 0; fields < 8; fields++) if (fscanf(dfile,"%f", &datum2) == EOF)
break;
    }
    else fields = 8;
    fclose(dfile);

    if(patsize == 0) patsize = lines * fields;
    if((lines * fields) != patsize) {
        printf("\n Error; incorrect pattern size in %s ", dfilename);
        exit(1);
    }

    //see if category is new
    for( datacat = 0; datacat <= cat; datacat++ ) {
        tmp = strlen(samplecat);
        if( strcmp(categories[datacat], samplecat, tmp) == 0 ) break;
    }
    //if new, then add to category list
    if ( datacat == cat + 1 ){
        datacat--;
        strcpy(categories[cat++], samplecat);
        //allocate mem: "fields" floats per line * number of lines
    }

    catcount[datacat]++; //increment the pattern count for this category
    n++;
}

//Print number of files processed and categories found
printf("\n\n %ld files have been processed.\n", --n);
vecs = n;
printf("\nCategories found... ");
for(n = 0; n < cat; n++) printf("\n%3d    %s    %d patterns", (int)n, categories[n],
catcount[n]);
rewind(lst);

//see that all categories are equally represented in the list file
// ie. compare the size of each category to the first category
for(n = 0; n < cat; n++) {
    if (catcount[n] != catcount[0]) {
        printf("\n Skew data set error - uneven category representation in list
file!");
        exit(1);
    }
}
}

```

```

//Now create arrays for each category (one element per pattern)
//Each array element will have value 1,2 or 3.
//Depending on the value of the element, the pattern will be written to the
1.training, 3.testing or 2.validation set.
//There will also be one more pattern file (the "output" file) that contains all
patterns.
trpats = (int)(TR * ((float)catcount[0]/100.0));
vapats = (int)(VA * ((float)catcount[0]/100.0));
tspats = catcount[0] - vapats - trpats;

for(n = 0; n < cat; n++) distrib[n] = (int *) calloc(catcount[0], sizeof(int));
for(n = 0; n < catcount[0]; n++) {
    if( n < trpats ) val = 1;
    else if (n < ( vapats + trpats) ) val = 2;
    else val = 3;
    for( x = 0; x < cat; x++) distrib[x][n] = val;
}
//now shuffle the order of data in these distrib[] arrays

printf("\nShuffling pattern set");
if (rndseed == 0) srand( (unsigned)time( NULL ) ); // set a new seed number - use random
seed
else srand(rndseed); //use user specified seed

for(n = 0; n < cat; n++){
    for( i = 0; i < 10000; i++){
        x = (int)((float)rand() / 0x7fff) * ( catcount[0] - 1));
        y = (int)((float)rand() / 0x7fff) * ( catcount[0] - 1));
        if( abs(x) > catcount[0] || abs(y) > catcount[0] ) continue;
        z = distrib[n][x];
        distrib[n][x] = distrib[n][y];
        distrib[n][y] = z;
    }
}
//for debug purposes
for(l = 0; l < cat; l++){
    for(m = 0; m < catcount[0]; m++) printf(" %d", distrib[l][m]);
    printf("\n");
}
//now that the numbers are shuffled we can write the corresponding patterns to the
various files.
//There will be no skew as the number of 1's, 2's & 3's are the same in each category.
//Open the pattern files in there original order - as indicated in the 1st file
//find the category of the pattern and look at the distrib array for that class.
//if the corresp distrib array element has value 1 then write the pattern to the training
set... etc
//a counter is required for each distrib array to show which is the current element
starting from 0.

//open the pattern ****OUTPUT**** file - will contain all patterns
sprintf( dfilename, "%s%3s%03d.pat", direc, ext, exp );
if( (pfile = fopen( dfilename, "w+t" )) == NULL ) {
    printf("\n could not open %s", dfilename );
    exit(1);
}

time(&t);
fprintf (pfile, "SNNS pattern definition file V1.4\n");
fprintf (pfile, "generated at %s\n\n", ctime( &t));
fprintf (pfile, "No. of patterns      : %d\n", vecs);
fprintf (pfile, "No. of input units   : %d\n", patsize);
fprintf (pfile, "No. of output units  : %d\n", cat);
fprintf (pfile, "\n");

//print output vector format to pattern file comments
fprintf(pfile, "\n\n# Output vector format for this pattern file.... ");
for(n = 0; n < cat; n++) fprintf(pfile, "\n# %3d   %s", (int)n, categories[n]);

//open the ****TRAINING**** pattern file - will contain the training set
sprintf( dfilename, "%s%3s%03dtr.pat", direc, ext, exp );
if( (trfile = fopen( dfilename, "w+t" )) == NULL ) {
    printf("\n could not open %s", dfilename );
    exit(1);
}

```



```

time(&t);
fprintf (trfile, "SNNS pattern definition file V1.4\n");
fprintf (trfile, "generated at %s\n\n", ctime( &t));
fprintf (trfile, "No. of patterns      : %d\n", cat * trpats);
fprintf (trfile, "No. of input units   : %d\n", patsize);
fprintf (trfile, "No. of output units  : %d\n", cat);
fprintf (trfile, "\n");

//print output vector format to pattern file comments
fprintf(trfile, "\n\n# Output vector format for this pattern file.... ");
for(n = 0; n < cat; n++) fprintf(trfile, "\n# %3d   %s", (int)n, categories[n]);

//open the ****VALIDATION**** pattern file - will contain the validation set
sprintf( dfilename, "%s%3s%03dva.pat", direc, ext, exp );
if( (vafile = fopen( dfilename, "w+t" )) == NULL ) {
    printf("\n could not open %s", dfilename );
    exit(1);
}

time(&t);
fprintf (vafile, "SNNS pattern definition file V1.4\n");
fprintf (vafile, "generated at %s\n\n", ctime( &t));
fprintf (vafile, "No. of patterns      : %d\n", cat * vapats);
fprintf (vafile, "No. of input units   : %d\n", patsize);
fprintf (vafile, "No. of output units  : %d\n", cat);
fprintf (vafile, "\n");

//print output vector format to pattern file comments
fprintf(vafile, "\n\n# Output vector format for this pattern file.... ");
for(n = 0; n < cat; n++) fprintf(vafile, "\n# %3d   %s", (int)n, categories[n]);

//open the ****TEST**** pattern file - will contain the test set
sprintf( dfilename, "%s%3s%03dts.pat", direc, ext, exp );
if( (tsfile = fopen( dfilename, "w+t" )) == NULL ) {
    printf("\n could not open %s", dfilename );
    exit(1);
}

time(&t);
fprintf (tsfile, "SNNS pattern definition file V1.4\n");
fprintf (tsfile, "generated at %s\n\n", ctime( &t));
fprintf (tsfile, "No. of patterns      : %d\n", cat * tspats);
fprintf (tsfile, "No. of input units   : %d\n", patsize);
fprintf (tsfile, "No. of output units  : %d\n", cat);
fprintf (tsfile, "\n");

//print output vector format to pattern file comments
fprintf(tsfile, "\n\n# Output vector format for this pattern file.... ");
for(n = 0; n < cat; n++) fprintf(tsfile, "\n# %3d   %s", (int)n, categories[n]);

//all output files are open and headers written - now go through the list again and get the
input data
vecs = 0;
for(i = 0; i < cat; i++) catcount[i] = 0; //reset category counters
while( !feof(lst)){
    //open a rawfile companion for the listed data file - raw file contains header for
the data file
    sprintf(line, "\0\0\0");
    fgets(line, 100, lst);
    if (strlen(line) <= 4) continue; //check for blank line
    vecs++;
    sscanf(line, "%3d_%4d.%s", &exp, &incr, &ext);
    sprintf(dfilename, "%s%03d_%04d.raw", direc, exp, incr);
    if( (dfile = fopen( dfilename, "rt" )) == NULL ) {
        printf("\n could not open %s", dfilename );
        exit(1);
    }
    //scan header info for category name
    while(!feof(dfile)){
        fgets(data, 100, dfile);
        if(sscanf(data, "Sample: %s\n", &samplecat) != 1) continue;
        if( strcmp(samplecat, "Air") == 0 || strcmp(samplecat, "blank") == 0 ||
strcmp(samplecat, "air") == 0 ) continue;

```

```

        break;
    }
    fclose(dfile);
    if (strcmp(samplecat, "\0")== 0) {
        printf("\n Could not find sample category in file %s", dfilename);
        exit(1);
    }

    //see if category is new
    for( datacat = 0; datacat < cat; datacat++ ) {
        tmp = strlen(samplecat);
        if( strncmp(categories[datacat], samplecat, tmp ) == 0 ) break;
    }
    //if new then error and exit.
    if ( datacat == cat + 1 ){
        printf("\nError new category encountered!!!!");
        exit(1);
    }
    //if not new, print the data followed by the output vector.
    //open the data file to read the input vector...
    sprintf( dfilename, "%s%03d_%04d.%s", direc, exp, incr, ext);

    printf("\n %s ---> %d", dfilename, distrib[datacat][catcount[datacat]]);
    putpattern( pfile, dfilename, samplecat, cat, datacat, vecs, ++pats[0]);
    switch (distrib[datacat][catcount[datacat]++){
        case 1:  putpattern( trfile, dfilename, samplecat, cat, datacat, vecs,
++pats[1]); break;
        case 2:  putpattern( vafile, dfilename, samplecat, cat, datacat, vecs,
++pats[2]); break;
        case 3:  putpattern( tsfile, dfilename, samplecat, cat, datacat, vecs,
++pats[3]); break;
    }

}

fclose(lst);
fclose(pfile);
fclose(trfile);
fclose(tsfile);
fclose(vafile);
printf("\ndone\n");

return 0;
}

```


A.5 Simulator utility source modification

The bignet utility that is provided with SNNS needed to be modified to accept command line parameters. This made it possible for training to be fully automated via Linux bash shell scripts. A code fragment is provided below indicating the modification. Full details and a precompiled modified utility for an Intel x86 based Linux platform may be found on the RESULT_CD:\Configuration\Software\Simulator.

bignet.c

```
/*      $State: Exp $ $Locker: $ */
/*****Modified by B.N. - 17/01/2000 *****/
/*****Added features - command line interface*****/
/*****Enables bignet to be called from a shell script*****/

/*****      NEW COMMAND LINE ARGS - B.N.      *****/
-i      number of input units
-h      number of hidden units
-o      number of output units
-n      network name
-f      network filename (to be saved)
*****/

/*****
FILE      : $Source: /usr/local/bv/SNNS/SNNSv4.1/tools/sources/RCS/bignet.c,v $
SHORTNAME : bignet.c
SNNS VERSION : 4.1

PURPOSE   : SNNS-Network Generator for special 3 Layer Feedforward Networks
NOTES     :

AUTHOR    : Niels Mache
DATE      : 01.10.90

CHANGED BY : Sven Doering
IDENTIFICATION : $State: Exp $ $Locker: $
RCS VERSION : $Revision: 2.4 $
LAST CHANGE : $Date: 1995/11/16 07:20:06 $

      Copyright (c) 1990-1995 SNNS Group, IPVR, Univ. Stuttgart, FRG
*****/
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

/* SNNS-Kernel constants and data type definitions */
#include "glob_typ.h"
/* SNNS-Kernel User-Interface Function Prototypes */
#include "kr_ui.h"

static void errChk( err_code )
int err_code;
{
    if (err_code != KRERR_NO_ERROR) {
        printf( "%s\n", krui_error( err_code ));
        exit( 1 );
    }
}
```

```

int main(int argc, char *argv[])
{
    int  ret_code, i, a, j, unit_no;
    int  IUnits, OUnits, HUnits;
    char netname[80], file_name[80];
    struct PosType  unit_pos;
    float initialize_params[5];

