Impact of algorithm design in implementing real-time active control systems

Impact of Algorithm Design in Implementing Real-time Active Control Systems

M A Hossain[1], M O Tokhi[2] and K P Dahal[3]

[1,3]Department of Computing, School of Informatics
The University of Bradford, Bradford, BD7 1DP, UK.
[2]Department of Automatic Control & Systems Engineering,
The University of Sheffield, Sheffield S1 3JD, UK
Email: m.a.hossain1@bradford.ac.uk;

Abstract. This paper presents an investigation into the impact of algorithm design for real-time active control systems. An active vibration control (AVC) algorithm for flexible beam systems is employed to demonstrate the critical design impact for real-time control applications. The AVC algorithm is analyzed, designed in various forms and implemented to explore the impact. Finally, a comparative real-time computing performance of the algorithms is presented and discussed to demonstrate the merits of different design mechanisms through a set of experiments.

1. Introduction

Although computer architectures incorporate fast processing hardware resources, high performance real-time implementation of an algorithm requires an efficient design and software coding of the algorithm so as to exploit special features of the hardware and avoid associated problems of the architecture. This paper presents an investigation into the analysis and design mechanisms that will lead to reduction in execution time in implementing real-time control algorithms. Active vibration control (AVC) of a simulated flexible beam based on finite difference (FD) method is considered to demonstrate the effectiveness of the proposed methods.

In practice, more than one algorithm exists for solving a specific problem. Depending on its formulation, each can be evaluated numerically in different ways. As computer arithmetic is of finite accuracy, different results can evolve, depending on the algorithm used and the way it is evaluated. On the other hand, the same computing domain could offer different performances due to variation in the algorithm design and in turn, source code implementation. The choice of the best algorithm for a given problem and for a specific computer is a difficult task and depends on many factors, for instance, data and control dependencies of the algorithm, regularity and granularity of the algorithm and architectural features of the computing domain [1], [2].

The ideal performance of a computer system demands a perfect match between machine capability and program behaviour. Program performance is the turnaround time, which includes, disk and memory accesses, input and output activities, compilation

time, operating system overhead, and CPU time. In order to shorten the turnaround time, one can reduce all these time factors. Minimising the run-time memory management, efficient partitioning and mapping of the program for concurrent system, and selecting an efficient compiler for specific computational demands, could enhance the performance. Compilers have a significant impact on the performance of the system. This means that some high-level languages have advantages in certain computational domains, and some have advantages in other domains. The compiler itself is critical to the performance of the system as the mechanism and efficiency of taking a high-level description of the application and transforming it into a hardware dependent implementation differs from compiler to compiler [3], [4], [5].

This paper addresses the issue of algorithm analysis, design and software coding for real-time control in a generic manner. A number of design methodologies are proposed for the real-time implementation of a complex control algorithm. The proposed methodologies are exemplified and demonstrated with simulation algorithm of an AVC system for a flexible beam. Finally, a comparative performance assessment of the proposed design mechanisms is presented and discussed through a set of experimental investigations.

2. Active Vibration Control Algorithm

Consider a cantilever beam system with a force $U(x,t)$ applied at a distance x from its fixed (clamped) end at time t. This will result in a deflection $y(x,t)$ of the beam from its stationary position at the point where the force has been applied. In this manner, the governing dynamic equation of the beam is given by

$$m^2 \frac{\partial^4 y(x,t)}{\partial x^4} + \frac{\partial^2 y(x,t)}{\partial t^2} = \frac{1}{m} U(x,t) \tag{1}$$

where, m is a beam constant and m is the mass of the beam. Discretising the beam in time and length using the central FD methods, a discrete approximation to equation (1) can be obtained as [6]:

$$Y_{k+1} = -Y_{k-1} - l^2 S Y_k + \frac{(\Delta t)^2}{m} U(x,t) \tag{2}$$

where, $l^2 = \left[(\Delta t)^2 / (\Delta x)^4\right] m^2$ with Δt and Δx representing the step sizes in time and along the beam respectively, S is a pentadiagonal matrix (the so called stiffness matrix of the beam), $Y_i \ \ (i = k+1,\ k,\ k-1)$ is an $(n-1)\times 1$ matrix representing the deflection of end of sections 1 to n of the beam at time step i (beam divided into $n-1$ sections). Equation (2) is the required relation for the simulation algorithm that can be implemented on a computing domain easily.

