Synchronous Systems for Behaviour Based Robot Control

Martin D. Westhead

February 23, 1995

Abstract

The operation of a robot is fundamentally tied to the the dynamics of the interaction of the the robot control system, the physical robot and the environment. Conventional programming techniques will either ignore the controller's dynamics or make many implicit assumptions about them. This has prompted some [Smi94] to suggest that classical models of computation should be completely abandoned in favour of a dynamical systems approach, where the entire system is described and implemented in terms of continuous differential equations. A less extreme, and potentially more tractable approach is that taken by synchronous systems. The synchrony hypothesis allows the programmer to abstract away many of the implicit timing problems faced in robot control. It is argued that this model could be usefully applied to Behaviour Based robot control

1 Introduction

Anyone who has programmed a robot will be aware of how closely tied the interaction of the robot is with the real world. The timings of these interactions can be critical: Can the robot stop in time to avoid the cliff edge? Can the arm move fast enough to catch the ball?

Many of the computational models we work with today, like the Turing Machine are based on the idea of an information processing system. A program has an input and an output and we are concerned only with the mapping between the two. There is no notion of time in the system, only the number of steps the machine takes to complete the problem. The system can only respond to a new problem once it has finished the current one. It is possible to augment this model with the addition of an interrupt system, that might give some degree of reactivity, but it doesn't solve the problem. The model even with interrupts presupposes that the principal type of computation is information processing.

The problem with this model involves the timing of the interaction of the robot control system and the real world. The languages we use for programming robots will generally be poorly equipped for real time programming. Even in languages such as Brooks' BL [Bro90] which are explicitly designed for real time use, the timing mechanisms are very haphazard.

An increasing number of workers [Smi94, vG94] believe that this is the wrong model for programming robots. They propose that we should abandon notions of computation altogether and model the whole interaction (robot, controller and environment) as a continuous dynamical system. This is a very difficult thing to try to do, and although some simple systems have been successfully analysed in this way [Bee93] it is far from clear if this approach will ever lead to practical techniques for programming robots.

This paper considers an another approach: that of synchronous systems. Several synchronous languages already exist, (Lustre [HCRP91], Argos [Mar91], Esterel [BG92]). They are designed for real time reactive control, and are being applied to a number of problems including robotics.

2 Reactive systems

Classical robotics tends to focus on the robot's *actions on* the environment. The dynamical systems perspective [Smi94] considers the robot's *interaction with* the environment. In contrast synchronous systems consider the robot to be a reactive agent, and think about its *reactions to* the environment.

There are many examples of computer applications, involving programs that maintain a continuous interaction with the environment, reacting to inputs by sending out outputs. The examples include computer games, word processors, digital watches, in fact most modern user interfaces. Such programs are called *reactive programs* [HP85], and the systems they control are called *reactive systems*. Every action of such a system is triggered by an event in the environment that causes an input signal.

Reactive programs can be thought of in terms of three components (taken from [BG92]:

- **Interface** — deals with input from and output to the environment. Handles interrupts, the reading of sensors, and the activation of actuators.
- **Reactive Kernel** — this is the part of the system that handles the real time control. It decides based on the system input, which signals should be output by the interface component, and which computations should be triggered in the data handling component.
- **Data handling component** — these are standard information processing routines that perform classical computations at the request of the kernel. They will typically run asynchronously with the kernel.

Synchronous system's provide a way of constructing the reactive kernel of a system which is the central and most difficult part.

2.1 Concurrency

In order for our reactive kernel to be capable of simultaneously reacting to a number of inputs, the system needs to allow concurrency. It could be argued that parallelism in a robot control system can provide speed (through the use of multiple processors) and robustness (even if one component fails the others can continue to operate). However most robot systems simulate parallelism on a single CPU, in this case neither of the above apply. Indeed even if we use more than one processor it is not clear that best decomposition for reactivity would provide us with a decomposition suited to obtaining significant speedup or robustness from a parallel machine.

It seems clear that the most important reason for parallelism therefore is for design description. We would like to be able to specify that if a soda can is detected we pick it up but *at the same time* if a cliff edge is spotted ahead we stop advancing. Of course we could construct a module that detected cans and cliff edges and stopped at one, and picked the other up, but ideally we would like to keep the two separate and run them in parallel. This allows the problem of the overall behaviour to be decomposed in terms of relatively independent behavioural modules, it is fundamentally what is behind the behaviour based decomposition.

2.2 Determinism

Most available parallel control systems are asynchronous in operation. That is each of the parallel components is timed independently of the others. Brooks' BL [Bro90] is an example of this as are ADA, OCCAM, and all concurrent operating systems environments. The big disadvantage with asynchronous systems is that they are non-deterministic in nature. That is the sequence of outputs that is generated depends not only on the sequence of inputs, but also on subtle timing interactions that can occur within the controller. In other words the internal dynamics of the controller itself can be critical in determining the systems operation. Non-deterministic systems are much harder to specify, debug and analyse than deterministic ones because all possible outputs must be considered at each stage.

A purely sequential system is deterministic, however a system does not have to be sequential to be deterministic. The non-determinism in all the systems mentioned above arises because they are based on asynchronous models of parallelism. The timing of interactions between processes is not specified, and so can result in more than one possible outcome.