    printf( "\n%s\n", krui_getVersion() );
    printf( "---- Network Generator for 3 Layer Feedforward Networks ----\n\n" );
    if( argc < 11) {
        printf( "No. of input units: " );
        scanf( "%i", &IUnits );
        printf( "No. of output units: " );
        scanf( "%i", &OUnits );
        printf( "No. of hidden units: " );
        scanf ( "%i", &HUnits );
    } else
    {
        for( a = 1; a < argc; a++){
            if(strcmp(argv[a],"-i")==0) sscanf(argv[i+1], "%i", &IUnits);
            if(strcmp(argv[a],"-h")==0) sscanf(argv[i+1], "%i", &HUnits);
            if(strcmp(argv[a],"-o")==0) sscanf(argv[i+1], "%i", &OUnits);
            if(strcmp(argv[a],"-n")==0) sscanf(argv[i+1], "%s", netname);
            if(strcmp(argv[a],"-f")==0) sscanf(argv[i+1], "%s", file_name);
        }
    }
}

```

Appendix B

SELECTED RESULTS

The extensive nature of the result set can easily cause one to get lost in the details. For this reason Chapter 5 presents a focussed set of results, this appendix expands that result set and the RESULT_CD contains a full result set which includes a large amount of contextually less-significant detail.

B.1 Organisation of Appendix B

This appendix groups results according to dataset. Dataset correlation tables and dataset-classifier training history graphs are presented for each considered dataset. The following datasets are considered:

- PUR & DCT – representing raw data in the source and transform domains
- D1C & D2C – representing DC and HF filtered data in the transform domain
- D3CA & D3CB – representing coefficient selected data in the transform domain

B.2 Input dimensionality comparison across selected datasets

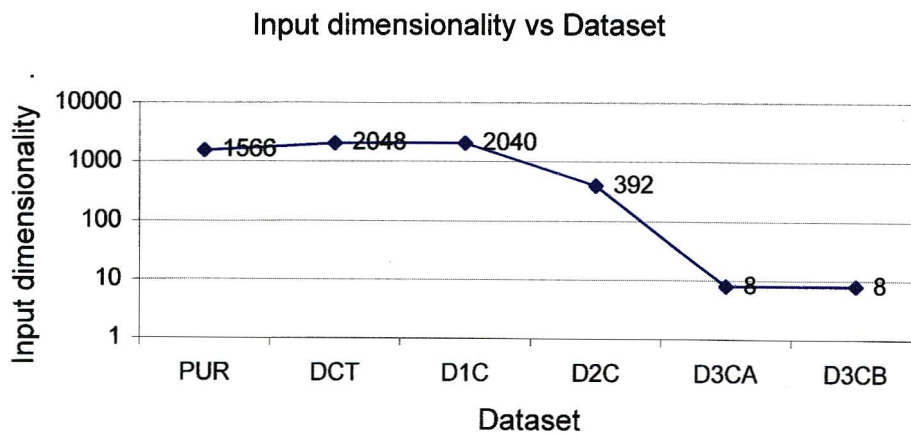


Figure B.1: Dataset input dimensionality for selected datasets.

Raw data is represented by the PUR dataset. The remaining datasets all represent transform domain data. Each transform domain dataset is made up of a unique selection of transform coefficients. In the DCT dataset, all 2048 transform coefficients are selected.

B.3 PUR dataset results

Table B.1: Category correlation coefficient matrix in upper triangular form (PUR dataset)

	<i>Camembert</i>	<i>Brie</i>	<i>BrieTB</i>	<i>Blue</i>	<i>Mozzarella</i>	<i>Chevin</i>	<i>Laberyl</i>
<i>Camembert</i>	0.94	0.91	0.89	0.89	0.88	0.84	0.93
<i>Brie</i>	0.00	0.94	0.92	0.92	0.80	0.88	0.92
<i>BrieTB</i>	0.00	0.00	0.94	0.92	0.79	0.88	0.92
<i>Blue</i>	0.00	0.00	0.00	0.95	0.81	0.89	0.94
<i>Mozzarella</i>	0.00	0.00	0.00	0.00	0.99	0.75	0.89
<i>Chevin</i>	0.00	0.00	0.00	0.00	0.00	0.92	0.85
<i>Laberyl</i>	0.00	0.00	0.00	0.00	0.00	0.00	0.97

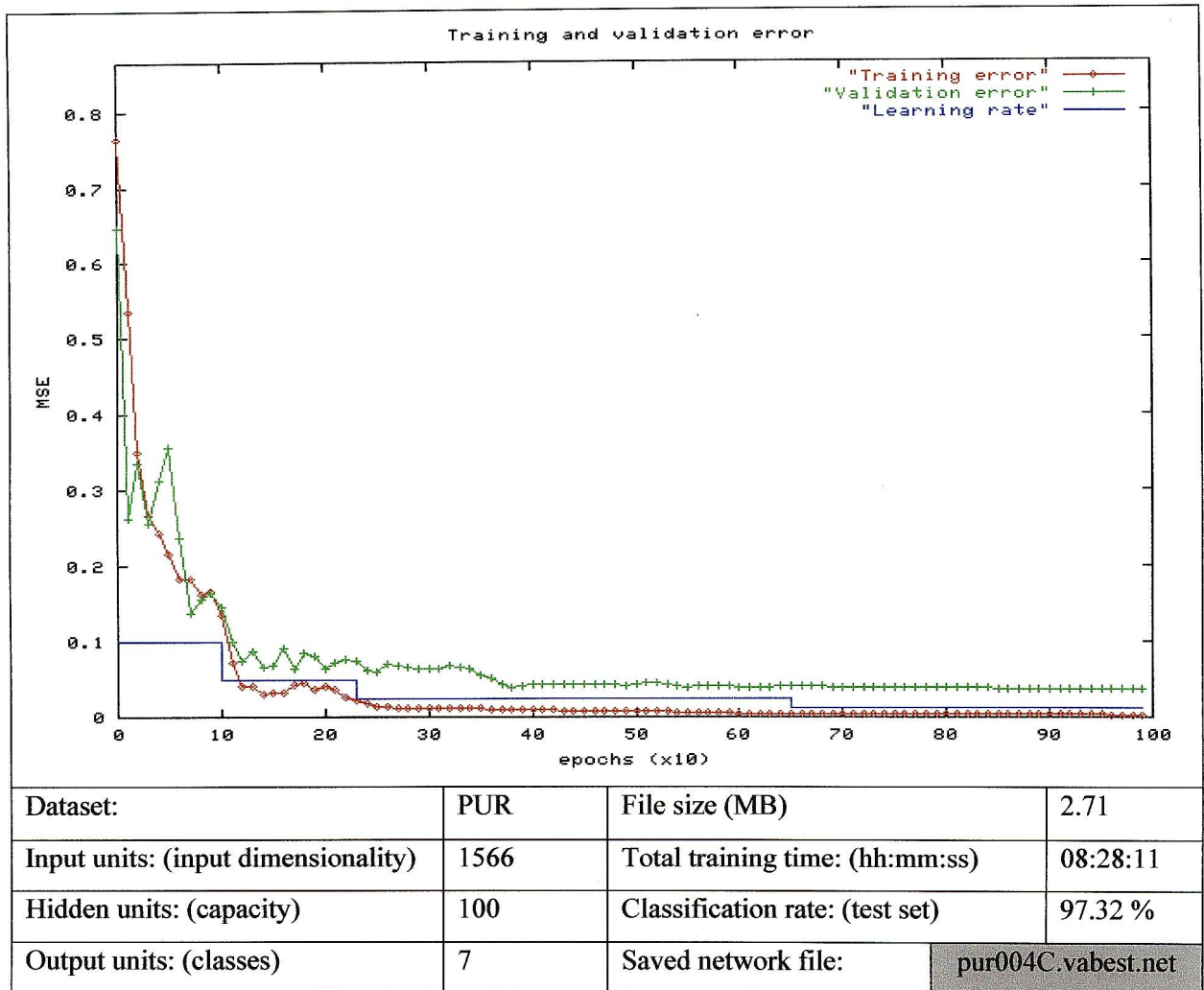


Figure B.2: Training result - 100 hidden unit neural network with the PUR dataset

Brief comment: A stable trajectory is achieved, thus indicating convergence of the learning algorithm. An acceptably high classification rate is achieved, but training time and input dimensionality (file size) are undesirably high for any embedded application.

B.4 DCT dataset results

Table B.2: Category correlation coefficient matrix in upper triangular form (DCT dataset)

	Camembert	Brie	BrieTB	Blue	Mozzarella	Chevin	Laberyl
Camembert	0.99	0.98	0.98	0.97	0.97	0.95	0.99
Brie	0.00	0.98	0.98	0.98	0.95	0.96	0.98
BrieTB	0.00	0.00	0.99	0.98	0.96	0.96	0.98
Blue	0.00	0.00	0.00	0.98	0.95	0.97	0.98
Mozzarella	0.00	0.00	0.00	0.00	1.00	0.93	0.98
Chevin	0.00	0.00	0.00	0.00	0.00	0.97	0.95
Laberyl	0.00	0.00	0.00	0.00	0.00	0.00	0.99

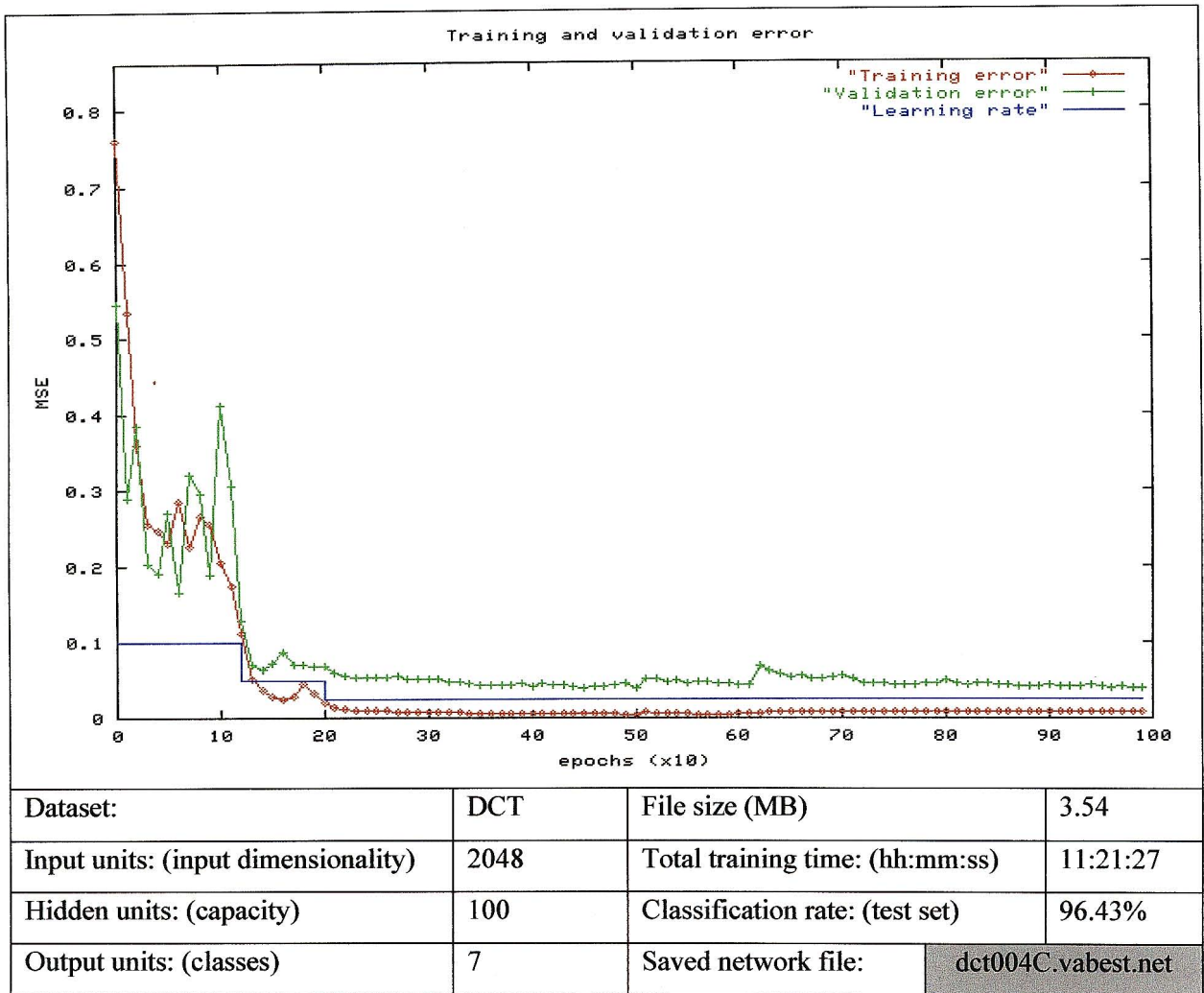


Figure B.3: Training result - 100 hidden unit neural network with the PUR dataset

Brief comment: Similar to the PUR result. A stable trajectory is achieved, thus indicating convergence of the learning algorithm. Classification rate is good, but training time, input dimensionality and file size are undesirably high. This result is worse than the PUR result.

B.5 D1C dataset results

Table B.3: Category correlation coefficient matrix in upper triangular form (D1C dataset)

	<i>Camembert</i>	<i>Brie</i>	<i>BrieTB</i>	<i>Blue</i>	<i>Mozzarella</i>	<i>Chevin</i>	<i>Laberyl</i>
<i>Camembert</i>	0.97	0.96	0.94	0.89	0.89	0.88	0.94
<i>Brie</i>	0.00	0.98	0.96	0.93	0.87	0.92	0.95
<i>BrieTB</i>	0.00	0.00	0.98	0.93	0.90	0.94	0.96
<i>Blue</i>	0.00	0.00	0.00	0.98	0.83	0.94	0.95
<i>Mozzarella</i>	0.00	0.00	0.00	0.00	0.99	0.85	0.91
<i>Chevin</i>	0.00	0.00	0.00	0.00	0.00	0.97	0.93
<i>Laberyl</i>	0.00	0.00	0.00	0.00	0.00	0.00	0.98

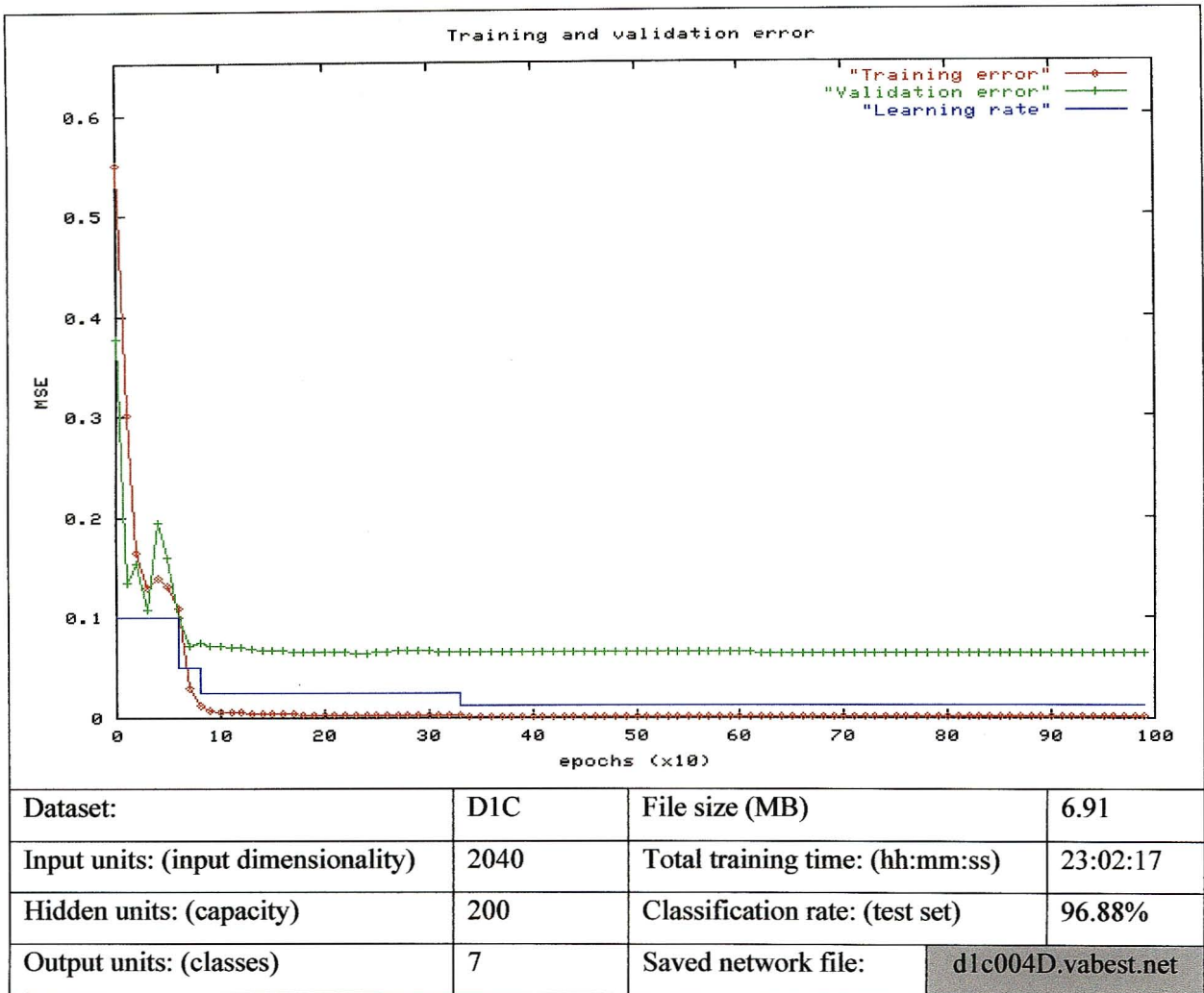


Figure B.4: Training result - 100 hidden unit neural network with the D1C dataset

Brief comment: Learning is more rapid than in DCT dataset. A stable trajectory is achieved, thus indicating convergence of the learning algorithm. Classification rate is good, but training time, network capacity and file size are excessively high. This is an undesirable result.

B.6 D2C dataset results

Table B.4: Category correlation coefficient matrix in upper triangular form (D2C dataset)

	<i>Camembert</i>	<i>Brie</i>	<i>BrieTB</i>	<i>Blue</i>	<i>Mozzarella</i>	<i>Chevin</i>	<i>Laberyl</i>
<i>Camembert</i>	0.97	0.96	0.95	0.89	0.89	0.88	0.94
<i>Brie</i>	0.00	0.98	0.96	0.93	0.87	0.92	0.95
<i>BrieTB</i>	0.00	0.00	0.98	0.93	0.90	0.94	0.96
<i>Blue</i>	0.00	0.00	0.00	0.98	0.83	0.94	0.95
<i>Mozzarella</i>	0.00	0.00	0.00	0.00	0.99	0.86	0.91
<i>Chevin</i>	0.00	0.00	0.00	0.00	0.00	0.97	0.93
<i>Laberyl</i>	0.00	0.00	0.00	0.00	0.00	0.00	0.98

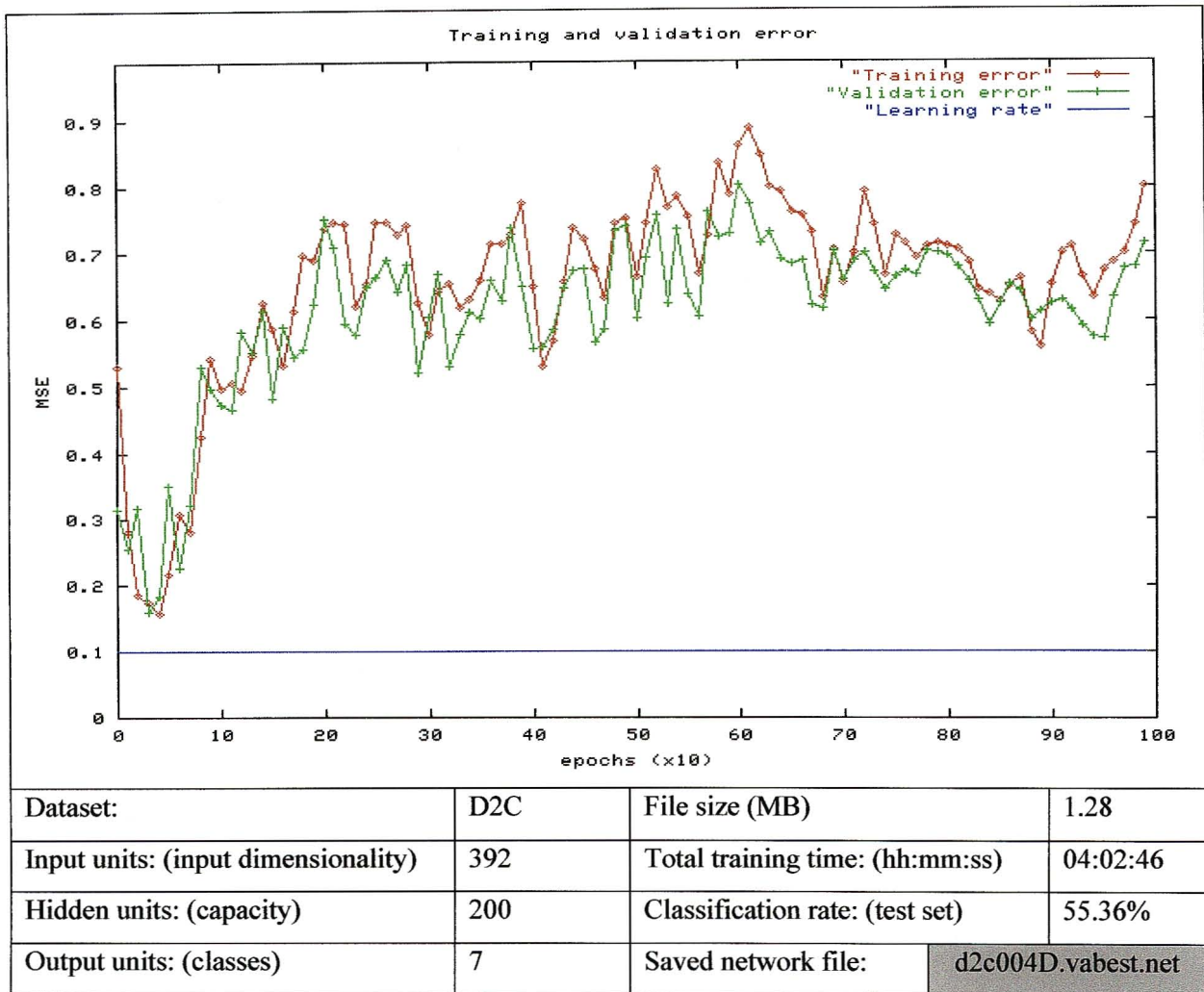


Figure B.5: Training result - 100 hidden unit neural network with the D2C dataset

Brief comment: An unstable trajectory is evident, thus indicating the inability of the learning algorithm to converge. Classification rate is unacceptably low, however, training time and input dimensionality are desirably reduced. This result is undesirable.

B.7 D3CA dataset results

Table B.5: Category correlation coefficient matrix in upper triangular form (D3CA dataset)

	Camembert	Brie	BrieTB	Blue	Mozzarella	Chevin	Laberyl
Camembert	0.98	0.97	0.97	0.96	0.91	0.95	0.98
Brie	0.00	0.98	0.98	0.97	0.87	0.97	0.98
BrieTB	0.00	0.00	0.99	0.97	0.89	0.97	0.98
Blue	0.00	0.00	0.00	0.98	0.82	0.95	0.97
Mozzarella	0.00	0.00	0.00	0.00	0.99	0.85	0.90
Chevin	0.00	0.00	0.00	0.00	0.00	0.97	0.96
Laberyl	0.00	0.00	0.00	0.00	0.00	0.00	0.99

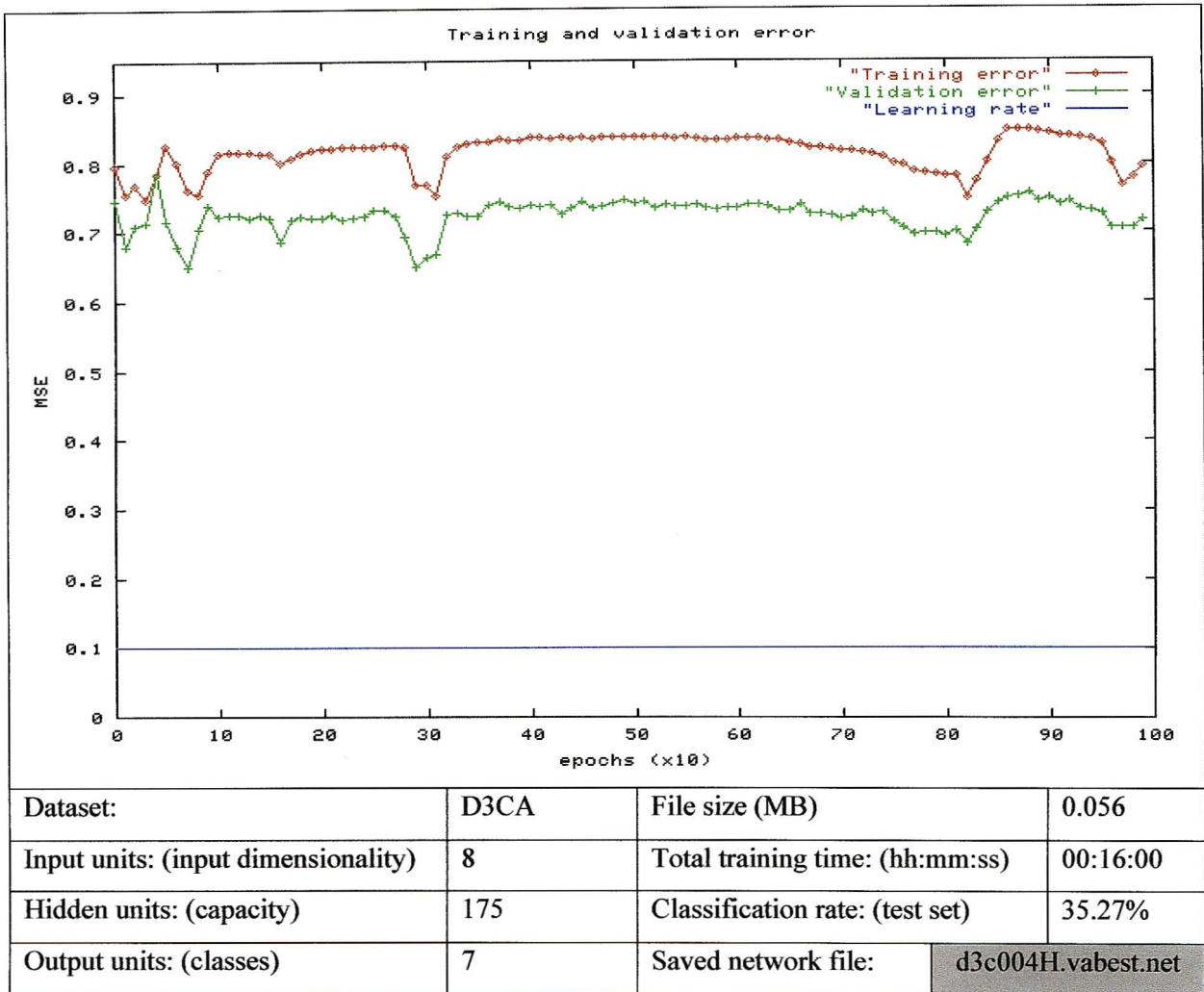


Figure B.6: Training result - 100 hidden unit neural network with the D3CA dataset

Brief comment: The trajectory indicates a consistently high error, thus indicating the inability to learn. The D3CA coefficient selection has produced a poor dataset. Classification rate is poor, but training time and input dimensionality are desirably low. This is an undesirable result.

B.8 D3CB dataset results

Table B.6: Category correlation coefficient matrix in upper triangular form (D3CB dataset)

	Camembert	Brie	BrieTB	Blue	Mozzarella	Chevin	Laberyl
Camembert	0.92	0.83	0.63	0.41	0.52	0.06	0.76
Brie	0.00	0.97	0.78	0.72	0.40	0.46	0.84
BrieTB	0.00	0.00	0.87	0.79	0.55	0.60	0.84
Blue	0.00	0.00	0.00	0.97	0.17	0.77	0.81
Mozzarella	0.00	0.00	0.00	0.00	0.96	0.24	0.39
Chevin	0.00	0.00	0.00	0.00	0.00	0.93	0.42
Laberyl	0.00	0.00	0.00	0.00	0.00	0.00	0.95

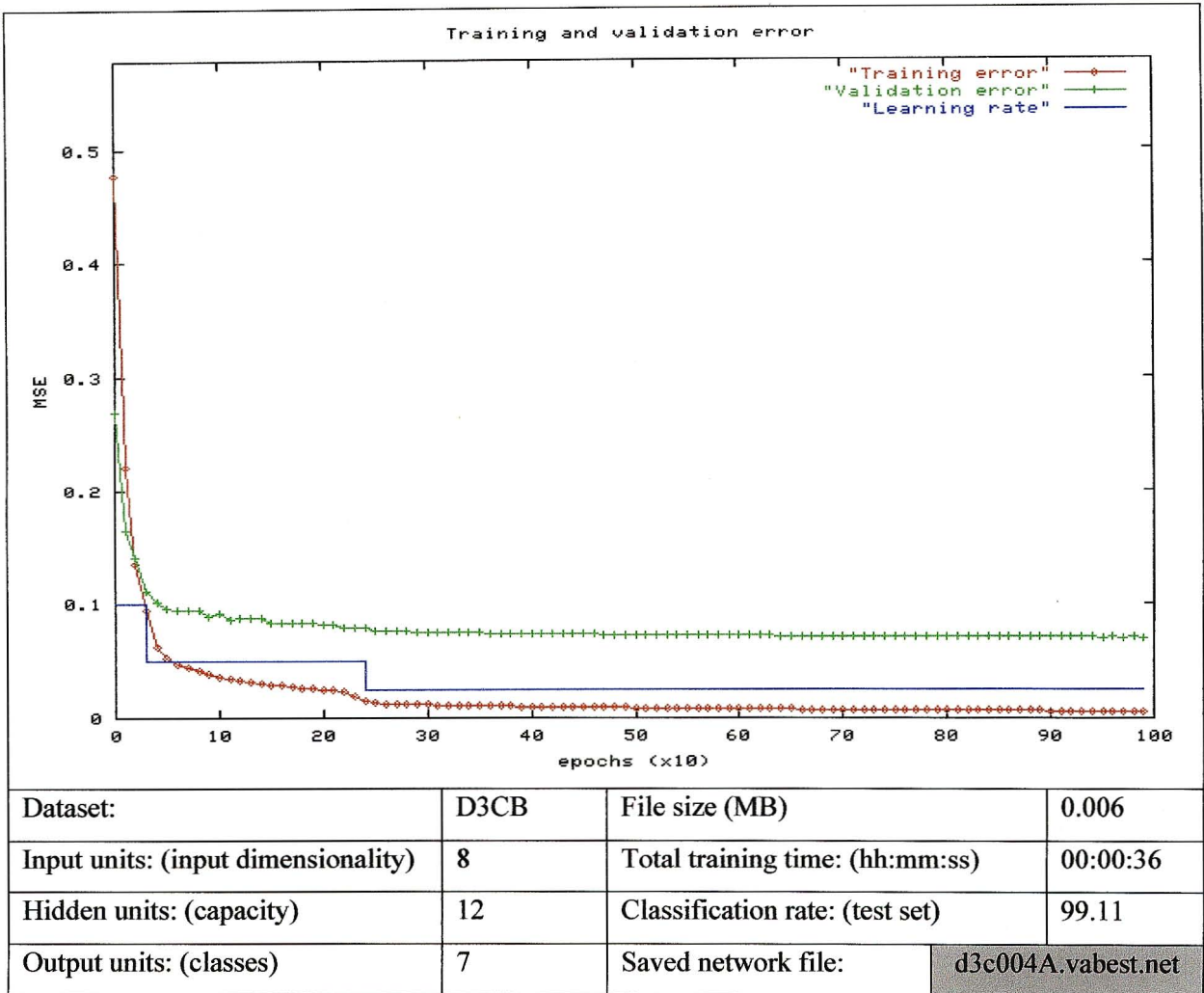


Figure B.7: Training result - 12 hidden unit neural network with the D3CB dataset

Brief comment: A stable trajectory is achieved, thus indicating convergence of the learning algorithm. Classification rate, training time and input dimensionality have all attained desirable levels. This is a desirable result. A detailed discussion of D3CB results is offered in Chapter 5.

CHAPTER 1

INTRODUCTION

1.1 A brief perspective

Olfaction is a chemical sense that is specifically adapted for the perception of gaseous or vaporous substances. Other chemical senses exist, for example the sense of taste and the vomeronasal sense that is involved in the specific detection of pheromones. These chemical senses are fundamentally similar but differ significantly in their intended purpose. The chemical senses are involved in a broad spectrum of application ranging from basic survival to abstract social and cultural functionality.

It is of utmost importance that a common misconception about olfaction is dispelled from the outset. The olfactory sense is not intended to identify chemical composition even though this may be possible in some instances. Instead it relates chemical odours to abstract objects through association. For example, the aroma emanating from a mug of fresh coffee ultimately results in an emotional response such as relaxation, and a cerebral classification such as “Mocha Java!”. This is similar to other senses such as hearing where a sound is classified as an abstract object such as the word “Hello” and no consideration is given to the elemental frequencies that constitute the sound. While the output of a sensor (cochlea for example) may be a fundamental measurement, the output of the entire sensory system is most often abstract.

When building an artificial olfactory system, it is the abstract associative output that must be emulated. Therefore, artificial olfactory systems or “electronic noses” as they are commonly known are not intended to perform fundamental gas analysis. Specialised analytical instruments such as Gas Chromatographs and Mass Spectrometers (GC/MS) exist for that purpose.

In their book “Electronic noses: Principles and applications” [1], Dr Julian Gardner and Dr Philip Bartlett traced the notion of artificial odour measurement back to 1920 when the concept was first proposed by Zwaardemaker and Hogewind [2]. In 1954 Hartman [3] published the first discovery of an electrochemical sensor that responded to gaseous odorants, but it was only in 1982 that Persaud and Dodd of Warwick University arguably produced the first successful artificial olfactory system [4]. In the two decades since then the field of artificial olfaction experienced rapid growth and was driven largely by the continued efforts of the of the Warwick University team under the principal leadership of Dr Gardner.

Although it is a young discipline, the field of artificial olfaction has spun off a variety of commercial ventures. In their article “The how and why of electronic noses” [5] Professors Troy Nagel and Susan Schiffman provide a useful review of the state of the art and industry as it stood in 1998 at the commencement of this study.

1.2 The olfactory sense

Perhaps the least understood of the biological senses, it is ironic that the chemical sense is also the oldest and most prolific of all biological senses. Found in plants and animals, from unicellular organisms to mammals the chemical senses are confounding by virtue of their variability in design and purpose. In lower life forms chemical sensitivity exists even in the absence of a nervous system, while in humans, although highly developed, the chemical senses are largely limited to emotional and subconscious responses and struggle to produce value in the domain of the conscious mind. Hold your breath and try to imagine the scent of strawberries. Most find it impossible, but how simple it is to recall an image of a strawberry with your eyes closed.

Furthermore, the value of the olfactory sense varies with age. Although children possess the same olfactory sensitivity as adults, they exhibit poor odour identification and memory abilities due to their level of mental development. The elderly also exhibit poor olfactory performance. Given that conscious olfactory ability is limited in early and advanced age [6], one may question the value and purpose of the olfactory sense in humans. Some believe that our olfactory sense is perhaps an evolutionary remnant that was previously more highly developed and critical for survival and is now fading away.

Survival aside, the olfactory sense has certainly found increased significance in modern culture [7]. It is therefore no surprise that the recent social relevance has brought the olfactory sense into the commercial limelight. Virtually every chemical consumer product now takes olfactory appreciation into account. The fragrance and high-end food and wine industry are premised largely on the importance of olfactory appreciation. The flavour impact that food producers seek is primarily an olfactory phenomenon. It is well known that expensive food, wine and scents are “feel good” products. Such food is consumed to create a feeling of well-being. The pivotal role that olfaction plays in the regulation of emotion is seldom realised.

“The persuasive power of an odour cannot be fended off, it enters into us like breadth into our lungs, it fills us up, imbues us totally. There is no remedy.” Patrick Suskind [8]

1.3 The electronic nose

Poorly defined and apparently mysterious, the olfactory sense seems to defy complete emulation. At first glance it may be argued that the purposes served by the biological sense are so different from those that are desired of an electronic nose that there is no need for close emulation. The question must be asked, what is expected from an electronic nose and how does that differ from its biological counterpart?

The electronic nose is involved in the detection of predefined odour conditions, which incorporates odour classification. That is, the electronic nose is primarily used as an odour discriminator, and this is remarkably similar to the human olfactory system, the main difference being that the discrimination is most often subconscious in humans where the discrimination result is propagated largely to the centres of emotion. For practical purposes, the addition of emotional value may be regarded as an extra-olfactory process and need not feature in our electronic nose model. The biological analogy may be regarded as applicable to the extent that it discriminates odours, consciously or otherwise.

Odour and chemical discrimination are not necessarily the same. Several chemically pure substances are known to have distinctly different odours at different concentrations. The non-linearity gets even more complicated when natural odours, which are usually complex mixtures of gases, some of which suppress the detection of the others, are considered. Chapter 2 offers a more detailed discussion on this issue.

The high-level organisational framework for biological olfaction, electronic odour discrimination and traditional gas analysis is the same, and may be described as a three step process namely, gas handling, sensing and signal processing. The uniqueness of the electronic nose is particularly apparent at the sensing and signal processing stages. It is at the sensory and signal processing levels that most of the current research effort is concentrated [9]. This study is focused on signal processing and uses commercially available sensing technology.

Given the obvious novelty and allure of this study it would be negligent to pass up its obvious recreational potential, so it was decided that the experimental nose should be used to distinguish fine wines. Unfortunately, the sensors' receptive fields did not permit effective wine discrimination. This

was discovered by trial and error. So the next best thing was done, the system was used to discriminate “fine cheeses”. From a technical perspective the choice of cheese as the subject of discrimination has many advantages. Cheeses produce complex and variable gas mixtures with subtle variations from one cheese to the next. Continuous bacterial action and the effects of oxidation also make for an interesting and challenging classification task. Cheese discrimination is certainly a demanding test of olfactory performance.

1.4 Applications of the electronic nose

There are several factors that influence the application of electronic noses. Foremost among these are:

- Low cost relative to traditional gas analysis systems,
- The possibility of continuous online measurement,
- Reduced size, and
- Discrimination of abstract user-definable odour conditions.

These factors contribute to the remarkable versatility of the electronic nose. Applications typically include online chemical and food process control, alarm condition detection and diagnosis of various medical conditions

In their article [5] “The how and why of electronic noses” Troy Nagle and Susan Schiffman provide a detailed table of commercially available electronic nose products, indicating their cost, size, and technology among other things. The smallest and simplest devices, not mentioned in the table, are pocket sized breath analysers and gas leak detectors. Intelligent gas leak detectors are able to determine when a gas detection event is due to a leak or normal usage in a kitchen environment.

Medical diagnosis includes the non-invasive detection of various conditions that give rise to oral malodour, for example liver cancer. Electronic nose technology is also being rolled out in the dairy industry where the monitoring of product quality and animal health are performed online while a cow is being milked. Conditions such as ketosis in cows produce detectable odour signatures in their milk [10].

It is difficult to say what the future holds for electronic nose technology. Perhaps the technology will be generalised and used to detect liquid phase odours as well. Research on electronic tongues has already begun [9]. Electronic noses have also been incorporated into miniature robots that are able to locate odour sources. Perhaps future research will produce chemosensory micro-machines that can be

injected into the human blood stream to seek out various pathologies from their chemical by-products. These devices may then administer localised treatment.

1.5 Specific contributions

This thesis builds a plausible biologically inspired basis for the design of an electronic nose. Biological system organisation is the result of evolutionary adaptation. These adaptations provide valuable indicators of olfactory functional requirements. Selected biological subsystems are studied, and analogous engineering solutions are proposed. The following specific functional adaptations of the biological organism were investigated: gas handling and headspace regulation, the detection mechanism, establishment of an elementary orthogonal feature map, and signal classification.

The following seminars, conference papers and publications were produced in the course of this study.

Naidoo, B.: 'Biologically inspired signal transformations and neural classification of odours in an electronic nose', pp. 69-74, Proceedings of the 14th annual symposium of the pattern recognition society of South Africa, Langebaan, ISBN 0-7992-2218-6, November 2003.

Naidoo, B.: 'Electronic noses turn up in Brighton: Overview of the 7th international symposium on olfaction & electronic nose (ISOEN 2000: 20-24 July 2000)', ChemoSense, ISSN: 1442-9098, Vol. 2, No. 4, September 2000.

Naidoo, B., and Broadhurst, A.D.: 'Sensor array data processing using a 2-d discrete cosine transform', pp. 153-158 in, Gardner, J.W., and Persaud, K.C., (eds.): 'Electronic Noses and Olfaction 2000', Series in sensors, Institute of Physics, Bristol, 2000.

Levy, D.C., and Naidoo, B.: 'How machines can understand smells and tastes: Controlling your product quality with neural networks', Ch 22, pp. 199-207 in, Bell, G.A., and Watson, A.J., (eds.): 'Tastes & Aromas: The chemical senses in science and industry', UNSW Press, 1999, Sydney, Australia.

Naidoo, B., Levy, D.C., Bell, G.A., and Barnett, D.: 'Food odour classification in an artificial olfactory system', International conference on engineering applications of neural networks (EANN - 1995), Finland.

Naidoo, B., Levy, D.C., Bell, G.A., and Barnett, D.: 'Classification of data from non-ideal gas sensors', Proceedings of the Sixth Australian Conference on Neural Networks (ACNN - 1995), p.61-64.

Naidoo, B.: 'Neural Network Based Classification of Sensory Data in an Artificial Olfactory System', Presented at the South African Institute of Electrical Engineers (SAIEE) Annual Post-Graduate Paperette Evening, 1994, Durban.

1.6 Thesis overview

This study investigates the notion that the emulation of biological system organisation will benefit the development of an artificial olfactory discriminator. Chapter 2 presents an analysis of biological olfactory system organisation and proposes a framework for the development of an artificial olfactory discriminator.

An artificial olfactory system is then deployed in the proposed framework. Chapter 3 describes the system front-end that is responsible for capture of raw olfactory measurements. This incorporates odorant handling, sensing and recovery.

Chapter 4 describes the software implementation that emulates low-level olfactory processing in the mammalian brain. This is a pre-classification processing stage and is typically used to extract relevant features in the data and to reduce the complexity of the classification problem. It is modelled on actual neural processing in the mammalian olfactory system.

Final odour classification and system performance is addressed in Chapter 5. Artificial neural networks are used to classify the processed odour measurements. Chapter 6 concludes with a brief discussion on the major outcomes and their significance in the context of this study.

CHAPTER 2

BIOLOGICALLY INSPIRED SYSTEM ORGANISATION

2.1 Introduction

The contemporary description of electronic noses and olfaction offered in Chapter 1 helps to build a general paradigm. Chapter 2 seeks to establish a technical basis on which to build a solution. Selected findings in the field of biological olfaction are used to produce a clear technical basis for the organisation of artificial olfactory systems.

Artificial olfaction is a young field with most of the research and development having taken place in the last decade. As such, there is no globally applicable specification framework for electronic nose development. Each electronic nose is specified in a manner that is technology and application specific. In such circumstances, it is beneficial to consider the organisational principles that lead to successful biological olfaction. Mammalian olfaction will be considered in particular.

Biological theory provides valuable insight into olfactory system architecture and functionality. Extensive investigations into olfactory receptor cell functionality continue to inspire and guide the development of electro-chemical sensors. Neural coding of olfactory stimuli and the associated biological signal processing pathway provide clues regarding necessary signal processing. This chapter presents selected details in the cross-disciplinary theories of gas handling, sensing and discrimination. At each stage inferences are made from the study of biological olfaction, and a specification framework for the electronic nose is evolved.

2.2 Elements of olfactory biology

Although the study of biological olfaction already spans several decades, the research community is still in the process of establishing a definitive and globally acceptable model of olfaction. In the light of recent findings, a consensus of opinion is emerging, and many of the older theories on chemoreception and neural processing have been replaced [11]. This section presents aspects of the generally acceptable model of olfaction. Major topics are touched on with the purpose of establishing a simple theoretical model.

The biological odour-processing pathway is decomposed into a complex sequence of processes. It is useful to arrange the various processes in four categories:

- Pre-receptor events: processes that influence the availability of stimulus molecules at the detection surface.
- Chemoreception: the process of stimulus detection and electrical (neural) representation.
- Low-level processing: transformation of the initial neural code.
- High-level processing: selection, association and other higher-brain processes.

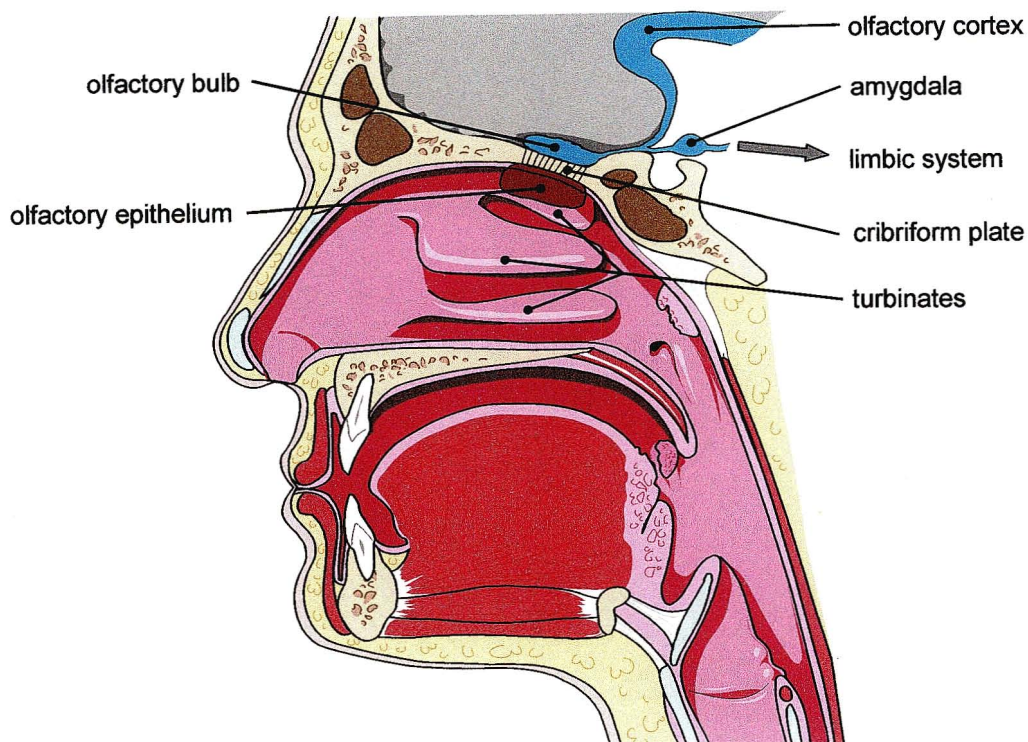


Figure 2.1: Simplified anatomy of the human olfactory system
[Modification of the original image (*uresp.wmf*) from *LifeART Super Anatomy 1*, Copyright © 2002, Lippincott Williams & Wilkins, a Walters Kulwer Company.]

2.2.1 Pre-receptor events

The process of conscious odour discrimination starts with the deceptively simple action of sniffing. Stimulus bearing air is drawn via the nostrils, past the turbinate bone structures and over the olfactory epithelium. Individual stimulus molecules eventually reach the receptive surfaces thereby triggering a cascade of neural processes. This sampling phase incorporates several invisible processes that influence the spatial and temporal profile of stimulus delivery to the receptive surfaces. Such factors

that influence the availability of odour molecules at the receptor surface are referred to as pre-receptor events [1]. In this section, some of these events and their implications for the electronic nose are considered.

Sniffing and air flow-rate: Contemporary wisdom has seldom acknowledged sniffing for the pivotal role it plays in natural olfactory processes. Recent studies [12, 13] have characterised sniffing as an integral part of a larger feedback loop that regulates the olfactory process. Sniffs are modulated in frequency, duration and depth, indicating a temporal dimension in odour perception [13]. Airflows resulting from ordinary respiration or deliberate sniffing impose a detectable modulation of olfactory nerve signals [12]. These facts imply that odour detection is episodic (i.e. not continuous) with a definite identification period that begins and ends with each sniff. The inhalation and exhalation transients are encoded in each detection cycle. This indicates the possible usefulness of airflow control in the electronic nose, a topic that is described in more detail in Chapter 3.

As a general rule, higher air flow-rates are believed to improve odorant detection [14]. However, actual flow rates through each nostril always differ in humans [14]. When sniffing through the high flow-rate nostril alone, sniffs are short and shallow. When using only the low flow-rate nostril, sniffs are longer [14]. This implies that odorant detection is integrated over time until some detection threshold is reached. Perhaps the electronic nose should therefore have a fixed odorant flow-rate into the detection chamber over a standard period of time. This will ensure that a standard volume of odorant bearing air is sampled in each cycle.

Furthermore, the normal birhinal (dual-nostril) sniff allows only the high flow-rate nostril and its associated epithelium to reach optimum threshold. The other nostril will detect at a sub-threshold level. This conveys two slightly different images of the detection event to the brain [14]. Sub-threshold olfactory images are not necessarily subsets of full-threshold images. Therefore, sub-threshold images are able to enrich the detectable feature set. The implications for the electronic nose are clear. Detection should be dispersed among two or more simultaneously sampled parallel channels, each operating at a different flow-rate, over the same fixed period. The parallel measurement system imposed a significant cost penalty and could not be implemented in this study. Parallel measurements can be serialised and executed sequentially on one set of sensors, provided that the sensors and sample gases are stable. Unfortunately, the SnO₂ sensors under consideration lack stability [15, 16] and the composition of the sampled headspace will be modified by each successive measurement.

Given that each measurement event dilutes the headspace, a sequential measurement system is not representative of the simultaneous sampling that takes place in two nostrils. Sequential measurement

systems would have to wait for very long periods between measurements for headspace equilibration, and there is no guarantee that the same equilibration point will be reached each time. Alternately, a portion of the original headspace can be maintained for a subsequent measurement. This was not done either, due to the cost penalty that it would incur. The system was therefore restricted to a single measurement channel operating at one predefined flow rate.

Temperature, Humidity and Concentration: Apart from air flow-rates, other factors such as temperature, humidity and concentration may also affect the detection process. In mammalian olfactory systems, temperature and humidity are regulated as air enters the warm, moist nasal cavity and passes over the turbinate bone structures and mucus membranes. The electronic nose should operate similarly over a restricted temperature and humidity band. The use of an air-conditioned environment and bottled synthetic air attempts to emulate this characteristic and so provide temperature and humidity stability. Chapter 3 deals with these issues in greater detail.

The vast majority of odours are complex mixtures of pure gases. The relative concentrations of gases in an odour mixture are a key factor in mixture classification. Where significant variations in total mixture concentration take place, it is believed that feedback signals from the brain cause compensatory adaptation in the olfactory sensor neurons. This is believed to extend the dynamic range of the detection mechanism [17]. It would therefore be prudent to take reasonable steps to limit the concentration range over which the experiment is to take place. The sensors selected for this experiment do not have a wide linear dynamic range and compensatory adaptation is an advanced topic left for a future study.

The rate of change of stimulus concentration in the nose (as the stimulus is introduced and removed) has also been shown to affect conscious perception. In fact, the rate of change of concentration may be more important than the absolute concentration itself [18]. This indicates that data capture should commence before stimulus introduction and terminate after stimulus removal. The captured transient could encode features that characterise stimulus identity.

The final concentration related phenomenon to be considered is speed of detection. The concentration of a stimulus in the nose has been shown to influence detection latency [12]. Detection thresholds are reached more rapidly for concentrated stimuli, thus reducing detection latency. Under controlled conditions, detection latency can be used as an indicator of stimulus identity. For example, one can discriminate between freshly sliced blue cheese and mild cheddar simply by the time it takes to obtain an effective sniff. The pungent blue cheese will always reach the detection threshold faster. In this situation, one wishes to identify the solid substance that produces the aroma. Each substance will have its own evaporation rate and vapour pressures. As long as all environmental variables are kept

constant, each substance should produce a characteristic concentration transient and response time. All that remains is to deploy a system with the necessary sensitivity and resolution to capture this information.

One barrier remains before odorant molecules reach the receptor surface, that is the mucus layer. Contrary to popular belief, the nose does not directly detect substances in the gaseous phase. The receptive surface is physically isolated from the nasal airflow by a mucus layer that is typically 35 μ m thick. Odorant molecules must dissolve into and traverse this aqueous layer for detection to take place. A typical odorant takes 300 milliseconds to traverse the mucus layer [19]. The layer is constantly replaced to prevent fouling of the detection mechanism. Mucus flows over the receptive surface at the surprisingly high speed of 10 to 60 millimetres per minute [20]. Experts in electronic olfaction are not convinced that the above-mentioned functionality need be emulated in any form [11].

The most important phenomenon at the mucosal surface (from the electronic nose point of view) must be the differential sorption of odorants [21]. The detectable spatial distribution of odorants at the mucus surface is believed to be a result of turbulent stimulus flows imposed by the turbinate bone structures [12]. There is also clear evidence of temporal separation of odour mixture components. This is imposed by differential diffusion rates of component gases through the mucus layer [12]. The effect is similar to gas separation in the separation columns of gas chromatographs [22, 23]. It is clear that the temporal signature may encode aspects of mixture identity. Some commercial electronic noses now use gas separation columns. This study did not implement gas separation due to cost and complexity considerations.

2.2.2 Chemoreception: The olfactory epithelium

Having traversed the mucus layer, odorants become available for detection at the receptor cells of the olfactory epithelium. Each receptor cell produces an electrical representation (codification) of stimulus detection. Once generated, the codewords are propagated to the olfactory bulb in the brain for further processing. The later subject is discussed in Section 2.2.3. Detection may be summarised as a process of odorant discovery wherein the presence of an odorant is expressed as a neural codeword. Several detection-related issues are considered in this section.

Detection: Theories of olfactory detection date back as far as the first century BC. The Roman philosopher Lucretius speculated that odours are delivered to the nose as a cloud of very small airborne particles. He also suggested that the detected shape of the particle would determine the character of the odour [1]. Two millennia later, in 1952, Amoore presented similar findings [24] based

on scientific investigation. In 1964 Amoore went on to discover highly specific receptor sites with complementary characteristics to those of the odorant molecules [25]. The odorant and its complementary receptor site form the basis of a “lock and key detection mechanism”. This model has since been expanded and refined, but remains the generally accepted foundation of modern theories.

When an odorant molecule binds with a complementary receptor protein, a complex electrochemical process is set in motion. This process, known as the transduction cascade, takes place entirely within the olfactory sensor neuron and is responsible for producing the neural (electrical) codeword. It will not be necessary to consider the transduction cascade any further for the purpose of this study, however, sensor physics is discussed in Chapter 3.

Detection mechanisms: In his study on detection mechanisms across several species, J. Boeckh [26] suggested three categories of detection:

- *Labelled-line detection:* for each pure stimulus there exists one receptor type such that the receptor completely identifies the stimulus.
- *Array detection:* were each pure stimulus produces a unique distributed response (pattern of multi-component intensities) across an array of dissimilar receptors with broad overlapping receptive fields.
- *Temporal signature:* a characteristic temporal response is created across any number of sensors configured as labelled-line or array detectors.

Labelled-line and array detection mechanisms produce very different types of data. The two classes of data necessitate very different modes of information processing. Consequently, there are two classes of olfactory system that cater to the needs of the detection mechanisms, these are:

- *Odour specialists:* where each odorant detection event is mapped onto a unique neural codeword by a highly selective (labelled-line) receptor [26]. The single receptor is able to completely identify the odorant. The detection event is asserted along a single line that is not shared by other odorant detections. Highly selective receptors with orthogonal (non-overlapping) receptive fields are required. The orthogonal codification of stimulus identity takes place at the receptor level. Therefore, there is no need for complex decoding of stimulus identity at post receptor levels. This simple architecture does have a significant drawback. A unique receptor type is required for every stimulus type. Given that an average human being may need to detect in excess of 10 000 stimulus types, over 10 000 distinct receptor proteins would be required at the receptor level. Since a unique gene is required for the production of each protein, the olfactory system would consume a disproportionate amount of the organism’s genetic code.

- *Odour generalists*: where the matrix of detectors is made up of mildly selective receptors that have large overlapping receptive fields [26]. Any single stimulus should produce a distributed response across the detection array such that the total response is unique for that stimulus. In such a system, each detection event is coded across multiple codewords. Since each receptor type is differentially receptive to a large spectrum of stimuli, the array is able to detect more stimuli than there are receptor types. The flexibility of generalised detection comes at a cost. Each detection event produces a complex multi-codeword pattern that has to be further decoded. Distinctions between patterns are usually fine and non-linear. This makes the discrimination mechanism very sensitive to variations in the transfer characteristic of the receptors.

Specificity: Boeckh [26] went on to state that labelled-line mechanisms were rare. Evidence showed that most olfactory systems, including human, were odour generalists with array detectors.

Spatial coding and lifespan: The olfactory epithelium in each nostril is broadly divided into four zones. Any given receptor is expressed only in one of the four zones, however, each receptor is randomly distributed within the zone amongst other receptors of that zone. [27]. Therefore, the olfactory epithelium itself expresses a broad (non-specific) categorisation of odorant identity based on the zone of detection. It is assumed that receptors are randomly distributed in order to prevent extinction of a receptor type in the event of localised epithelial damage. This morphological adaptation and the high receptor cell redundancy imply a harsh operating environment where sensor neurons are often damaged. In fact, olfactory sensor neurons in the rat are replaced every 28 days on average due to rapid degradation. This may be used as an indicator that electronic sensors will also require frequent replacement due to environmentally imposed degradation [11]. It would therefore be wise to deploy electronic sensors in a controlled and safe environment. These issues are addressed further in Chapter 3.

There is no shortage of research articles describing olfactory detection. This is due largely to the fact that a definitive detection model was so elusive. However, recent discoveries by Dr L. Buck and her colleagues at Howard Hughes Medical Institute (HHMI) at Harvard Medical School have precipitated a seemingly robust and complete description of olfactory detection.

