"Software optimisation techniques for real-time applied adaptive filtering using the TMS320C6201 Digital Signal Processor"

N. Dahnoun, M. Hart and F. Umbarila, University of Bristol

ABSTRACT

Adaptive filtering is now an integral part of most modern communication systems where it is involved with both channel equalisation and estimation techniques.

In these systems, filters are generally fed with a short training sequence to which they have to adapt prior to receiving data. This training sequence is often multiplexed with the data, reducing the amount of data transmitted in each frame.

To maximise the efficiency of a system, training sequences need to be as short as possible requiring that adaptation occurs in as few iterations as possible. Also as bit rates of communication systems increase, the time available to complete one iteration decreases. All of these factors place increasing demands on implemented algorithms, requiring fast digital signal processors with highly efficient optimised software.

In this paper, optimisation techniques for real-time adaptive algorithms based on Wiener and Kalman filter theory were developed. Two algorithms in particular were implemented on the TMS320C6201 evaluation module, these being the Least Mean Squares and Recursive Least Squares. Benchmarking of the algorithms was performed allowing the evaluation of the maximum bit rate that can be supported in various situations. The two algorithms were also compared in other areas such as code size, ease of implementation, stability, reliability and data memory required for implementation.

1 INTRODUCTION

Conventional filters are used in a variety of applications where the specifications required and the system into which it will be employed are known prior to the design of the filter.

To enable filters to be used in applications where their specifications are required to change with the nature of the application and time, an adaptation mechanism is required.

This mechanism needs to be defined in some way so that it adjusts the filter coefficients in an attempt to meet some criteria. In most applications this criteria is the minimisation of some form of error based on the success in the adjustment of the filter specifications to that required by the application.

There are many situations that require the use of adaptive filters and these include channel equalisation and estimation, speech processing, echo and noise cancellation, control systems and medical imaging and instrumentation.

There are also several algorithms available for performing the adaptation process. However, they generally fall into one of two categories, these being either Least Mean Squares (LMS) or Recursive Least Squares (RLS). There are also variants of these that are improved versions for more specific applications.

In this paper these two algorithms are defined and then implemented on the Texas Instruments most powerful fixed-point digital signal processor, the TMS320C62x. The software used to implement these algorithms is then optimised followed by benchmarking and testing of the algorithms both quantitatively and qualitatively.

2 BACKGROUND THEORY

Adaptive Filtering

The objective of the adaptive mechanism within the filter is to minimise the error in estimating the desired signal.

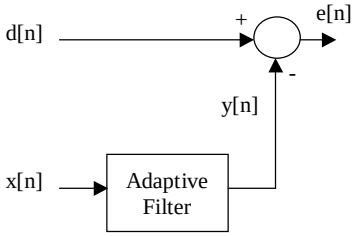

Figure 1 – Basic Adaptive Filter Structure

The output of such a filter and the estimation error is:

$$y = \mathbf{H}^T\mathbf{X} \qquad e = d - y$$

Where **H** is the vector containing the tap coefficients and **X** is the vector containing the tap inputs.

The aim of the two adaptive algorithms implemented in this report is to adjust **H** such that the error is minimised in some way. The method used to minimise the error in both cases is highlighted first in a post processing

environment where all the data is gathered prior to optimisation of the filter coefficients.

These methods are then used to derive the two recursive algorithms implemented.

Minimum Mean Square Error (MMSE)

The idea of the LMS algorithm is to minimise the Mean-Square Error (MSE) resulting from the estimation of the desired response. The cost function, *J*, is therefore [5]:

$$J = E\left[e(n)^2\right] \tag{1}$$

Where *E* is the expectation. It can be shown that the optimal solution for the tap coefficients in the MMSE sense is [5]:

$$\mathbf{RH} = \mathbf{P} \tag{2}$$

Where **R** and **P** are respectively the auto- and cross-correlation matrices. A recursive method for optimally solving Equation 2 is presented in the form of the LMS Algorithm.

The LMS algorithm is based on the method of steepest descent. The basic aim is to minimise the MSE by adjusting the tap weights based on the gradient of the error surface, an example of which is shown in Figure 2. (This also shows why FIR filters are always used so that there is only one well-defined minima.)