A schematic diagram of an AVC structure is shown in Figure 1. A detection sensor detects the unwanted (primary) disturbance. This is processed by a controller to gen-

erate a canceling (secondary, control) signal so that to achieve cancellation at the observation point. The objective in Figure 1 is to achieve total (optimum) vibration suppression at the observation point. Synthesizing the controller on the basis of this objective yields [7]

$$C = \left[1 - \frac{Q_1}{Q_0}\right]^{-1} \tag{3}$$

where, Q_0 and Q_1 represent the equivalent transfer functions of the system (with input at the detector and output at the observer) when the secondary source is *off* and *on* respectively.

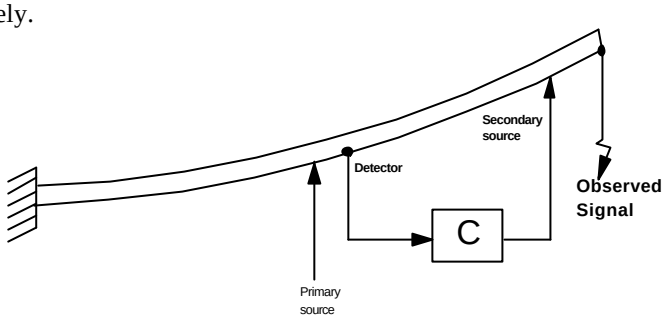

Fig. 1. Active vibration control structure

To investigate the nature and real-time processing requirements of the AVC algorithm, it is divided into two parts, namely control and identification. The control part is tightly coupled with the simulation algorithm, and both will be described in an integral manner as the control algorithm. The simulation algorithm will also be explored as a distinct algorithm. Both of these algorithms are predominately matrix based. The identification algorithm consists of parameter estimation of the models Q_0 and Q_1 and calculation of the required controller parameters according to equation (3). However, the nature of identification algorithm is completely different as compared with the simulation and control algorithms [8]. Thus, for reasons of consistency only the simulation and control algorithms are considered in this investigation.

3. Algorithm Analysis and Design

3.1 Flexible Beam Simulation Algorithm

The flexible beam simulation algorithm forms a major part of the control algorithm. Thus, of the two algorithms, the simulation algorithm has higher impact due to data dependency on real-time AVC. To demonstrate the real-time implementation impact, the simulation algorithm is designed in seven different methods [9, 10]. Three of these are considered here to explore real-time AVC. These are briefly described below.

Simulation Algorithm–1: Shifting of data array. The 'Simulation Algorithm–1' incorporates design suggestions made by Hossain, 1995 [8], is listed in Figure 2. It is noted that complex matrix calculations are performed within an array of three elements each representing information about the beam position at different instants of time. Subsequent to calculations, the memory pointer is shifted to the previous pointer in respect of time before the next iteration. This technique of shifting the pointer does not contribute to the calculation efforts and is thus a program overhead. Other algorithms were deployed to address this issue at further levels of investigation.

```
Loop {

//Step 1
  y0[2]=-y0[0]-lamsq*(a*y0[1]-4*y1[1]+y2[1]);
  y1[2]=-y1[0]-lamsq*(-4*y0[1]+b*y1[1]-4*y2[1]+y3[1]);
        :
  y18[2]=-y18[0]-lamsq*(y16[1]-4*y17[1]+c*y18[1]-2*y19[1]);
  y19[2]=-y19[0]-lamsq*(2*y17[1]-4*y18[1]+d*y19[1]);

//Step 2
  // Shifting memory locations
  y0[0]=y0[1]; y0[1]=y0[2]; y1[0]=y1[1]; y1[1]=y1[2];
        :
  y18[0]=y18[1]; y18[1]=y18[2];  y19[0]=y19[1]; y19[1]=y19[2];
}
```

Fig. 2. Design outline of 'Simulation Algorithm–1'