Combinatorial coding: There are as many as 1000 distinct odorant receptor proteins in mammals. Each sensor neuron is known to express only one receptor protein [28]. These receptors form the binding sites in a “lock and key” detection mechanism. Here it is believed that stimulus molecules bind with receptor proteins that possess a complimentary molecular structure. However, Ressler, Sullivan and Buck have also shown [34] that each receptor interacts with multiple odorants and that each odorant is detected by multiple receptors. A many-to-many mapping! Taken at face value, this

appears to contradict the unique binding requirements of the lock and key theory. Buck and Malnic et al [29, 30] explain that the receptors do not detect entire molecules as previously assumed. Instead, they detect smaller structural features in odorant molecules, and individual structural features may be common to a diversity of odorants. Therefore, odorants that are described by multiple features are detected across multiple receptor types. Given that a particular feature may be common to a group of distinct odorants, the entire group may be detected by the one receptor that is designed for the feature in question.

Every odorant is identified by a unique combination of receptors. Given that there are approximately 1000 distinct receptor types, the combinatorial coding scheme [30] is able to code far more than 1000 distinct odours. Whether pure substances or complex mixtures, odorants are coded across multiple codewords thus preserving the notion that mammals are odour generalists.

It is also known that some pure odorants (not mixtures) are perceived differently under different concentrations. For example, indole an organic compound smells putrid at high concentrations and flowery at low concentrations. Combinatorial coding provides an explanation for this phenomenon. Larger amounts of a chemical cause it to bind to lower affinity receptors thus increasing variety of bindings and hence changing its coded representation and perceived character [31]. This indicates that the stimulus must be made uniformly available to all sensors in an electronic nose. Otherwise, a partial characterisation and misclassification may result.

Suppression: Since natural odours are usually complex mixtures of gases, odour perception depends on reception and neural processing of multiple components [28]. Given that the low flow-rate nostril and its associated olfactory epithelium produce sub-threshold detection [14], it seems obvious that the character of the odorant at that nostril may be perceived differently.

Individual odour components have been shown to compete for receptor sites at the olfactory epithelium. This effectively allows one component to suppress the detection of another [32]. The phenomenon takes place at the receptor cell level [32] and is sensitive to the concentration of suppressing odorants [33]. Therefore, the reduced concentration of the suppressing odorant at the low flow rate nostril may retard suppression thus exposing constituent odorants that could be suppressed in the high flow-rate nostril. This would effectively expand the detectable feature set for odour mixtures.

It is clear that odour mixtures do not express themselves as linear combinations of individual component detections. The mechanism is non-linear and superposition of single component detections

does not strictly occur when detecting a mixture. Perhaps non-linear discrimination mechanisms would be required in the electronic nose.

2.2.3 Low-level processing: The olfactory bulb

Each sensor neuron projects its single axon through the cribriform plate into the olfactory bulb. On their way to the olfactory bulb, these axons fasciculate or converge to form the olfactory nerve. The sensory image presented in the olfactory nerve is a complex and dispersed depiction of a multitude of sensory events across the four zones of the olfactory epithelium. The information that is broadly organised by zonal affiliation in the olfactory epithelium [27] is refined in the olfactory bulb to produce a clear and unique combinatorial map [34] for that odour stimulus. A brief look at olfactory bulb structure reveals some inner workings of this transformation.

The olfactory bulb is a distinctly layered structure [35] that implements the next processing stage for olfactory stimuli. The first synaptic contacts in the olfactory pathway take place in the glomeruli of the olfactory bulb. Neural interactions in the bulb are complex and are still the subject of investigation. Only the core functionality is presented here. Many finer details that are still the subject of biological research are not considered.

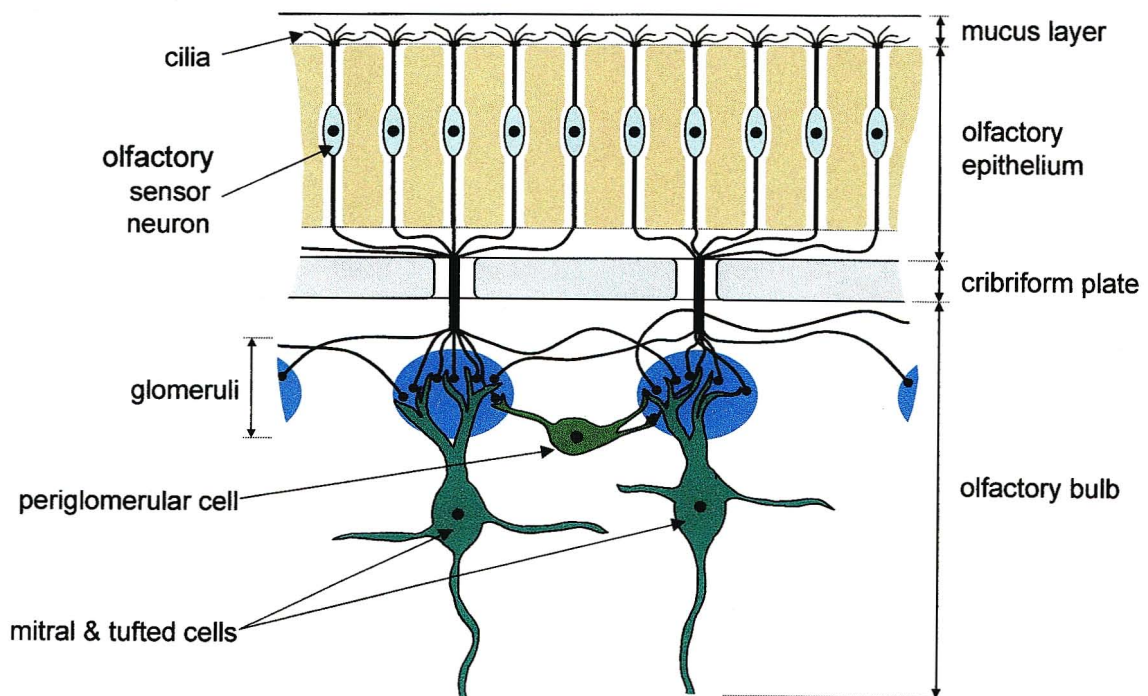


Figure 2.2: Schematic of selected epithelial and bulbar structures

Figure 2.2 presents a partial depiction of olfactory bulb structure. Axons from the olfactory epithelium project into the bulb and terminate in prominent spherical structures called glomeruli. These glomeruli are located in the outer (glomerular) layer of the olfactory bulb. Each glomerulus is mainly a collection of synaptic contacts between a single mitral or tufted cell and a multitude of axons from olfactory sensor neurons. Mitral and tufted cells propagate the sensory signal to higher processing levels in the brain, while periglomerular cells mediate inter-glomerular interactions. These lateral interactions serve to refine the sensory input [36] through processes of inhibition or excitation.

A breakthrough discovery published in 1994 [34] showed that the randomly distributed sensor neurons projected their axons onto small and distinct subsets of olfactory bulb glomeruli, such that each glomerulus expressed axons for one receptor type only. Therefore, receptor genes that are expressed in random and dynamically distributed sensor neurons later converge on specific locations in a discrete and fixed glomerular map. It was also shown that these glomerular maps are consistent across distinct individuals of a species [37].

The significance of the finding may be expressed as follows. In the nose, an odour is coded by a unique combination of randomly distributed active sensor neurons. This distribution varies across individuals of a species and varies with time as the olfactory epithelium constantly regenerates. When a detection event is expressed along an axon, it is clear that detection has taken place, but there is no clear indication of stimulus identity because axons are associated with randomly mixed receptor types. When this signal reaches the olfactory bulb of the brain, the axon will terminate in a glomerulus that is distinctly associated with a single receptor gene. This glomerulus occupies a fixed position in an ordered spatial map of distinct glomeruli. By virtue of its stereotyped spatial location, the active glomerulus indicates a clearly categorised detection event.

Each receptor protein detects a specific structural feature of an odorant molecule. Therefore, different structural features are mapped onto specific stereotyped locations on the olfactory bulb. The olfactory bulb therefore presents a combinatorial feature map called the bulbar map to the higher processing centres in the brain. Furthermore, given that the underlying receptor proteins each detect structurally unique features, the feature map may be regarded as orthogonal. It can be seen that the bulbar map describes each odour as a unique combination of archetypal bases. This is remarkably similar to the orthogonal discrete trigonometric transform (DTT) domain representation of measured signals, where signals are decomposed into linear combinations of unique basis vectors.

2.2.4 High-level processing: Cortical and limbic pathways

The stereotyped bulbar map is projected directly to the higher brain. This region of the brain that receives direct bulbar input is called the olfactory cortex. It is believed that the olfactory cortex is responsible for integrating spatio-temporally coded information from the olfactory bulb [38]. This spatial integration is clearly expressed in the structure of neural connectivity between the bulb and cortex. The cortex possesses an array of localities that are similar to bulbar glomeruli. It is convenient to refer to this collection of cortical loci as a cortical map.

Studies of connectivity between bulbar and cortical maps have shown that:

- Individual bulbar outputs project onto multiple cortical loci [39], and
- Each cortical locus receives multiple bulbar outputs [40].

Based on the above experimental findings, it is assumed that each cortical locus receives a unique selection of bulbar outputs [1]. This means that each cortical locus may be able to encode some rudimentary odour quality. It should be noted that the glomeruli code a detailed and elementary map of structural features in the odour molecule. Individual loci in the bulbar map are far too elementary to code odour quality. These structural features may be regarded as the building blocks of odour quality. Therefore, multiple “building blocks” are integrated at each cortical locus to generate a higher-level representation of odour quality.

The value added cortical outputs then project to several parts of the brain [38]. These regions may be grouped into two major pathways [1, 41]:

- The olfactory association pathway, and
- The limbic pathway.

The olfactory association pathway forms the basis of associative odour memories. This is where conscious discrimination and perception takes place. The limbic pathway is responsible for emotional and hormonal responses to odour stimuli. The neural network classification mechanisms described in Chapter 5 emulate the conscious discrimination functionality of the olfactory association pathway. The limbic system or emotional brain is not considered here due to a present difficulty in obtaining applicable findings. It is known that the processing at this level is sub-conscious and poorly defined. The limbic system should be revisited in a future study to establish its olfactory significance and perhaps to incorporate new findings into this body of work.

2.3 Summary

This review of natural olfaction and mammalian olfaction in particular has provided insights and suggestions for the development of an electronic nose. Figure 2.3 provides a graphical summary of olfactory processes and the implementation mechanism of each process in biological and electronic systems.

A summary of inferences made from the study of biological systems is given in the following three sections. Justifications for the inferences are omitted here as they have already been dealt with in preceding sections. Chapters 3, 4 and 5 describe the implementation details.

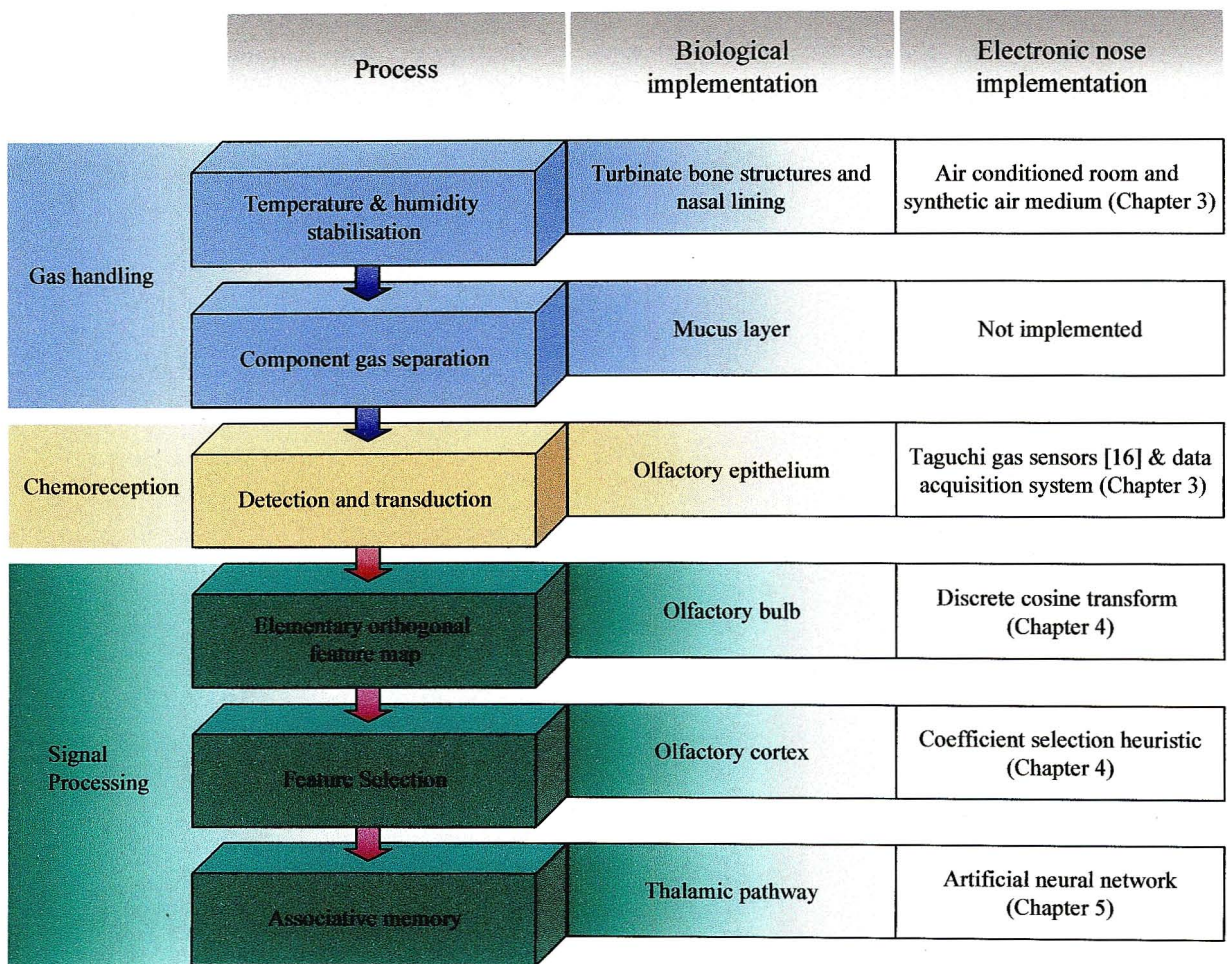


Figure 2.3: Biological and electronic processing analogy

2.3.1 Gas handling

To mimic the biological system, a standardised measurement procedure must be employed. Each measurement should capture the transient response to stimulus introduction and removal while delivering stimuli at fixed flow-rates over standard time intervals.

It was also shown that measurement across several parallel channels operating at different flow-rates could be useful. This was not implemented due to cost and complexity considerations in this study.

Environmental temperature and humidity must be stabilised, and for this purpose an air-conditioned room and a synthetic air medium were used.

The concentration of stimuli must be kept within sensory range to avoid saturation, as electronic sensors do not have the wide dynamic range and adaptation mechanisms that biological systems possess.

It is known that gas separation columns can enhance the temporal signature of odour mixtures thus making them more identifiable. However, separation columns were not implemented in this study due to cost and complexity considerations.

2.3.2 Chemoreception

It is preferable to model the system on biological odour generalists with array detection as used by the mammalian olfactory system. Therefore, sensors should have broad overlapping receptive fields thus ensuring wider sensitivity, and signal processing techniques should be used to extract stimulus identity.

It was shown that a controlled and safe environment would be necessary to protect the sensors from degradation. The use of a synthetic air medium also reduces the possibility of chemical attack on the sensors.

The enclosure for the different sensors must ensure that stimulus gases are made uniformly available to all sensors. This prevents partial characterisation of stimuli.

2.3.3 Signal processing

It has been established that the olfactory bulb expresses a decorrelation of receptor array signals. To mimic this in this study, the measured signals are decomposed to a linear combination of orthogonal bases in order that the limited number of sensors can identify many different stimuli. Various discrete trigonometric transforms can achieve this, such as the discrete cosine transform, which is described in Chapter 4 and which was selected in this study.

The olfactory cortex was shown to select specific subsets of signals from the olfactory bulb for use as inputs to the final classification process. In this study specific transform domain coefficients are selected so that they are able to efficiently code stimulus identity. Chapter 4 describes this selection mechanism.

The discrimination mechanism in biological odour generalist systems is a non-linear neural-network. Artificial neural networks, which are described in Chapter 5, are used to mimic this discriminator. Selected transform coefficients are used as input to this final classification stage.

2.4 Conclusion

This chapter describes the mammalian olfactory system organisation and function. Various principles and processes of olfaction have been identified which are relevant to artificial olfaction. A functional and organisational description is a traditional strength of the biological sciences and the review of biological literature presented here has been most valuable in establishing an organisational framework for the deployment of the electronic nose.

CHAPTER 3

ODOUR RESPONSE MEASUREMENTS

3.1 Introduction

Chapter 3 provides a description of the chosen technology and methodology for odorant sensing. The development of odour measurement hardware and software formed an integral part of this study. The aim was to use existing sensor technology to build a sensory front-end that could detect (measure) odours for classification at a subsequent stage.

Selected aspects of the hardware, software and operating principles are presented here. Figure 3.1 illustrates the general system architecture. This chapter emphasises three technical issues:

- The chosen sensor technology,
- Gas/odorant handling, and
- Measurement configuration.

Post measurement signal processing and pattern recognition are dealt with in Chapters 4 and 5 respectively.

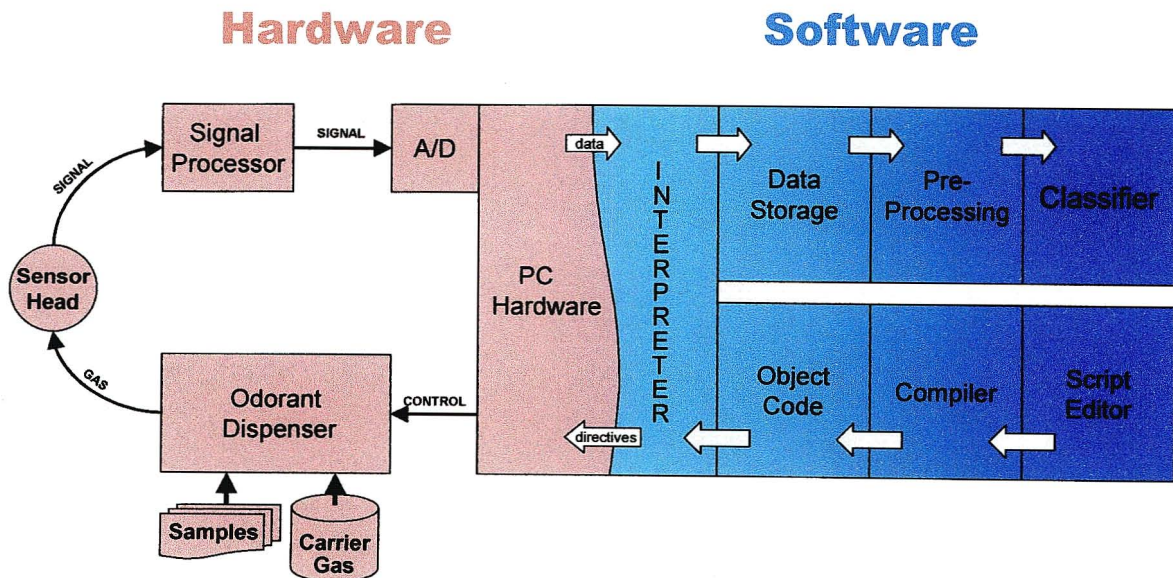


Figure 3.1: System architecture

3.2 Measurement system front-end

An overview of the system hardware is presented in this section. Section 3.2.1 introduces the concept of headspace sampling that guided the hardware development. Section 3.2.2 describes gas reticulation and the associated control strategies. Sensor technology is discussed in Section 3.2.3, and methods of improving post-measurement sensor recovery are discussed in Section 3.2.4.

3.2.1 Headspace sampling

Naturally occurring odours have concentrations that are variable in space and time. Biological systems attempt to compensate for the variation through several mechanisms. At the peripheral level, the processes of controlled sniffing and threshold detection are used to present a partially stabilised olfactory image to the brain. The controlled volume of odorant bearing air that is sampled in the nose is called the odorant headspace. As described in Section 2.2.1, the parallel or simulated parallel sampling system that emulates a birhinal sniff could not be implemented.

Aggregate vs. specific molecular sampling: Electronic noses attempt to sample the character or “flavour impact” [42] of an entire odorant headspace in a fashion that emulates the biological sense. This is different from conventional gas chromatograph or mass-spectrometer analysis of specific molecular composition [16]. Molecular analysis requires a costly sensory front-end that is able to resolve the identities of individual molecular constituents. Aggregate headspace sampling generally uses sensors that possess receptive fields with wide footprints in molecular space. The single sensory output is representative of an aggregate detection that spans the receptive field. Precise molecular composition and molecular dynamics such as suppression (masking of one detectable molecular species¹ by another) are not explicitly resolvable. However, the consequences of molecular combinations are implicit in the sensory output. Aggregate sampling therefore provides a broad characterisation of odour character, which is similar to the final characterisation that takes place in biological systems.

Equilibration: Prior to sensing, the electronic nose must produce a gaseous headspace from the solid cheese sample. This is achieved by placing the solid cheese sample in a clean sampling cell that is free of extraneous odorants. Time is then allocated for volatiles from the sample to evolve into gaseous phase and reach an acceptable equilibrium. This equilibration period must not be so long as to allow the effects of oxidation and bacterial action to become evident. Thus the cheese must maintain its

¹ An ensemble of chemically identical molecular entities that can explore the same set of energy levels on the timescale of the experiment – IUPAC Compendium of Chemical Terminology [43].

characteristic odour. Once produced, the headspace is transferred to the measurement cell or sensor head where it is sampled.

Internal dynamics: Cheese odour headspaces are typically unstable. That is, the odorant mix evolves over time. Chemical composition of the cheese sample, bacterial action, chemical decomposition and equilibration rates of volatiles all affect headspace character. These intrinsic processes differ from one cheese to another and are responsible for the unique odour of each cheese.

External dynamics: External dynamics such as temperature variation also affect headspace character. In order to reduce spurious variation in headspace character the following external factors were controlled:

- Temperature,
- Measurement cell volume,
- Equilibration time,
- System pressure,
- Gas flow rate, and
- Carrier gas purity.

The regulation of external variables ultimately results in more precise odour discrimination. Headspace regulation also finds application in traditional gas analysis technologies such as mass spectroscopy and gas chromatography. These technologies apply headspace regulation for exactly the same reason as biological systems – the reduction of spurious variability.

Static and dynamic headspace: Headspace may be described as static or dynamic depending on whether it is sampled in-situ or introduced into a gas flow and moved to a remote measurement cell [44]. This study employs static headspace equilibration followed by dynamic transfer to a measurement cell. The static phase permits equilibration of volatiles in a sealed sample chamber for a fixed period. During the dynamic phase, the sample chamber is merged with a carrier gas flow that travels through the measurement cell. Dynamic transfer causes a desirable dilution of the headspace en route to the measurement cell. Dilution is addressed in greater detail in Section 3.2.4.

The design of the system hardware was guided by the principle of headspace management. Section 3.2.2 provides a brief overview of the gas handling system design.

3.2.2 Odorant handling

The gas circuit has four main components:

- The carrier gas source,
- Eight sample equilibration cells,
- A sensor head or measurement cell, and
- The gas reticulation and flow control system.

Gas source: Bottled synthetic air is used as a carrier medium. This provides a standardised source of clean air for odorant transport. Section 3.2.4 addresses carrier gas purity in greater detail. A secondary advantage of using bottled gas is that it can travel through the system under its own pressure and removes the need for a gas pump that could introduce its own odour into the headspace. In applications such as these, specialised non-lubricated and non-fouling pumps are often used at considerable cost. The synthetic air source is pressure regulated and flow-restricted. Pressure gauges and a flow meter were used to monitor carrier gas delivery.

Sample cells: Eight identical sample cells were used to allow for automated measurement of eight odour sources, one of which could be a control. Figure 3.2b shows the glass sample cells. Each sample cell has a volume of $1.2 \times 10^{-4} \text{ m}^3$ or 120 cubic centimetres. Cheese samples were cut into blocks of approximately one cubic centimetre and placed in the sample cells for headspace equilibration. The gas reticulation system transferred each headspace in turn to the measurement cell for measurement.

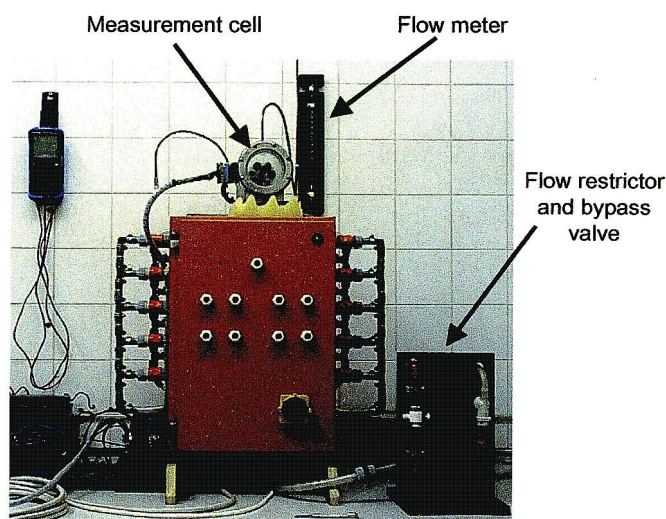


Figure 3.2a: Gas circuit hardware

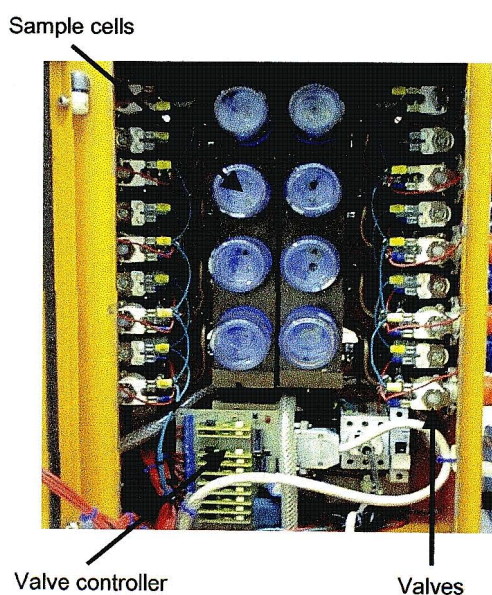


Figure 3.2b: Sample cells and valve sets

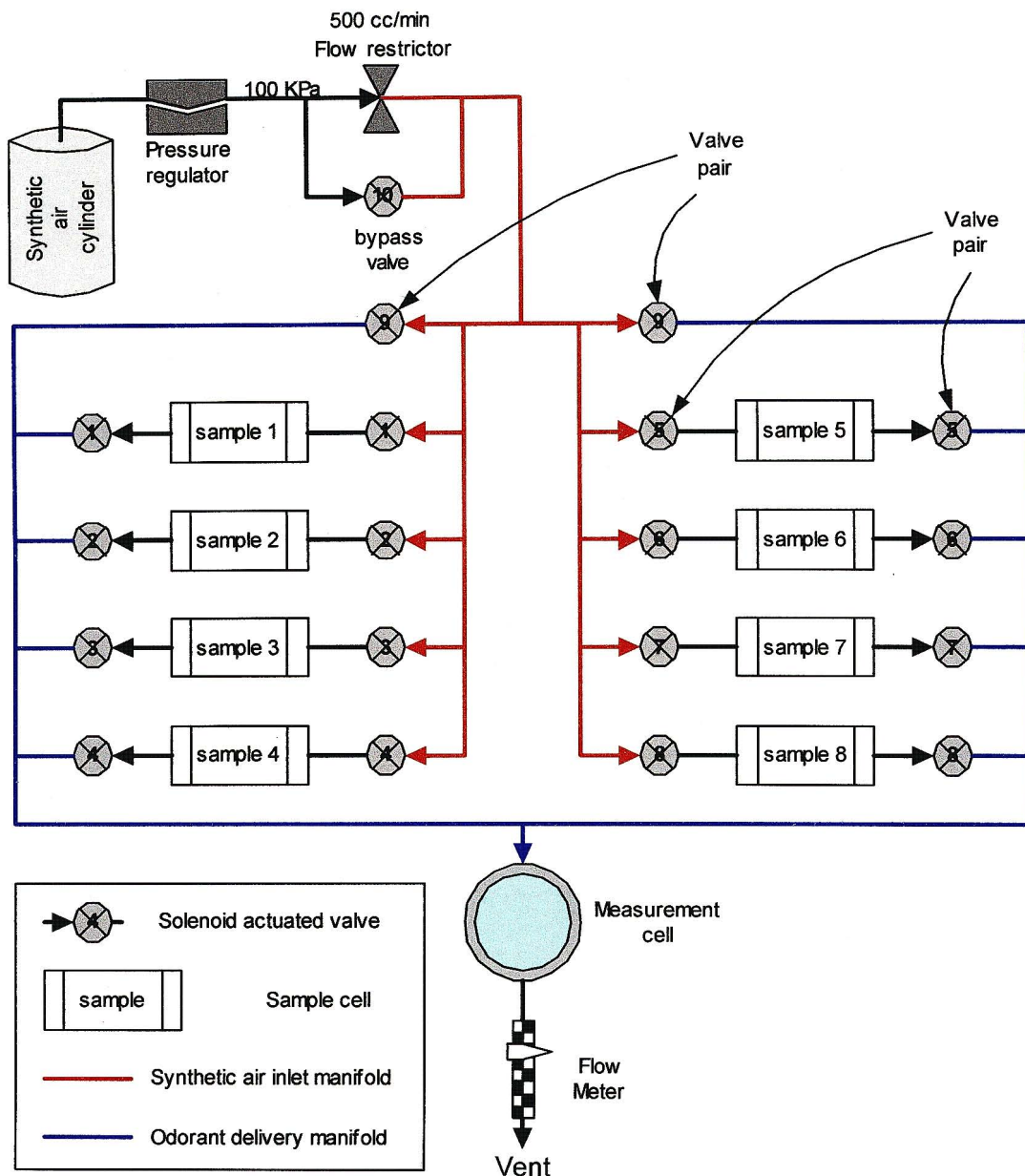


Figure 3.3: Gas circuit schematic

Measurement cell:

The gas delivery and electrical measurement systems converge in the measurement cell, which has a volume of $4.08 \times 10^{-4} \text{ m}^3$ (or 408 cm^3). Six metal oxide semiconductor sensors were used to detect the cheese odorants that were transferred to the measurement cell. Section 3.2.3 deals with sensor technology in greater detail. Figures 3.4a & 3.4b show the measurement cell.

Gas reticulation: The gas reticulation system transports gas through the system under software control at a flow rate of approximately $1 \times 10^{-5} \text{ m}^3/\text{s}$ (or $10 \text{ cm}^3/\text{s}$). This software is described in greater detail in Section 3.3. Figure 3.3 illustrates the complete gas circuit. All valves are normally closed. Some

valves are hardwired together (as indicated by valve sets 1 to 9 in Figure 3.3) and always work in pairs. Valve 10 may be used to bypass the flow-restrictor on the carrier gas source. The flow restrictor is used to limit the flow rate during headspace transfer to the measurement cell. Valves 9 and 10 are opened simultaneously to flush the odorant delivery manifold and measurement cell at maximum flow rate. These valves remain closed during other operations. Valve sets 1 through 8 may be operated individually or in any combination to transfer headspaces or combinations thereof to the measurement cell.

3.2.3 Sensing

There are two main approaches to chemical sensing [45]:

1. The use of highly selective sensors with minimal signal processing, and
2. The use of less selective sensors and pattern recognition which when combined, produce the necessary specificity.

Electronic noses generally fall into the latter category where a broad spectrum of chemicals can be detected (but not necessarily identified) with a minimal set of sensors. Within this category, there are currently two main classes of sensors [46]:

1. Inorganic Metal Oxide Semiconductor (MOS) devices such as tin-dioxide sensors, and
2. Organic conducting and insulating polymer devices, which were not commercially available at the time of hardware development.

This study uses tin-dioxide (SnO_2) MOS sensor technology manufactured by Figaro Engineering (Osaka, Japan). These are the most widely used of all gas sensors [11]. Figure 3.4a shows the sensor head with its array of six Figaro 8-Series Taguchi Gas Sensors.

T. Seiyama and N. Taguchi discovered metal oxide semiconductor sensor technology separately in 1962 [47]. Figaro Engineering produced the first commercial release of tin-dioxide sensors in 1967. These devices were designed for application in gas alarm systems. By 1988 the notion of odour sensing had already been proposed, and K. Takahata of Figaro Engineering predicted that MOS devices could be used in such applications [16]. He also identified the relationship between sensor output and actual “smell” as a technical problem that needed clarification. This remains the subject of research today. Chapters 4 and 5 explain the approach that was adopted in this study.

Advantages of MOS technology: The simple construction, low cost and small weight and size of MOS sensors make prolific application of the technology possible. Traditional Gas-Chromatography/Mass-Spectroscopy (GC/MS) technologies have analytical power, but they are cumbersome and expensive. MOS technology allows continuous [48], online, real-time in-situ detection [49]. Furthermore, MOS

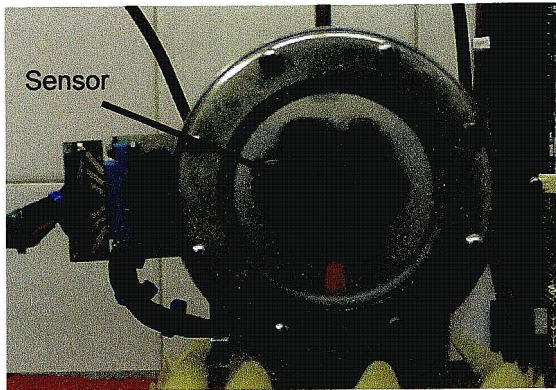


Figure 3.4a: Measurement cell front view

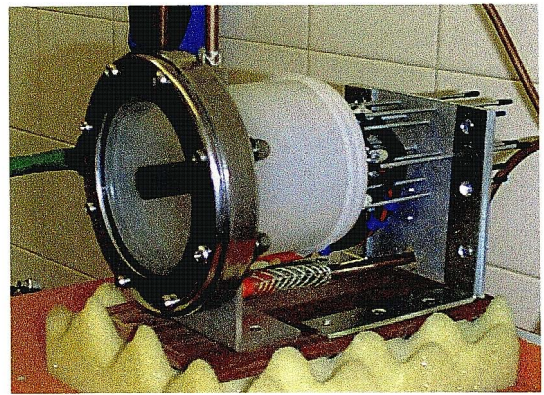


Figure 3.4b: side view

technology is easily produced [48]. This introduces new possibilities such as affordable odour monitoring in the food industry, compact breath analysers and low cost intelligent gas alarms. MOS devices are also able to detect over a wide concentration range relative to other sensors, and perform well at low concentrations [16,50]. However, they still remain limited in terms of dynamic range and adaptability relative to olfactory receptor neurons.

Problems associated with MOS sensors: When compared to GC/MS, MOS sensors provide reduced analytical ability [15]. Each sensor has a broad and poorly defined receptive field. Various combinations of input gases at various concentrations can produce the same sensory output. Therefore, the single output cannot be used as an accurate indicator of concentration or chemical composition. This problem may be addressed by the use of arrays of sensors and modern signal processing techniques. Sensor stability is another source for concern. Sensory characteristics can be temporarily or permanently modified by the sensor's exposure to certain gases or operating conditions. This is remedied by protecting the sensors from adverse operating conditions such as corrosive atmospheres and high temperatures. Instability is discussed further in this section.

Sensor construction: Figure 3.5 illustrates the internal construction of the Figaro 8-Series gas sensor [51]. A mass of sintered crystalline SnO_2 particles is deposited around a ceramic tube. The tube fulfils the dual role of supporting the substrate and providing electrical insulation from a heater element that passes through it. The heater keeps the substrate at an operating temperature of 400 °C. Gold electrodes are deposited at either end of the substrate and lead wires are bonded to them. These lead wires connect to an external resistance measurement circuit. The heated sensor element is housed in a flameproof enclosure. A fine SUS316 stainless steel mesh allows gases to reach the sensor substrate while preventing the passage of flame. This is necessary in the presence of combustible gases for which these sensors are specified.

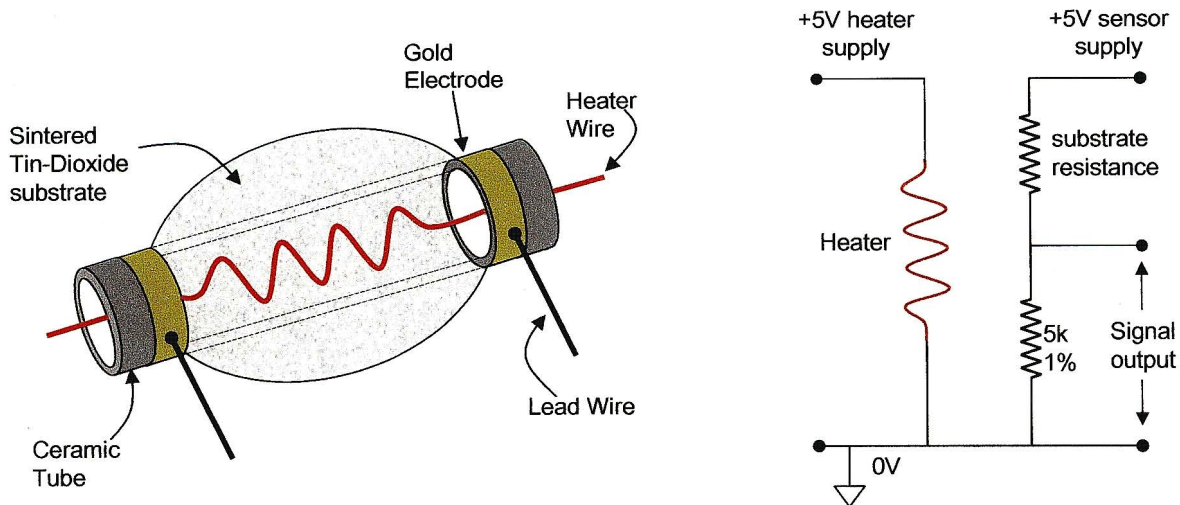


Figure 3.5: Figaro 8-series sensor construction and electrical connection diagram

Operating principle: The operating characteristics of all gas sensors are dependent on their working temperature [46]. The electrochemical processes that take place in MOS sensors are no exception. The following discussion considers only the specified operating temperature of 400 °C.

At 400 °C, and in the absence of gaseous oxygen, electron mobility is high, and the substrate offers little resistance to the flow of electrons across crystal grain boundaries [52]. This results in a low resistance measurement across the sensor electrodes. The opposite extreme is encountered in clean air that contains oxygen, where the resistance reaches its peak value. For the purposes of this discussion, clean air may be considered as natural oxygen bearing air that is devoid of any detectable combustible gas species.

In clean air, oxygen, which has a high electron affinity, is adsorbed² on the SnO₂ particle surfaces where there is an abundance of mobile (valence) electrons. These donor electrons are transferred to the adsorbed oxygen atoms creating a positive (electron depleted) space charge layer under the surface of the particle [53]. This surface potential acts as a barrier to electron flow between the particles [54], and increases the total electrical resistance across the sintered particle substrate.

The high resistance state is the normal sensor condition in the absence of a detectable gas. When a detectable gas is present, oxygen sorbates are consumed in a catalytic oxidation of the gas [53], where the substrate functions as the catalyst. During this reaction the adsorbed oxygen atom releases the

² Sorption is a generic term that refers to the process whereby sorbates (odorants) attach to sorbants (solid or semi-solid base material) either as a surface attachment (adsorption) or by migrating into the solid phase (absorption). Desorption is the reverse process where sorbates are liberated back into the atmosphere.

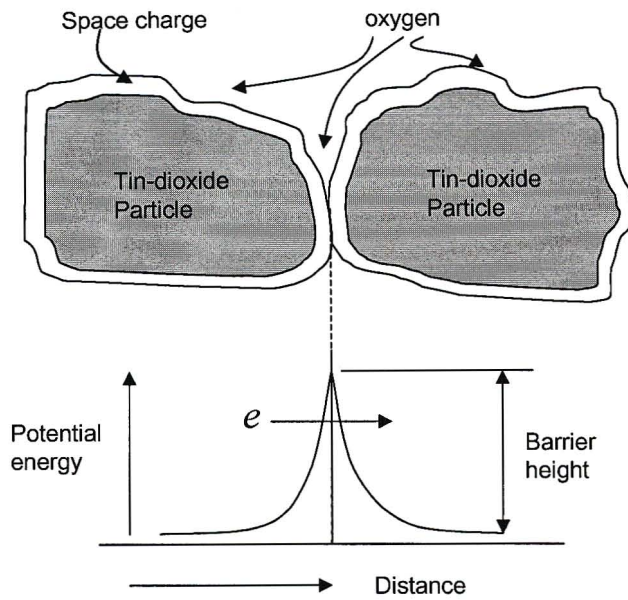
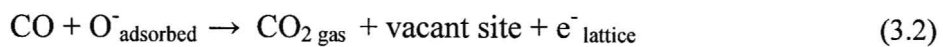
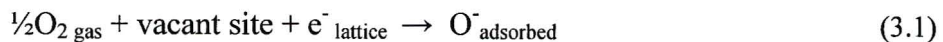


Figure 3.6: Potential barrier at the crystal grain boundary

bound lattice electron back into the SnO₂ lattice, and bonds with the detectable gas. The release of an electron increases total electron mobility, decreases the space charge layer and ultimately increases conductivity. The reaction products then desorb, creating a vacant site for further oxygen sorption. The original space charge is restored as soon as a new oxygen atom occupies the vacant site. The formulae below describe the detection of carbon monoxide (CO) [52]. This is the simplest type of reaction that takes place at the surface.



Equation (3.1) describes the sorption of oxygen, which produces an elevated potential barrier as illustrated in Figure 3.6. Equation (3.2) describes the oxidation of carbon monoxide and desorption of the reaction product (CO₂), which reduces the height of the same potential barrier and restores the vacant site and valence electron. Oxygen adsorption (3.1) reduces conductivity while the catalytic conversion of CO (3.2) increases conductivity. The total conductivity at any given time is related to the rates of these two reactions.

Depending on the specific gas, some of the products desorb immediately after the catalytic oxidation reaction as with CO, while others remain in a sorbed state and gradually desorb after further decomposition [15]. This delayed desorption has a cascade of consequences. During detection, the adsorbed oxygen atom releases the bound lattice electron in the catalytic oxidation of the gas.

Substrate conductivity increases indicating a detection phenomenon. Thereafter, the following happens:

- The reaction products remain sorbed thus occupying a potential oxygen sorption site.
- The local space charge is prevented from recovering to its pre-detection distribution.
- Sensor conductivity exhibits a memory of the detection event long after the fact. This impedes sensor recovery and future detections.
- Reaction products eventually desorb creating a vacant site for oxygen adsorption.
- The local pre-detection space charge and conductivity are restored after sorption of oxygen at that site.

The sensor's operating temperature of 400 °C is chosen so as to maximise desorption and sensor recovery rates. Further details of the complex surface reactions are beyond the scope of this project. However, Heiland and Kohl's [15] "Physical and chemical aspects of oxidic semiconductor gas sensors" may be consulted for further information.

Gas specificity: The sensors used in this study achieved a measure of gas selectivity through the variation of sensor materials and operating temperature [52]. The controlled addition of catalysing materials or surface chemical modifiers to the substrate, for example palladium [15] or silicon dioxide [55], achieves limited gas selectivity. Further proprietary details were not available for the sensors used in this study. It should be noted that the receptive fields of such sensors remains large not withstanding the selectivity enhancement. As mentioned at the start of this section, the sensors used in this study have broad and poorly defined receptive fields. Table 3.1 lists the six sensors that were used and indicates the principal region of their receptive fields. Note that all the sensors responded to cheese odours regardless of the specified receptive field.

Table 3.1: List of Figaro MOS sensors used in this study

<i>Sensor</i>	<i>Receptive field specified by manufacturer</i>
TGS800	General air contaminants
TGS813	General hydrocarbons
TGS826	Ammonia, amine
TGS822	Solvent vapour
TGS842	Methane, natural gas
TGS880	Fumes from food

3.2.4 Recovery

Several mechanisms were put in place to promote rapid recovery of the sensory system after each odorant measurement. The underlying approach was to control the quality of air to which the sensors were exposed.

Construction materials: Construction materials in the gas circuit were selected so that they were:

- Chemically stable and unlikely to react with odorant gases,
- Unlikely to absorb or produce detectable gases, and
- Did not contain any silicone rubber or adhesives, which damage MOS devices [54,55].

These measures reduce the quantity of noise gases generated by the materials in the gas circuit. During the initial commissioning of the system, the gas circuit was flushed with pure oxygen for an hour. This was done in order to minimise and stabilise any oxidisable agents in the system.

Synthetic air carrier: Bottled synthetic air was used as the transport medium for all odorants and for general flushing of the system. The gas 'BOC special gases Air IG Zero' (supplied by Afrox South Africa Ltd) has near zero humidity and is composed of high purity oxygen and nitrogen in a proportion that simulates a natural atmosphere. The use of bottled synthetic air keeps environmental noise gases out of the system.

High dilution: Field trials of MOS sensor technology have shown that exposure to high food odour concentrations caused significant sensor drift and reduced sensor lifespan [53]. The increased reaction rate in the presence of high gas concentrations raises the temperature of the substrate. Experiments have shown an increase in operating temperature from 400 °C to 500 °C [16] in some cases.

Concentration induced temperature peaks affect the sensitivity of the sensor in two ways:

- In the short term: adsorption and desorption rates, and surface reactions are modified. Restoring the normal operating temperature reverses this phenomenon.
- In the long term: crystal growth is accelerated and the number of grain boundaries is reduced. This phenomenon is not reversible and results in permanent damage to the sensor.

Fortunately, MOS sensors are able to detect gases at low concentrations. Equation (3.3) shows that a power law relationship exists between detectable gas concentration and sensor resistance [1]. Furthermore, sensitivity or the slope of the curve increases at lower concentrations [56].

$$\Delta\sigma \propto X^r \quad (3.3)$$

where: $\Delta\sigma$ = change in sensor's electrical resistance
 X = gas concentration
 $0.5 \leq r \leq 1.0$

All these factors indicate that low concentrations should be used. The dynamic headspace transfer mechanism was designed so as to dilute the headspace in a fixed volume of synthetic air. This transfers odorants into the measurement cell at a reduced concentration. Figure 3.7 illustrates the shape of the curve described by Equation (3.3), take note of the raised sensitivity at low concentrations.

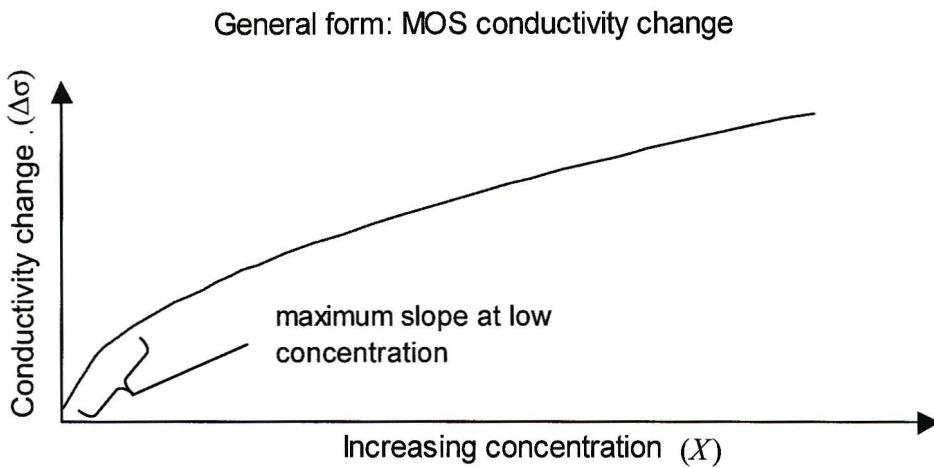


Figure 3.7: The relationship between concentration and conductivity change

System flushing: Immediately after a measurement is taken, the odorant delivery manifold and sensor head are flushed with synthetic air. This ensures that residual gases are removed from the system so that the sensors can re-establish their base references and new measurements can be taken.

3.3 Measurement configuration tools

The gas reticulation system (Figure 3.3) is managed by electrically actuated valves. These valves in turn are controlled by software running on a Personal Computer (PC). This section provides a brief description of the software. A more comprehensive description is provided in Appendix A.

In order to maximise flexibility, a simple interpreted language was created. Table 3.2 provides a summary of the language command set. Measurement control scripts were written in this language and were compiled to a byte code that was interpreted and executed. The compiler that produces the byte code was written in the AWK scripting language. Compiled byte code is interpreted and executed by an interpreter that runs on a PC. The interpreter was written in the C programming language. An Integrated Design Environment (IDE) was also developed for the programming language. Figure 3.8 illustrates the IDE window as it appears immediately after a successful compilation. The compiler, interpreter and IDE are described in greater detail in Appendix A.

Table 3.2: Measurement control language command set

<i>Command</i>	<i>Description</i>
stop	Shutdown the system
open(<i>valve</i>)	Open the specified valve or valve set
close(<i>valve</i>)	Close the specified valve or valve set
flush(<i>begin</i>)	Begin flushing the odorant delivery manifold and sensor head
flush(<i>end</i>)	Stop flushing
rate(<i>high</i>)	Activate high flow rate
rate(<i>low</i>)	Activate low flow rate (measurement flow rate)
capture	Capture the current output of the six sensors (one sample only)
delay(<i>period</i>)	Delay for the specified number of seconds
loop(<i>dest</i> , <i>iterations</i>)	Go <u>back</u> to destination no more than the specified number of iterations
log(<i>open</i>)	Create a new log file for sensor data (filename is auto-generated)
log(<i>close</i>)	Close the currently open log file
macro name(<i>paras</i>)	Begin macro definition with name and parameters specified
mend	End current macro definition

The simple compiler offers no syntax checking or error detection. The author of the control script must verify its correctness. The control script in Listing 1 was used to capture data for this study. The five-phase measurement event that is coded in this script is described in Section 3.4.

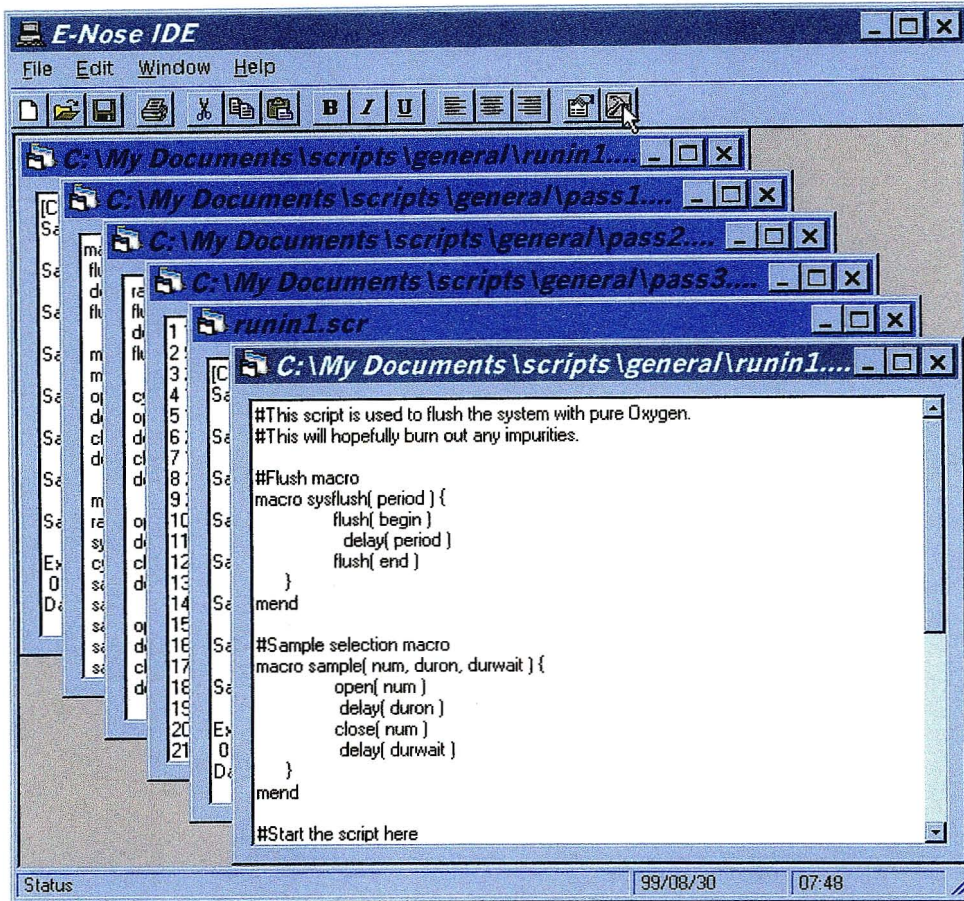


Figure 3.8: Screen capture of the IDE indicating various compilation output windows

Listing 1: Measurement control script used to capture data

