Real-Time Intelligent Behaviour in Dynamic Environments: Soccer-playing Robots

by

Michael K. Sahota

B.A.Sc. in Engineering Science,
University of Toronto, 1991

A THESIS SUBMITTED IN PARTIAL FULFILLMENT OF
THE REQUIREMENTS FOR THE DEGREE OF
MASTER OF SCIENCE

in

THE FACULTY OF GRADUATE STUDIES
DEPARTMENT OF COMPUTER SCIENCE

We accept this thesis as conforming
to the required standard

THE UNIVERSITY OF BRITISH COLUMBIA
August 1993

Abstract

An autonomous robot operating in a dynamic environment is confronted by the question of, "What to do now?" The problem of designing a controller for a car-like robot competing with another robot in a game of soccer is considered. This is a dynamic environment; the locations of the ball and the robots are constantly changing. Rapid and appropriate responses to changes in the world are central to intelligent behaviour.

There are no established robot architectures that seem adequate for the challenges posed in dynamic domains. Traditional work in Artificial Intelligence has focused on the construction of plans for future execution and has resulted in architectures that are not extensible to dynamic environments. Even recently developed "reactive" architectures such as the subsumption architecture, situated automata, and RAP do not seem satisfactory.

Reactive deliberation has been designed to present a set of structural elements needed in dynamic domains. Reactive deliberation makes three contributions to robot architecture. First, the decisions of what task to achieve and how to achieve it are best resolved in unison. Second, the transient goals of a robot must be evaluated at a rate commensurate with changes in the environment. Third, goal-oriented modules called behaviours are a useful abstraction that allow effective goal-arbitration and sharing of scarce computational resources.

The effectiveness of reactive deliberation has been demonstrated through a tournament of one-on-one soccer games between robots. Current functionality includes motion planning, ball shooting, and playing goal, with accurate motion control at speeds of 1 m/s. The results of the soccer tournament suggest that the architectural elements in reactive deliberation are sufficient for real-time intelligent control in dynamic environments.

Table of Contents

List of Tables

List of Figures

Acknowledgments

Without the help and support of the Clod-Buster/Dynamo group this thesis would never have happened. Stewart Kingdon, an extraordinary vision programmer and DataCube guru, is the creator of the vision system that I have relied on for months. Rod Barman handled the electronic fiddlybits and was a key participant in getting this project on the road. Credit also goes to Heath Wilkinson whose simulator I used extensively.

After a long debate last summer, I was finally able to convince Alan Mackworth to be my supervisor. This turned out to be a good thing. Over the past year, we have had a number of meetings that were so good that they went on and on and . . . Thanks to Alan, I have kept on the straight and narrow, hence avoiding the seductive trap of Brooksian robotics.

I have received invaluable comments on draft versions of this thesis; I would like to recognize the following individuals for taking the time to review it: Jeff Brewster, Stewart, Alan, Keiji Kanazawa, Ying Zhang, Dinesh Pai, and Michael Horsch.

Thanks also goes to my wife Beverly who has put up with my long hours, sudden plan changes, and late arrivals. In time, perhaps, will I be weaned off my desire to show the latest copy of the Dynamo video to everyone we know.

Chapter 1
Introduction

1.1 Motivation

Ever since the nineteen-fifties when Isaac Asimov identified robots as machines that could be helpful to society, people have dreamed of having robots that can operate side by side with human beings [Asi90]. One obvious characteristic of our environment is that it is dynamic — it changes over time. As robot designers, we need robot architectures to guide the design of robots that can function in dynamic environments.

Robot architecture is both the art and science of building robots as well as the structure of robots. The computational aspects robot architecture are of interest to computer scientists. Although this includes the type and organization of computing elements, this thesis will be restricted to considering only the software. The term *robot architecture* will be used to describe approaches taken in the design of software for sensing, thinking, and acting.

There are no established architectures that seem adequate for the challenges posed in dynamic domains. Traditional work in Artificial Intelligence has focused on the construction of plans for future execution and has resulted in architectures that are not extensible to dynamic environments. Even recently developed "reactive" architectures such as the subsumption architecture [Bro86], situated automata [KR90], and RAP [Fir89] do not seem satisfactory.

1.2 Research Goals and Objectives

The main goal of this thesis is to describe an architecture that can generate *real-time intelligent behaviour in dynamic environments.* A robot operating within the real-time constraints placed by the external environment must answer the question: "What to do now?" It is not sufficient for a robot to react and interact with its environment; it must act in goal-oriented ways to produce intelligent behaviour and not just any behaviour. Intelligent behaviour simply means that both the achievement of goals and the manner in which they are accomplished are important. The importance of real-time control is identified by the following quote: "An oncoming truck waits for no theorem prover." [Gat92] The moral is that robots operating in dynamic domains must keep pace with changes in the environment.

This thesis investigates *robotic intelligence* as opposed to *general intelligence.* Robotic intelligence is about generating intelligent behaviour in robots, while general intelligence is the pursuit of human level intelligence through connectionist and symbolic approaches.

Soccer has been chosen as a domain for experiments with mobile robots since it has characteristics prevalent in the real-world that are absent from typical robot problem domains. Soccer-playing is a dynamic environment because the ball and the cars are all moving. One feature is that the score of a game provides an objective measure of a robot's ability to function. The inadequacy of robot architectures suitable for soccer motivates the need for better architectures.

The research strategy pursued is to: (1) determine a research problem, (2) develop a solution, and (3) test the solution with an experiment. The research problem is the development of an architecture suitable for dynamic domains such as soccer. The solution presented in this thesis is a robot architectures called *reactive deliberation.* The experiment is a tournament of one-on-one games of soccer that are conducted using the *Dynamite testbed.*

The testbed consists of a collection of radio-controlled robots under visual control. The use of real-world robots to demonstrate the effectiveness of the proposed architecture is an essential component of this thesis. Hence, the construction of a system of *soccer-playing robots* is the secondary goal of this thesis.

The architecture presented in this thesis, reactive deliberation, is focussed on the problems that arise in dynamic domains. Further, it is an incomplete robot architecture that does not address important issues such as sensor processing and fusion, the development of maps, and world modeling.