Simulation Algorithm–2: Array rotation. The 'Simulation Algorithm–2' incorporates design suggestions made by Hossain et al, 2000 [9]. A listing of the algorithm is given in Figure 3. In this case, each loop calculates three sets of data. Instead of shifting the data of the memory pointer (that contains results) at the end of each loop, the most current data is directly recalculated and written into the memory pointer that contains the older set of data. Therefore, re-ordering of array in the 'Simulation Algorithm–1' is replaced by recalculation. The main objective of the design effort is to achieve better performance by reducing the dynamic memory allocation and, in turn, memory pointer shift operation. Thus, instead of using a single code block and data-shifting portion, as in 'Simulation Algorithm–1', to calculate the deflection, three code blocks, are used with the modified approach in 'Simulation Algorithm–2'. Note that in 'Simulation Algorithm–2', the overhead of 'Simulation Algorithm-1' due to memory pointer shift operation is eliminated and every line of code is directed towards the simulation effort.

Simulation Algorithm–3: Two-element array rotation. The 'Simulation Algorithm–3' is listed in Figure 4. This makes use of the fact that access to the oldest time segment is only necessary during re-calculation of the same longitudinal beam segment. Hence, it can directly be overwritten with the new value. The 'Simulation Algorithm–3' is optimized for the particular discrete mathematical approximation of the governing physical formula, exploiting the previously observed features.

```
Loop {

//Step 1
   y0[2]=-y0[0]-lamsq*(a*y0[1]-4*y1[1]+y2[1]);
   y1[2]=-y1[0]-lamsq*(-4*y0[1]+b*y1[1]-4*y2[1]+y3[1]);
         :
   y18[2]=-y18[0]-lamsq*(y16[1]-4*y17[1]+c*y18[1]-2*y19[1]);
   y19[2]=-y19[0]-lamsq*(2*y17[1]-4*y18[1]+d*y19[1]);

//Step 2
   y0[0]=-y0[1]-lamsq*(a*y0[2]-4*y1[2]+y2[2]);
   y1[0]=-y1[1]-lamsq*(-4*y0[2]+b*y1[2]-4*y2[2]+y3[2]);
         :
   y18[0]=-y18[1]-lamsq*(y16[2]-4*y17[2]+c*y18[2]-2*y19[2]);
   y19[0]=-y19[1]-lamsq*(2*y17[2]-4*y18[2]+d*y19[2]);

 //Step 3
   y0[1]=-y0[2]-lamsq*(a*y0[0]-4*y1[0]+y2[0]);
   y1[1]=-y1[2]-lamsq*(-4*y0[0]+b*y1[0]-4*y2[0]+y3[0]);
         :
   y18[1]=-y18[2]-lamsq*(y16[0]-4*y17[0]+c*y18[0]-2*y19[0]);
   y19[1]=-y19[2]-lamsq*(2*y17[0]-4*y18[0]+d*y19[0]);

}
```

Fig. 3. Design outline of 'Simulation Algorithm–2'

```
 Loop {

    // Step 1
    y0[0]=-y0[0]-lamsq*(a*y0[1]-4*y1[1]+y2[1]);
    y1[0]=-y1[0]-lamsq*(-4*y0[1]+b*y1[1]-4*y2[1]+y3[1]);
          :
    y18[0]=-y18[0]-lamsq*(y16[1]-4*y17[1]+c*y18[1]-2*y19[1]);
    y19[0]=-y19[0]-lamsq*(2*y17[1]-4*y18[1]+d*y19[1]);

    // Step 2
   y0[1]=-y0[1]-lamsq*(a*y0[0]-4*y1[0]+y2[0]);
    y1[1]=-y1[1]-lamsq*(-4*y0[0]+b*y1[0]-4*y2[0]+y3[0]);
          :
    y18[1]=-y18[1]-lamsq*(y16[0]-4*y17[0]+c*y18[0]-2*y19[0]);
    y19[1]=-y19[1]-lamsq*(2*y17[0]-4*y18[0]+d*y19[0]);
 }
```

Fig. 4. Design outline of 'Simulation Algorithm–3'