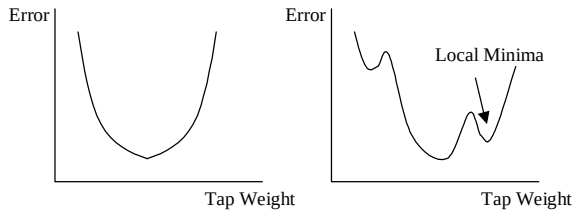

Figure 2 - Error Surface of a FIR and an IIR Adaptive Filter

The tap weights are therefore updated recursively using the following equation:

$$\mathbf{H} = \mathbf{H} - \mu\nabla \tag{3}$$

(Where μ is a constant that controls the rate and stability of convergence).

The steepest descent algorithm requires prior knowledge of both the auto- and cross-correlation matrix, **R** and **P**, which is not available in a real time adapting situation. The LMS overcomes this by using instantaneous estimates based on the currently available data. This difference is highlighted in Equation 4 which shows the different gradient calculations.

$$\begin{aligned} \nabla &= -2\mathbf{P} + 2\mathbf{RH} && \text{Steepest Descent} \\ \nabla &= -2\mathbf{X} + 2\mathbf{X}\mathbf{X}^{\mathrm{T}}\mathbf{H} && \text{LMS} \\ &= -2e\mathbf{X} \end{aligned} \tag{4}$$

Least Squares Error (LSE)

The idea of Least Squares (LS) filtering is to minimise the sum of all the estimation error squares, with the following cost function [5]:

$$J = \sum_{i=M}^{N} e^2(i) \tag{5}$$

It can be shown that the optimal solution for the tap coefficients in the LSE sense is [5]:

$$\mathbf{RH} = \mathbf{P} \tag{6}$$

Although we are effectively trying to solve the same equation as for the MMSE case, we are using two different methods. In the LSE method the definitions are based on time summations, whereas for the MMSE case they were based on expectations.

The Recursive Least Squares (RLS) algorithm computes updated estimates of the tap weight vector based on the least-square error and is a special case of a Kalman filter. The RLS algorithm exploits the matrix inversion lemma to overcome the need to compute lengthy matrix inversions required to update tap weight vector.

First consider the update equation for the auto-correlation matrix:

$$\mathbf{R_n} = \mathbf{R_{n-1}} + \mathbf{X_n}\mathbf{X_n^T} \tag{7}$$

A recursion for the inverse of the auto-correlation can now be derived by using Equation 7 and the matrix inversion lemma, as given below:

$$\mathbf{A}^{-1} = \mathbf{B} - \mathbf{BC}\left(\mathbf{D} + \mathbf{C}^{\mathbf{H}}\mathbf{BC}\right)^{-1}\mathbf{C}^{\mathbf{H}}\mathbf{B} \tag{8}$$

Where: $\mathbf{A} = \mathbf{R}_n$, $\mathbf{B} = \mathbf{R}_{n-1}^{-1}$, $\mathbf{C} = \mathbf{X_n}$ and $\mathbf{D} = 1$.

Using the recursive form for auto-correlation matrix, a similar recursion for the cross-correlation matrix and Equation 6, the update equation for the tap weights can be shown to be [5]:

$$\mathbf{H}_n = \mathbf{H}_{n-1} + \mathbf{R}^{-1}\mathbf{X}e \tag{9}$$

Summary of Recursive Adaptive Algorithms

Table 1 summarises the LMS and RLS in terms of the equations required and the initialisation conditions required. It also shows the equations required for a complex version of the LMS.

Update Equation	$\mathbf{H_n} = \mathbf{H_{n-1}} + \mu e \mathbf{X}_n$
Error	$e[n] = d[n] - y[n]$
Output	$y[n] = \mathbf{H_n^T X_n}$
Initialisation	$\mathbf{H^T} = 0$

Table 1a– Summary of the LMS Algorithm

Update Equations	$\mathbf{H}_I(n) = \mathbf{H}_I(n-1) + \mu\left(e_I \mathbf{X}_I + e_Q \mathbf{X}_Q\right)$ $\mathbf{H}_Q(n) = \mathbf{H}_Q(n-1) + \mu\left(e_I \mathbf{X}_Q - e_Q \mathbf{X}_I\right)$ $\mathbf{H} = \mathbf{H}_I + j\mathbf{H_Q}$
Error	$e[n] = e_I[n] + j e_Q[n]$ $= \left(d_I[n] - y_I[n]\right) + j\left(d_Q[n] - y_Q[n]\right)$
Output	$y[n] = \mathbf{H^H X}$ $\mathbf{X} = \mathbf{X}_I + j\mathbf{X}_Q$
Initialisation	$\mathbf{H^H} = 0$