1.3 Thesis Outline

The purpose of Chapter 2 is to consider the suitability of established architectures for robots operating in dynamic environments. The role of robot architecture and the challenges in the area of mobile robots provide a context for the work performed in this thesis. Landmark robotic architectures are analyzed in the context of their suitability to dynamic environments.

In Chapter 3, the principal results of this thesis are presented through a robot architecture called *reactive deliberation.* It can be viewed as a coherent summary of the essential elements of an architecture for dynamic domains.

The purpose of Chapter 4 is to demonstrate the link from theory to practice through the Dynamite testbed that has been designed for experiments with multiple mobile robots. A robot controller based on reactive deliberation has been designed so that a robot may compete with another robot in a game of soccer. A series of soccer experiments called the LCI Cup are described.

The experimental results and conclusions are contained in Chapter 5. In addition, the theoretical limitations and advantages of reactive deliberation in relation to other architectures are discussed.

Chapter 6 summarizes the contributions of this thesis.

The appendices provide greater detail on subjects not directly connected with the objective of this thesis: the vision subsystem, the simulator, and path planning algorithms.

It is possible to avoid most of the theory on robot architecture by omitting Chapters 2 and 3. This leaves the reader with the more application-oriented details in Chapters 4 and 5. On the other hand, Chapter 3 provides a detailed look at reactive deliberation and contains the bulk of the contributions of this thesis.

Chapter 2
On Robot Architecture

The purpose of this chapter is to consider the suitability of existing architectures for robots operating in dynamic environments. Before considering specific architectures, some background is introduced, such as the role of robot architecture, the challenges in the area of mobile robots, and a classification scheme for robot architectures. The body of the chapter describes landmark robot architectures belonging to each class, and a discussion of the differences is contained in the final section. Section 2.4, additional related literature, describes previous work relevant to this thesis that has not been presented as part of a coherent architecture.

2.1 Setting The Stage

There is more to robotics than just computer science; the real world is not an abstract data type. Effecting change in the environment is not as simple as writing to some output registers. Few intelligent robotic systems function without carefully monitoring changes in the environment. Usable information about the environment has to be synthesized from complex sensor data.

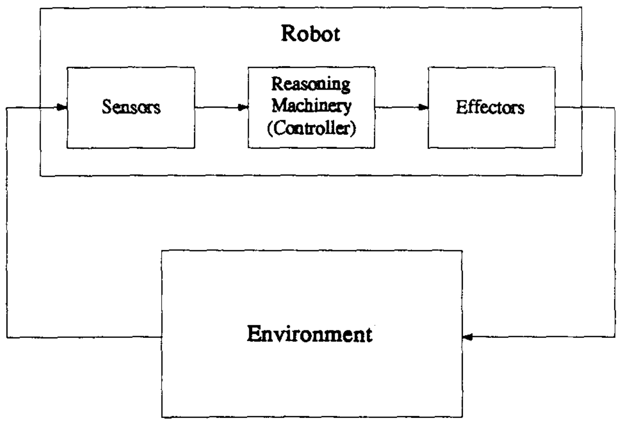

Figure 1: An abstraction of the robot domain

2.1.1 The World of Autonomous Mobile Robots

The purpose of this section is to describe the complexities faced by mobile robots. Figure 1 provides an abstract view of the domain of robots. Robots interact with the world through sensors and effectors/actuators. Sensors measure physical properties in the world such as the light impinging on a camera, or the closure of a switch (contact sensor). Effectors allow a robot to effect changes in the world. For example, the motors and steering servos in a mobile robot allow it to move about and push objects. The part of the robot labelled *reasoning machinery* is typically a monoprocessor computer, although it could be a combinational logic circuit. A robot architecture seeks to describe the content and organization of the controller.

A typical view of the design process for robots is as follows. Given a specification of the desired behaviour of the robot and a description of the robot and environment, produce a design for the robot controller [ZM92a]. The triple of sensors, actuators, and environment is considered fixed. For example, it is difficult to modify the robot hardware and nearly impossible to modify the environment. The role of robot architecture is to aid the design of a controller that will produce the desired behaviour given a fixed triple.

There are a number of characteristics of the environments of mobile robots that complicate the design of controllers. Typically the environments are unstructured, unpredictable, complex, dynamic and perceived through noisy sensors.

Mobile robots are often intended to operate in *unstructured* or unconstrained environments. The real world is usually not organized so that it is easy for a robot to navigate and identify hazards. An example of a highly structured environment is a room without furniture with smooth and level floors, fixed lighting, and orthogonal adjacent walls. However, a normal office usually has irregular structure. A robot would have to contend with different kinds of obstacles such as chairs, tables, and filing cabinets. The floor might have extension cords, video cables, or even a thick Persian carpet on it. Most buildings have windows so the optical sensors of a robot must work under varying illumination conditions. Even the domain of an office, however, is highly structured in comparison with outdoor environments such as a forest.

Most environments are *unpredictable*. For instance, a robot may be unable to predict the outcome of an action. Consider a robot rolling a die; unless it knows how to cheat, the outcome of the roll is unpredictable. A more practical example is that of a robot opening a door. The robot can make a partial prediction of what will be on the other side based on prior knowledge. Things may have changed since the last time the robot opened the door, so this can be characterized as partially unpredictable.

Agre and Chapman [AC87] characterize the world as *complex*. The essential idea is that most real world situations cannot be fully represented inside a computer. There is so much to know that only a subset of all the information available can be absorbed and reasoned about in a finite amount of time. The issue of complexity places limits on the suitability and completeness of world models internal to a robot.

As illustrated in Figure 1, robots obtain knowledge of the outside world through *sensors*. Sensors provides a limited and incomplete view of the world. The raw data provided by the sensors has finite resolution and is often noisy. Many robots are limited to sonar range sensors, although some have video and other sensors. As a result, robots are limited to incomplete information. Robots need to interpret raw data so that they are able to function in the world. As anyone in computer vision will testify, this is a difficult task

A fundamental problem for robots is that the world is *dynamic*. The environment changes with time due to: actions taken by the robot, actions taken by other agents, and continuing processes in the environment (e.g. a rolling ball). In dynamic worlds, a robot may have to respond to changes within a limited period of time (e.g. the problem of an oncoming truck).