3.2 AVC Algorithm

As mentioned earlier, the AVC algorithm consists of the beam simulation algorithm and control algorithm. For simplicity the control algorithm in equation (3) can be rewritten as a difference equation as in Figure 5 [8], where b0, …, b4 and a0, …, a3 represent controller parameters. The arrays y12 and yc denote input and controller output, respectively. It is noted that the control algorithm shown in Figure 5 has similar design and computational complexity as one of the beam segment described and discussed in 'Simulation Algorithm-1'. Thus, the control algorithm can also be rewritten for recalculation in a similar manner as discussed in 'Simulation Algorithm-2' and 'Simulation Algorithm-3'.

```
yc[n]=b0*y12[n] + b1*y12[n-1] + b2*y12[n-2] + b3*y12[n-3]+ b4*y12[n-4]-(a0*yc[n-
1]+a1*yc[n-2] +a2*yc[n-3] +a3*yc[n-4]);
//Shift data array
   y12[n-4]=y12[n-3] ; y12[n-3]=y12[n-2] ; y12[n-2]=y12[n-1] ; y12[n-1]=y12[n] ;
   yc[n-4]=yc[n-3] ; yc[n-3]=yc[n-2] ; yc[n-2]=yc[n-1] ; yc[n-1]=yc[n] ;
```

Fig. 5. Design outline of the control algorithm (data array shifting method)

4. Implementation and Results

The AVC algorithms based on three different methods of the simulation and corresponding similar design of the control algorithms were implemented with similar specification [7], with 0.3ms sampling time. It is worth mentioning that the AVC Algorithm-1 was implemented combining the 'Simulation Algorithm-1' and the data array shift method of control algorithm as shown in Figure 5. The AVC Algorithm-2 implemented in combination of the 'Simulation Algorithm-2' and similar recalculation method of control algorithm. Finally, AVC Algorithm-3 was implemented combining the 'Simulation Algorithm-3' and similar recalculation method of control algorithm. For reasons of consistency, a fixed number of iterations (250,000) were considered in implementing all the algorithms. Therefore, the execution time should be 75 sec in implementing each algorithm to achieve real-time performance.

Figure 6 depicts a comparative performance of the AVC Algorithm–1 and Algorithm–2 for 20 to 200 beam segments. It is noted that the execution time for both algorithms increases almost linearly with the increment of the number of segments. It is also noted that Algorithm-2 performed better throughout except for 100 segments.

Figure 7 shows a comparative real-time performance of implementing Algorithm-2 and Algorithm-3. It is observed that Algorithm-3 performs better throughout except for smaller number of segments. It is also noted that the performance variation of Algorithm-3 as compared to Algorithm-2 was not linear and performed best when the number segments was 80. This is further demonstrated in Table 1, which shows the performance ratio of Algorithm-2 and Algorithm-3 relative to Algorithm-1. It is observed

that the transition towards weaker performance occurred in AVC Algorithm–3 halfway between the transitions of Algorithm–1 and Algorithm–2. In spite of being outperformed by Algorithm–1 in a narrow band of around 100 segments, Algorithm–3 offered the best performance overall. Thus, the design mechanism employed in Algorithm–3 can offer potential advantages in real-time control applications.