```
# Control script to capture odour responses from the new sensors - to test cheese response
# Experiment number 4
# NOTE: same control strategy is used as exp3 for wines

# Cheeses:
#1 FVcamembert: Fairview Camembert
#2 FVbrie:
#3 FVbrietb: Brie with tomato and basil
#4 FVbluerock:
#6 SBmozza: Simonsberg tradition Mozzarella
#7 FVchevinsrp: Chevin with sweet red peppers

#NB: sample chamber 5 is not used due to manufacturing defect

####SAMPLING MACRO####
macro sample (bottle, dur, Bstationary, Bflowing, Oflowing, Ostationary, flushing) {
log(open)

#Phase 1: Capture "Bstationary" samples of blank odour (air) in stationary mode
I0:
capture
delay( dur )
loop( I0, Bstationary)

# Phase 2: capture "Bflowing" samples of flowing blank odour (air) - gas flow to measurement cell via flush circuit
flush(begin)
I1:
```



```

capture
delay( dur )
loop( I1, Bflowing)
flush(end)
delay(dur)

# Phase 3:   Capture "Oflowing" samples of flowing odour – gas flow to measurement cell via sample cell
open( bottle )
I2:
capture
delay( dur )
loop( I2, Oflowing)
close(bottle)

# Phase 4:   Capture "Ostationary" samples of stationary odour – all valves closed – static headspace
I3:
capture
delay( dur )
loop( I3, Ostationary)

# Phase 5:   Capture "flushing" samples of flushing odour – gas flow to measurement cell via flush circuit
flush( begin)
I4:
capture
delay( dur )
loop( I4, flushing)
flush( end)
log(close)

}
mend
#####SAMPLING MACRO ENDS#####

#####RECOVERY MACRO#####
macro recover(cycles,dur){
r1:
flush(begin)
delay(dur)
flush(end)
delay(dur)
loop(r1, cycles)
flush(begin)
delay(5)
flush(end)
delay(5)
}
mend
#####RECOVERY MACRO ENDS#####