The controller for a mobile robot is an embedded system with fixed computational hardware. Limited time coupled with fixed computational hardware results in a limited amount of computation that can be performed. Regardless of advances in technology, robots will only be able to perform bounded computations. Effective robot architectures must provide a strategy for dealing with scarce computational resources [DB88]. A *computational resource* is a multipurpose computing engine (typically a Von Neumann architecture computer) that can be used for a variety of tasks. The problem of distributing computational resources is often the problem of sharing computer cycles.

Robot domains can be classified according to how well the above characteristics apply. It is interesting to note that many architectures are designed for specific domains where certain problems are ignored. This taxonomy could be more rigorously developed to classify domains and architectures, but this is beyond the scope of this thesis.

2.1.2 Robots have goals

All robots have goals. These goals may be explicit in the mind of the designer, in the internal structures of the robot, or externally attributed to the robot by an outside observer. In Figure 2 a robot arm is trying to pick up a block. It is easy to imagine how the goals might be represented in each of the designer, robot, and observer. It is possible for the robot to have either explicit or implicit goals. In the figure, these alternatives are shown by the predicate Pickup(Block) and the binary encoding 1001110011000. For example, some programming languages compile down to circuit descriptions, so there are no predicates stored in the robot. (The circuit description is shown in the figure as a series of bits.)

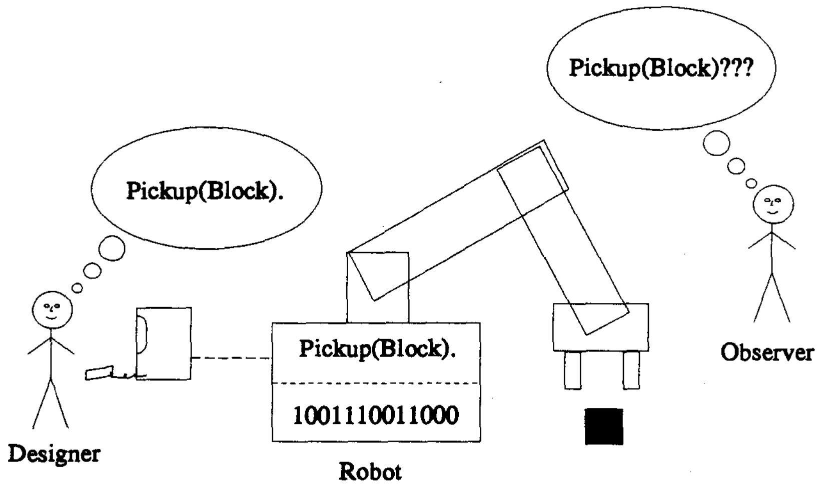

Figure 2: Where are the goals in a robot system?

An observer may or may not be able to ascribe goals to a robot exhibiting complex behaviour, so the goals of the robot need not necessarily be represented in observers. In contrast, the designer is responsible for constructing the robot program and so must have explicit goals to program the robot. Goals describe function and purpose, and yet if the designer has no goals (for the robot), then what is the meaning of the robot program?

Hence, a purposeful robot must contain implicit or explicit goals that are generated from the explicit goals of the designer.

When discussing or designing the architecture of a robot it is useful to describe the sub-goals that the robot is trying accomplish. The term *transient goals* describes the current concrete low level goals that a robot is trying to achieve through its actions. Transient goals should be interpreted as active/current/operative goals. They have extent in time and typically cannot be accomplished through a single atomic action. Transient goals change with time: a robot may first try to accomplish one particular goal, while later a different one. For example, a robot may have many goals that motivate movement from one room to another. The transient goal in this situation would be to move through a certain doorway. It is not the motivation, but rather the action attempted that reflects the transient goal. If only a single action can be performed at a time, then there is only one transient goal active at a time. Transient goals are used to compare and classify approaches to robot architecture.

Why are transient goals important? A robot operating in a dynamic world should seek to produce intelligent behaviour through the selection of appropriate transient goals. If transient goals are seldom evaluated, then a robot may continue to pursue a transient goal that is no longer appropriate. Some important questions that robot architectures must answer are "How are transient goals chosen?" and "When are transient goals changed?"

Robots use *control signals* to drive actuators. At the lowest level of a robot controller, control signals are used to drive motors and other actuator mechanisms. Control signals can be thought of as the level below which there is no further processing. Typically robotic systems have a control rate associated with them that is a function of the hardware design. This is the maximum frequency that the robot can drive its actuators at.

Robot architectures can be compared and classified using transient goals and control signals. They will be differentiated from one another based on when transient goals are evaluated and can be changed. A *complete reactive architecture* is an architecture that selects transient goals at the same rate as control signals. In such architectures, only control signals are generated, and transient goals are implicit in them. A *partitioned architecture* is one where transient goals are formed independently and usually at a slower rate than control signals. Both of these classes of architectures will be illustrated in the following sections through landmark robot architectures.

The word *reactive* means the rapid response to stimulus. When applied to the word behaviour, it means that actions (1) occur within bounded time and (2) are in response to changes in the environment. The term *reactive architectures* is used in this sense and many of the architectures discussed in this chapter would fall under this classification. It does not mean that the robot has no internal state, although it has occasionally been used in this manner.

Some, but not all, reactive architectures are able to select transient goals at the same rate as control signals. These are named complete reactive architectures to note this distinguishing feature. The word complete is used to indicate that a complete decision is made that includes the selection of transient goals. Other architectures are labelled reactive because they respond to changes in the environment while pursuing specific transient goals, although they are unable to change transient goals.

2.1.3 Criteria for Robot Architectures

There is no consensus on how to build robot controllers. Every year new architectures are proposed that are different from previous ones, yet the majority seem not to be designed to test out a theory or to satisfy general criteria, but rather to work with a particular hardware configuration on a specific problem. It is difficult to compare architectures that are associated with different domains, and to make matters worse, the target domains are rarely described clearly.

In this thesis, architectures will be evaluated according to their suitability for dynamic domains. Architectures are judged on their underlying model and not on the performance of a specific implementation. The following questions establish the criteria for comparing robot architectures:

- When are transient goals selected?
- How are transient goals selected?
- How are real-time constraints met?
- What kinds of computations are permitted?
- Are suitable abstractions provided?