Table 1b– Summary of the Complex LMS Algorithm

Coefficient Update Equation	$\mathbf{H}(n) = \mathbf{H}(n-1) + \mathbf{R}_{\mathbf{XX}}^{-1}(n)\mathbf{X}(n)e(n)$
Inverse Matrix Update Equation	$\mathbf{R}_{\mathbf{XX}}^{-1}(n) = \mathbf{R}_{\mathbf{XX}}^{-1}(n-1) - \frac{\mathbf{R}_{\mathbf{XX}}^{-1}(n-1)\mathbf{X}(n)\mathbf{X}^{\mathbf{T}}(n)\mathbf{R}_{\mathbf{XX}}^{-1}(n-1)}{1 + \mathbf{X}^{\mathbf{T}}(n)\mathbf{R}_{\mathbf{XX}}^{-1}(n-1)\mathbf{X}(n)}$
Error	$e[n] = d[n] - y[n]$
Output	$y[n] = \mathbf{H_n^T X_n}$
Initialisation (I is the identity matrix, δ is a small constant.)	$\mathbf{H^T} = 0$ $\mathbf{R} = \frac{1}{\delta}\mathbf{I}$

Table 1c – Summary of the RLS Algorithm

3 SOFTWARE OPTIMISATION TECHNIQUES

The software tools supplied with the TMS320C62x allow code to be written in three different formats, C, linear assembly and assembly. Texas Instruments quote that the optimising C compiler is about 80% efficient compared to hand optimised assembly [1] and linear assembly is 95-100% efficient.

However, writing code directly in assembly is often difficult and time consuming. Hence the following optimisation procedure is recommended [1][2][3][4][11] and was followed in the development in this paper.

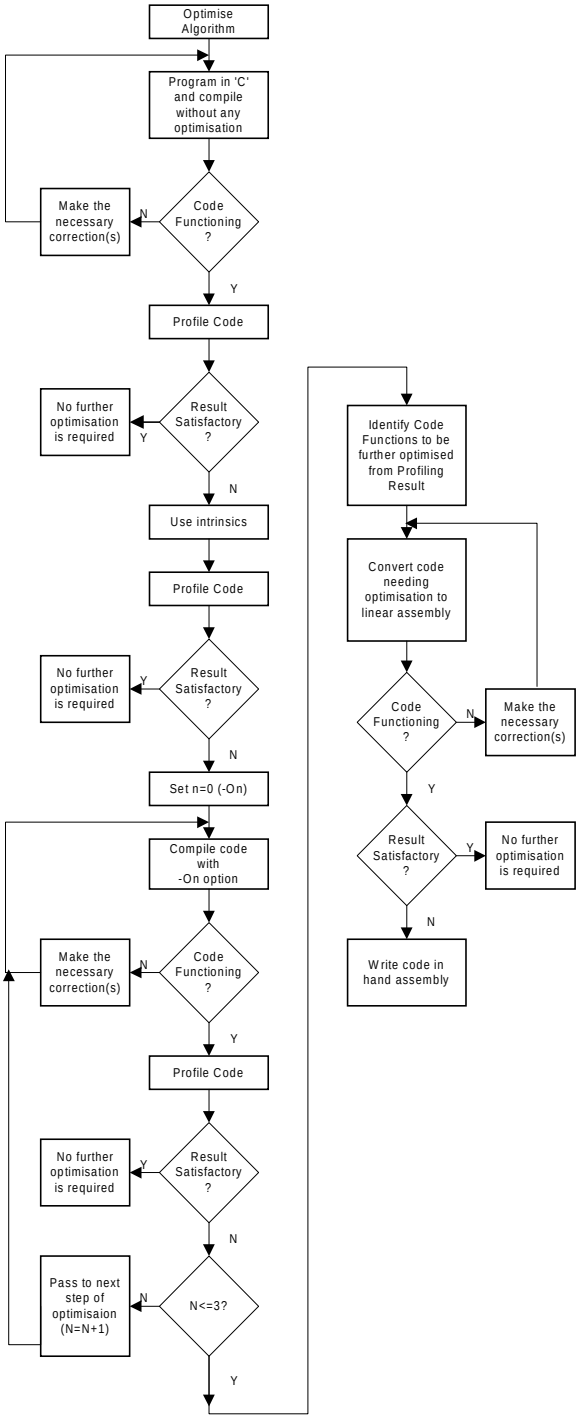

Figure 3 – Software Optimisation Procedure [1]