3 Synchronous systems

Synchronous systems allow the programmer to abstract away many of the internal, and external timing problems, and consider the control program in terms of sequence. This is done by adopting the *synchrony hypothesis*: "Every reaction is assumed to be *instantaneous*". In other words we assume that the system reacts in *no time*. Two assumptions are being made here:

1. A reaction takes no time with respect to the external environment
2. Each subprocess reacts in no time with respect to the other internal subprocesses

This last point is critical. Communication in synchronous systems is achieved by broadcast communication. Since all the processes in the system react together, and instantaneously, they all share the same environment (of internal and external signal values). Consequently the program's operation is completely deterministic.

Time can be incorporated as an input to the system. Statements take time *if and only if* they say they do. The statement "`await 10 SECOND`" will wait for 10 inputs of the signal `SECOND`. This means that:

- Statements take exactly as long to execute as they say they will "`await 10 SECOND`" will wait exactly 10 seconds. "`every 60 SECOND do emit MINUTE end`" will emit a `MINUTE` exactly on the 60th second. This can't be done using an asynchronous system.
- Other signals such as distance are treated in exactly the same way as time. You could as easily write:"`every 100 CENTIMETRES do emit METER end`"

Treating computation time in this way is very useful. All delays must be explicit, and so their presence, and duration is explicit in the code. Time can be, and often is, used in behaviour based systems to mark changes in the real world. For example the operation of BL and the subsumption architecture is heavily dependent on delays. It is possible to specify the distance a mobile robot moves across the floor, by turning the motors on waiting a specified time interval and then turning them off again. However using timing delay as a measure of distance is to introduce an implicit model of the robot's operation. It is far better to sense the distance directly, and "use the real world to model itself" as Brooks would have it.

4 Justifying infinitely fast computation

The notion of any computation taking place in "no time" might sound a little hard to swallow, particularly to the ears of AI workers who are often faced with large NP-hard search problems. Clearly the computations will take time, the question is: Is the assumption of instantaneous computation in the reactive controller a reasonable one for the types of control system that we want to design? In other words can it provide us with a useful model? In fact for most of the control problems that are being tackled in the Behaviour Based field it probably can.

There are many precedents for such a hypothesis in classical science, Newtonian mechanics for instance assumes instantaneous interactions between bodies. Kirchoff's laws assume the instantaneous propagation of electricity. VLSI circuit design assumes that all reactions take place within a single clock cycle.

In order for the hypothesis to be considered reasonable here, we only need to ensure that the time taken for these computations is negligible with respect to the dynamics of the robot-environment interaction. Due to the efficiency of the compiled code, this should be the case for a large proportion of robot systems with ordinary micro-processor control.

The compilers for synchronous languages are extremely efficient. Because of the determinism all parallelism and internal communication can be compiled away [1] to leave a very efficient piece of sequential executable code. So a single reaction could occur in a matter of tens of machine cycles.

For the most part the computations carried out by the reactive kernel of a system will be relatively simple. Large planing problems, vision problems, matrix manipulations etc, are all handled by the data processing components of the system.

Of course the computation cannot actually happen infinitely fast, however it is a simplifying assumption that makes the design and analysis of the system much easier. Providing the design rules are not violated (i.e. problems that are going to be computationally intensive are done outside the central kernel) it should provide a reasonable model for a large range of robot control problems.

5 Conclusion

Synchronous systems provide a number of advantages over existing techniques for programming robots. Most significantly:

- They simplify the complex problem of concurrent programming by providing a clearly defined deterministic programming model.
- They simplify the nature of the interaction between the robot controller and the outside world, by allowing us to assume that computation delays are negligible with respect to the dynamics of the system. Hence consideration of the dynamics of the

[1] Internal communication really does happen infinitely fast since it occurs at compile time.

system can be restricted to the plant (sensors and actuators) and its interaction with the environment.

It is hoped that the introduction of synchronous systems may pave the way for the increasing use of theoretical methodologies in the design and analysis of behaviour based robots.

References

[Bee93] Randall D. Beer. A dynamical systems perspective on autonomous agents. *Artificial Intelligence*, 1993. Unsure of year, submitted for a special issue on Computational Theories of Interaction.

[BG92] G. Berry and G. Gonthier. The ESTEREL synchronous programming language: design, semantics, implementation. *Science of Computer Programming*, 19(2):87–152, November 1992.

[Bro90] Rodney A. Brooks. The behaviour language; users guide. A.I. Memo 1227, Massachusetts Institute of Technology, 1990.

[HCRP91] N. Halbwachs, P. Caspi, P. Raymond, and D. Pilaud. The synchronous dataflow programming language lustre. *Proceedings of the IEEE*, 79(9):1305–1320, September 1991.

[HP85] David Harel and A. Pnueli. On the development of reactive systems: Logic and models of concurrent systems. volume 13 of *NATO ASI Series F,*, pages 477–498. NATO Advanced Study Insititute on Logics and Models for Verification and Specification of Concurrent Systems, Springer-Verlang, 1985.

[Mar91] F. Maraninchi. Argos : a graphical synchronous language for the description of reactive systems. Spectre report c29, LGI-IMAG, Grenoble, mar 1991.

[Smi94] Tim Smithers. What the dynamics of adaptive behaviour and cognition might look like in agent-environment interaction systems. In *On the Role of Dynamics and Representation in Adaptive Behaviour and Cognintion*. DRAB'94, 1994.

[vG94] Tim van Gelder. What might cognition be if not computation. In *On the Role of Dynamics and Representation in Adaptive Behaviour and Cognintion.* DRAB'94, 1994.