The main criteria in this thesis is based on when and how the transient goals of a robot are selected. Other issues of general importance to robot architecture such as sensory processing, the role of internal state (world modeling), general design issues are not addressed in this thesis.

Another omission from this thesis is a treatment of the correctness of robotic systems. Formal concepts from concurrent systems such as safety and liveness can be applied to robotic systems. A *safety* property guarantees that an undesirable event never happens [MP92]. A *liveness* property guarantees that a desirable event eventually happens (this means that the system is not susceptible to either deadlock or livelock). Constraint Nets, a computational model for real-time systems, provides a thorough treatment of this [ZM92a; ZM92b; ZM93].

Individual robots can be compared, but differences may be the result of technical details which have nothing to do with their architectures. However, implemented systems provide a lower bound on the utility of an architecture since limitations in the architecture are often reflected in the functionality of a robot. The robot controller described in this thesis will be experimentally tested to establish a lower bound on the utility of reactive deliberation. This criterion will also be used to highlight significant limitations of other architectures.

Appropriate questions to ask of a robot are, "What tasks can it perform?" and "How well are they accomplished?" These measures of utility are determined by the observer of a robot. A nice wording for these ideas is: "Intelligence is determined by the dynamics of interaction with the world." and "Intelligence is in the eye of the observer." [Bro91]. The essence of these ideas is that robots must achieve goals in an appropriate manner and interact intelligently with the world.

2.2 Complete Reactive Architectures

The descriptor *complete* is used to describe a class of reactive architectures where *both* transient goals and control signals are evaluated at a system specific rate. In this section, the following architectures are described: the subsumption architecture, the concrete-situated approach and the situated automata. The common features of complete reactive architectures are summarized in the last subsection.

2.2.1 The Subsumption Architecture

Brooks proposes the *subsumption architecture* that uses a hierarchy of task achieving *behaviours* rather than the traditional Artificial Intelligence sense-model-plan-act framework [Bro86; Bro91]. The behaviours in the subsumption architecture are generic types of behaviour, not specific instances. For example, a hierarchy of behaviours that include generic ones like "plan changes to the world" is given in Figure 3. Some activity-specific examples of behaviours are: seek-light, avoid-darkness, avoid-obstacles, and recharge [Kub92].

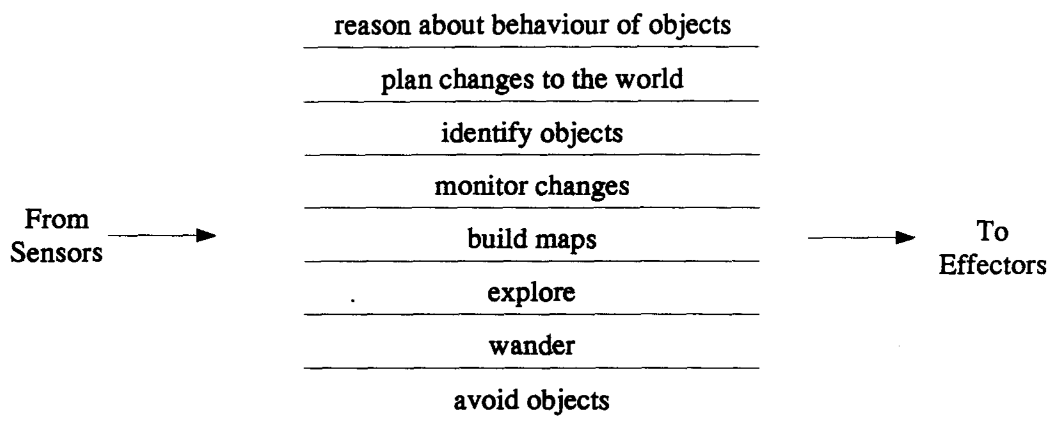

Figure 3: Task-achieving Behaviours in the Subsumption Architecture [Bro86]

Each task-achieving behaviour independently maps sensor inputs into desired control signal outputs. Behaviours principally interact with one another through a fixed set of inhibition and suppression connections, although message passing between behaviours is allowed. These connections allow higher level behaviours to *subsume* the function of lower levels. Higher levels can suppress the outputs of lower levels when they wish to take control of the robot. This form of arbitration, where the highest level behaviour wins, makes sense in the subsumption architecture where the highest level is the most sophisticated.

Each behaviour is composed of a collection of simple computing elements: augmented finite state machines. The elements can store state, perform simple computations and communicate through fixed-topology connections. These design choices result in some interesting features/limitations: there is no central locus of control; there is no central model of the world; pointers and data structures cannot be easily implemented (and are discouraged since they result in costly computations); and the search space of computations is bounded.

In the subsumption architecture there is no explicit representation of goals and knowledge. Brooks argues that this is a good thing and promotes the slogan, “The world is its own best model.” [Bro91]. Since there is no explicit representation, the transient goals of the robot are implicit in the control signals it generates. The subsumption architecture is clearly a complete reactive architecture since the transient goals and the control signals are evaluated at the same rate. One additional note is that a behaviour does not guarantee an output. For instance, if a behaviour is performing a time consuming computation it may not produce an output on time. This feature permits computations that require more time than a sensing-action cycle. It is not clear, however, that slow behaviours can be effectively combined with more responsive ones.

The focus of the subsumption architecture is on engineering robots by starting out as simple as possible and developing them incrementally. This approach has been demonstrated on at least ten robots from the 50 gram Squirt to the much larger Herbert that has a robot arm attached [Bro90]. However, the commitment to avoid representation seems to be a significant hinderance in the development of more sophisticated robots.

2.2.2 Concrete-Situated

The *concrete-situated* approach take by Agre and Chapman characterises real world situations as complex, uncertain, and immediate [AC87; Cha91]. Their approach has been used to build software *agents* that play two different video games. Their system, although implemented in LISP, is based on a connectionist approach to architecture where expensive (according to [Cha91]) data structures such as pointers and dynamic storage allocation are prohibited.