Prior to writing the initial C code, the algorithms were implemented in MATLAB to initially check they functioned correctly. They were then implemented in C for execution on a PC and again tested. Following this the necessary alterations were made for implementation in a fixed-point format.

One of the advantages of writing code in C is that the code is portable, however, the major disadvantage is that the code may be severely inefficient. If the portability is discarded and the C code written with some knowledge

of the processor in mind, then improvements can be made. Intrinsics can also be used to improve C code.

To summarise, the overall procedure for developing a highly optimised algorithm was adopted:

[1] MATLAB
[2] C for PC
[3] C using fixed point arithmetic (include intrinsics)
[4] Linear Assembly
[5] Assembly (Un-pipelined)
[6] Assembly (Pipelined)

4 IMPLEMENTATION AND OPTIMISATION

To allow implementation of the algorithms developed in C, the matrix-based equations were converted into normal equations.

$$y(n)=\sum_{k=0}^{N-1} h(k)x(n-k) \tag{10}$$

LMS Equations:

$$h_n(k)=h_{n-1}(k)+\mu e x(n-k), \quad k=0,1,2,\ldots,N-1 \tag{11}$$

RLS Equations:

$$R_{l,m}^{-1}(n)=R_{l,m}^{-1}(n-1)-\frac{\sum_{j=1}^{N}\left[x_j(n)\left(\sum_{i=1}^{N}R_{l,i}^{-1}(n-1)x_i(n)\right)R_{j,m}^{-1}(n-1)\right]}{1+\sum_{i=1}^{N}\left[x_i(n)\left(\sum_{j=1}^{N}R_{i,j}(n-1)x_j(n)\right)\right]} \tag{12}$$

$$H_k(n)=H_k(n-1)+e\sum_{i=1}^{N}R_{k,i}^{-1}(n)x_i(n) \tag{13}$$

The flow charts shown in Figure 4 were then used to outline the required logical flow of both algorithms [8].

The two algorithms were implemented within an interrupt function so that whenever a new value was sampled by the CODEC the function was called so that the tap weights could be updated and the filtering be performed. All the filters implemented comprised of eight taps.

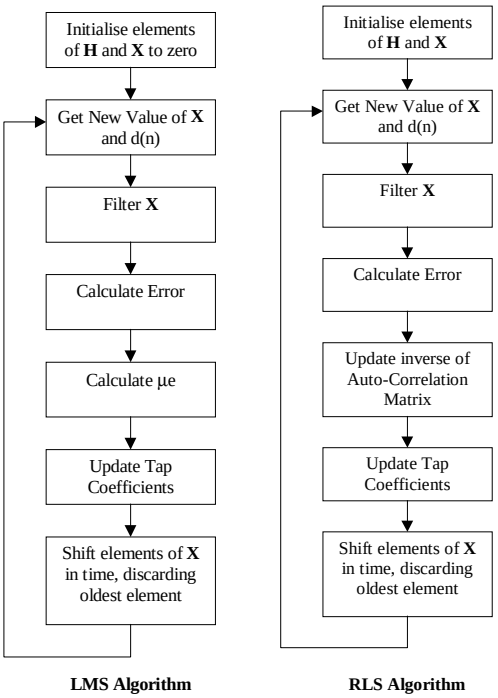

Figure 4 – LMS and RLS Algorithm Flow-Charts

The procedures for optimising the code highlighted earlier were followed, and benchmarking performed at each stage. The result of this benchmarking is summarised in Table 2, followed by an evaluation of the efficiency of the C compiler in Table 3.