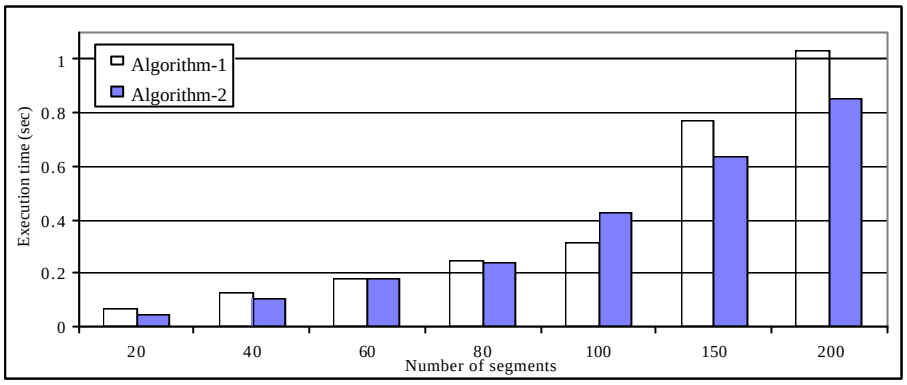

Fig. 6. Performance comparison of Algorithm–1 and Algorithm–2

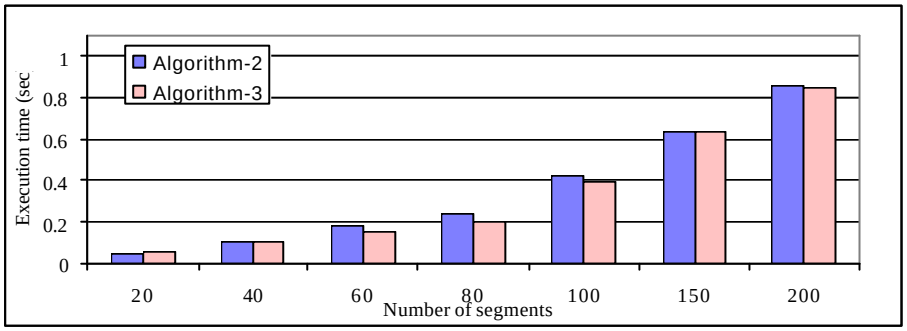

Fig. 7. Performance comparison of Algorithm–2 and Algorithm–3

Table 1: Performance ratio of Algorithm–2 (A2) and Algorithm–3 (A3) as compared to Algorithm–1 (A1).

Segments	20	40	60	80	100	150	200
A2/A1	0.67	0.83	1.0	1.4	1.6	0.83	0.83
A3/A1	0.83	0.83	0.83	0.83	1.3	0.83	0.82

5. Conclusion

An investigation into algorithm analysis, design, software coding and implementation so as to reduce the execution time and, in turn, enhance the real-time performance, has been presented within the framework of real-time implementation of active vibration control algorithms. A number of approaches have been proposed and demonstrated experimentally with the AVC algorithm of a flexible beam system. It has been observed that all three algorithms have achieved real-time performance. Although, execution time and in turn, performance of the algorithm varies with different approaches. Designs leading to large instructions cause non-linear transitions at certain stages where internal built-in instruction cache is unable to handle the load. It is also noted that such transitions with the AVC algorithm considered occur for computation of different number of segments. Thus, none of the designed algorithms performed best for the whole range of computation. Therefore, identification of the suitability of source code design and implementation mechanism for best performance is a challenge.

References

1. Thoeni, U. A: Programming real-time multicomputers for signal processing, Prentice-Hall, Hertfordshire, UK (1994).
2. Tokhi, M. O., Hossain, M. A: CISC, RISC and DSP processors in real-time signal processing and control, Journal of Microprocessors and Microsystems, Vol. 19(1995), pp.291-300.
3. Bader, G. and Gehrke, E: On the performance of transputer networks for solving linear systems of equation, Parallel Computing, 1991, Vol. 17 (1991), pp. 1397-1407.
4. Tokhi, M. O., Hossain, M. A., Baxter, M. J. and Fleming, P. J: Heterogeneous and homogeneous parallel architectures for real-time active vibration control, IEE Proceedings-D: Control Theory and Applications, Vol. 142, No. 6 (1995), pp. 1-8.
5. Tokhi, M. O., Hossain, M. A., Baxter, M. J. and Fleming, P. J: Performance evaluation issues in real-time parallel signal processing and control, Journal of Parallel and Distributed Computing, Vol. 42 (1997), pp. 67-74.
6. Kourmoulis, P. K: Parallel processing in the simulation and control of flexible beam structure system, PhD. Thesis, Department of Automatic Control and Systems Engineering, The University of Sheffield, UK (1990).
7. Tokhi, M. O. and Hossain, M. A: Self-tuning active control of noise and vibration, Proceedings IEE - Control Conference-94, Vol. 1, Coventry, UK (1994), pp. 263-278.
8. Hossain, M. A: Digital signal processing and parallel processing for real-time adaptive noise and vibration control, Ph.D. thesis, Department of Automatic Control and System Engineering, The University of Sheffield, UK (1995).
9. Hossain, M. A., Kabir, U. and Tokhi, M. O: Impact of data dependencies for real-time high performance computing, Journal of Microprocessors and Microsystems, Vol. 26, No. 6 (2002), pp. 253 – 261.
10. Tokhi, M. O., Hossain, M. A. and Shaheed, M. H: Parallel Computing for Real-time Signal Processing and Control, Springer Verlag, ISBN: 1-85233-599-8, (2003).