The main thesis of the work is that intelligent activity can be produced from moment to moment improvisation and need not result from the pursuit of detailed plans. The first system designed on this basis, Pengi, has no internal state, while the more sophisticated Sonja has a small amount of distributed local memory. The action of the agent is determined principally by the current situation. The implicit assumption is that the portion of the world observable by the agent contains sufficient information for making an intelligent decision. In such an environment, there is no need for memory, since the world itself acts as memory that has perfect fidelity. The video games that Agre and Chapman's software agents live in are instances of environments that contain complete information. Unfortunately, the real world is not composed of discrete blocks that form a finite 40 by 30 grid.

Patterns of behaviour, called *routines*, result from interaction with the environment and not from the explicit encoding of repetitive actions. Agre and Chapman have demonstrated that simple rules for action can generate intelligent coherent behaviour. One criticism made of this work is that the designer of the system is responsible for analysing the environment and abstracting appropriate rules for behaviour [Mae90]. However, this is no different from what is needed of the builders of other systems; the problem of automatic programming has not yet been solved.

The system consists of a *visual routine processor* and a collection of action *proposers* as can be seen in Figure 4. The proposers correspond quite closely to behaviours. There is a proposer for each internal and external action that the agent can make. Internal actions are computation requests sent to the vision routine processor, while external actions are moves that the agent makes in the environment. Internal actions provide a central feature of the system: task directed sensing (or more accurately, situation directed computation). This approach to vision is gaining increasing recognition for solving some of the complexity problems in computer vision. Conflicts between proposers are ultimately resolved through a fixed priority scheme, so that only one action is enabled at a time. In general, multiple internal actions will take place before the proposal for an external action is accepted.

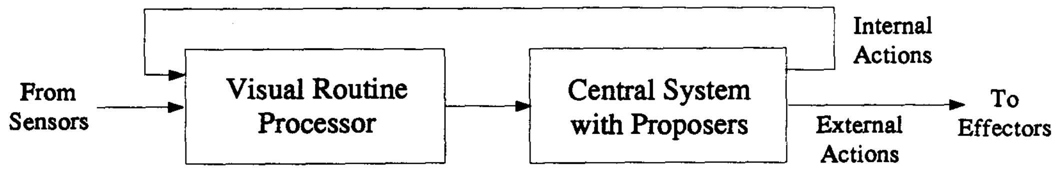

Figure 4: An abstraction of the Concrete-Situated Architecture

Each change in the environment results in a new action. There is no separation between the selecting transient goals (through proposers) and control signals (external actions); the winning external action proposal is the control signal. Since transient goals and control signals are evaluated at the same rate, the concrete-situated approach is a complete reactive architecture.

A valid criticism of the concrete-situated approach is that there are many problems in the real world that do not arise in video games. Despite this, the work has made a significant contribution to the *theory* of building intelligent systems. In the two video-game systems the software agents could examine complex situations and generate intelligent actions through a distributed network of behaviours.

2.2.3 Situated Automata

Kaelbling and Rosenschein have created two separate robot architectures based on the Rex programming language. The architecture presented in "An Architecture for Intelligent Reactive Systems" [Kae90] appears to be a strongly related to the subsumption architecture, while the later situated automata approach [KR90], is more rigorous with its treatment of semantics. To avoid confusion, the architectures will be described in chronological order. However, outside of this subsection, only the situated automata approach will be referred to.

The architecture described in "An Architecture for Intelligent Reactive Systems" [Kae90] is based on the Rex programming language. Rex is designed for the implementation of real-time embedded systems. It supports recursion and functional programming. Programs can be compiled into hardware specifications that can be built or simulated by sequential code. All computations in Rex are guaranteed to terminate in constant time. Rex also supports an incremental planning system that is compatible with real-time operation. The planner verifies that its goal is unchanged and the context of the plan is satisfied by the environment before continuing planning for a fixed period of time.

The architecture proposed in [Kae90] looks like a cleaned-up version of the subsumption architecture, although there are some significant differences. One notable difference is the separation of the perception and action components of the robot. The perception component stores the world model of the robot with data types ranging from raw sensor values to abstract world models. The similarity with the subsumption architecture lies in the organization of the action component into hierarchically mediated behaviours. The behaviours described in [Kae90] are similar to the ones in [Bro86]. A fundamental difference, however, is that the former fosters the use of meaningful symbols. The idea of situating planning in real-time activity (mentioned earlier) is a good one, but the details provided in the paper do not explain the connection between the behaviour-based approach and planning.

In [KR90] a more formal approach to robot synthesis, situated automata, is presented. The only connection with the previous paper appears to be the use of the Rex programming language. Rather than using the earlier behavioural style of architecture, situated automata builds on the perception-action split through two metalanguages, Ruler and Gapps, which compile into Rex expressions. Ruler is used to describe the perception component by establishing connections between raw sensor data and propositional descriptions of the environment. (This is essentially the same as the approach to sensing in the earlier architecture.) Gapps is used to describe the action component of a robot. It has a formal semantic grounding for describing robot programs.

The behaviour of the robot is specified through a set of goal reduction rules as well as a prioritized goal list. The goal reduction rules describe how actions of the robot lead to the accomplishment of the goals. The prioritized goal list ranks the goals according to importance. Just, as the subsumption architecture, Gapps has a fixed hierarchy of goals. The behavioural specification are compiled into a finite set of condition-action pairs that form a robot program and can be compiled further into a circuit description.

In situated automata a clock tick is defined as the minimum-cycle time needed to read inputs, perform some computation, and set outputs. A single action is generated from the set of condition-action pairs. The action component of the robot has no state: it is purely reactive. There is, however, state in the perception component to reduce the effects of sensor noise and uncertainty. The action produced is a control signal as well as a transient goal. Thus situated automata is a complete reactive architecture.

A serious drawback to Gapps is that it has a fixed ranking of goals and does not support arbitrarily complicated computations such as planning. Some remedies to the planning problem are discussed but it is not clear if they can be cleanly integrated with situated automata. It is claimed in [KR90] that the situated automata approach has been used in a variety of robotic applications, although none are described.

2.2.4 Summary

Complete reactive architectures select transient goals at the same rate as control signals, so the transient goals certainly can keep pace with the environment. The transient goals are implicit in the control signals and not explicitly represented. The architectures described in this section have a strong focus on reactive behaviour. They are capable of moment to moment activity, but there is no projection or planning of future states.