Optimisation Level	LMS	Complex LMS	RLS
C	1020	-	44035
C+o3	196	-	7947
C+intrinsics	979	2261	38417
C+intrinsics+o3	197	495	10149
Linear Assm	302	-	24732
Linear Assm + Pipelining	76	-	12626
Assembly	42	-	-
Fully Optimised Assembly	16	-	-

Table 2 – Benchmarking for one Iteration of each Eight Tap Algorithm

	Efficiency of C Compiler	
Comparison	LMS	RLS
Linear Assm + Pipeling	39%	159%
Assembly	21%	-
Fully Optimised Assembly	8%	-

Table 3 – Evaluation of the Efficiency of the Optimising C Compiler

5 TESTING OF THE IMPLEMENTED ALGORITHMS

A simple set-up where the desired signal was also the input to the filter was used. The implemented algorithms were tested with a variety of input and initialisation conditions. Figure 5(a) and (b) shows the LMS and RLS algorithms adapting and Figure 5(c) shows the output from the complex LMS adaptive filter in the form of an I-Q adapting constellation diagram. Note that the testing environments were effectively noise-free.

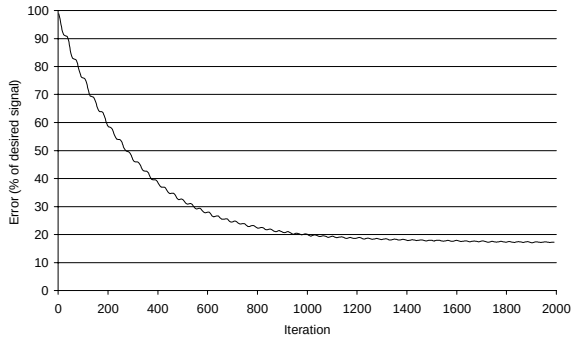

Figure 5(a) – Estimation Error as a Function of Iteration number for the LMS Algorithm

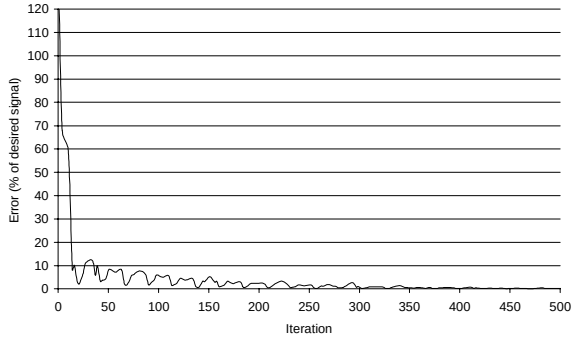

Figure 5(b) – Estimation Error as a Function of Iteration number for the RLS Algorithm

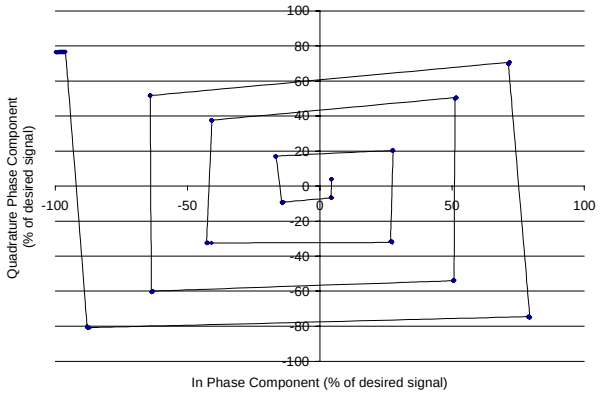

Figure 5(c) – Adapting constellation diagram for the Complex LMS Adaptive Filter

6 COMPARISON OF IMPLEMENTED LMS AND RLS ALGORITHMS

Besides the usual comparison made in various adaptive signal processing books [5][6][11][15], the results found allow the LMS and RLS algorithms to be compared in terms of speed, total time to converge and code size. (Based on a clock speed of 200MHz)

Algorithm and Implementation	Max Sample Rate	Total Time to Converge	Code Size (bytes)	Data Mem. (bytes)
LMS (C)	1.02MHz	980µs	1120	36
LMS Linear Assembly	2.63MHz	380µs	192	70
LMS Optimised Assembly	12.5MHz	80µs	288	0*
RLS (C)	25.2kHz	1192µs	2720	164
RLS Linear Assembly	15.8kHz	1894µs	864	324

*Only registers were used and these were not saved or restored in between interrupts as the processor was idle.

Table 4 – Quantitative Comparison of the LMS and RLS Algorithms

The total time to converge is effectively the amount of processing time that is required to achieve a pre-defined error level. The values given in Table 4 are based on the algorithms working at the maximum possible sampling rate and the LMS taking 1000 cycles to converge and the RLS 30, both of which are typical values in moderate noise conditions [10].