#####MAIN PROGRAM#####
rate(low)
cyc:
recover(10,2)
sample(1, 1, 20, 30, 50, 40, 116 )
recover(10,2)
sample(2, 1, 20, 30, 50, 40, 116 )
recover(10,2)
sample(3, 1, 20, 30, 50, 40, 116 )
recover(10,2)
sample(4, 1, 20, 30, 50, 40, 116 )
recover(10,2)
sample(6, 1, 20, 30, 50, 40, 116 )
recover(10,2)
sample(7, 1, 20, 30, 50, 40, 116 )
loop( cyc, 19 )
stop
#####MAIN PROGRAM ENDS#####

```


3.4 The standard measurement event

Once off ‘snap-shot’ measurements are only of value if the sensors reach a meaningful steady state after the introduction of the cheese headspace [57]. Given the unstable nature of the sensors and the dynamic headspace, a stable and repeatable steady state (measurement plateau) is unlikely to occur. It is therefore necessary to capture the sensor’s temporal response to the stimulus. This also increases the total information that is provided by the sensors [45,57] and complies with the methodology proposed in Chapter 2 (Sections 2.2.1 & 2.3.1).

The measured temporal response includes the detection and recovery transients, that is, the sensory response to the introduction and removal of the stimulus headspace. Given that each cheese produces an unstable but chemically distinct headspace, and each sensor has an unstable but unique transfer characteristic, the detection and recovery transient of each sensor is likely to be a function of headspace identity. It is also reasonable to assume that headspace identity can be extracted from such data using appropriate signal processing and pattern recognition techniques. Measurement of temporal responses effectively implements a simulated sniff and is able to expose:

- The initial base reference,
- Detection transient (ramp-up),
- Detection plateau,
- Recovery transient (ramp-down), and
- Final base reference.

Measured temporal dynamics are likely to vary from sensor to sensor as well as from one cheese to another, thereby providing a wealth of information [57] that could be used to identify cheeses.

A temporal measurement technique was developed and standardised for this investigation. Each measurement is called a measurement event. The measurement event spans 256 measurements taken across six sensors at a sampling rate of 1Hz. The output of each sensor is sampled to 12-bit accuracy (4096 discrete levels) over the range 0V to 5V. It should be noted that 8-bit resolution (256 discrete levels) would have sufficed provided that the measured signals spanned the 8-bit range. Unfortunately, the signal range in response to cheese headspace could not be predicted in advance. All that was known was that the measured signals would stretch out somewhere within the extremes of 0V to 5V. This 5-volt range was therefore sampled to 12-bit precision in the knowledge that the signal would span a subset of that range. As it turned out, the typical measurement spanned 20% of the full 12-bit range (see Figure 3.9), which is well in excess of the desired minimum 8-bit resolution.

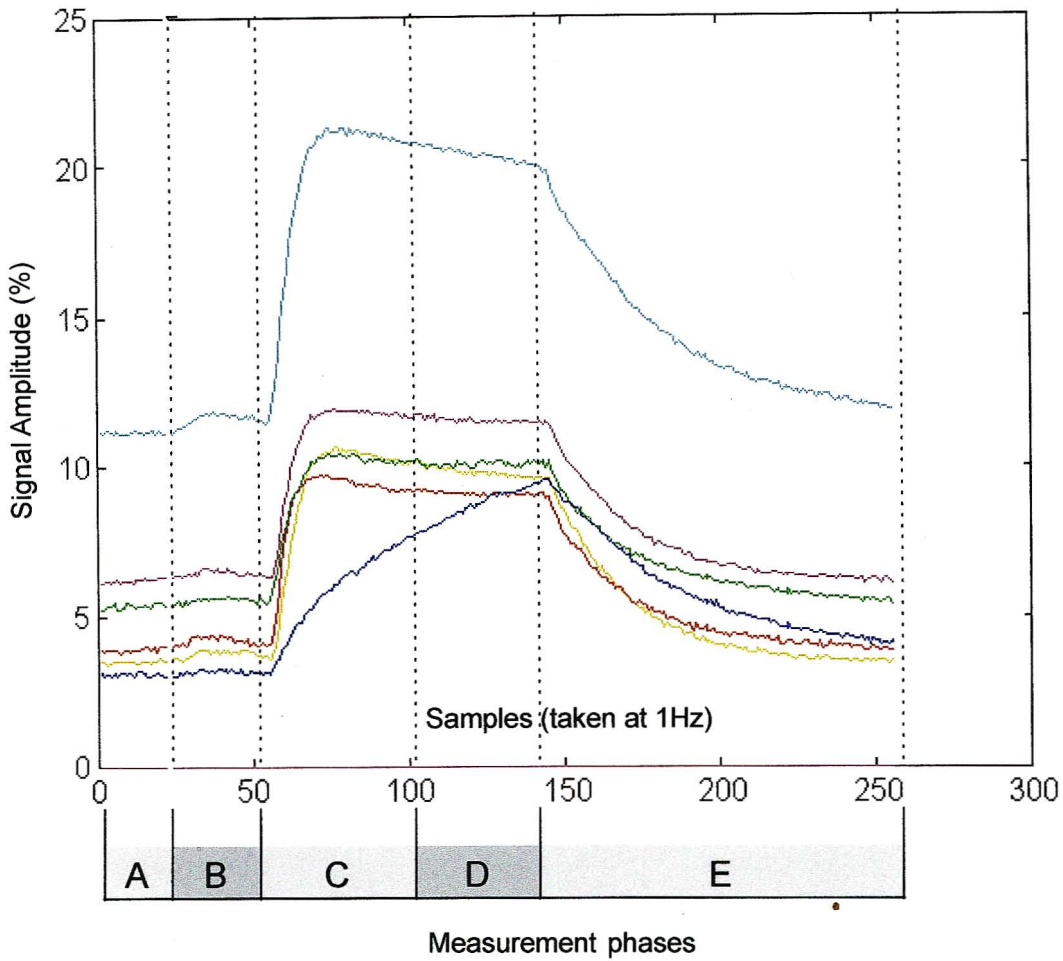


Figure 3.9: A plot of the outputs of the six different sensors for a typical measurement event.

The typical measurement plot in Figure 3.9 shows to varying degrees, that the sensors are not able to achieve a flat measurement plateau. The measurement event is comprised of five distinct phases:

Phase A

Description: Capture data with sensors in their resting state. That is, in the presence of static (not flowing) carrier gas only.

Purpose: To establish the base reference.

Duration: 21 samples taken at 1Hz.

Note: Even in pure, static (not flowing) synthetic air some degree of noise is evident.

Phase B

Description: Capture data with pure carrier gas flowing at the measurement flow rate.

Purpose: To establish sensor response to flowing carrier (control odour).

Duration: 31 samples taken at 1Hz.

Note: simply making the pure carrier gas flow at the measurement flow rate produces a noticeable change in sensor output. This exposes an element of the sensor instability.

Phase C

Description: Capture data during dynamic headspace transfer at the measurement flow rate.

Purpose: To establish the detection transient ramp-up.

Duration: 51 samples taken at 1Hz.

Note: Sensor response appears delayed due to time taken for the headspace constituents to reach the sensor's active surfaces and for surface reactions to commence. Five of the six sensors reach their peak activation during this phase.

Phase D

Description: Halt the headspace transfer and capture data with the static headspace that remains in the sensor head.

Purpose: To establish the detection plateau.

Duration: 41 samples taken at 1Hz.

Note: Some sensors show a significant change in their outputs and continue to produce a dynamic output after 40 seconds of static headspace.

Phase E

Description: Capture data while flushing the odorant delivery manifold and sensor head with pure carrier gas at the measurement flow rate.

Purpose: To establish the recovery transient (ramp-down).

Duration: 117 samples taken at 1Hz, truncated to 112 samples (discussed in Chapter 4, Section 4.2.)

Note: Sensor response appears delayed due to time taken for the headspace constituents to purge from the measurement cell and the sensor's active surfaces. After 116 seconds of purging with pure carrier gas, some sensors still remain significantly far off from their original base references.

Each measurement is logged to a separate file and stored with an auto-generated filename. File naming conventions are discussed in Chapter 4.

3.5 Conclusion

This chapter presents the design of the sensory front-end that produces raw measurements. Various principles and practices that are pertinent to the production of raw measurements are addressed. The

design philosophy favoured a conservative and limited hardware deployment with a maximal amount of functionality deployed in software. This in turn permits greater flexibility.

Both the hardware and software functionality remain closely guided by biological principles discussed in Chapter 2. Most notable are the environmental stabilisation measures that were incorporated in the hardware, and the standardised simulated sniff deployed in the software controlled measurement event.

CHAPTER 4

A DATA PREPROCESSING STRATEGY

4.1 Introduction

Having investigated biological system organisation in Chapter 2, a biologically inspired measurement and signal processing strategy was developed. This in turn led to the development of a measurement event that was distributed across time and sensor-space (presented in Chapter 3). Each measurement event is therefore represented as a two-dimensional matrix of measurements that spans the six sensors and 256 sampling instants. This chapter presents the manner in which that data is refined and prepared for final classification.

The processing methodology adopted here can be broadly arranged into two phases. The first phase transforms or maps the data onto an orthogonal or decorrelated vector space. This emulates the role of the olfactory bulb in mammalian olfaction. The domain of the new vector space should preferably be described by a basis set that is fixed (i.e. never needs subsequent modification). Each new odour measurement is then expressed as a linear combination of these standard bases. Several orthogonal vector space transformations can be used to achieve this. Section 4.2 addresses this issue.

Phase two is comprised of an assortment of basis or coefficient selection strategies [57]. Coefficient selection emulates the selective convergence of bulbar outputs at each cortical locus where classification and association processes commence. Various coefficient selection strategies are proposed in Section 4.3.

4.2 The discrete cosine transformation

It is important to establish the reason for which decorrelation is employed. Continuing with the biological analogy, the olfactory nerve presents to the brain a confusing mass of signals from a variety of receptor types. The olfactory bulb separates and sorts these signals so that each glomerulus expresses exactly one receptor type. Each glomerulus then produces a single output that is correlated purely with one receptor type and therefore decorrelated from other receptor types. The total bulbar output consequently expresses a weighted combination of decorrelated bases, where each receptor type is a basis and the glomerular output is the weighting factor.

Orthogonal transform bases represent source domain patterns such that any given transform basis cannot be reproduced by a combination of other bases in that transformation. The orthogonal Fourier spectrum provides a case in point. It is impossible to represent any pure sinusoidal wave as a linear combination of other sinusoids. Therefore each orthogonal basis represents a unique source domain pattern. The arbitrary input signal is simply a linear combination of these unique patterns. Classification is made simpler when the signal is expressed in this fashion; an explanation follows.

Individual coefficients (and the bases they represent) are orthogonal features and may be included or excluded from the final feature set without affecting other features. Therefore, disruptive features may be removed and useful ones preserved. This is exactly what happens in the olfactory cortex where cortical loci accumulate only those glomerular outputs that are of special interest thus reducing the dimensionality of the specific classification problem.

The functionality described above can be emulated with any orthogonal transformation followed by an informed (intelligent) coefficient selection mechanism. The immediate task would be to find a suitable transform.

When it comes to information distribution, one transform is known to be optimal. It is a well-established fact that the Karhunen-Loève Transform (KLT), also known as Principal Component Analysis (PCA), stores the maximum amount of information in the fewest coefficients [58]. This effectively reduces the number of significant basis vectors. The majority of bases, which encode little information, could be eliminated and a reduced and focused feature set would remain for classification purposes.

Several studies have already shown that KLT/PCA performs well for any fixed set of olfactory measurements [e.g. 59,60]. The success of the KLT can be attributed to a single fact; the KLT statistically optimises its basis functions to suit the prevailing measurement signal statistic [61]. Before use, the KLT must be allowed to optimise its basis functions. To do this it needs a set of olfactory measurements that fully describes the general measurement statistic for the expected duration of the experiment. The covariance matrix is computed for the data set, and the eigenvectors of the matrix are used as the bases. The eigenvectors with the largest eigenvalues encode the most information. The eigenvectors (bases) are arranged in decreasing eigenvalue order.

The KLT cannot be regarded as a fixed transform. It modifies its own transformation to suit the training data that it is presented with. If the signal statistic were to change at a later stage, the transformation would no longer be optimal and the underlying covariance matrix and eigenvectors

would need to be recomputed. To summarise, the KLT requires a stable signal statistic and a training set that describes fully the domain and range of that stable input signal.

The KLT is described as the optimal¹ vector space transformation. The remaining non-optimal orthogonal transformations still offer benefits in other areas where the KLT does not excel. Frequency spectral analysis, for example, does not require the representational efficiency of the KLT. Rather, it requires a basis set of uniform amplitude sinusoids that span a frequency spectrum. The Discrete Fourier Transform (DFT) would be better suited to that application.

The electronic nose places two requirements on the transform, representational efficiency and insensitivity to sensor drift. The KLT performs best only on representational efficiency. All of the “sub-optimal” orthogonal transforms have fixed bases and offer insensitivity to sensor drift. They also offer varying degrees of representational efficiency. A comprehensive study of Discrete Trigonometric Transforms (DTTs) and Discrete Orthogonal Transforms (DOTs) by Elliot and Rao [61] shows that the Discrete Cosine Transform (DCT) [58] offers the best representational efficiency of all sub-optimal DOTs. The DCT is often referred to as the near-optimal transform. A further benefit offered by the DCT is computational speed. Several fast implementations of the DCT now exist, while no globally applicable fast implementations exist for the KLT.

In the nose, each basis vector represents a geometric feature of an odorant (as detected by a geometry specific receptor). The question then arises, what do the DCT basis vectors represent in terms of the current application? Given that DCT bases are fixed for any given transform size (independently of the application), one immediately observes that the bases do not offer any special relationship to electronic nose data. However, these bases offer near-optimal compression of arbitrary information content into a small number of coefficients. That is, the transformed vector space expresses a concentration of information in a reduced subspace. Classification can then be effected in this low dimensionality subspace. Section 4.3 expands the concept of coefficient (subspace) selection.

As with all DTTs, the DCT was originally specified for a one-dimensional data sequence. This was subsequently generalised for two-dimensional data, and several fast implementations developed. This thesis makes use of the fast implementation proposed by Cvetković and Popović in 1992 [62]. Their algorithm provides exceptional computational efficiency [63] with regard to the number of multiplications and additions. This experiment processed all measured data in an offline batch mode. The improved speed of the efficient DCT algorithm was not essential, but its use was retained given the possibility of future development of an online system.

¹ The word optimal applies only to the ability of the transform to pack the maximum amount of information into the fewest coefficients. It does not apply to other issues such as implementation complexity and speed.

The 1D DCT is defined as follows:

Given any one-dimensional sequence of raw data

$$\mathbf{x} = \{x_0, x_1, \dots, x_{N-1}\} \quad (4.1)$$

for fast computation $N = 2^p$, $p \in \mathbb{Z}$

The DCT produces a transformed sequence of equal size

$$\mathbf{X} = \{X_0, X_1, \dots, X_{N-1}\} \quad (4.2)$$

where each element is the result of the DCT described below

$$X_n = \frac{2}{N} e(n) \sum_{k=0}^{N-1} x_k \cos \frac{(2k+1)n\pi}{2N}, \quad n = 0, 1, \dots, N-1 \quad (4.3)$$

$$e(n) = \begin{cases} \frac{1}{\sqrt{2}} & \text{for } n = 0 \\ 1 & \text{otherwise} \end{cases}$$

A two-dimensional DCT is achieved in several steps. Firstly, applying the 1D DCT to the rows of the $(m \times n)$ measurement matrix \mathbf{x} shown in (4.4).

$$\mathbf{x} = \begin{pmatrix} x_{0,0} & x_{0,1} & \dots & x_{0,(n-1)} \\ x_{1,0} & & \ddots & \\ \vdots & & & \\ x_{(m-1),0} & & & x_{(m-1),(n-1)} \end{pmatrix} \quad (4.4)$$

This produces an intermediate matrix \mathbf{X} that is transformed in the horizontal direction only by the application of (4.3).

$$\mathbf{X} = \begin{pmatrix} X_{0,0} & X_{0,1} & \dots & X_{0,(n-1)} \\ X_{1,0} & & \ddots & \\ \vdots & & & \\ X_{(m-1),0} & & & X_{(m-1),(n-1)} \end{pmatrix} \quad (4.5)$$

Taking the 1D DCT of the columns in \mathbf{X} produces the final matrix \mathbf{X} . This is achieved by transposing \mathbf{X} and then applying the horizontal transformation (4.3) a second time and transposing the result.

$$\mathbf{X} = \left[DCT(\mathbf{X}^T) \right]^T \quad (4.6)$$

Alternatively, the fast DCT algorithm [62] achieves the same result but exploits the symmetry of the cosine function to reduce the number of computational steps.

As indicated by (4.1), the fast transform places a constraint on data dimensionality that requires the transform length to be a positive power of two. The two-dimensional problem transforms both the rows and columns of the measured data matrix. The input matrix must therefore have dimensionality

$$2^p \times 2^q, \quad \text{where } \{p, q\} \in \mathbb{N} \quad (4.7)$$

In this experiment raw measurement matrices had a dimensionality of 6×261 , corresponding to the six sensors that were sampled across 261 sampling instants (described in Chapter 3). In order to satisfy the transform, simple padding and pruning were used to adjust measurement matrix dimensionality by:

- Adding two dummy channels with zero output, thus increasing the sensor-channel dimensionality from 6 to 8, and
- Deleting the last 5 sampling instants of Phase E (sensor recovery), thus reducing the sampling-instant dimensionality from 261 to 256.

Figure 4.1 illustrates a two-dimensional view of a typical measurement matrix prior to dimensionality adjustment, and Figure 4.2 illustrates the 2D DCT of an adjusted matrix.

Both the source and transform domains are two-dimensional, therefore, the distribution of information in each domain may be viewed as the volume under the curve. Figures 4.1 & 4.2 clearly show that the information distribution is more dispersed in the source domain and localised towards the lower temporal frequencies in the transform domain.

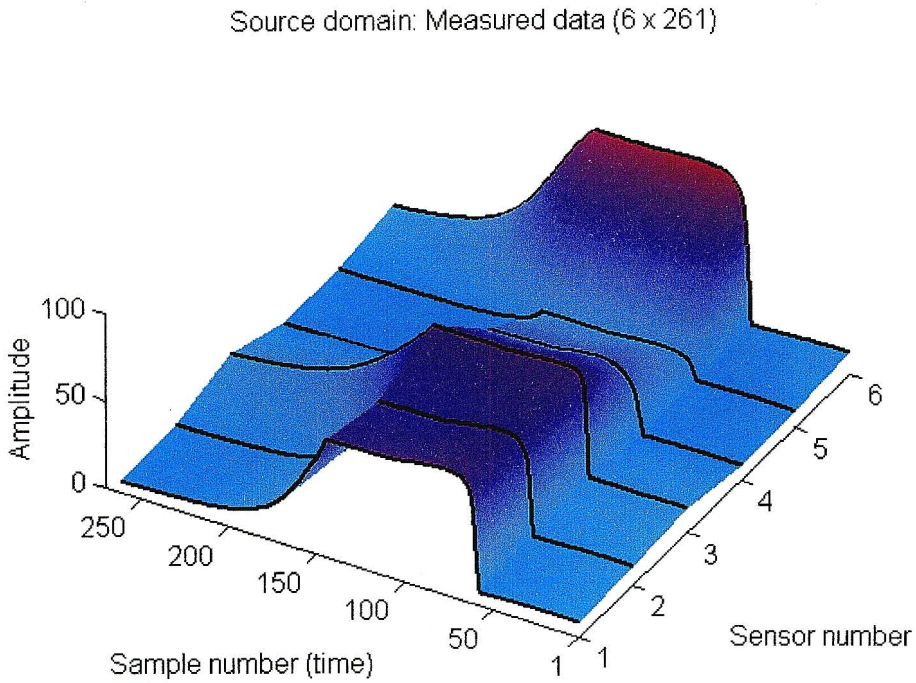


Figure 4.1: 2-D plot of a typical measurement event, indicating spatially and temporally distributed information content before transforming with the DCT. Dark lines indicate actual sensor traces and shading is used only to assist in visualisation.

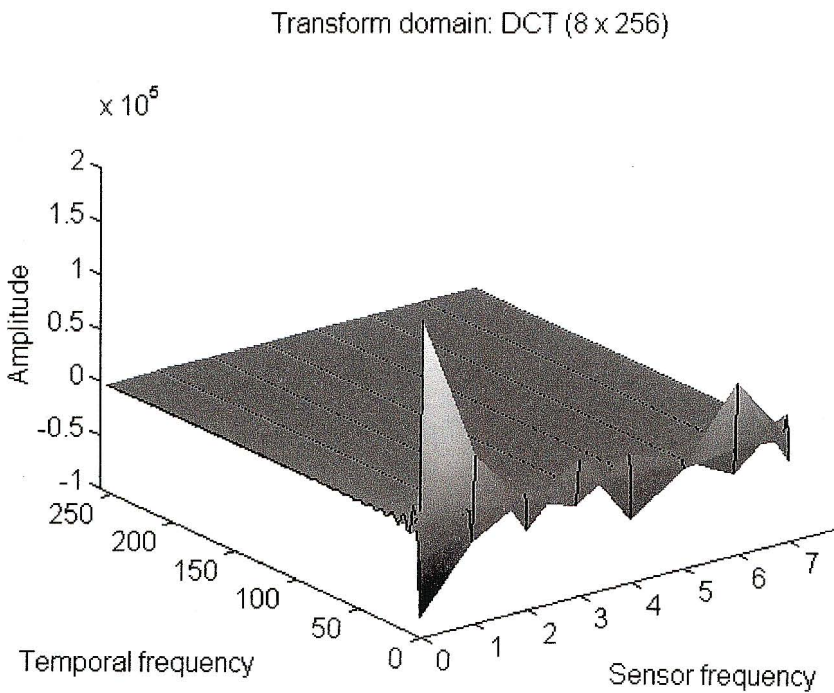


Figure 4.2: 2-D DCT of typical measured data, indicating the frequency information content in the sensor and temporal (sample) domains.

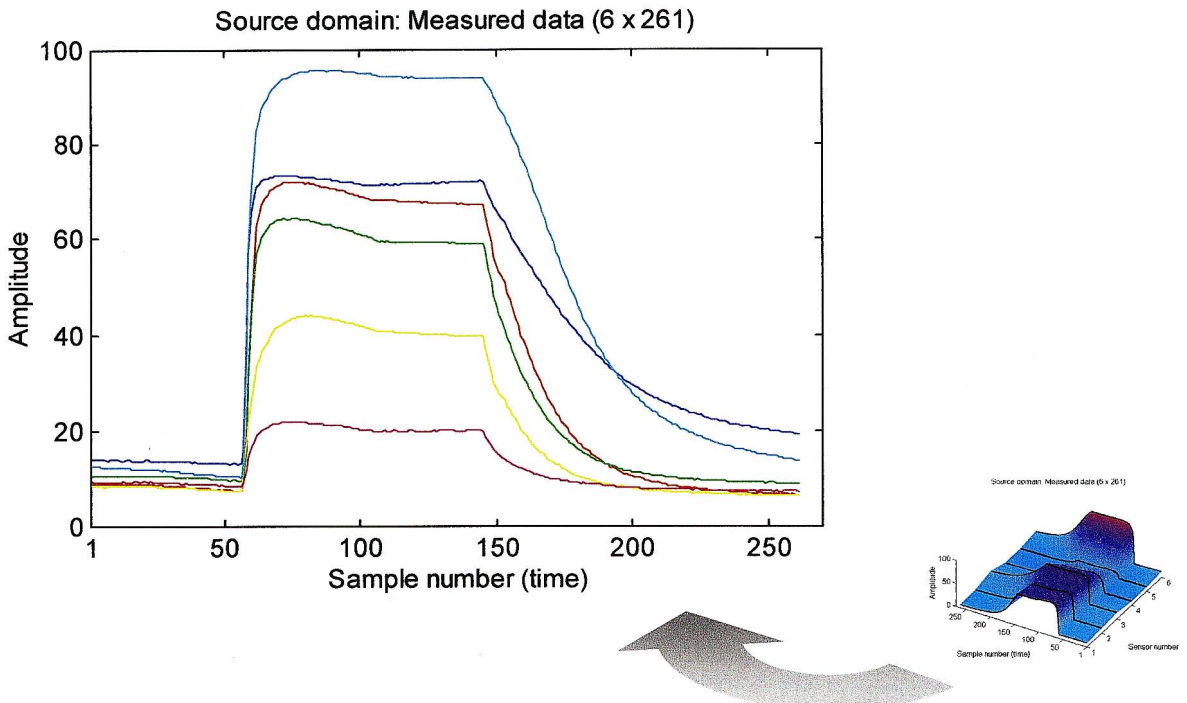


Figure 4.3: Measured data projected onto the temporal axis

Figures 4.3 & 4.4 are projections of the source and transform domain curves onto their respective temporal axes. When one moves from source to transform domain, the graphs show:

- A reduction in area under the curves, and
- Information localisation or compression towards the lower temporal frequency subspace.

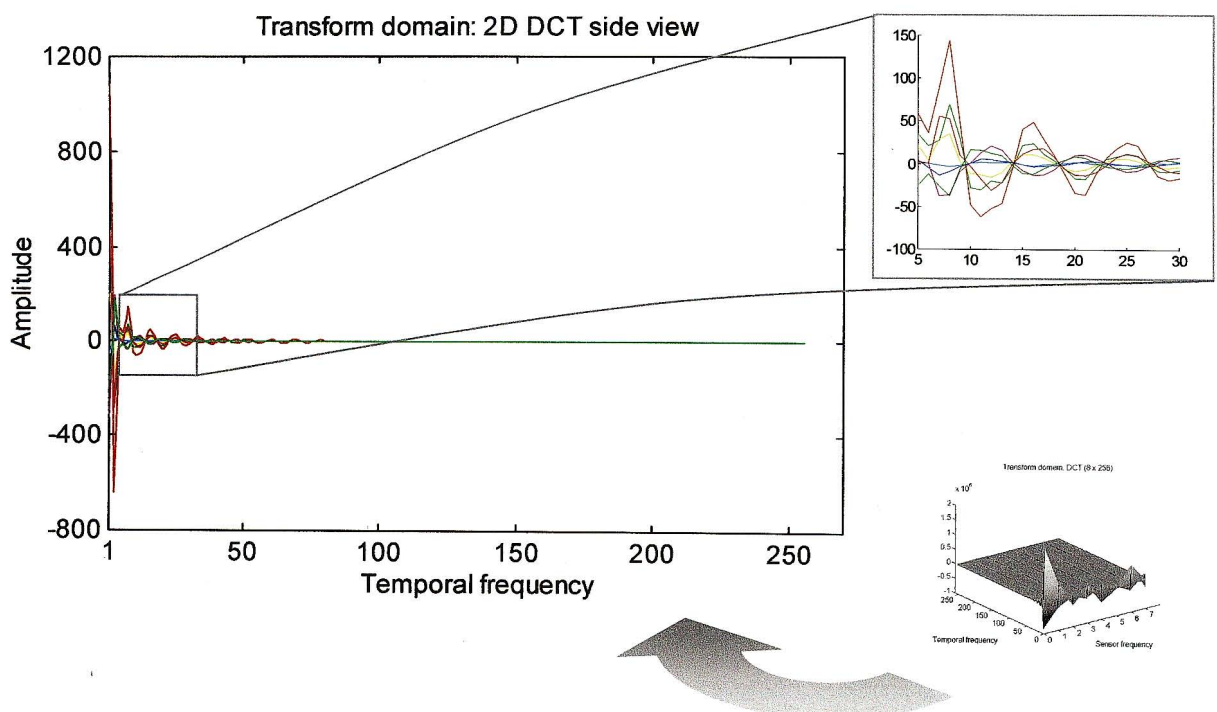


Figure 4.4: 2D DCT of measured data projected onto the temporal frequency axis

4.3 Transform coefficient selection

The underlying intentions in coefficient selection are simple:

- Pack as much data as possible into a few transform components, and then select a number of these high value components for classification purposes. This reduces the dimensionality of the classification problem.
- To facilitate classification by selecting those components (dimensions or bases) in which separation is less complex, in a selection process based on a statistical separability measure.

Basic theory: The DCT is called an orthogonal transform [64] because it produces a set of orthogonal basis vectors that span the transform vector space. That is, an arbitrary vector \mathbf{x} in the transform domain can be represented as a linear combination of transform bases x_1, x_2, \dots, x_n , e.g.

$$\mathbf{x} = \alpha_1 x_1 + \alpha_2 x_2 + \dots + \alpha_n x_n \quad (4.8)$$

The transform domain has a finite number of bases and is therefore a finite dimensional vector space. The dimensionality of the transform vector space V is equal to the number of bases. In this case,

$$\dim V = 8 \times 256 = 2048 \quad (4.9)$$

When a subset of the orthogonal bases is selected for further processing, a subspace H , with reduced dimensionality, is being selected. The classifier then classifies orthogonal projections of data from V onto H . Provided that the projections onto H encode substantial and significant information, it is assumed that classification results in the reduced vector space H may be generalised to the larger vector space V . It is further assumed here that classification in the subspace is simpler or more effective than in the full transform domain. The degree of accuracy in these assumptions may be deduced from the final classification results presented in Chapter 5.

The theory and assumptions discussed above led to the development of a signal processing regime. Figure 4.5 illustrates the signal processing pathway that generates various datasets along the way. These datasets are summarised in Table 4.1. The rest of Section 4.3 describes the reduced (coefficient selected) datasets and their associated selection methodologies. The dataset PUR is composed of all raw sensor measurements. These were transformed using the discrete cosine transform and saved as the DCT dataset. Subsequent selections were performed on the DCT dataset or derivatives thereof.

Table 4.1: Overview of datasets generated by the signal processing pathway

<i>Full (non-selected) datasets</i>		
Description		Dataset
Raw data as measured by the sensors		PUR
Discrete cosine transformation of the PUR dataset		DCT
<i>Frequency selected datasets</i>		
Description		Dataset
DCT dataset with all (eight) DC temporal components removed		D1C
Inverse discrete cosine transform of the D1C dataset		ID1
D1C dataset with high frequency components removed by a Tukey filter		D2C
Inverse discrete cosine transform of the D2C dataset		ID2
<i>Variance selected datasets</i>		
Ranking	Selection of ranked components	Dataset(s)
Variance	Top 2 to top 8 components (saved in different datasets)	D3C2 – D3C8
Separability indicator	Top 8 components only (do not span any axes)	D3CA
Separability indicator	Top 8 components that span the sensor frequency axis	D3CB

Data selection was divided into two categories viz. frequency based selection and variance based selection. Frequency based selection dealt primarily the removal of DC and high frequency components (discussed in Section 4.3.1 and 4.3.2). Variance based selection first ranked the bases according to variance or a separability indicator (details in Sections 4.3.3 and 4.3.4), and then selected from the ranked list.

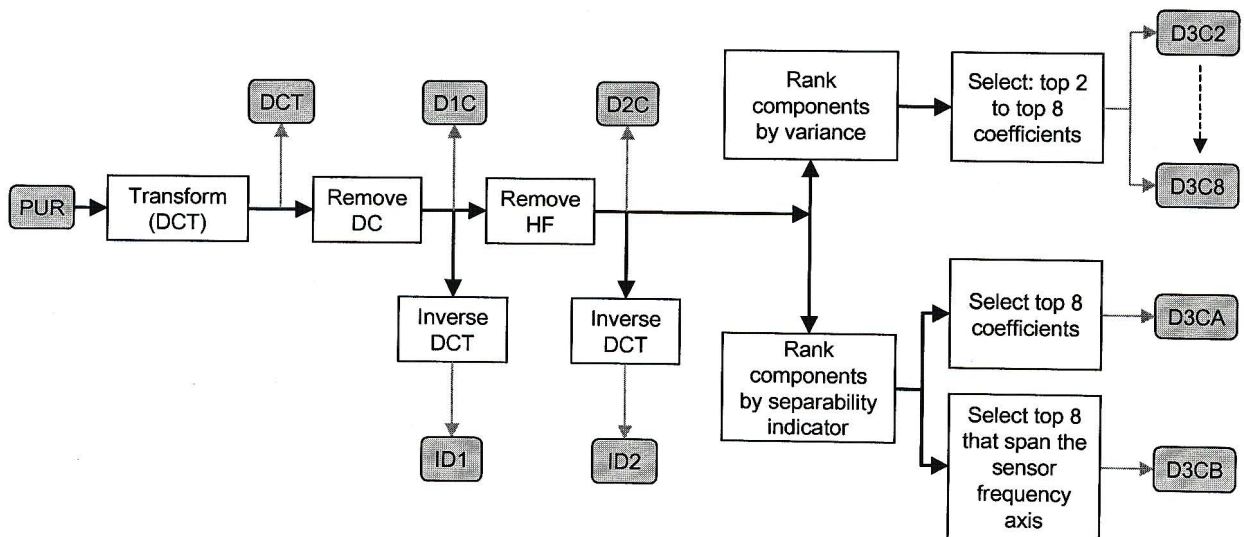


Figure 4.5: Signal processing pathway showing dataset generation

4.3.1 DC coefficient removal

Apart from their lack of selectivity, the most significant disadvantage with MOS sensors is sensor base-reference drift. Sections 3.2.3 and 3.2.4 described the underlying causes for drift and some of the counteracting measures that were employed during hardware development. Signal processing also plays a significant role in the stabilisation of sensory data.

Drift manifests as a modification to sensor base references in the absence of an odour stimulus. These base-reference variations are difficult to predict with regard to their duration and magnitude. Fortunately, short-term drift is minimised by a highly diluted headspace (Section 3.2.4), and long-term drift is expressed on a time-scale (days to months) that is large relative to the 256-second measurement window. It is therefore assumed that drift is expressed as a DC level shift in the time-scale of each measurement. The first coefficient selection strategy entails the removal of DC components. The intention is to preserve only the dynamic response of each sensor.

The component (0,0) in the transform domain is the DC component and represents the vertical displacement of the entire measurement matrix in the source domain. If this component is zeroed and an inverse transform taken, the resultant source domain data will show that:

- The 2D measurement matrix (such as that in Figure 4.1) retains its shape,
- The entire measurement matrix appears level shifted (vertically) such that the measurement matrix has a zero average, and
- Relative DC offsets between sensor channels are preserved within the matrix.

Although the matrix as a whole has a zero average, each channel taken separately may still have a non-zero average, and hence continue to encode the undesirable effect of drift. The ideal would be to extract the purely dynamic response of each sensor channel.

When using linear transforms the source domain signal is expressed as a linear combination of weighted transform bases (4.8). The DCT weighting factors (i.e. transform coefficients) are expressed in the DCT coefficient matrix (4.10). Some of the DCT bases are plotted in Figure 4.6.

$$\begin{array}{c}
 X = \left(\begin{array}{cccc}
 X_{0,0} & X_{0,1} & X_{0,2} & \dots & X_{0,255} \\
 X_{1,0} & X_{1,1} & & & \\
 \vdots & & & \ddots & \\
 X_{7,0} & X_{7,1} & & & X_{7,255}
 \end{array} \right) \begin{array}{l} \uparrow \\ \text{Sensor frequency axis} \end{array} \\
 \begin{array}{c} \rightarrow \\ \text{Temporal frequency axis} \end{array}
 \end{array} \tag{4.10}$$

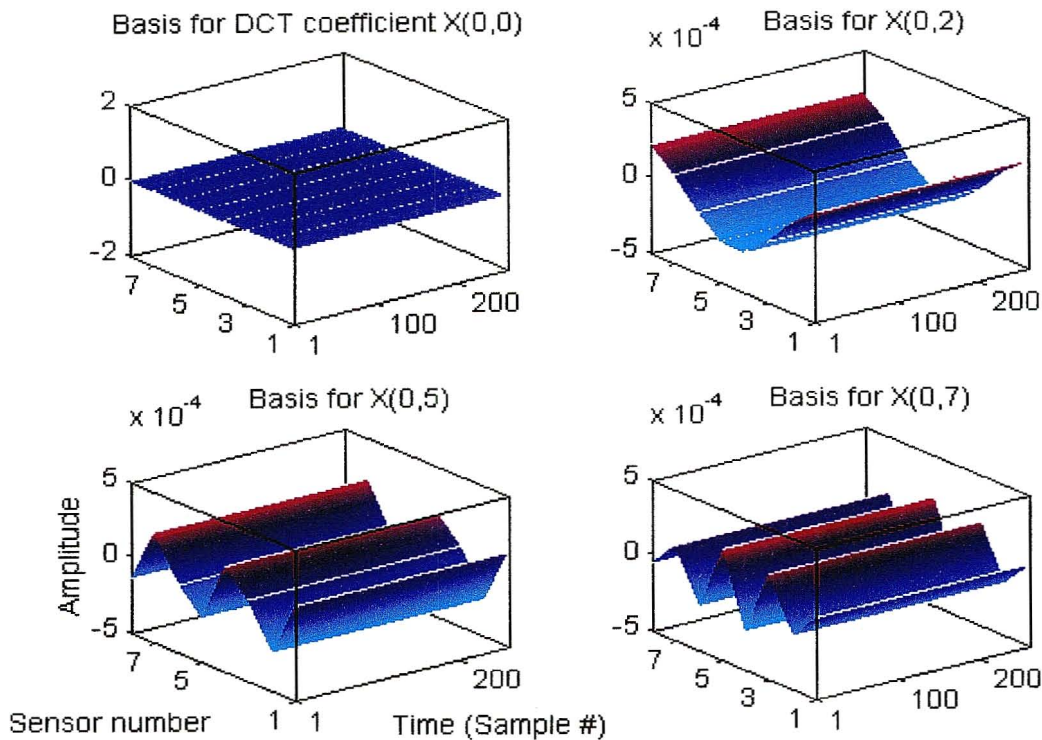


Figure 4.6: Source domain DCT basis functions corresponding to selected coefficients from the first column vector of the DCT coefficient matrix (Equation 4.10)

All components in the first column vector $\{X_{0,0} \dots X_{7,0}\}$ of (4.10) encode basis functions with zero derivatives in the source domain temporal dimension. That is, they represent constant level shifts over the time dimension only. The global DC basis, represented by the $X_{0,0}$ term is plotted in Figure 4.6. This basis has a zero derivative over both sensor and temporal dimensions. The bases represented by terms $X_{1,0}$ to $X_{7,0}$ (some of which are also plotted in Figure 4.6) vary over the sensor dimension but remain constant (DC) in the temporal dimension. In summary, all coefficients in this column vector represent bases that remain constant over time. When this entire column vector $\{X_{0,0} \dots X_{7,0}\}$ is zeroed in the transform domain, the signal in the source domain expresses the desired dynamic response. Figure 4.7 illustrates the source domain effect of this modification where individual sensor traces express purely dynamic responses with a zero average value for each trace.

Two new data sets were generated:

Dataset ID1

Description: The transform domain column vector $\{X_{0,0} \dots X_{7,0}\}$ is zeroed and a two-dimensional inverse DCT is taken. This extracts the source domain dynamic response curve.

Domain: This dataset stores modified source domain data.

Dimensionality: 2048 (Each measurement in the ID1 dataset has 8×256 values)

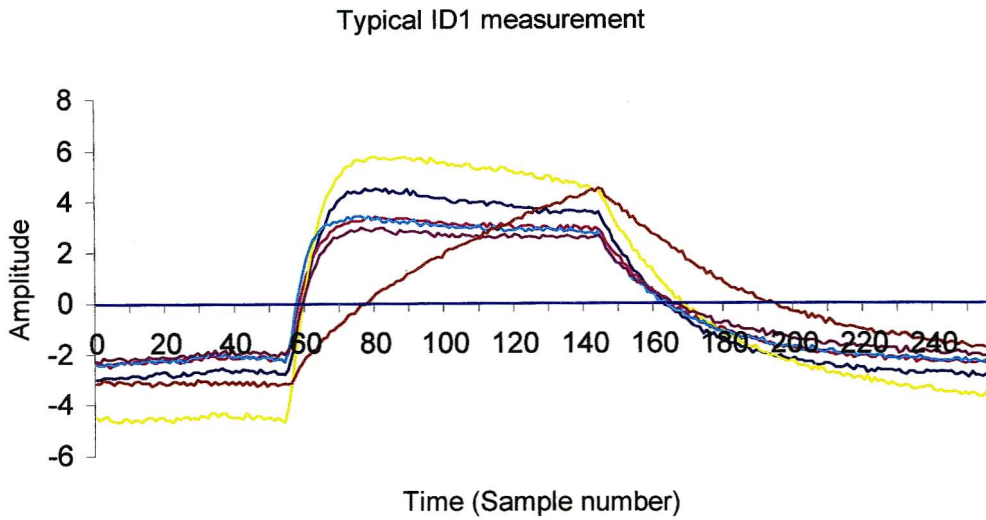


Figure 4.7: Typical source domain ID1 data showing the six sensor traces with the removal of all DC

Dataset D1C

Description: The transform domain column vector $\{X_{0,0} \dots X_{7,0}\}$ is removed.

Domain: Transform domain subspace

Dimensionality: 2040 (Each measurement in the D1C dataset has 8×255 values)

Significance of the modification: The sensor base reference or response to a stimulus-free environment is made up of two components.