Figure 1 contrasts the complete reactive architectures. The architectures are listed in the rows, and the distinguishing features in the columns.

Feature / Architecture	Model of Computation	Limited Computation?	Focus on:	Systems
Subsumption Architecture	AFSM	No	Engineered Solution	Many Robots (10)
Concrete Situated	Circuit	Yes	Improvisation through Routines	Video Game Players (2)
Situated Automata	Circuit	Yes (Possibly No)	Formal Semantics	Variety of Applications (?)

Table 1 Distinguishing Features of Complete Reactive Architectures

The heading *Model of Computation* refers to the computational model used in the architecture. All of the architectures can compile down to a circuit level description. The restriction of the computational mechanism to circuits has significant impact on the capabilities of these architectures. For instance, all these systems can respond quickly to changes in the environment. A limitation of these architectures is the inability to perform search type algorithms needed for planning, since they have little or no memory.

The architectures with *limited computation* are guaranteed to complete all computations each system cycle. This means that only a small class of algorithms can be used by these architectures. Although the subsumption architecture permits unbounded computations, other commitments within the design philosophy limit the applicable algorithms. The issue of limited computation may lead to problems extending these architectures to more sophisticated systems.

The column *Focus on* contains the motivation or guiding principle of each architecture. Although these architectures share a large number of features, they have significantly different motivations. The only one concerned with semantics and correctness is the situated automata approach.

The *Systems* column lists robotic systems built using the architecture. It is included because experiments are needed to substantiate theories of robot architecture. The concrete-situated approach has only been demonstrated in a video game, so its extensibility has yet to be demonstrated. The subsumption architecture has had simple applications in dynamic domains documented; little has been published on the experimental use of the situated automata approach.

2.3 Partitioned Architectures

Partitioned architectures are architectures where transient goals are generated independent of, and typically at a slower rate than, control signals. The approaches discussed in this section are: Shakey the robot, Reactive Action Packages (RAP), ATLANTIS, and the dynamics of action selection. This section concludes with a discussion of the common features and differences between architectures.

2.3.1 Shakey

"Shakey" is the name of a mobile robot that was used in experiments in the late sixties and early seventies [Nil84]. This was one of the first attempts at creating an intelligent autonomous robot and strongly influenced successive attempts at building robots. Shakey is considered the "traditional" Artificial Intelligence approach to building robots.

Figure 5 shows the architecture of Shakey the Robot. Shakey is partitioned into the STRIPS planner and the PLANEX executor. The planner is responsible for "high-level" reasoning and instructs the executor through plans. The executor is responsible for monitoring the execution of the plan and communicates with the planner through an abstract world model that stores information in predicate form. The partition between planning and execution monitoring that began with Shakey is reflected in all the partitioned architectures in this section.

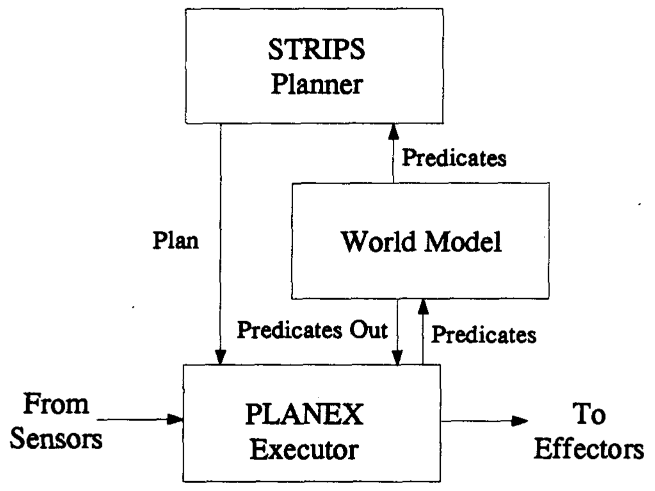

Figure 5: The Architecture of Shakey the Robot

The reasoning component of the robot is the STRIPS planner. It is assumed that the robot is given a high level goal by an external authority. The STRIPS planner performs a directed search through the space of possible actions to produce a plan. The plan consists of a sequence of operators (actions) that should result the goal state when they are executed. An example operator actually used in Shakey is: GOTOADJROOM(ROOM1,DOOR,ROOM2) [Nil84]. This reads as: Go To Adjacent Room from Room1 through Door to Room2.

The PLANEX component of the system is responsible for executing the plan. PLANEX iterates over a series of perceptions and actions. First, it checks to see if the goal state has been achieved. If it has not, PLANEX selects an operator whose preconditions are true. If no operator is valid, then plan failure is reported to the STRIPS planner. The operators are typically selected in sequence, although some may be skipped if some external agent assists Shakey.

Operators are associated with action routines called Intermediate-Level Actions (ILAs). ILAs are composed of a sequence of Low-Level Actions (LLAs). LLAs are the bottom of the abstraction hierarchy and are implemented in LISP. Each LLA has termination conditions based on simple sensory data. It is the LLAs that form feedback loops with the environment to perform simple tasks. The ILAs allow the STRIPS planner to plan at a higher level of abstraction the LLAs. The operator (ILA) GOTOADJROOM is composed of the following LLAs: DOORPIC, TURN, PLANJOURNEY, ROLL, etc. [Nil84].

The operators defined in this system are composed of LLAs. LLAs are related to transient goals since they are close to the idea of a simple action or activity. Hence, a STRIPS plan can be thought of as a sequence of transient goals. The transient goals of Shakey are established during planning and are committed to long afterwards. New plans are generated when the current plan has been completed or there is a problem in execution. Shakey is a partitioned architecture because transient goals are evaluated independent of and usually far in advance of the control signals.

The assumptions made in building the planner for Shakey are very restrictive: there is a single agent in the world, the world is static, actions are discrete, and all actions are predictable. Each of the problems stated in Section 2.1.1 were engineered away or ignored.

It is worth noting that all of the assumptions made in Shakey are invalid in an environment with people in it. Shakey is not at all appropriate for dynamic environments.