In the linear assembly implementations the required data memory was larger as integers were used instead of shorts for **X** and **H** (and **R** in the RLS) to reduce the number of shift instructions required in fixed point arithmetic.

A comparison of the level of optimisation achieved showed that the LMS was highly optimised by following the procedures outline in Section 3. However, the procedures did not increase the optimisation of the RLS in terms of cycles required to complete one iteration.

To achieve higher optimisation for the RLS, the linear assembly code would have to be rewritten with a more intricate pre-planning.

The investigations also indicate that for the case of the LMS, the optimising C compiler was only 8% efficient compared to hand optimised assembly.

7 CONCLUSION

The results presented in this paper give an insight into some of the implementation factors that need to be considered when using the TMS320C62x.

Although the actual processing time to converge for both of the algorithms implemented in C is very similar, in general, the RLS algorithm is preferred as it is more reliable with change in the noise environment, although it can suffer 'blow up' due to several factors. The RLS algorithm also requires less iterations to converge hence allowing shorter training sequences than that required for the LMS.

The LMS does have the advantage that it will always be able to complete one iteration faster than the RLS, as shown. Hence in high bit rate systems it may be the only algorithm that can be used. It has also proved to be much simpler to implement, requiring less CPU resources, program and data memory than the RLS algorithm.

However, the LMS algorithm presented has been implemented in a highly optimised form, whereas there is still scope for optimising the RLS further and it is therefore possible that the implemented RLS could possibly out perform the LMS in terms of time to converge.

REFERENCES

[1] Dahnoun, N., "DSP Implementation using the TMS320C62x Processors", Addison Wesley, to be published late 1999.

[2] Dahnoun, N., Tong, H.K., "Software Optimisation for Modems using TMS320C62xx Digital Signal Processor (DSP)", 9th International Conference on Signal Processing Applications and Technology (ICSPAT), Toronto 1998.

[3] Dahnoun, N., Tong, H.K., "Implementation of JPEG Image Coding standard on the TMS320C62xx", Texas Instrument's Second European DSP Education and Research Conference. ESIEE Paris September 1998.

[4] Dahnoun, N., Tong, H.K., "Viterbi Decoder Design and Implementation using TMS320C62xx Digital Signal Processor (DSP)", Texas Instrument's Second European DSP Education and Research Conference. ESIEE Paris September 1998.

[5] Haykin, S., "Adaptive Filter Theory", Prentice-Hall, 1991.

[6] Haykin, S., "Adaptive Signal Processing", Prentice-Hall, 1986.

[7] Haykin, S., "Communication Systems", Wiley, 3rd Edition, 1994.

[8] Ifeachor, E., Jervis, B., "Digital Signal Processing: A Practical Approach", Addison Wesley, 1998.

[9] McClellan, J.H., Schafer, R.W., Yoder, M.A., "DSP First – A Multimedia Approach", Prentice-Hall, 1998.

[10] Mulgrew, B., Grant, G., Thompson, J., "Digital Signal Processing: Concepts and Applications", Macmillian Press Ltd., 1999.

[11] Texas Instruments, "TMS320C62x/C67x Programmers Guide", Texas Instruments (SPRU198B), 1998.

[12] Texas Instruments, "TMS320C62x/C67x CPU and Instruction Set", Texas Instruments (SPRU189C), 1998.

[13] Texas Instruments, "TMS320C6x Evaluation Module", Texas Instruments (SPRU269), 1998.

[14] Texas Instruments, "TMS320C6x Optimizing C Compiler", Texas Instruments (SPRU187C), 1998.

[15] Widrow, B., "Adaptive Signal Processing", Prentice-Hall, 1985.

[16] Widrow, B., McCool, J., Ball, M., "The Complex LMS Algorithm", Proceedings of the IEEE, pp719-720, 1975.

ACKNOWLEDGEMENTS

The authors would like to thank Robert Owen, Hans Peter Blaettel, Neville Bulsara, Greg Peake and Maria Ho of Texas Instruments for their continuous help and support.

M. Hart would also like to thank everyone concerned at Fujitsu Telecom R&D Centre, especially Sunil Vadgama, for sponsoring his MSc at the University of Bristol.