$$\text{base reference} = \text{operating point} + \text{drift component} \tag{4.11}$$

Drift is a variable, age-related output and is not a function of present operating conditions. The

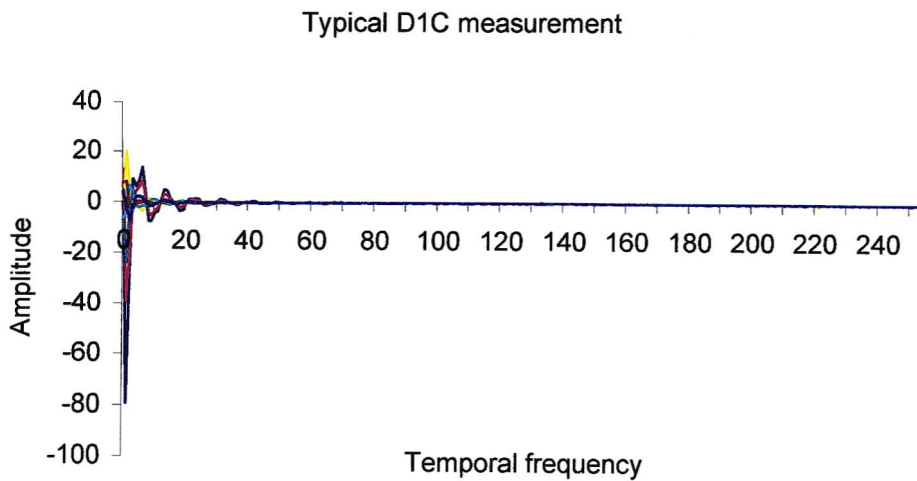


Figure 4.8: Typical D1C data viewed along the temporal frequency axis.

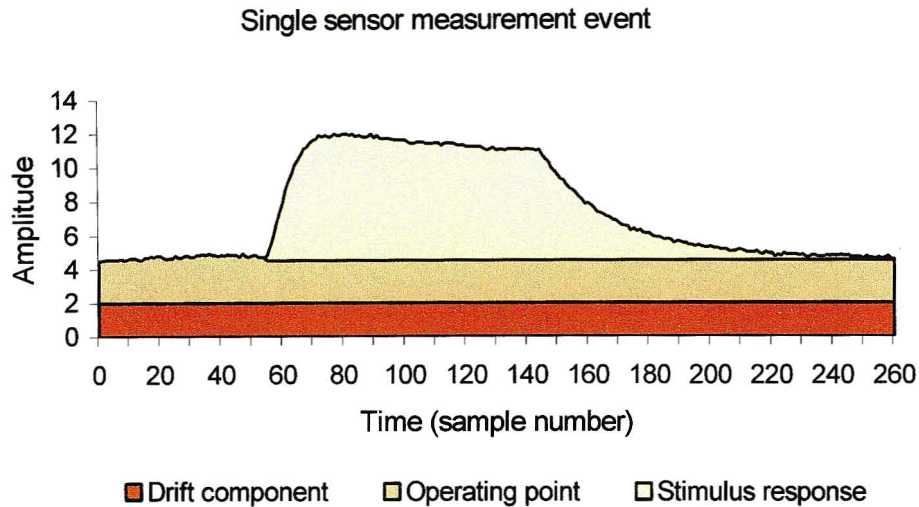


Figure 4.9: Sensor output is composed of stimulus response, operating point and drift.

operating point is the current response to pure carrier gas. In an ideal situation drift does not exist, and the base reference and operating point are equivalent. The operating point expresses potentially useful information that could assist in classification. When the base reference is measured it is impossible to determine (under present operating conditions) the separate magnitudes of drift and operating point. The unpredictable drift component is undesirable so the entire base-reference is removed, thus removing potentially useful operating point information.

As discussed earlier, DC drift is constant during the course of a single measurement but does vary over time causing unpredictable level shifts across measurements taken at different times. Independence from this type of drift provides a long-term benefit. Unfortunately drift is removed along with useful operating point information that encodes the sensor response to the stimulus free carrier gas. If accurately extracted, operating point data can potentially assist in the classification process. Therefore there exists a trade-off between short-term separability and long-term stability, the cost of which can only be discovered experimentally.

The data produced here is stored in two datasets, one in the source domain, and the other in the transform domain. Both datasets were applied initially to the classifier because it was not known which signal type was more conducive to separation by the neural network classifier.

4.3.2 High frequency removal

The information distribution illustrated in Figure 4.8 still retains the general form of the original DCT (Figure 4.4). As the high frequency temporal components are small they encode very little information. The removal of these components, which is common in image compression, is sometimes useful in classification applications as well. The justification, however, is different.

A generally accepted rule of thumb in neural network based classification states that more training data is required to characterise high dimensionality input spaces (measurements). This requirement is further strengthened when the data is noisy or when the measurements across categories are highly correlated. It has already been established (in Chapter 3) that MOS sensor arrays are highly correlated. It would therefore seem logical that the reduction of input dimensionality via the removal of coefficients with low information content would benefit classification. In this way, a significant reduction in dimensionality can be achieved with minimal information loss.

The DCT (Figure 4.4) and D1C (Figure 4.8) distributions exhibit low information content in the high frequency temporal components. An analysis of area under the transform domain curve of a typical measurement showed the following. Consider the high temporal-frequency subspace beyond temporal component 50.

$$\mathbf{X} = \begin{pmatrix} X_{0,0} & \dots & X_{0,50} & \boxed{X_{0,51} \quad \dots \quad X_{0,255}} \\ \vdots & & & \vdots \\ X_{7,0} & & X_{7,50} & \boxed{X_{7,51} \quad \dots \quad X_{7,255}} \end{pmatrix} \quad (4.12)$$

\uparrow
 High temporal frequency subspace

The high temporal-frequency subspace indicated in (4.12) contains 1640 coefficients. This accounts for more than 80% of the 2048 DCT coefficients, yet it contains approximately 7% of the total area under the curve. Deletion of this subspace results in an 80% dimensionality reduction with a loss of 7% of total information. The benefit, if any, of this trade-off would be evident in the classification results (Chapter 5).

The removal of high frequency components was effected by the application of a Tukey window in the transform domain. The Tukey window function (4.13) produces a symmetric window centred on zero temporal frequency. Figure 4.10 illustrates the modified Tukey window as it was implemented in this study. Filter coefficients were adjusted to produce a total window length of 50 samples in the positive temporal frequency domain with cosine roll-off occupying the last 25% of the window. For completeness, the window is zeroed at zero temporal frequency to prevent the passage of DC. Note

Tukey window
 $\alpha = 0.75$ $N=100$

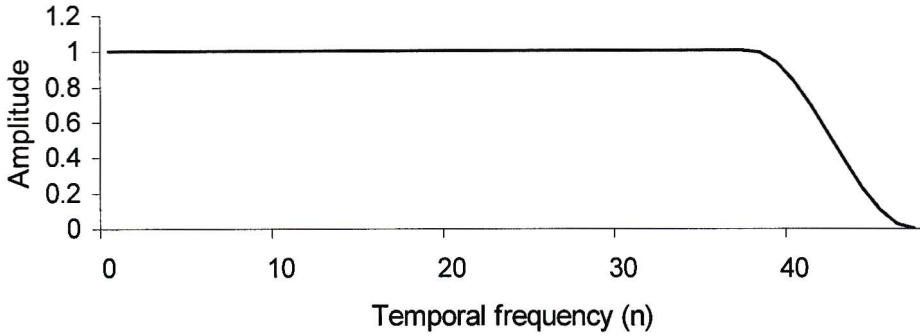


Figure 4.10: Tukey window used to remove high temporal frequencies

that temporal frequency DC components have already been removed by this stage. The window is extended across the channel component axis.

$$w(n) = \begin{cases} 1.0, & 0 \leq |n| \leq \alpha N/2 \\ 0.5 \left[1.0 + \cos \left(2\pi \frac{|n| - \alpha N/2}{2(1-\alpha)N/2} \right) \right], & \alpha N/2 \leq |n| \leq N/2 \end{cases} \quad (4.13)$$

n Sample number
 N Window length
 α Roll-off commencement $0 \leq \alpha \leq 1$

Two new data sets were generated:

Dataset ID2

Description: The transform domain column vector $\{X_{0,0} \dots X_{7,0}\}$ and high temporal-frequency subspace are zeroed. A two-dimensional inverse DCT is then taken. Figure 4.11 illustrates a typical ID2 curve. The signal is smoothed by the removal of high frequency components.

Domain: This dataset stores modified source domain data.

Dimensionality: 2048 (Each measurement in the ID2 dataset has 8×256 values)

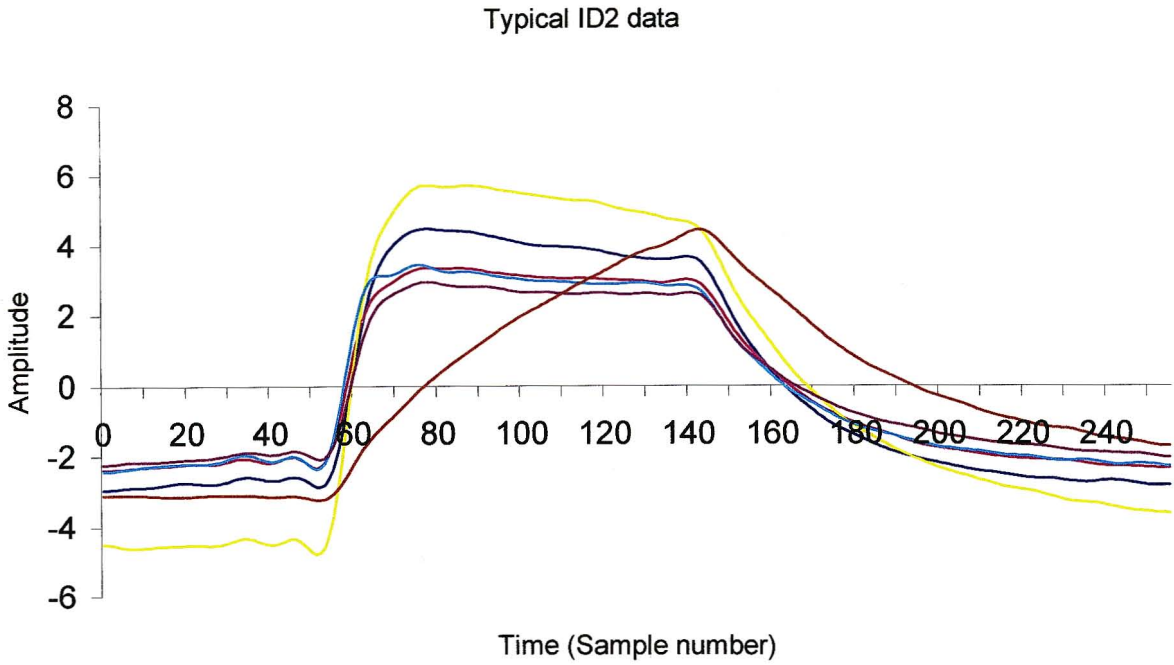


Figure 4.11: Typical source domain ID2 data

Dataset D2C

Description: The transform domain column vector $\{X_{0,0} \dots X_{7,0}\}$ and high temporal-frequency subspace are removed. Figure 4.12 illustrates a typical D2C curve.

Domain: Transform domain subspace.

Dimensionality: 392 (Each measurement in the D2C dataset has 8×49 values)

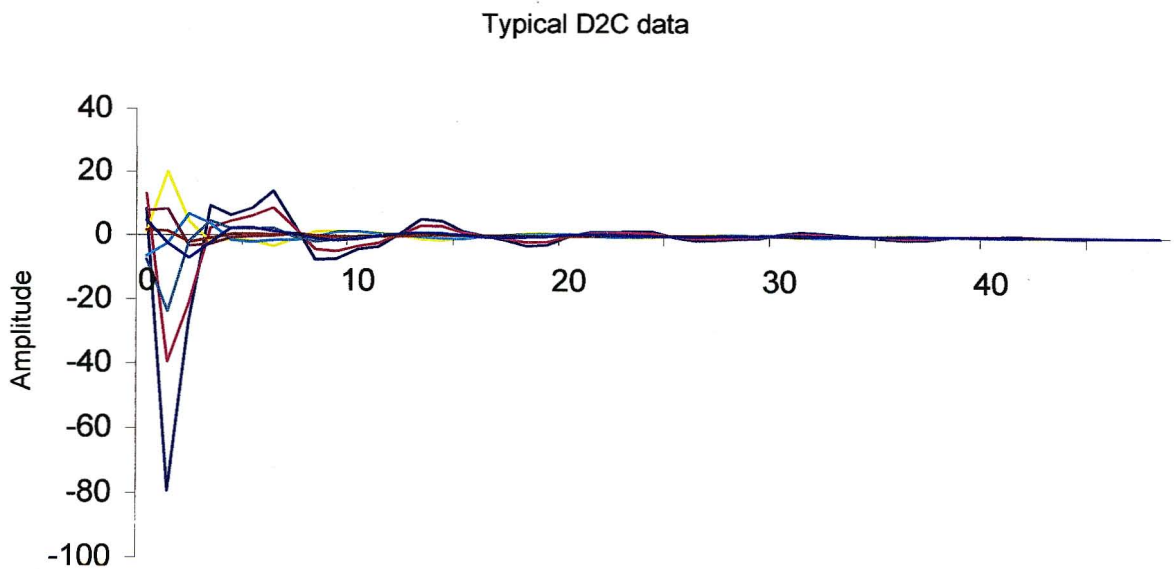


Figure 4.12: Typical transform domain D2C data illustrating the removal of DC and HF components

4.3.3 Simple variance-based selection

Sections 4.3.1 and 4.3.2 discussed selection strategies that were based on frequency. The filtering out of specific frequency bands is common practice in the fields of digital and analogue communications where it is often used for bandwidth and noise reduction. The advent of digital image processing reinforced the need for efficient data compression. It also introduced the notion of data selection based on coefficient variance.

H.C. Andrews proposed variance-based selection in his 1971 paper “Multidimensional rotations in feature selection” [65]. The technique is based on a statistical analysis of transform domain information distribution. Given a set of transform domain measurements, the covariance matrix Ψ can be computed. The main diagonal of this matrix indicates the variances of the transform coefficients. Andrews proposed that those elements with the largest variances be retained.

The question arises, what do these retained coefficients represent? It is widely believed that main diagonal coefficients with high variances encode high levels of information and are therefore incorporated into some image compression schemes. This belief is not strictly true. Retained coefficients encode high levels of information variance or deviation, which differs from an absolute indication of information content. This type of information is very likely to benefit classification, which seeks those components with high spread (variation) in the pattern space. Provided that data is separable, the high variance (spread) in pattern space allows more flexibility for the formation of decision boundaries.

New variance-selected data sets were generated as follows. The covariance matrix for D2C data was computed. Main diagonal elements were then ranked in decreasing order of variance. The highest ranked coefficients were selected and retained. There was no clear indication of the number of coefficients that should be retained. A leading authority in electronic olfaction, Dr. Julian Gardner (University of Warwick), published works where the optimal KLT was used as the orthogonal transform. In some investigations data could be separated with as few as two principle components being preserved [42]. The DCT was not expected to perform as well as the KLT, so it was anticipated that more components would be required. A range of datasets was produced with each dataset retaining a different number of coefficients. The smallest dataset retained two components and the largest retained eight.

Datasets D3C2 to D3C8

Description: The top 2 to top 8 variance ranked components are retained in their respective datasets. All other components are eliminated.

Domain: Transform domain subspaces.

Dimensionality: Dimensionality ranges from 2 to 8.

4.3.4 Coefficient selection heuristic

The coefficient-selected data sets considered thus far were all generated by traditional feature extraction methodologies. In this section, a modified technique is developed that addresses some of the limitations of the previously considered techniques. As a starting point, the fundamental requirements of classification are considered and a selection heuristic is evolved that attempts to satisfy those requirements. A heuristic approach may be defined² as any technique that uses knowledge of the problem to improve the average performance without necessarily influencing the best or worst case scenarios [66].

So what are the fundamental requirements of successful classification? Given that a set of data is separable into distinct classes, the two fundamental requirements that facilitate separation are:

- Individual instances of a particular class must be closely clustered around the class centroid in the feature space.
- The distinct class centroids must be spread as far apart as possible in the feature space.

A feature space is any domain that describes features of the data on orthogonal axes. During training, the classifier evolves boundaries between class clusters in feature space, such that the individual classes are separated and made distinct by these boundaries. Tightly clustered classes with highly separated centroids give the classifier more space in which to develop its decision boundaries.

The aim of the heuristic is to identify those coefficients that best satisfy the two classification requirements described above. The selection strategies considered thus far eliminated the following coefficients:

- DC components which are influenced by drift of the sensors,
- HF components which possessed low information content, and
- Components that expressed small variance.

D2C coefficients are now re-evaluated in terms of the two classification requirements.

² The meaning of the word “heuristic ” has changed over time. The current definition in the machine intelligence community goes beyond the traditional association of heuristics with “rules of thumb”.

It is traditionally assumed that components with high variances represent spread-out data and should be more separable [65,58]. This is not always applicable. In a classification problem, variance alone is not necessarily an adequate measure of a coefficient's usefulness. In natural data, coefficients with high average values will usually exhibit high variances and vice versa. In mathematical terms, the variance of a component is often proportional to its mean. This exposes a possible scaling problem where components with lower means are less likely to be preserved even if their variances are large relative to their means.

The coefficient selection heuristic extends variance-based selection and addresses the limitation in that approach when applied to classification problems. There exists at least one case where the variance-based selection fails. Consider Figure 4.13.

Both distributions (Figure 4.13a & 4.13b) will produce the same global³ variance (Figure 4.13c) even though one distribution is clearly more separable. A dilemma is consequently introduced when interpreting separability according to global dataset variance. That is, if an arbitrary coefficient n is considered, a large global variance could indicate either that:

- The categories are distinct and far apart as in Figure 4.13a (at one extreme), or the
- Categories are spread out and overlapping as in Figure 4.13b (other extreme).

Large global variance is not a good basis on which to infer the degree of separability. Additional information is required to resolve the dilemma.

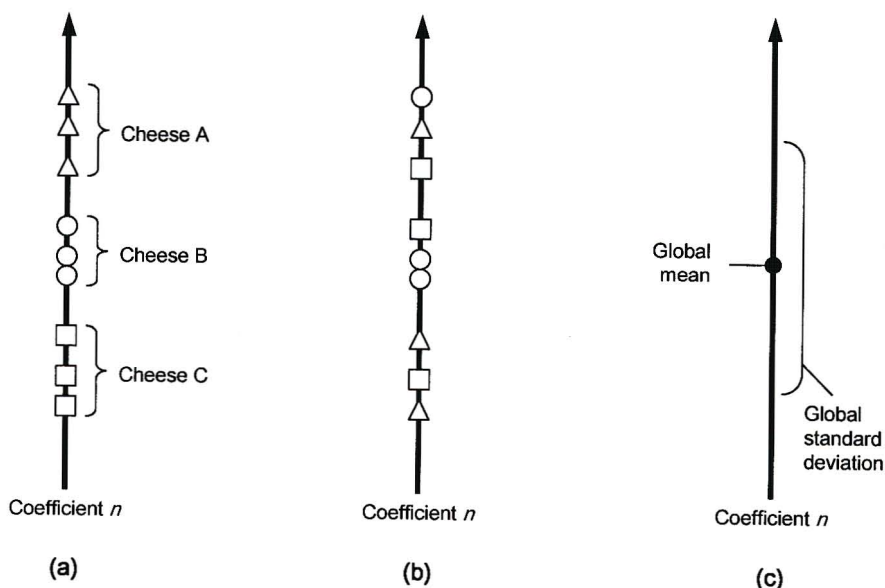


Figure 4.13: Separable (a) and inseparable (b) class distributions can produce the same variance (c)

³ Global variance refers to the variance across the entire dataset irrespective of class memberships of individual measurements.

For the sake of simplicity, let us analyse the idealised scenario depicted in Figure 4.14, which illustrates the orthogonal projection of a separable dataset on one dimension in feature space. The intention is to determine if this dimension (or basis and coefficient) should be retained or discarded. The illustrated data spans three classes that are clearly separable in the chosen dimension because:

- The classes are distinct and tightly clustered, and
- Individual clusters are far apart.

These observations may be expressed in terms of statistical means and standard deviations as follows. A set of representative training data, for which all class memberships are known, is projected onto a single dimension. The arithmetic mean for each class is calculated in this dimension using (4.14). The class means locate the centroid of each class in the current dimension (Figure 4.14c). The standard deviation for each distinct class is calculated in a similar fashion using (4.15). The class standard deviations give an indication of the compactness of each class in the current dimension (Figure 4.14c).

Consider the set of projected class A measurements $\{x_1^a \ x_2^a \ \dots \ x_k^a\}$

The mean of class A in the current dimension is

$$\bar{A} = \frac{1}{k} \sum_{i=0}^k x_i^a \tag{4.14}$$

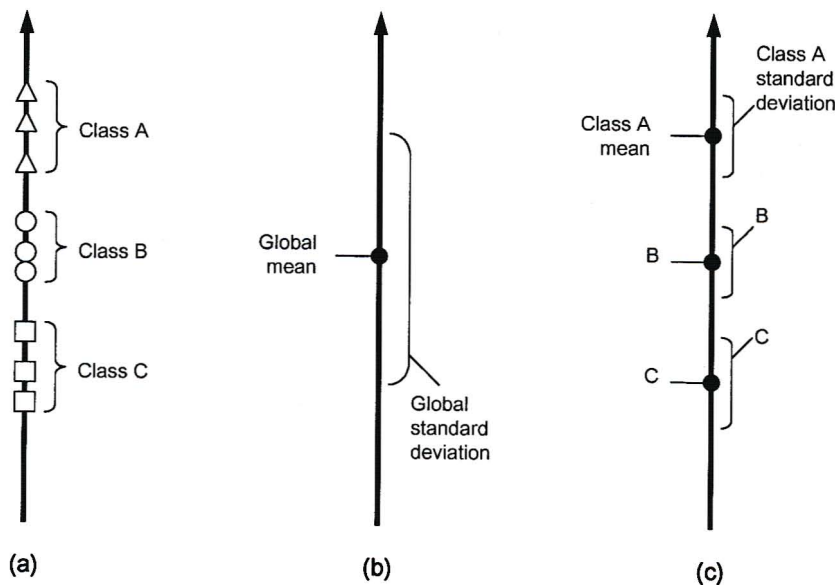


Figure 4.14: Calculation of individual class statistics

The sample standard deviation of class A in the current dimension is

$$\sigma_A = \sqrt{\frac{\sum_{i=0}^k (x_i^a - \bar{A})^2}{k-1}} \quad (4.15)$$

By applying these calculations to all classes in the dataset, a set of class means (CM) and a set of class standard deviations (CS) are generated.

$$\begin{aligned} \text{CM} &= \{\bar{A} \quad \bar{B} \quad \bar{C} \quad \dots\} \\ \text{CS} &= \{\sigma_A \quad \sigma_B \quad \sigma_C \quad \dots\} \end{aligned} \quad (4.16)$$

By finding the mean of CS, single measure is established, that provides an indicator of general class compactness in the selected dimension. The smaller the value of $\overline{\text{CS}}$ the more compact the classes.

Similarly, by finding the sample standard deviation of CM, a single figure σ_{CM} is established, that represents the spread of the category centroids. The larger the value, the more spread out the class centroids will be.

Finally, the two measures are combined to produce a single indicator of separability (S) in the current dimension.

$$S = \frac{\alpha \sigma_{\text{CM}}}{\overline{\text{CS}}} \quad (4.17)$$

α is a constant of proportionality. The separability measure is biased in favour of class compactness for $\alpha < 1$, and for $\alpha > 1$ it will favour centroid dispersal as an indicator of separability. In this study α was set to 1, equally weighting compactness and centroid dispersal.

A new data set was generated much like the D3C2 – D3C8 datasets, however, components were ranked according to the separability indicator S.

Dataset D3CA

Description: D2C components are ranked according to the separability indicator (equation 4.17), and the top 8 components are retained. All others are eliminated.

Domain: Transform domain subspace.

Dimensionality: 8 components.

The final step in the heuristic's development was based on a brief analysis of information distribution in the transform domain. Figure 4.2 clearly shows that information content diminishes significantly as

one proceeds along the temporal frequency axis towards the high temporal frequency components. The same cannot be said for the sensor frequency axis. Some degree of reduction is generally evident towards the high sensor frequency components, however, the reduction is not as consistent or pronounced and the level of information remains significant across the axis.

This is not surprising when one considers that the sensor frequency axis encodes the responses of differentially selective sensor channels, and selectivity differentiation across the sensor array is the principal means of odour separation. Chapter 2 explains that the temporal dimension was not expected to produce as much value as the spatial (sensor) dimension due to the lack of a gas separation column. This expectation is validated by the transform domain information distribution.

It was therefore decided to bias the coefficient selection mechanism in favour of the sensor frequency axis. This is achieved by choosing the eight highest separability-ranked components such that they span the sensor frequency axis. A new dataset, which may be viewed as a modification of the D3CA dataset, was generated.

Dataset D3CB

Description: D2C components are ranked according to the separability indicator (equation 4.17), and the top 8 components that span the sensor frequency axis are retained. All others are eliminated.

Domain: Transform domain subspace.

Dimensionality: 8 components.

4.4 Preliminary classification results

Section 4.3 describes the rationale and methodology employed in the development of a signal processing pathway. It should be noted here that the process was not conducted in an entirely open loop fashion. Sample measurements were taken upfront and subjected to the signal processing mechanism as it developed. New datasets were produced and classified by artificial neural networks, which is explained in greater detail in Chapter 5. Classification results were used as indicators of the potential usefulness of the signal processing to that stage. This effectively closed the loop and assisted in the further refinement of the signal processing methodology.

This study de-emphasises the role of these interim results for good reason. The sample dataset was captured on old sensors prior to the final hardware deployment. Furthermore, given that the

measurement set was small and not representative of the general signal statistic, the outcomes would at best, provide only weak indicators of performance. The temptation to extract deep significance from preliminary data such as this must be avoided. This is a well-established principle in the field [67]. Nonetheless, the interim results (Table 4.2) provided a measure of confidence in the developing strategy.

Table 4.2: Summary of results

<i>Dataset</i>	<i>Classification rate</i>	<i>Training time hh:mm:ss</i>	<i>Required hidden layer size</i>
PUR	97.77%	17:01:22	200
D1C	96.88%	23:02:17	200
D2C	99.11%	10:20:42	500
D3C8	35.27%	00:03:13	40
D3CA	35.27%	00:16:00	175
D3CB	99.11%	00:00:36	12

The classification rate is defined as the percentage of measurements that are correctly classified and the hidden layer size is a measure of the complexity of the artificial neural network that was used. These issues are explained in greater detail in Chapter 5. For the moment, three criteria are considered with respect to the classifier:

- Optimal hidden layer size must be found experimentally, and smaller sizes are preferable,
- Training time must be minimised, and
- Classification rate must be maximised.

The D3CB dataset, as produced by the final selection heuristic, produced the highest classification rate with the smallest hidden layer size and the shortest training time.

Other datasets did not perform as well:

- Low classification rate: D3C8 and D3CA
- Large hidden layer size: D2C
- Long training time: PUR, D1C and D2C

Full datasets as well as explanations for the classification results are considered in Chapter 5.

4.5 Conclusion

The signal processing methodology proposed in this chapter was developed for the specific classification problem under consideration. The strategy proposed is related to and derived from existing signal and image processing methods, however, there is a major departure with regard to intention or purpose. In image processing, coefficient selection is usually implemented with the

intention of compressing and subsequently restoring the data stream with minimum information loss. In classification problems, the burden of restoration is removed. This chapter proposed that it is possible to select a feature subspace (i.e. a subset of an orthogonal feature space) in such a manner that classification results within that subspace may be generalised over the original feature space. The technique was developed such that the classification problem is simpler in the subspace and yet remains representative of the larger more difficult problem. Unlike image compression where selection methods are guided by the need to restore, in this approach they are guided by the need to remain representative of the larger problem.

Significant benefits included a dimensionality reduction factor of 256 (decrease from 2048 to 8 coefficients), and preliminary results indicated a significant reduction in classifier size and training time while exceeding the classification rate for the original PUR dataset.

This chapter proves by way of example that “informed” dimensionality reduction alone could greatly improve classification results. D3CB data is a pure subset of the datasets DCT, D1C and D2C. That is, the improvement was achieved only by modifying the selection mechanism while preserving exactly the retained coefficients. This result shows that retention of only those dimensions that are known to encode the most separable data can improve classification performance.

CHAPTER 5

CLASSIFICATION BY NEURAL NETWORK

5.1 Introduction

Chapter 4 presented the evolution of a signal processing methodology, dataset generation and preliminary classification results. This chapter discusses the final experiment, and takes a deeper look at selected concepts and principles related to classification by artificial neural networks. A full dataset is captured and final classification results are presented.

After construction of the system hardware, new sensors were installed. These were powered up and allowed to stabilise for three weeks according to the manufacturer's instructions. The system remained powered up for the duration of the experiment and a backup power source ensured that sensors remained energised continuously.

The full dataset was captured over a period of three months. Data was archived and processed offline. The various datasets described in Chapter 4 were then assembled into training, validation and test sets for the neural network classifier. This chapter describes the configuration, control and outcomes of the classification process.

5.2 Software environment

Many different types of artificial neural network classifiers exist. The most commonly used classifier is the multilayer feed-forward network, which is usually trained by the back-propagation algorithm or some variant thereof. This investigation employed the prolific back-propagation neural network paradigm. A single hidden layer network is used, as it is known to approximate any continuous function [68]. The theory of back-propagation [69] is well established and easily accessible and is not discussed in any detail here. Only selected issues pertaining to training configuration are considered where appropriate.

5.2.1 Stuttgart Neural network Simulator

All commercial and public domain neural network simulators support some form of back-propagation algorithm and multi-layer feed-forward neural architecture. However, implementation details and configuration facilities vary greatly across simulators. The Stuttgart Neural Network Simulator (SNNS) version 4.1 for Linux [70] was selected for this study. It offered the necessary neural network implementation, performance and flexibility.

5.2.2 Neural network configuration system

A total of 700 cheese odour measurements were taken across 7 distinct categories. Table 5.1 shows the cheese categories and the respective number of measurements taken. The measurement quantities were influenced by sample availability during the three-month test period. It is a basic requirement that classes are uniformly represented in the training set. The most under-represented category (FVlaberyl) therefore dictated that the final number of samples used per category be restricted to 80.

Table 5.1: Odorant categories

<i>Cheese</i>	<i>Category label</i>	<i>Measurements</i>	<i>Used</i>
Fairview Brie with tomato & basil	"FVbrietb"	100	80
Fairview Brie	"FVbrie"	100	80
Fairview Camembert	"FVcamembert"	100	80
Fairview Laberyl	"FVlaberyl"	80 ¹	80
Fairview Chevin with sweet red pepers	"FVchevinsrp"	100	80
Fairview Blue Rock	"FVbluerock"	100	80
Simonsberg Traditional Mozzarella	"SBmozza"	120	80

Only the first 80 measurements of each cheese category were used. These measurements were taken in a fixed sequence as indicated by the measurement script in Chapter 4. Fairview Laberyl was the first cheese to run out. All measurements taken after that point were not used in this study. The final database of 560 distinct raw measurements was processed into the various datasets as described in chapter 4. Given the large number of separately stored raw and transformed measurements, it is useful to review the naming convention for measurement files. Each measurement or transformed measurement is stored to a separate file that is named according to the following convention:

¹ The lower measurement count was related to the commercial availability of "Fairview Laberyl"

004_XXXX.YYY

where:

004_ represents the experiment number. Experiments 001 to 003 belong to the system test and commissioning phases, and are not discussed in this thesis. This study deals exclusively with cheese data captured in experiment 004.

XXXX refers to the measurement increment number. The first measurement starts at increment 0001.

YYY refers to the data-type or dataset. These include *raw*^{*}, *pur*^{*}, *dct*, *d1c*, *d2c*, *id1*, *id2*, *d3c*^{**}.

Note:

^{*}The *pur* dataset is simply a reformatted version of the original *raw* measurement files. Actual measurement data remain unchanged across these file types.