2.3.2 Reactive Action Packages

The Reactive Action Packages (RAP) system is a descendant of Shakey with some significant improvements [Fir89]. Like Shakey, the decision making component of the robot is a planning system. Reactive action packages play exactly the same role as operators and ILAs in Shakey. The operator role of the reactive action packages is to maintain the planner's delusion that there are abstract primitive actions in a continuous world [Fir92]. The ILA role is to provide a means for describing robot programs. The RAP Executor acts as a bridge between an abstract planner and a controller embedded in the real world. The entire architecture can be seen in Figure 6.

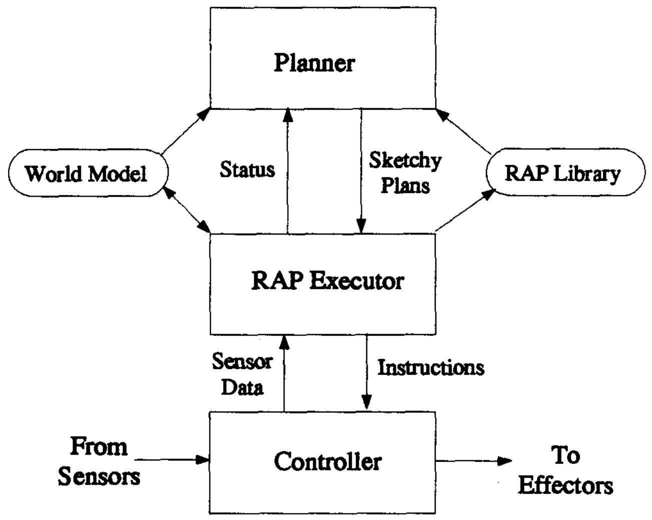

Figure 6: The RAP Architecture

Each RAP is a robot program that consists of a task, success criteria, and a number of alternate methods for carrying out the task [Fir89]. RAP provides a more structured approach to robot programing than ILAs since each method within a RAP can be thought of as a simple behaviour-based program. In the latest version of RAP theory, methods enable and disable collections of control and sensing modules in the controller rather than directly performing sensing and controlling the actuators [Fir92]. The controller continuously interacts with the world to produce reactive behaviour. A notable improvement over Shakey is that dynamic activities such as following someone across the room are now supported. Further, the behaviour-based approach has resulted in more robust control and the system tolerates the failure of actions.

The RAP system generates a "sketchy plan" composed of a sequence of tasks. The sequentially executed methods within each task are equivalent to transient goals. Like Shakey, plans and transient goals are generated only when the preceding plan has been completed or has failed. As a result the problem of producing appropriate behaviour (as opposed to reactive behaviour) in a dynamic world is not addressed. RAP is a partitioned architecture for the same reasons as Shakey.

2.3.3 ATLANTIS

ATLANTIS (A Three-Layer Architecture for Navigating Through Intricate Situations) is described as a direct intellectual descendent of the original RAP system [Gat92]. It has a three layer architecture that consists of a deliberator and a controller connected by a sequencer that is similar to the RAP executor. It allows an asynchronous traditional planner to operate with a reactive control mechanism. In ATLANTIS, the deliberator can perform any computations, so it is difficult to draw a clear schematic for the architecture. However, the figure for RAP, Figure 6, is fairly close, except that the planner/deliberator has access to sensor data and can add to the world model.

The deliberator is restricted to computations that are interruptible so that complex computations will not degrade the real-time performance of the robot. The architectural model makes no commitment to any particular deliberation mechanism, although all lengthy computations must be performed by the deliberator. The current implementation includes a vision system, a world modeler, and a planner in the deliberator.

The controller is composed of behaviour producing modules called primitive actions. Each primitive action is very similar to the control modules in the RAP system. Although they are called primitive actions, there is no commitment to discrete actions and continuous activities are supported.

The deliberator shares computational resources with the controller and can operate while the controller pursues the current plan. Simultaneous planning and control allows the deliberator to consider alternate plans. Plans (and hence transient goals) are evaluated in ATLANTIS as available processing time allows. This makes it possible for a robot to produce intelligent behaviour in dynamic environments.

In conclusion, ATLANTIS attempts to alleviate some of the difficulties with the traditional Artificial Intelligence approach to robotics. It has some features appropriate to dynamic domains, such as the consideration of alternate plans. However, the commitment to the plans-as-communication view [AC90] prevents specific plan details from being computed until they are needed, resulting in a greater latency in response.

2.3.4 The Dynamics of Action Selection

The *dynamics of action selection* is a theory of action selection intended for autonomous agents [Mae89; Mae90]. It describes a mechanism for action selection and is not a complete robot architecture. The dynamics of action selection provides an alternative to traditional planning systems. It is discussed in this section because the dynamics of action selection could be inserted in the "planner" box of any of the other partitioned architectures. In addition, this material is relevant because the problems of action selection and the selection of transient goals are essentially the same.

Maes argues that a mechanism for action selection is needed for a robot operating in a complex dynamic environment [Mae90]. Any such mechanism must be fast, favour actions relevant to the current situation, and be able to exploit opportunities. Further, there is a trade-off between "goal-oriented" and "situation-oriented" behaviour. "Goal-oriented" behaviour means that a robot prefers to follow an active plan rather than respond to the current situation.

The commitment to pursue a plan irrespective of changes in the environment is inappropriate. The decision made by a robot of what to do *now* should be based on the expected utility of each action, not on the arbitrary notion of following a plan. The postulated trade-off does not apply. However, the inspiration for a systematic means of evaluating the appropriateness of actions is insightful.

The dynamics of action selection mechanism accepts high-level goals and predicates about the world as inputs. At each time step, it outputs an action. There is a *competence module* [Min86] for each possible action that is represented as a node in an activation network. Competence modules are connected through successor, predecessor and conflicter links that represent dependencies and conflicts among the actions. Energy is injected into the network for each active goal and valid proposition. A small number of tunable global parameters are used to regulate the flow of energy through the links connecting the nodes. After a set number of energy spreading iterations, the competence module with the highest energy is selected as output.