^{**}Datasets D3C2-D3C8, D3CA & D3CB all use the *d3c* file extension. In order to prevent confusion, datasets are separated into appropriately named directories. When dealing with any *d3c* file, one must consider the source directory for that file.

The set of raw measurements are made available on the accompanying CD in “SUPPORT_CD:\measurements\raw”. Raw measurements were processed into the various datasets as described in Chapter 4, and the datasets were assembled into pattern sets for the neural network training process.

Pattern sets: The word “pattern” is used by SNNS to describe the basic unit of training or testing data. A pattern is composed of two parts,

- The “input pattern” or “input vector” which is applied at the neural network input, and
- The corresponding “output pattern” or “output vector” which represents the desired optimal output given the current input.

A pattern set is a collection of several patterns in a single file, which may be used for training or testing the neural network. The 560 transformed measurement files in each dataset were reformatted into (560) patterns and assembled into appropriate pattern sets. For example, Figure 5.1 illustrates the header and first two patterns from a pattern file in the D3CB dataset.

Each output vector indicates the class identity of the input vector. Note that the output dimensionality of 7 corresponds to the 7 cheeses, and the input dimensionality of 8 is determined by the D3CB specification. The neural network learns the input-output relationship from these patterns. Four distinct pattern sets were generated for each dataset. These are described in Table 5.2.

SNNS pattern definition file V1.4
generated at Fri Mar 24 12:57:23 2000

No. of patterns : 560
No. of input units : 8
No. of output units : 7

Output vector format for this pattern file.....

0 FVchevinsrp
1 FVbrie
2 FVbrietb
3 FVbluerock
4 SBmozza
5 FVcamembert
6 FVlaberyl

***** pattern 1 *****

#input vector 1
#File: c:\mydocu~1\exp4\004_0560.d3c
#Category: FVchevinsrp
15.808782 -19.079769 16.483618 -2.622664 -39.133820 13.431553 16.924824 20.718163
#output vector 1
1 0 0 0 0 0

***** pattern 2 *****

#input vector 2
#File: c:\mydocu~1\exp4\004_0002.d3c
#Category: FVbrie
1.111362 13.482435 -0.468171 -7.890170 -10.961367 -16.139294 10.383916 2.977654
#output vector 2
0 1 0 0 0 0

Figure 5.1: Partial listing of a pattern file including the header and first two patterns

Table 5.2: Pattern sets used in training and testing of neural networks

<i>Pattern set</i>	<i>Number of patterns</i>	<i>Use</i>
Full	560 – all patterns	Testing only (not very useful, see discussion)
Training	168 – unique 30% of full set	Training only (used to teach the neural network)
Validation	168 – unique 30% of full set	Validation only (used to control training)
Test	224 – unique 40% of full set	Testing only (unseen data for performance appraisal)

Given any dataset (PUR, DCT, D1C, D2C, ID1, ID2, D3C2-D3C8, D3CA or D3CB) of 560 appropriately transformed patterns, a random selection of 168 patterns (24 per category) is stored in the training set. The pattern set generator, for which the source code is provided in RESULT_CD:\configuration\software\preprocessors\patgen.c, was written such that the random number generator seed could be specified and the same random selection could be made across all datasets. That is, all training sets contained the same appropriately transformed patterns in the same sequence. Validation and test sets were generated in a similar fashion. The full pattern-set is the combination of the training, validation and test sets.

Automated training: An automated script driven training system was developed due to the large number of datasets and the need to train multiple neural networks in the search of an optimal classifier in each dataset. The collection of training scripts may be found on the attached RESULT_CD. Consult the file “RESULT_CD:\configuration\training_config.txt” for further information.

Schedule files: Training schedule files were used to exert broad control over the whole training process and various scripts were used to implement subtasks. Each dataset has its own training schedule file. Consult the file “RESULT_CD:\configuration\training_config.txt” for further information.

Each record (row) in a schedule file contains the necessary information to train a single neural network. The significance of each field (column) is explained in the schedule file header. The training script “RESULT_CD:\configuration\training_scripts\batsh” extracts each schedule record in turn and trains the neural network according to the settings stipulated in the record. Figure 5.2 describes the algorithm and organisation of this high-level script. The scripting engine was developed as part of this study. In order for it to work, minimal source code modification to SNNS and subsequent recompilation was necessary. “RESULT_CD:\configuration\software\simulator\installation.txt” contains further details in this regard.

During training, the training set is processed repeatedly. Each pass through the training set is referred to as an epoch. The neural network weights are adjusted at the end of each epoch so that the network improves its approximation of the input-output relationship that is expressed in the training set. One must strive to ensure that the training set is representative of the general class of data, or signal statistic from which it is derived.

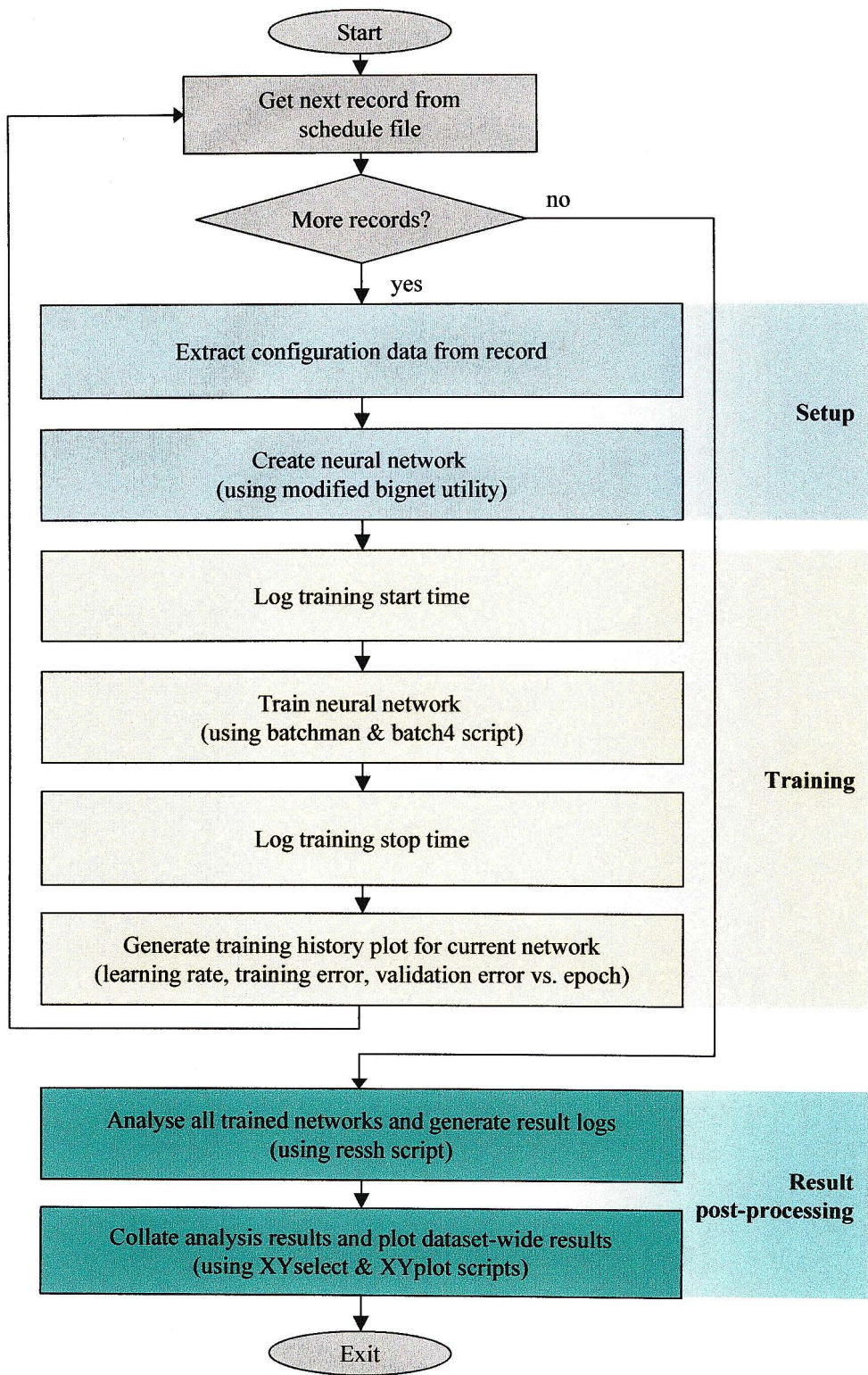


Figure 5.2: High-level training script (batsh) that processes all networks in a training schedule file

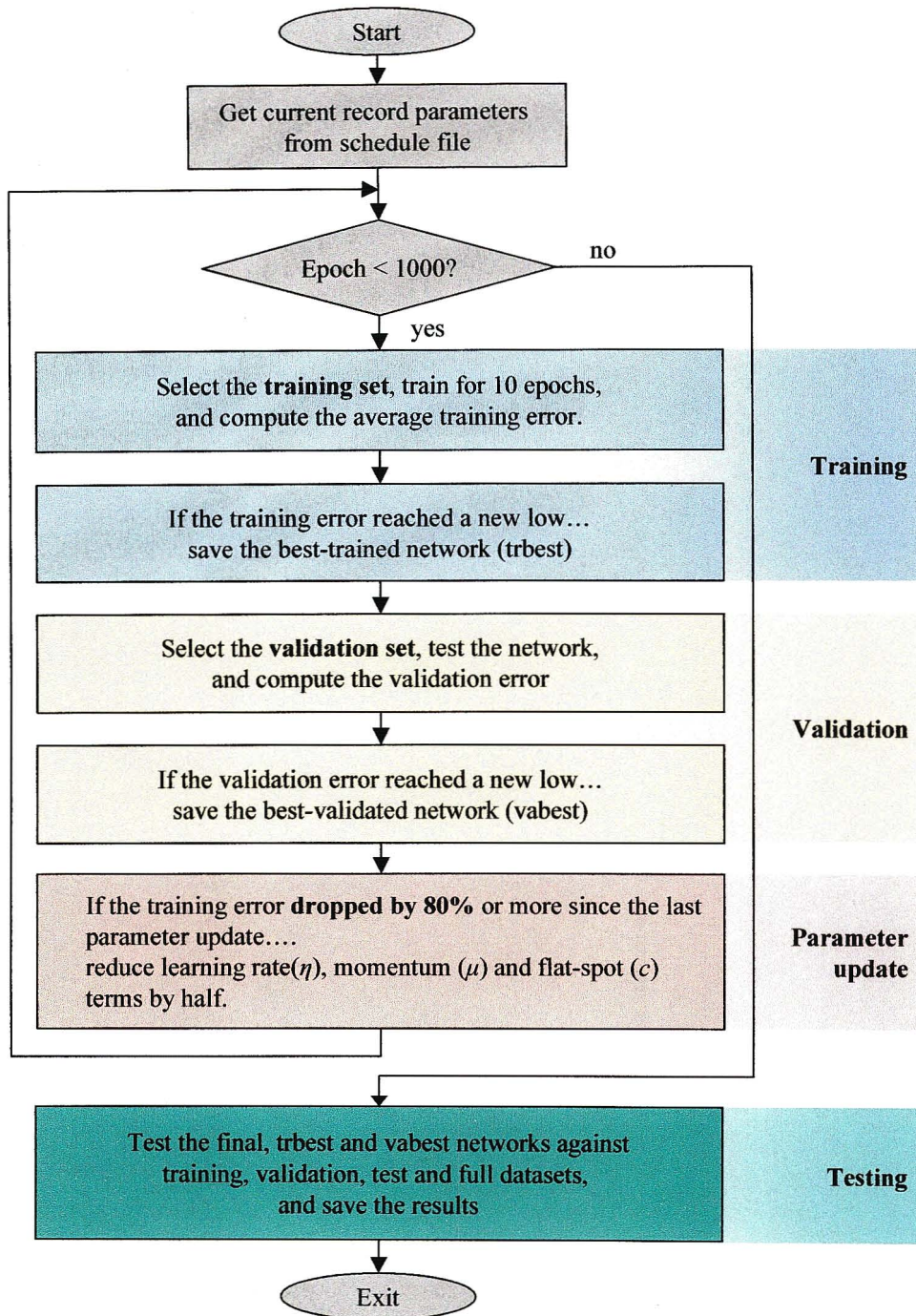


Figure 5.3: Low-level training algorithm (batch4) that is invoked (by batsh) to train a network

During the iterative processing of training data, the network develops a model of the general class of data from which the training set was derived. This is called generalisation, which is the network's ability to learn a general trend from specific data.

When assessing the quality of this learned generalisation, it is necessary to test the neural network with a separate test set that is produced from the same general class of data that produced the training set, such that the two sets have no intersection. Any intersection of test and training data will skew and invalidate the test result because the network would have had the opportunity to learn the intersecting portion of the test set during training.

A further pattern set (validation set) was used to determine when to stop training. Excessive training results in a phenomenon called over-training which results in a loss of generalisation. That is, after extracting the desired general signal characteristic, the learning process continues to the point that the network learns the exclusive specifics of the training data. In this scenario, the neural network is over-optimised for the specific training data. The over-trained network can therefore classify training data very accurately but usually performs poorly on other (validation or test) data even though such data is derived from the same general class of data as the training set. As training proceeds, the validation error (classification error for validation data) drops, but when over-training commences, the validation error rises again and diverges from the declining training set error. This divergence is used to detect over-training.

Validation process: Training is stopped at regular intervals (every 10 epochs) during the training phase. At each instance, the partially trained network is tested against the validation set. Every time the validation error reaches a new low, the network is saved as the best-validated network. When training eventually stops, it is the best-validated network that is regarded as the end product. The validation set may be regarded as a second test set, and the validation error as an auxiliary test error. In the event of over training, validation errors will increase above the recorded minimum and the network will not be saved. The temptation to use the test set for validation purposes must be avoided. For valid test results, the test set should have no influence on the training process. This includes determining when to save the network or stop training.

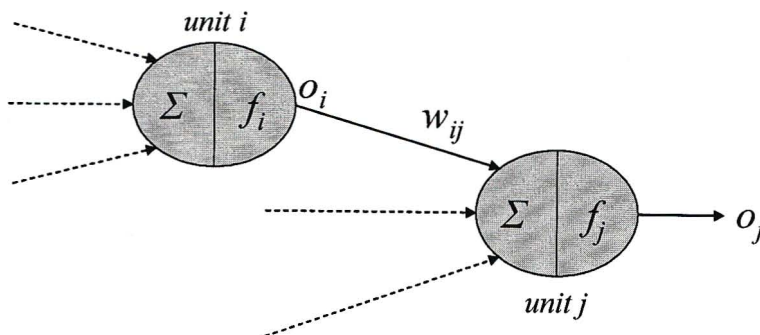


Figure 5.4: Two connected units (“neurons”) in a feed forward neural network

Parameter control: There are several variants of the backpropagation algorithm. This study makes use of “backpropagation with momentum and flat-spot elimination”. The weight update equation is provided below. The equation calculates the update for a single weight w_{ij} that connects unit (neuron) i to unit j .

$$\Delta w_{ij}(t+1) = \eta \delta_j o_i + \mu \Delta w_{ij}(t) \quad (5.1)$$

where

$$\delta_j = \begin{cases} f'_j \left(\left(\sum_i w_{ij} o_i \right) + c \right) d_j - o_j & \text{for output units} \\ f'_j \left(\left(\sum_i w_{ij} o_i \right) + c \right) \sum_k \delta_k w_{jk} & \text{for hidden units} \end{cases}$$

and d_j desired output at unit j
 o_j actual output at unit j
 f'_j first derivative of the sigmoid transfer function
 c flat spot elimination term that is optionally added to the weighted sum to ensure a non-trivial gradient and thereby prevent training stagnation.

The formula may be described as follows:

$\Delta w_{ij}(t+1)$ - The next weight update

$\eta \delta_j o_i$ - Scaled product of local gradient and unit i activation

η Scaling factor called the learning rate

δ_j Error-surface-gradient term (describes local gradient)

o_i Unit i output

$\mu \Delta w_{ij}(t)$ - Scaled previous weight update

μ Scaling factor called momentum

Learning Rate (η) is a scaling factor for the weight update. It influences the magnitude of the update or the step size across the error surface. The Momentum (μ) term is used to add a fraction of the previous update to the current one. This effectively smoothes the trajectory due to the integrating effect of memory (i.e. retaining a vestige of the previous update), and it reduces the sensitivity of the update to the immediate error surface gradient. Momentum is intended to help the neural network

escape local minima. In the event of the error surface gradient becoming very small, the neural network will rapidly lose momentum and run the risk of stagnating. The Flat-Spot (c) term is an offset that is added to the gradient calculation in such instances.

The general training objectives are:

- To find the global minimum in the error surface,
- To avoid getting trapped in local minima along the path to the global minimum, and
- To encourage entrapment in the global minimum.

This study implemented an online control mechanism for the Learning Rate (η), Momentum (μ) and Flat Spot (c) parameters. The purpose of training is to traverse the error surface and to find the global minimum while avoiding entrapment in intervening local minima. Large η and μ cause the network to move over the error surface in large and bold steps. Momentum reduces the sensitivity to the local gradient causing the network to occasionally go against the gradient (hence the reference to “boldness”). Given that the neural network usually commences training at a considerable distance from that global minimum, the larger steps allow the neural network to rapidly converge on the general vicinity of the global minimum. The step size also allows the network to step over and escape some local minima. However, when the global minimum is approached, the intention is to get trapped or drawn in as deep as possible. A large step size does not serve this end. In fact a large η (which translates into large step size) often causes the network to oscillate in the general vicinity of the global minimum. A reduction in step size (η) at this point will increase the search resolution and a reduction in μ will make the training process more sensitive to the immediate local gradient. In effect, the network will take smaller more tentative steps and is likely to find and be trapped (drawn in) by the global minimum.

The intention of the control strategy is to appropriately reduce the η , μ & c parameters as the global minimum is approached. The average training error is calculated after every 10 training epochs. If at any time the calculated error drops below 20% of its initial value, the parameters η , μ and c are halved. This action is repeated each time an 80% (or higher) drop in training error is detected.

5.2.3 Result post-processing

After 1000 training epochs (as indicated by Figure 5.3), the final network is saved. By this stage the training script would have produced three saved versions of the current network:

- The best-validated network (saved with file extensions “.vabest.net”),
- The best-trained network (saved with file extensions “.trbest.net”), and
- The final network after 1000 epochs (saved with file extensions “.final.net”).

The best-validated and best-trained networks are the networks with the lowest validation and training errors respectively. The control scripts were written as general research tools and produce more output than is necessary for this study. This study deals principally with the best-validated network. All saved networks can be found in the appropriate directory under “RESULT_CD:\results\saved_nets”.

The final script stages are all related to the testing of the saved networks. In its final stage, the training script (Figure 5.3) tests the classification performance of all three saved networks against all four pattern-sets (training, validation, testing and final pattern-sets). The results are saved to file (with extension “.res”) and can be found in the appropriate dataset directory under “RESULT_CD:\results\network_output”. For the purposes of this study, the final classification results are obtained by testing the best-validated network against the test pattern-set.

Control is then handed back to the high-level script (Figure 5.2) where graphs of results are produced. A training history graph for the current network is produced first. Figure 5.9 to Figure 5.12 are selected examples of such graphs. The full set of graphs may be found in the appropriate dataset subdirectories under “RESULT_CD:\results\network_output”.

Finally, the high-level script runs a result processing script (ressh) that analyses all result (“.res”) files and extracts classification rate statistics. The statistics are saved to log files such that each file contains the classification statistics for all networks in a given dataset. The directory “RESULT_CD:\results\log_files” contains the full set of logs. These logs are used to produce graphs of “Classification rate vs. Hidden layer size”. Figure 5.14 illustrates a selection of these graphs. The file “RESULT_CD:\results\log_files\result_logs.txt” describes the log-file format.

In summary, the result processing script sections do the following:

- Test and analyse trained networks against all datasets,
- Generate training history graphs for each network, and
- Generate result logs for each data set and plot classification rate vs. hidden units graphs.

5.3 Training configuration

The final experiment produced large amounts of data and results. To avoid getting lost in less significant detail, this thesis only considers selected data and results. The full data and result set is provided on the CD set, and Appendix B provides a more comprehensive set of graphs and result tables.

Making sense of the complexity: Raw data is stored in the PUR dataset. Fully pre-processed data is stored in the D3CB dataset. Several intermediate datasets were extracted along the pre-processing pathway. A separate neural network classifier classified each dataset. The training of each network, referred to as a dataset classifier, entailed a distinct sub-experiment.

The sub-experiment: The required hidden layer size (learning capacity) of each dataset classifier had to be determined experimentally. This meant that for each dataset:

- Several neural networks of different capacities had to be trained,
- Each network needed to be tested and the results analysed,
- One of them would be appropriately chosen as the final dataset classifier, and
- There were 15 such sub-experiments corresponding to the number of datasets.

Section 5.2 introduced the concept of training schedule files that were created to exert broad control over the experiment. These files also record aspects of experimental procedure. They are used to guide the high-level (Figure 5.2) and training (Figure 5.3) scripts by providing configuration information for each neural network that is to be trained. Training schedule files may be found in the appropriate dataset directories under “RESULT_CD:\configuration\datasets”. The script “RESULT_CD:\configuration\datasets\TrainAll” is used to initiate training across all datasets.

The schedule file for the PUR dataset is provided in Figure 5.5. The file header describes the significance of each field. The numbered rows represent four different neural networks that must be trained to classify this dataset. After training, one of them is selected as the dataset classifier. The only parameters that change across these four networks are the hidden layer size, neural network name and result file name. In order to simplify the stored information, all filenames are stated without file extensions. The scripts read the filenames from the schedule file, append the appropriate standard extensions and continue processing.

Column descriptions:
seq - sequence point number from 1, 2, 3,...
LR - learning rate
MM - Momentum term
FS - Flat spot elimination coefficient
fnet - name of network file (exclude the ".net" extension)
fres - name of result file
ftr - name of training set file
fts - name of test set file
fva - name of validation set file
ffull- name of full data set file
seed - seed integer for random number generator
trcyc- number of training epochs between validations
cycmax- maximum number of training cycles (multiple of trcyc)
hid- Hidden layer size

seq	LR	MM	FS	fnet	fres	ftr	fts	fva	ffull	seed	trcyc	cycmax	hid
1	0.1	0.1	0.05	pur004A	pur004A	pur004tr	pur004ts	pur004va	pur004	125	10	1000	25
2	0.1	0.1	0.05	pur004B	pur004B	pur004tr	pur004ts	pur004va	pur004	125	10	1000	50
3	0.1	0.1	0.05	pur004C	pur004C	pur004tr	pur004ts	pur004va	pur004	125	10	1000	100
4	0.1	0.1	0.05	pur004D	pur004D	pur004tr	pur004ts	pur004va	pur004	125	10	1000	200
-1													

Figure 5.5: Schedule file for networks in the PUR dataset

After training the four networks, one may add new networks to the schedule file (from sequence point 5 onwards in this case). The training script can then be informed to reprocess the schedule file from the new sequence point onwards. This facility was used extensively when processing the final D3CB dataset (see Figure 5.20). For example, initial results may show that optimum capacity lies somewhere between 100 and 200 hidden units. One can then go back and insert new networks with intermediate capacity and continue in the cycle until an optimum size is found.

The search for optimum capacity was fully implemented only on the final D3CB dataset. All other datasets were coarsely searched. This choice is related to network training time and network size. Both size and training time must be minimised for a solution to be regarded as a good one. This is a general Artificial Intelligence (AI) performance measure referred to as “space and time complexity” [66]. It is applied to various AI technologies including neural networks. Section 5.4 provides a more detailed perspective on the issue.

From a practical point of view, any solution with low space-time complexity requires less processing power and little storage space. This reduces the cost of the solution and often permits the solution to be deployed as a low cost embedded system.

5.4 Results

5.4.1 Pre-processing results

Chapter 4 described the evolution of a data pre-processing methodology. Starting with original data (PUR dataset), various datasets were produced and stored along the route to the final D3CB dataset. Some of the intermediate datasets were analysed and subjected to classification in order to test the notion that the final stage dataset does indeed produce the best performance.

Dataset correlation: The basic separability criteria described in Chapter 4 may be evaluated in terms of correlation coefficients. The covariance matrix C is calculated for each dataset X . The correlation-coefficient matrix R is then calculated as follows:

$$R(X) = \frac{C(X)}{\sqrt{C(X)C(X)}} \quad (5.2)$$

The main diagonal of matrix $R(X)$ represents the level of correlation between measurements in the same class (autocorrelation levels), while the off-diagonal components represent the level of correlation between data from different classes (cross-correlation levels). Auto-correlation levels provide an indication of measurement similarity within a class, while cross-correlation levels indicate inter-class similarity. Matrices C and R are symmetric about the main diagonal. Table 5.3 illustrates the correlation coefficient matrix R for the PUR dataset in upper triangular form.

Recall that the separability criteria discussed in Chapter 4 required

- Tight clustering of data around their respective class centroids, and
- Maximum dispersal of class centroids.

In terms of correlation coefficients, each class must have a high autocorrelation (tight clustering), and distinct classes must have a low cross-correlation (high spread). Correlation coefficient matrices were calculated for all datasets. Some of these may be found in Appendix B.

The correlation coefficient matrices may be illustrated graphically as in Figure 5.6, where the correlation coefficients of PUR and D3CB datasets are plotted. It is clear that the original measurements (PUR dataset) have an undesirably high cross-correlation. For example the correlation between “Blue” and “Laberyl” (0.94) is higher than the autocorrelation of the class “Chevin” (0.92) and equivalent to the autocorrelations of classes “Camembert”, “Brie”, and “BrieTB”. This clearly

indicates an unfavourable spread of data for this dataset (PUR). The D3CB dataset on the other hand indicates a considerable improvement in correlation data.

Table 5.3: Category correlation coefficient matrix in upper triangular form (PUR dataset)

	<i>Camembert</i>	<i>Brie</i>	<i>BrieTB</i>	<i>Blue</i>	<i>Mozzarella</i>	<i>Chevin</i>	<i>Laberyl</i>
<i>Camembert</i>	0.94	0.91	0.89	0.89	0.88	0.84	0.93
<i>Brie</i>	0.00	0.94	0.92	0.92	0.80	0.88	0.92
<i>BrieTB</i>	0.00	0.00	0.94	0.92	0.79	0.88	0.92
<i>Blue</i>	0.00	0.00	0.00	0.95	0.81	0.89	0.94
<i>Mozzarella</i>	0.00	0.00	0.00	0.00	0.99	0.75	0.89
<i>Chevin</i>	0.00	0.00	0.00	0.00	0.00	0.92	0.85
<i>Laberyl</i>	0.00	0.00	0.00	0.00	0.00	0.00	0.97

Table 5.4: Category correlation coefficient matrix in upper triangular form (D3CB dataset)

	<i>Camembert</i>	<i>Brie</i>	<i>BrieTB</i>	<i>Blue</i>	<i>Mozzarella</i>	<i>Chevin</i>	<i>Laberyl</i>
<i>Camembert</i>	0.92	0.83	0.63	0.41	0.52	0.06	0.76
<i>Brie</i>	0.00	0.97	0.78	0.72	0.40	0.46	0.84
<i>BrieTB</i>	0.00	0.00	0.87	0.79	0.55	0.60	0.84
<i>Blue</i>	0.00	0.00	0.00	0.97	0.17	0.77	0.81
<i>Mozzarella</i>	0.00	0.00	0.00	0.00	0.96	0.24	0.39
<i>Chevin</i>	0.00	0.00	0.00	0.00	0.00	0.93	0.42
<i>Laberyl</i>	0.00	0.00	0.00	0.00	0.00	0.00	0.95

The description of dataset correlation may be simplified further.

- The average value of the main diagonal components provides a single generalised indicator of autocorrelation for all classes in the dataset.
- The average value of all off-diagonal components provides a similar measure of cross-correlation for all classes in the dataset.

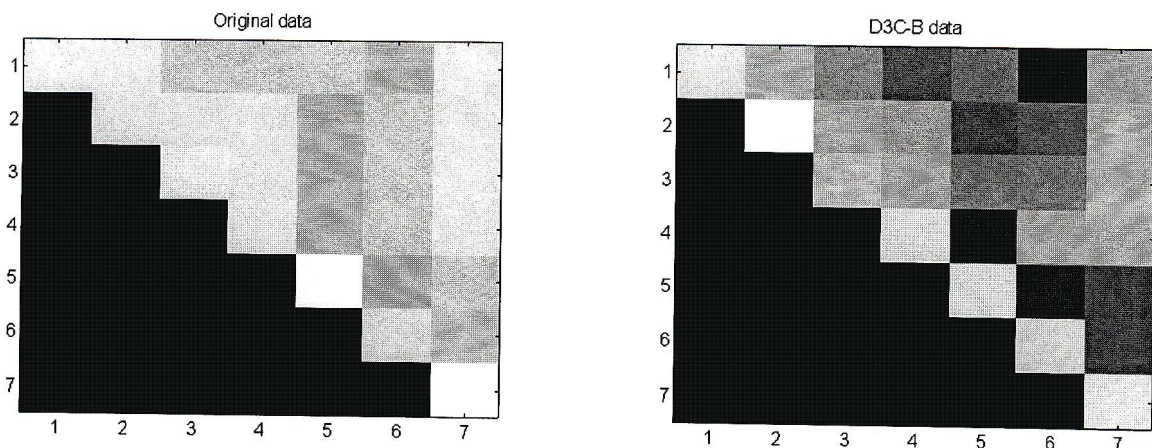


Figure 6: Original (PUR) and D3CB correlation coefficient matrices

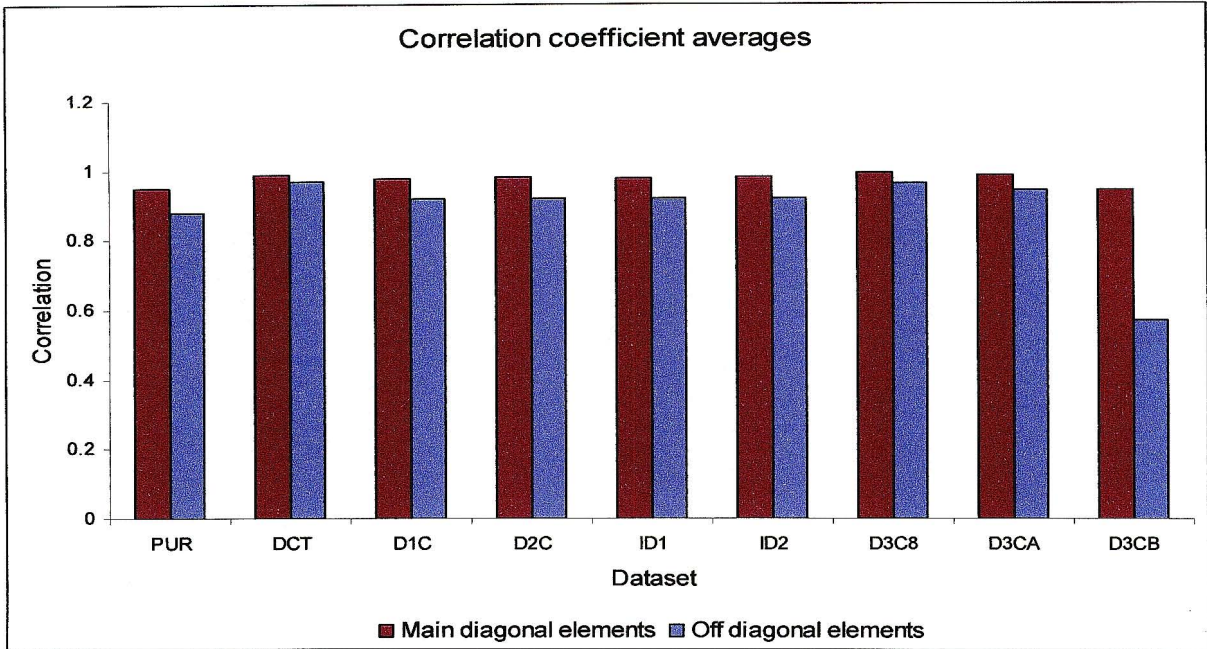


Figure 5.7: Average diagonal (auto) and off-diagonal (cross) correlation coefficients

The dataset D3CB as produced by the heuristic technique (presented in Chapter 4) shows drastically improved indicators of separability. The ratios of diagonal to off-diagonal elements were calculated for each dataset and plotted in Figure 5.8.

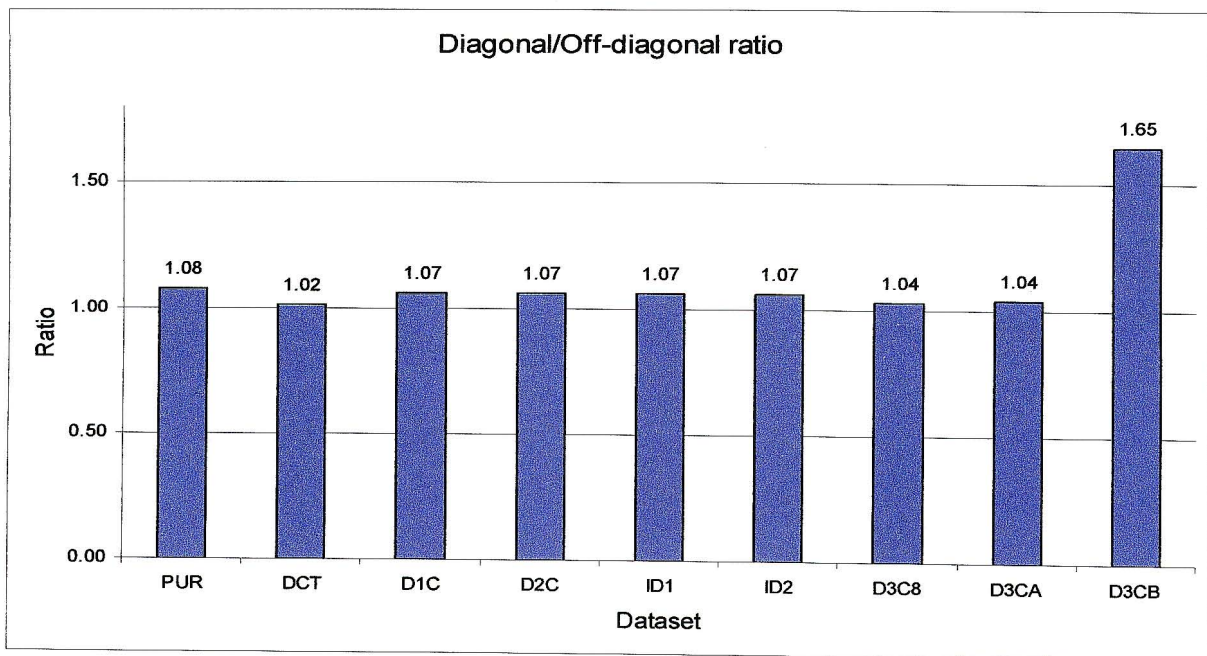


Figure 5.8: Ratios of average auto and cross-correlation values in each dataset

These correlation coefficient averages were computed for all datasets. Some of the results are illustrated in Figure 5.7. The figure reveals several facts.

- With the exception of D3CB, all datasets exhibit high cross-category correlations. Thus indicating that data are similar across classes and potentially difficult to separate.
- Cross-correlation was worse in the transform domain (DCT) than the source domain (PUR).
- Removal of the DC term in the transform domain (D1C) marginally improved (reduced) the cross-correlation figure.
- The Tukey window was used to remove 80% of the D1C data. The resultant D2C dataset did not show any degradation in the correlation ratio.
- Transform domain datasets D1C and D2C produced the same correlation ratio as their source domain counterparts, ID1 and ID2. This indicated no correlation advantage in either domain.
- The drastically reduced datasets D3C8 and D3CA produced a degraded correlation ratio, however, their figures still improved on that of the DCT dataset.
- Drastic dimensionality reduction coupled with the use of a-priori knowledge (the need to span the channel frequency axis) produced the desired improvement in correlation ratio as evident in the D3CB dataset.

The results shown here support some of the assumptions made in Chapter 4, namely,

- DC coefficient removal is beneficial,
- HF removal can be achieved without significant degradation, and
- A coefficient selection heuristic based on a-priori knowledge can produce the best result.

5.4.2 Training results

As described in Section 5.2.2, each dataset was arranged into pattern sets, and these were used to train a suitable dataset classifier. Several networks were trained in the search for each dataset classifier. The choice of dataset classifier was based on a subjective analysis of post-training results.

Some of these post-training results are discussed in this section. Due to the large amount of results produced, a comprehensive discussion would be too tedious therefore selected results are discussed. Appendix B contains a more comprehensive result set.

The schedule file for the PUR dataset is presented in Section 5.3. The training results of the four networks stipulated in that schedule are now discussed. The graph in Figure 5.9 shows the training history of the 25 hidden unit network (pur004A) in the PUR dataset. Training error, validation error and learning rate are plotted with respect to the training epoch number.

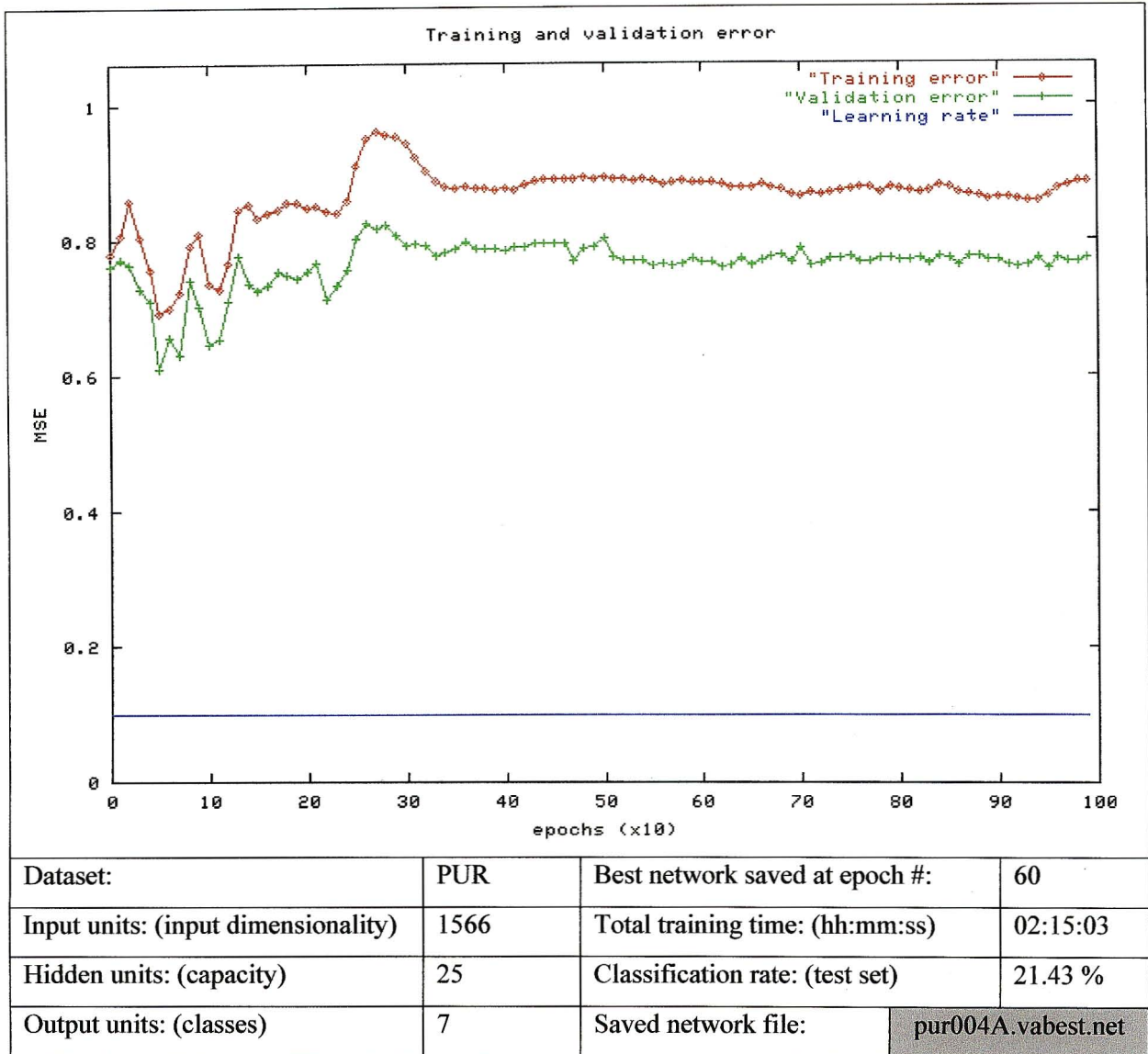


Figure 5.9: Training result - 25 hidden unit neural network with the PUR dataset

Error rates remained high throughout the training period so the training parameter control mechanism remained inactive as indicated by a constant learning rate over 1000 epochs. The initially high learning rate, momentum and flat-spot elimination parameters remained unchanged and should have encouraged rapid learning. However, there was no indication of significant learning. Assuming that the training data and algorithm were sound, the cause for poor training performance was most likely a lack of learning capacity in the network. In simpler terms, there were too few hidden units and a resultant shortage of trainable weights. With too few weights, a network is unable to model (describe) the system that produced the training data. It is simply not possible to store the required knowledge in a learning mechanism that lacks the necessary learning capacity.

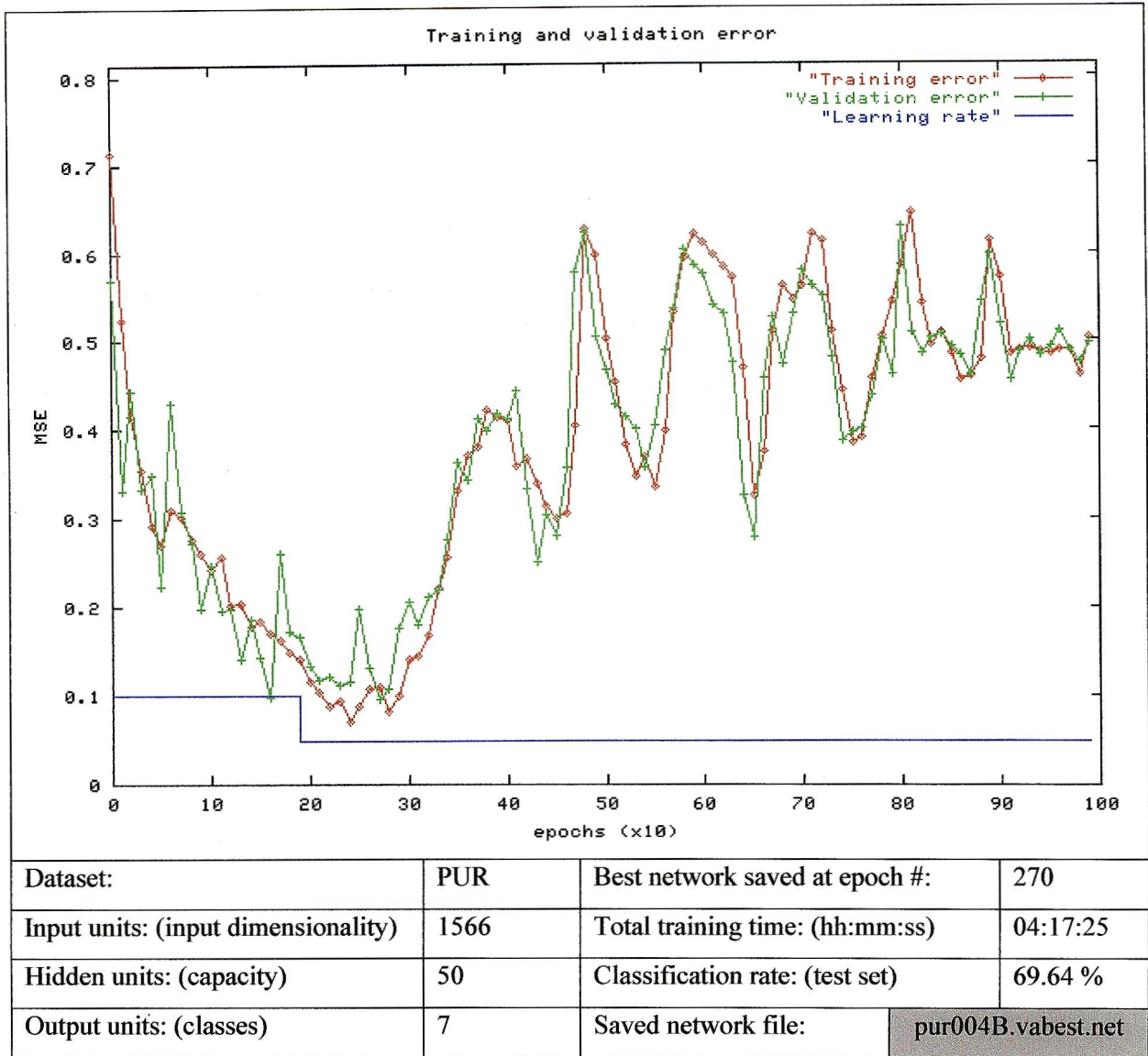


Figure 5.10: Training result - 50 hidden unit neural network with the PUR dataset

The number of hidden units was increased to 50. The network exhibited improved performance but this improvement was not sustained and the learning process became unstable. It has already been established that a network with 25 hidden units lacks the capacity to model this system. Perhaps the increase to 50 hidden units is still lacking in this regard. It is generally accepted that under-capacity produces drift and instability in the training process because the network is unable to describe sufficient modelling parameters in its small number of weights. Therefore, as new knowledge is committed to memory, the same small set of weights is modified, and old knowledge is lost or “forced out” of memory. At any given time, the under-capacity network can only store a partial model of the system. During training, one finds that the network is unable to stabilise, and its error levels fluctuate depending on the partial model it stores at that time.

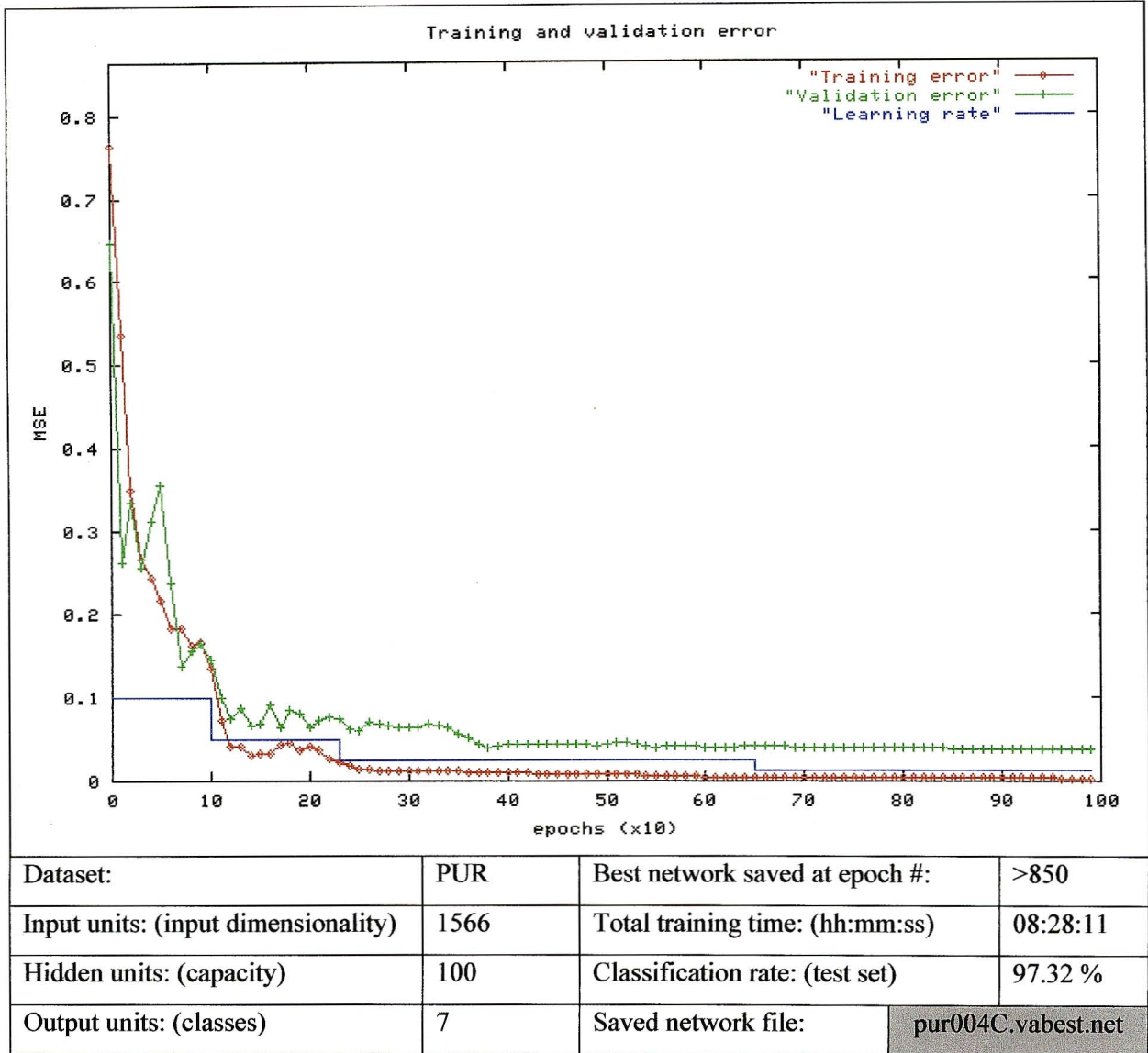


Figure 5.11: Training result - 100 hidden unit neural network with the PUR dataset

A further increase in capacity (to 100 hidden units) produced successful training, a stable trajectory and a high classification rate. The increased number of trainable weights extended the training time to 8½ hours. Automated training parameter control was triggered three times due to significant error reduction.

Validation and training errors followed similar trajectories. Validation error did not diverge significantly or rise relative to training error in the latter stages of training, thus indicating no evidence of over-training in the first 1000 epochs. Over-training is described in Section 5.2.2. The best network is saved when minimum validation error is achieved. The point at which this “best validated network” is saved was not explicitly recorded by the training script, however the graph indicates that minimum validation error was achieved somewhere beyond epoch 850. It is therefore presumed that the best network was saved somewhere between epoch 850 and 1000.

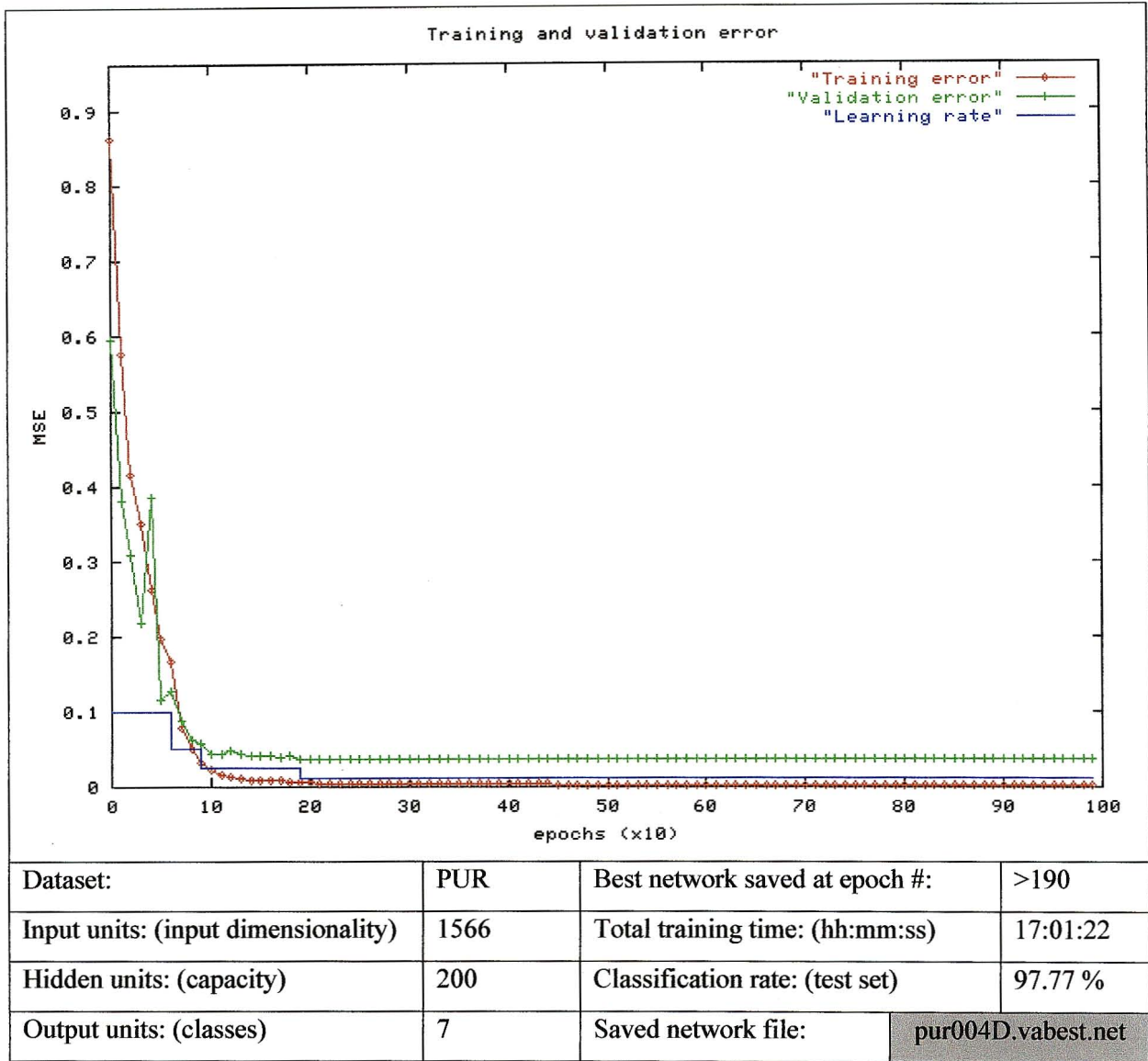


Figure 5.12: Training result - 200 hidden unit neural network with the PUR dataset

The final network in the PUR dataset was trained with 200 hidden units. This incurred a considerable training time penalty (17 hours) and produced a marginally improved classification rate. The four networks (pur004A to pur004D) all have the same input and output dimensionality and train on the same pattern sets. The variation in training time may therefore be attributed to changes in hidden layer sizes. Figure 5.13 illustrates the linear relationship between hidden layer size and training time. The gradient of the graph indicates a training time of approximately 5 minutes and 4 seconds per hidden unit.

Given that the previous network (pur004C) provided a relatively good model with 100 hidden units, this network possibly has an excessive learning capacity. The network assimilated most of the knowledge within the first 100 epochs. Automated training parameter control was triggered three

times by epoch 190 due to rapid error reduction. After that, there was little change in the error trajectory.

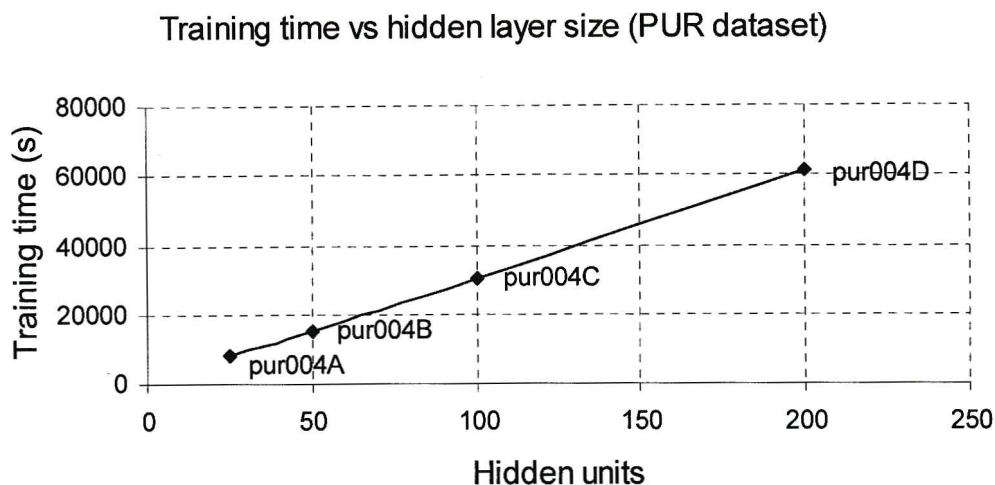


Figure 5.13: Linear relationship between training time and capacity for PUR dataset

The network `pur004D.vabest.net` is not regarded as a viable solution for the following reason: The training set contains 168 patterns and the network contains 200 hidden units. It is known that a multi-layer neural network can completely separate an arbitrary training set of p patterns, if the network has $p-1$ units in a single hidden layer [71]. In fact, this criterion describes a worst-case scenario (upper bound) where, for instance, the p patterns may belong to p distinct classes. Given that natural datasets contain fewer classes than patterns, and the classes are generally clustered, fewer than $p-1$ hidden units are usually required to classify the pattern set. It therefore stands to reason that a 200 hidden unit network cannot be regarded as a good solution when there are only 168 training patterns representing 7 classes. Given the high learning capacity of the network, the question may be asked; why was the 200-unit network unable to perfectly separate the 168 training set patterns? There are two obvious possibilities:

- The data was not separable, where the clustering is such that clusters overlap (interpenetrate) in a manner that prevents the formation of substantially error-free decision boundaries, and
- The network is trapped in a strong local minimum and was unable to find the global minimum. This can happen as a result of the random initial conditions and trajectory on the error surface.

The long training time and large memory footprint (5.445MB weight set in RAM) also serve as indicators of inadequacy.

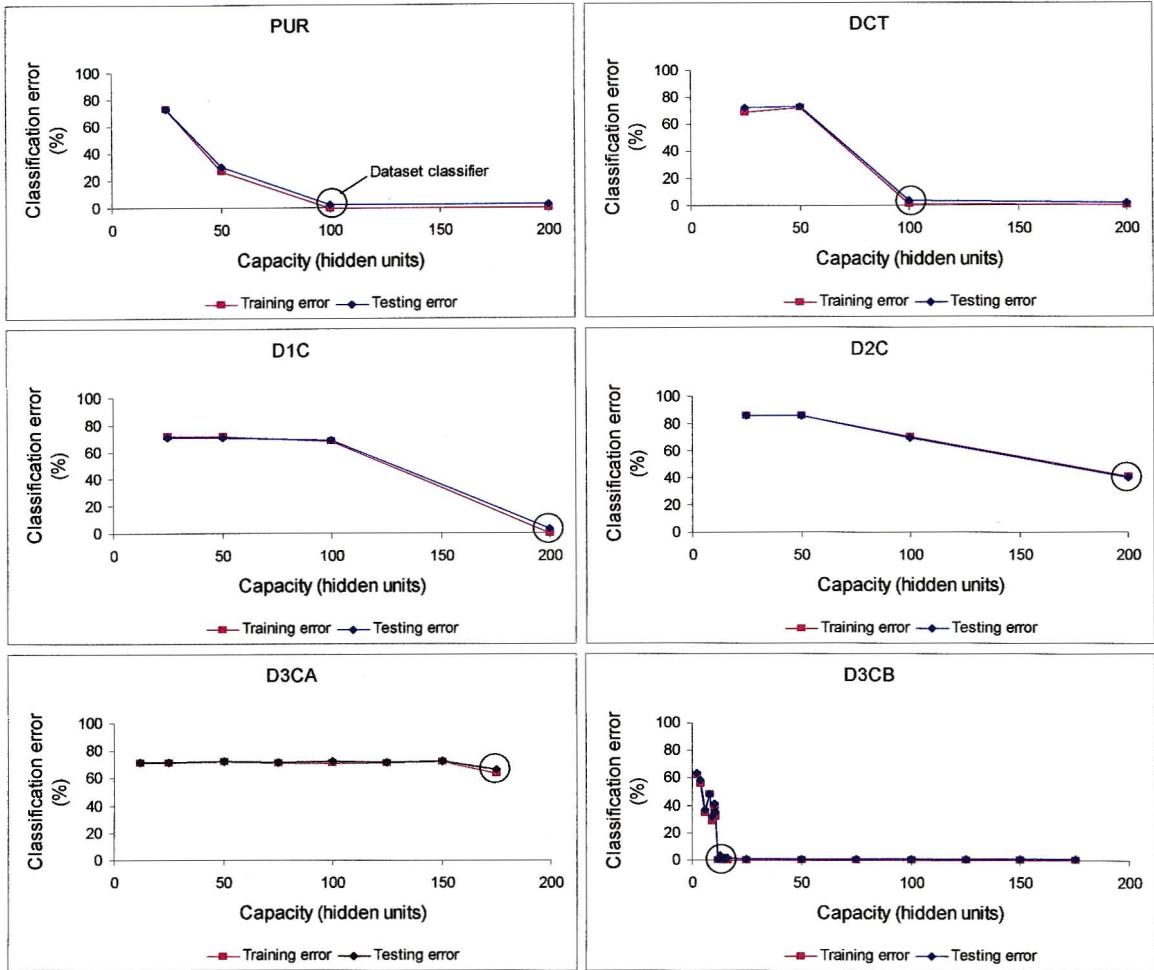


Figure 5.14: Classification error vs. capacity for selected datasets

Performance comparison: As discussed in Section 5.3, several networks of different capacity were trained in each dataset. In each dataset, one network was chosen as the dataset classifier. This choice was based on three criteria:

- Preferably high classification rate (obtained from the application of the test set to the best validated network),
- Preferably short training time, and
- Preferably small capacity (i.e. Fewer hidden units and smaller memory footprint).

These criteria needed to be appropriately balanced. For example, several networks with marginally higher classification rates were rejected on the basis of an unjustifiable increase in size and training time. In any given application, the final classifier choice must always be made subject to the immediate requirements. There is no consistent and objective approach in this regard, and high classification rates alone do not always justify a choice. The initial requirements of this experiment were that the classifier exceed 95% classification rate, and classifier size and training time be minimised. This would enable future embedded deployment. Furthermore, the smaller the capacity of a network, the better it generalises [72].

Figures 5.14 to 5.18 provide result comparisons for the following significant datasets:

- PUR & DCT – representing raw data in the source and transform domains,
- D1C & D2C – representing DC and HF filtered data in the transform domain, and
- D3CA & D3CB – representing coefficient selected data in the transform domain.

The signal processing mechanisms that produced these datasets are described in Chapter 4.

Figure 5.14 shows the classification errors for the various networks that were trained in each dataset. The subjectively chosen dataset classifier is circled in each graph. More networks were trained in the D3CA and D3CB datasets because training was much faster (see Figure 5.16). It will later become clear that a comprehensive search was only feasible and necessary in the D3CB dataset.

Figures 5.16 and 5.17 illustrate time and space complexities for the six selected datasets. Logarithmic scales were used due to the wide spread of data. From an engineering perspective, there are several objectives that must be simultaneously achieved or balanced. With respect to classification, the following major objectives exist:

- Classification performance (Figure 5.15) – the most basic requirement.
- Speed or time complexity (Figure 5.16) – fast training and execution require cheaper hardware platforms with less processing power.
- Memory footprint or space complexity (Figure 5.17) – small classifiers require cheaper hardware platforms with smaller memories.

If space and time complexity is sufficiently low, the classifier may be implemented in an embedded, possibly hand-held system. It is clear from Figures 5.16 and 5.17 that the D3CB dataset provides the best result in terms of space-time complexity.

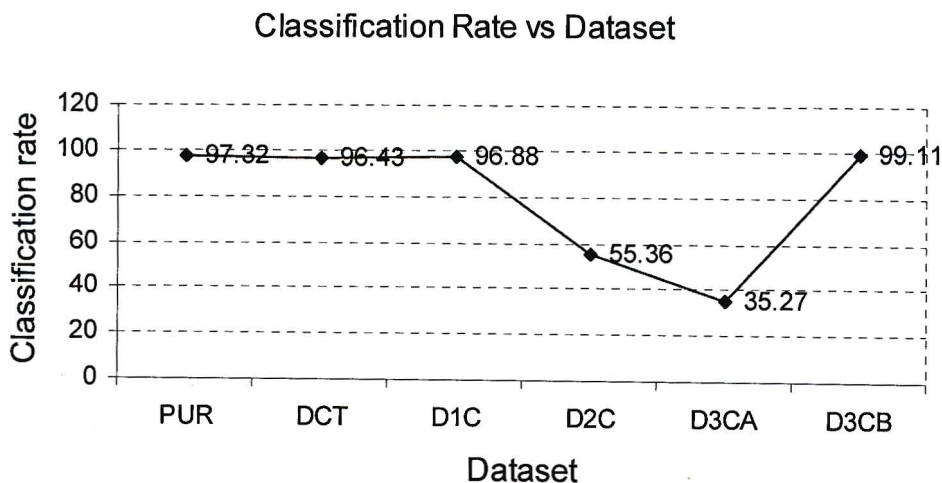


Figure 5.15: Dataset classifier classification rates for selected datasets

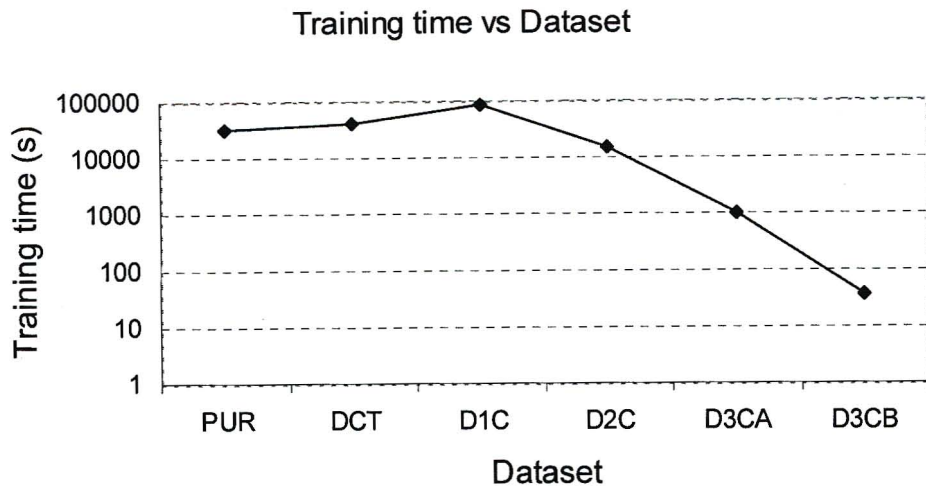


Figure 5.16: Dataset classifier training time for selected datasets

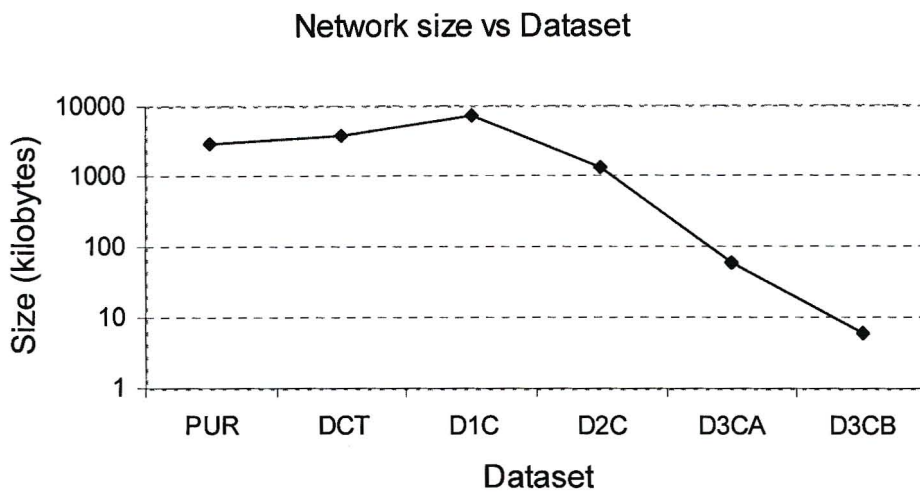


Figure 5.17: Dataset classifier file size for selected datasets

Combined performance measure: Given that the classification rate must be maximised while training time and network size must be minimised, a combined performance measure may be computed for each dataset classifier.

$$\text{performance measure} = \frac{\text{classification rate}}{(\text{network size})(\text{training time})} \quad (5.3)$$

Figure 5.18 clearly shows that dataset D3CB outperforms other datasets in terms of the performance criteria.

Combined Performance vs Dataset

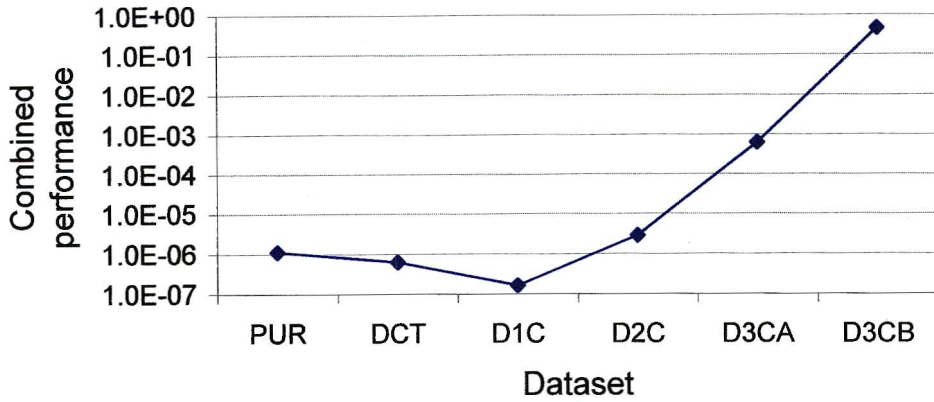


Figure 5.18: Combined performance measures for selected dataset classifiers

Capacity optimisation: It is clearly evident that the best classifier would be found in the fully pre-processed dataset D3CB. This dataset produced classifiers with the best classification rates and the smallest space-time complexity. It was therefore decided that a comprehensive search be performed for the optimal capacity classifier in the D3CB dataset. The rapid training speed of the D3CB dataset also assisted in this regard.

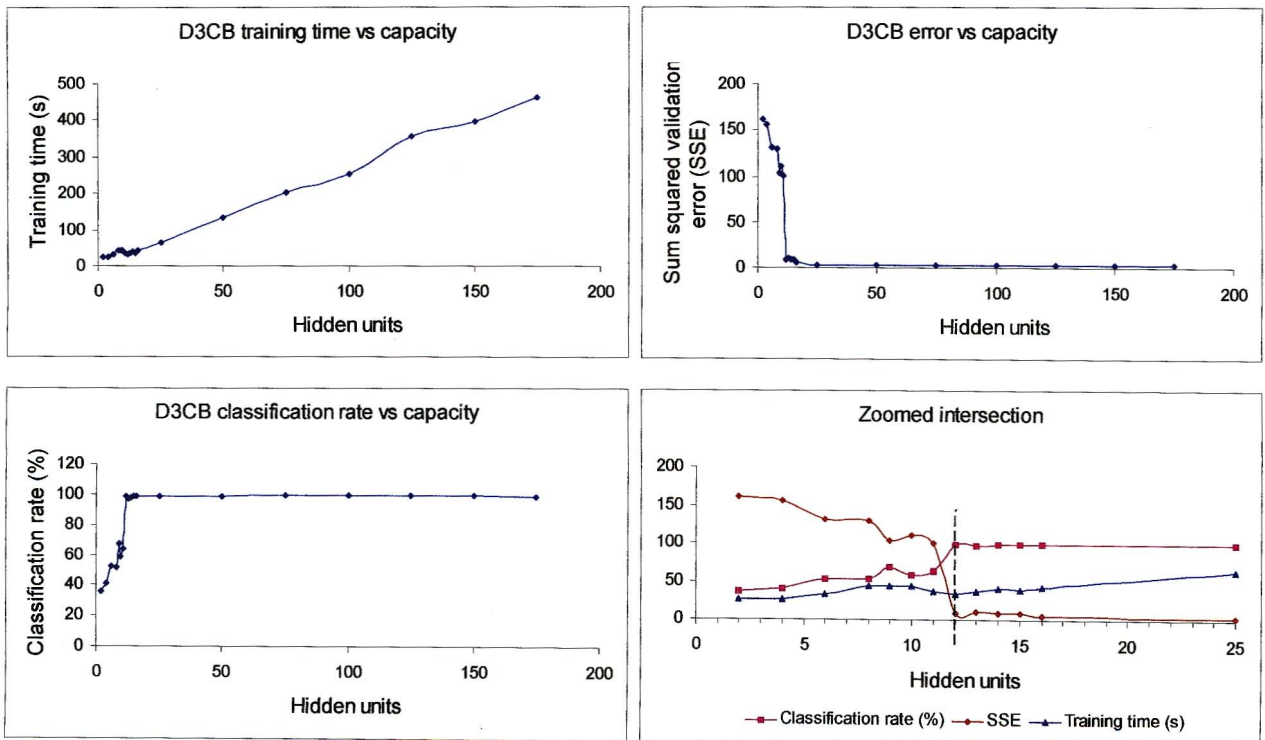


Figure 5.19: Locating the optimal D3CB neural network (i.e. the dataset classifier)

Figure 5.19 illustrates training time, test set classification rate, and validation error for all networks in the D3CB dataset. The intention was to find the smallest network (lowest capacity) that produces high classification rates, low training times and low validation errors. The zoomed intersection of the three graphs clearly shows that the network with 12 hidden units strikes the ideal balance of low capacity and good performance.

Column descriptions:													
seq - sequence point number from 1, 2, 3,...													
LR - learning rate													
MM - Momentum term													
FS - Flat spot elimination coefficient													
fnet - name of network file (exclude the ".net" extension)													
fres - name of result file													
ftr - name of training set file													
fts - name of test set file													
fva - name of validation set file													
ffull- name of full data set file													
seed - seed integer for random number generator													
trcyc- number of training epochs between validations													
cycmax- maximum number of training cycles (multiple of trcyc)													
hid- Hidden layer size													
seq	LR	MM	FS	fnet	fres	ftr	fts	fva	ffull	seed	trcyc	cycmax	hid
1	0.1	0.1	0.05	d3c004A	d3c004A	d3c004tr	d3c004ts	d3c004va	d3c004	125	10	1000	12
2	0.1	0.1	0.05	d3c004B	d3c004B	d3c004tr	d3c004ts	d3c004va	d3c004	125	10	1000	25
3	0.1	0.1	0.05	d3c004C	d3c004C	d3c004tr	d3c004ts	d3c004va	d3c004	125	10	1000	50
4	0.1	0.1	0.05	d3c004D	d3c004D	d3c004tr	d3c004ts	d3c004va	d3c004	125	10	1000	75
5	0.1	0.1	0.05	d3c004E	d3c004E	d3c004tr	d3c004ts	d3c004va	d3c004	125	10	1000	100
6	0.1	0.1	0.05	d3c004F	d3c004F	d3c004tr	d3c004ts	d3c004va	d3c004	125	10	1000	125
7	0.1	0.1	0.05	d3c004G	d3c004G	d3c004tr	d3c004ts	d3c004va	d3c004	125	10	1000	150
8	0.1	0.1	0.05	d3c004H	d3c004H	d3c004tr	d3c004ts	d3c004va	d3c004	125	10	1000	175
9	0.1	0.1	0.05	d3c004I	d3c004I	d3c004tr	d3c004ts	d3c004va	d3c004	125	10	1000	2
10	0.1	0.1	0.05	d3c004J	d3c004J	d3c004tr	d3c004ts	d3c004va	d3c004	125	10	1000	4
11	0.1	0.1	0.05	d3c004K	d3c004K	d3c004tr	d3c004ts	d3c004va	d3c004	125	10	1000	6
12	0.1	0.1	0.05	d3c004L	d3c004L	d3c004tr	d3c004ts	d3c004va	d3c004	125	10	1000	8
13	0.1	0.1	0.05	d3c004M	d3c004M	d3c004tr	d3c004ts	d3c004va	d3c004	125	10	1000	10
14	0.1	0.1	0.05	d3c004N	d3c004N	d3c004tr	d3c004ts	d3c004va	d3c004	125	10	1000	9
15	0.1	0.1	0.05	d3c004O	d3c004O	d3c004tr	d3c004ts	d3c004va	d3c004	125	10	1000	11
16	0.1	0.1	0.05	d3c004P	d3c004P	d3c004tr	d3c004ts	d3c004va	d3c004	125	10	1000	13
17	0.1	0.1	0.05	d3c004Q	d3c004Q	d3c004tr	d3c004ts	d3c004va	d3c004	125	10	1000	14
18	0.1	0.1	0.05	d3c004R	d3c004R	d3c004tr	d3c004ts	d3c004va	d3c004	125	10	1000	15
19	0.1	0.1	0.05	d3c004S	d3c004S	d3c004tr	d3c004ts	d3c004va	d3c004	125	10	1000	16
20	0.1	0.1	0.05	d3c004T	d3c004T	d3c004tr	d3c004ts	d3c004va	d3c004	230	10	1000	12
21	0.1	0.1	0.05	d3c004U	d3c004U	d3c004tr	d3c004ts	d3c004va	d3c004	108	10	1000	12
22	0.1	0.1	0.05	d3c004V	d3c004V	d3c004tr	d3c004ts	d3c004va	d3c004	123	10	1000	12
23	0.1	0.1	0.05	d3c004W	d3c004W	d3c004tr	d3c004ts	d3c004va	d3c004	99	10	1000	12
24	0.1	0.1	0.05	d3c004X	d3c004X	d3c004tr	d3c004ts	d3c004va	d3c004	0	10	1000	12

Figure 5.20: D3CB schedule file

The search procedure is effectively encoded in the D3CB schedule file. The file initially contained schedule entries 1 through 8. This performed a coarse search for optimal hidden layer size in the range 12 to 175. As it turned out the first schedule entry (sequence point 1) with 12 hidden units produced the best result. Thereafter, schedule entries 9 to 19 were added to refine the search around the 12 hidden unit network. As it turned out, the 12 hidden unit network (sequence point 1) remained the optimal network.

Finally schedule entries 20 to 24 were added. These were 12 hidden-unit networks similar to sequence point 1, but with different random number seeds. They produced results that were similar to the network specified in sequence point 1. This network (d3c004A.vabest.net), the first network to be trained in the D3CB dataset, was selected as the dataset classifier. The training history of the chosen D3CB dataset classifier is considered next.

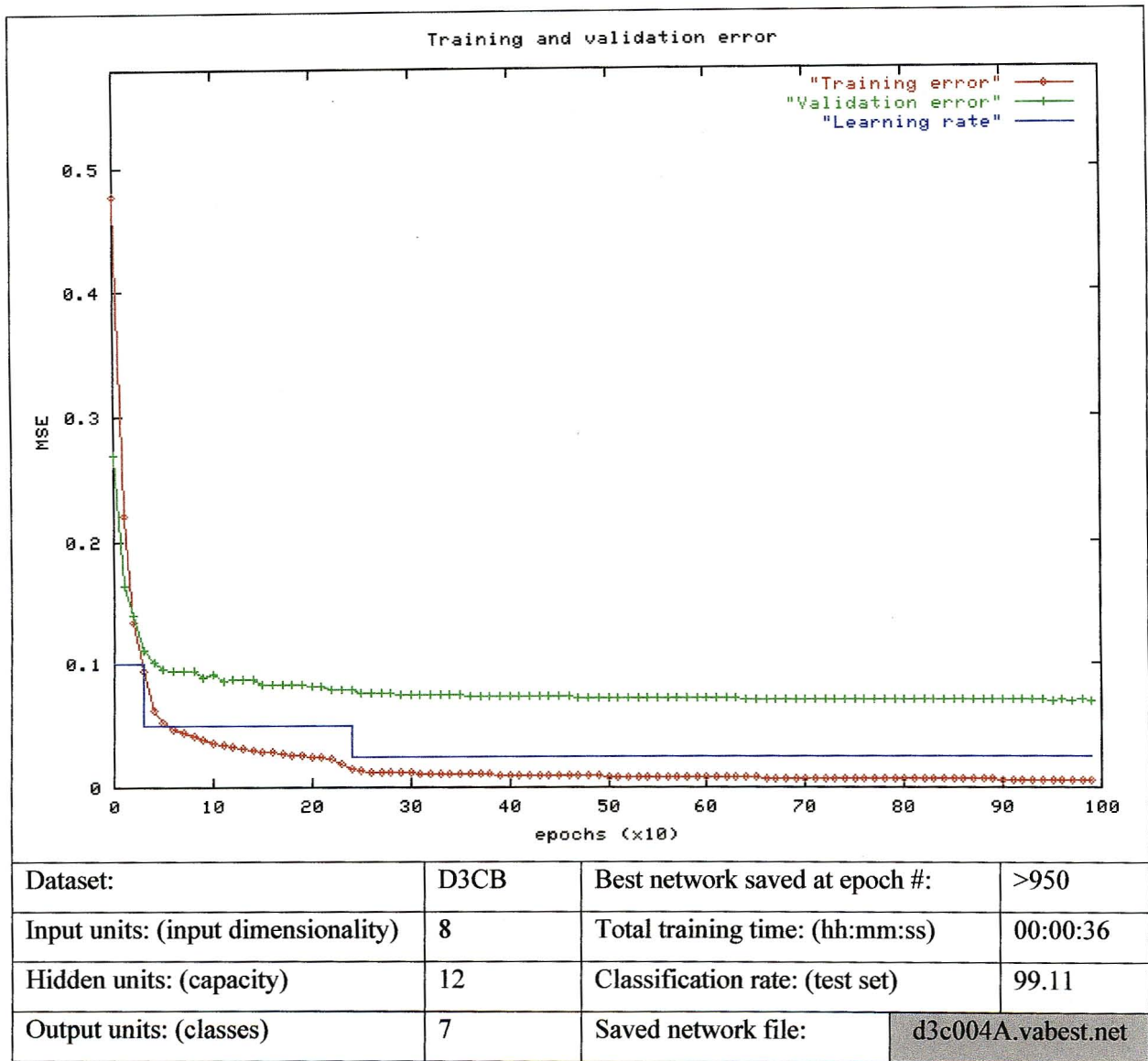


Figure 5.21: Training result - 12 hidden unit neural network with the D3CB dataset

Most notable are the low capacity, high classification rate and the very short total training time. The trajectory indicates rapid and stable learning with no evidence of over-training. The classification rate is raised regardless of the very low space and time complexity. Input dimensionality is reduced to only 8 and the total training time is reduced to 36 seconds.

5.4.3 Final Result

A result comparison of the initial (PUR) and final (D3CB) datasets follows. Table 5.5 shows comparative results. Result improvements were noted across all measured criteria. Significant and desirable reductions were achieved in input dimensionality, capacity, training time and network file size. Notwithstanding these reductions in information content and capacity, classification rates increased by from 97.32% to 99.11%.

Table 5.5: Final comparative result

	<i>PUR Dataset (Original data)</i>	<i>D3CB Dataset (Fully pre-processed)</i>	<i>Result modification (Relative to PUR)</i>
<i>Dataset classifier</i>	pur004C.vabest.net	d3c004A.vabest.net	
<i>Input space dimensionality</i>	1566	8	Dimensionality reduced to 0.51% of 1566
<i>Hidden units</i>	100	12	Hidden units reduced to 12% of 100
<i>Training time (hh:mm:ss)</i>	08:28:11	00:00:36	Training time reduced to 0.12% of 08:28:11
<i>Network file size (bytes)</i>	2,847,856 (2.71MB)	5,887 (5.74KB)	File size reduced to 0.22% of 2,847,856
<i>Classification rate (%)</i>	97.32	99.11	Absolute increase 1.79%

5.5 Conclusion

This chapter presents the design, implementation and analysis of the classification stage in the olfactory system. The classification output represents the final system output. The biologically inspired organisational framework (Chapter 2), design of the olfactory measurement front-end

(Chapter 3), and signal processing mechanism (Chapter 4) all influenced the final classification result. Encoded in these results are the effects of system organisation, measurement, pre-processing and classification.

The search for optimal classifiers across 15 datasets would have been onerous and was avoided. A coarse search and result analysis quickly identified the fully processed D3CB dataset as the best candidate for full investigation. The search for an optimal classifier for the D3CB dataset showed by pure coincidence that the first network to be trained was optimal. Several similar (12 hidden unit) networks were trained with different random seeds but all of them produced the same result thereby lending further credibility to the discovery of the 12 hidden unit network as the optimal classifier for the dataset.

The D3CB dataset produced the benefit of low measurement dimensionality at 256 orders of magnitude less than raw DCT data. The final network produced the best result across all performance categories, namely lowest space-complexity, lowest time-complexity and highest classification rate.

CHAPTER 6

CONCLUSION

Biological olfactory systems typically define the upper limit of olfactory performance. This study briefly investigates the biological organisation or functional design that ultimately leads to successful olfactory discrimination in animals. An organisational framework is produced, and an artificial olfactory system is deployed in that framework. Although the underlying organisation is conceptualised in a biological paradigm, the artificial olfactory system is very much a study in engineering with inspirational cues drawn from biological olfactory organisation.

The biological olfactory sense contains many unusual specialisations such as the continuous replacement of receptor neurons on a monthly cycle and orthogonal signal decorrelation in the olfactory bulb. Specialisations such as these shed light on the nature of the problem and the biological solutions that have evolved over millions of years. These biological solutions are composed of elements that have close parallels in the engineering paradigm. As explained in Chapter 2, the core biological logic can be expressed in engineering terms.

Contributions of this thesis: The main contribution made by this study is the proposition and demonstration of the fact that biological olfactory organisation provides a sound basis on which to specify an artificial olfactory system. The following specific contributions were made.

- The importance of temperature, humidity and flow-rate regulation was discovered and used to guide the design of a gas handling front-end.
- Gas separation and sub-threshold detection in parallel detection channels were found to be important.
- The notion of controlled sniffing was investigated and used to develop a standardised temporal measurement procedure.
- The bulbar map was shown to be similar to a discrete orthogonal transformation.
- The architecture of axonal projections from the bulbar map to each cortical locus was shown to be a subspace selection mechanism in the bulbar coefficient space, and was emulated in the electronic nose.
- A separability based coefficient selection mechanism was developed for the classification problem. The mechanism should be generally applicable to other classification problems.
- A gas auto-sampler, measurement control language and automated neural network training mechanism were developed.

Descriptions of the major conclusions follow.

Temporal episodic nature of olfactory discrimination: The entire process of conscious olfactory discrimination in animals, from pre-receptor processes to final classification, is regulated by the simple act of sniffing. Among other things, the regulated sniff creates a standard temporal context in which odour measurements may be compared. This standardised measurement control system is partially emulated in the odour measurement event that is described in Chapter 3. It is a fundamental organisational cue derived from biological olfaction and is used to standardise the measurement procedure.

Regulation of the measurement conditions: Several environmental variables are regulated in the biological nose at the pre-receptor level. Quite predictably, the aim is to keep the detection mechanism in its optimal sensing state. The same analogy is applied to the electronic nose where temperature, humidity, carrier purity, gas flow-rate and concentration are regulated. This need to regulate environmental conditions is a significant limiting factor in the industrial application of electronic noses. The solution to the problem lies mainly in the development of more robust sensor technology, which is beyond the scope of this study.

Broad-spectrum array detection: Electronic noses are distinct from gas analysers by virtue of their ability to discriminate abstract user-definable odour conditions. It is for this purpose that the detection mechanism must be specified as an array of sensors with distinct but broad and overlapping receptive fields. This allows the array to sense a wider variety of odorants but necessitates advanced signal processing for condition discrimination. Chapter 2 explains that most biological olfactory systems are array based odour generalists. That is, they have evolved into generic odour detectors, which is similar in purpose to the electronic nose.

The need for signal processing: Broad-spectrum array detectors are able to detect a wide range of odorants, but they have greatly reduced discriminatory ability. Any given odorant will produce a poorly defined and variable pattern of excitation across the array. It is therefore necessary to perform olfactory discrimination at some subsequent signal processing stage.

Fixed orthogonal transformation: An orthogonal transformation takes place in the mammalian olfactory bulb and is stereotyped or unchanging across individuals of a species. Similar to the family of discrete trigonometric transforms, the bulbar transformation expresses arbitrary input stimuli as weighted combinations of fixed orthogonal transform bases. The DCT may be used to emulate the bulbar transformation.

Heuristically optimised coefficient selection: Bulbar outputs represent the fundamental building blocks of odour character. Various combinations of these outputs converge in different regions of the olfactory cortex where classification processes begin. It is presumed that the bulbar signals that converge on a particular cortical locus provide focused information that is useful in a particular classification task. In a similar fashion, a small focused selection of DCT coefficients are used to classify cheese odours in the electronic nose. The selection methodology for bulbar outputs is not well understood and was not emulated. A heuristic coefficient selection approach was developed with the specific requirements of classification problems in mind.

The strategy proposed in Chapter 4 is related to and derived from existing signal and image processing methods, however, there is a major departure with regard to intention or purpose. In image processing, coefficient selection is usually implemented with the intention of compressing and subsequently restoring the data stream with minimum information loss. In classification problems, the burden of restoration is removed. Chapter 4 proposed that it is possible to select a feature subspace (i.e. a subset of an orthogonal feature space) in such a manner that classification results within that subspace may be generalised over the original larger feature space. The technique was developed such that the classification problem is simpler in the subspace and yet remains representative of the larger more difficult problem. There remains significant scope for further research into subspace selection.

Capacity optimised neural classification: The number of hidden units in a neural network affects training time, memory utilisation and final classification rate. A capacity optimised network is able to simultaneously produce low time and space complexity, and high classification rates. There is no fast method for finding optimal capacity. Given that some networks took as long as 24 hours to train, the search for optimal classifiers across 15 datasets would have been onerous and was avoided. A coarse search and result analysis quickly identified the fully processed D3CB dataset as the best candidate for comprehensive capacity optimisation.

The D3CB dataset produced the benefit of low measurement dimensionality at 256 times smaller than DCT data. The final network produced the best result across all performance categories namely, space-complexity, time-complexity and classification rate. A classification rate of 99.11% was achieved across 560 odour measurements in 7 cheese categories. The optimised classifier trained in 36 seconds and required 5.74 KB of storage.

Suggestions for further work: This study covered a vast research domain that spans biological research areas, measurement, signal processing, and pattern recognition. There were many topics that

could not be addressed in this initial study. The following specific suggestions are made for further work in this area:

- Further investigation is required into bulbar, limbic and cortical functionality and integration of new knowledge into the system architecture.
- A parallel-measurement, multi-flow-rate system, and gas separation column should be incorporated into a future system.
- Use of a hybrid sensor-array that incorporates various technologies such as newer tin-dioxide, conductive polymer, optical and quartz crystal microbalance sensors will improve performance in future systems.
- A deeper investigation should be undertaken into other orthogonal transforms, PCA and coefficient selection at the pre-processing level. It is suggested here that multiple parallel transforms can be used to feed the coefficient selector.
- Other classification methods that are not biologically inspired such as support vector machines and statistical classifiers should be considered for the final classification stage.
- Adaptive classification that is capable of constant online learning may provide better emulation of biological functionality.
- Structured knowledge systems may be considered for high-level decision-making and the management and integration of multiple specialised pre-processors and classifiers into a dynamically configurable unified system.