The dynamics of action selection is identified as a fast mechanism for robots, although it is not ideal in a number of ways. First, predicates describing the world are readily available and are used to the exclusion of probabilistic models (actually, this is a planned modification). Second, all goals input the same amount of activation energy into the network, although it is unlikely that all goals are of equal importance. Third, in an experimental evaluation of different action selection mechanisms the dynamics of action selection fared very poorly [Tyr93].

2.3.5 Summary

The partitioned architectures are, roughly speaking, composed of a planning component and an execution monitoring component. Transient goals are selected independently of and less frequently than control signals. This is advantageous because it is typically not necessary to select transient goals as frequently as control signals in dynamic environments. Transient goals are explicitly represented as actions. Actions are associated with activity producing modules that abstract away system specific implementation issues. This allows modules to be tested independently and reduces the complexity of the entire system.

Table 2 focuses on the selection of transient goals in partitioned architectures. The architectures are listed in the rows, and the distinguishing features in the columns. The issues that are important for partitioned architectures are different from those that are relevant to the complete reactive architectures. Hence, the differences between the headings of this table and the table on complete reactive architectures.

The question, "When are transient goals selected?" is the most important in dynamic worlds, since transient goals must be able to change with the environment. The heading *Systems* is included since robot theories must be tested to ensure validity. The bottom row provides a sneak preview of reactive deliberation, the architecture presented in this paper, since it is a partitioned architecture.

Feature / Architecture	When are transient goals selected?	How are transient goals selected?	How far into the future do transient goals extend?	Systems
Shakey	On failure or completion of plan.	Planner	Arbitrary Length Plans	"Office" Robot
RAP	On failure or completion of plan.	Planner	Arbitrary Length Plans	Office Robot (1?)
ATLANTIS	On failure or completion of plan and as frequently as possible.	Planner. (Possibly other mechanisms)	Arbitrary Length Plans	Outdoor Robots (>2)
Dynamics of Action Selection	As frequently as possible.	Activation Network	One Action	Tested in Simulator
Reactive Deliberation	As frequently as possible.	Estimates of the utility of behaviours	One Action	Multiple Soccer Playing Robots

Table 2 Partitioned Architectures in Perspective

The top three architectures, Shakey, RAP, and ATLANTIS, exhibit similar characteristics, since they are all designed around planning systems. In Shakey and RAP, arbitrarily long sequences of actions are planned and then executed without much consideration for other actions. ATLANTIS is similar, but allows alternate plans to be considered. In all three, sequences of actions are generated, and not just a single action. The danger with these approaches is that much of the robot's success lies in its ability to predict future worlds A robot must be able to operate in a dynamic world where there are multiple agents, actions are continuous, and the effects of non trivial actions are unpredictable. To operate in a real environment, transient goals must be evaluated frequently: the blind pursuit of plans is poor paradigm to follow.

The dynamics of action selection generates a single action at each time step, so it is able to select actions appropriate under the current circumstances. This approach is more appropriate for robots in dynamic environments, than the approaches based on planning. However, the dynamics of action selection is hardly an ideal scheme.

One notable omission from this chapter is a survey of hierarchical architectures such as NASREM where the problem of controlling a robot is considered analogous to the problem of controlling an army or business [Alb81]. Hierarchical architectures in their general form have been successfully used for the control of robot work cells in manufacturing. However, their use in mobile robots has been limited and their applicability in uncertain and dynamic environments has yet to be demonstrated. For these reasons, hierarchical architectures have not been given detailed treatment. It is worth noting that partitioned architectures can be considered a special case of hierarchical architectures.

2.4 Additional Related Literature

This section describes additional previous work relevant to this thesis. The work is related to mechanisms that arbitrate between conflicting goals. Planning systems are not suitable for this; they do not decide which goal the agent should pursue, but rather accept a single goal as an input. One possible solution is to develop a heuristic mechanism that arbitrates between goals such as the dynamics of action selection. A more rigorous approach is to use decision theory to evaluate the expected utility of each action.

Decision theory can be used to answer the question "What to do now?" (It can also be used to compute a sequence of actions.) Given an initial state w_0 and finite sets of decisions D and world states W, decision theory requires that a value be assigned to each world state $v[w]$and for each potential decision that the conditional probability of reaching each world state be specified $p[w \mid d, w_0]$. The expected value of each decision $EV[d]$ is then the sum of the values of each world state weighted by its conditional probability as shown in the following equation: $EV[d] = \sum_{w \in W} v[w] \times p[w \mid d, w_0]$. The best decision is then simply the decision with the greatest expected value.

One problem with decision theory is that there are a lot of world states and this leads to a very large number of conditional probabilities. The above instructions must be performed for each initial state, so for *n* world states and *m* decisions[1], the number of conditional probabilities needed are n^2m. Continuous properties must be discretized; This forces the designer of a system using decision theory to abstract away (hopefully) unimportant details of the world to reduce the number of conditional probabilities to be computed. A decision model for robots based on decision theory has been proposed [KD89], however it does not handle continuous variables, nor is it possible to do sophisticated spatial reasoning. For realistic robotic environments, decision theory does not appear to be good approach, however it motivates the approach taken in reactive deliberation of generating estimates of the utility of actions.

Bidding as a mechanism for action selection has been proposed before for autonomous robots. Minsky describes a central marketplace where *mental proto-specialists* compete for control [Min86]. Each proto-specialist represents a different goal and generates a bid based on the internal urgency of the goal. He argues that this approach is bound to result in unstable behaviour, but this conclusion is reached since only the urgencies of the goals and not the current situation are considered in his scheme. Once bids are based on both the importance of the goal *and* the current situation the objections raised are invalid. The

[1] The situation is actually worse than this. If there are *k* independent propositions, then $n=2^k$. Also, *m* is exponential in the number of independent decisions, and quadratic in the number of different control values.

problem of mediating behaviours is also discussed in [Kae90] and follows the same (flawed) line of arguments as in Minsky.

Action selection mechanisms based on bidding are referred to as *drives* in psychological literature. Different mechanisms for action selection are compared through a simulation of a zebra living in an African savannah in [Tyr93]. He found that while the drives approach is effective, a free-flow decision hierarchy (neural network) works better due to its ability to combine preferences (actions). It makes sense to combine preferences in the simulation when the world is a discrete grid and there are only a small number of actions. It is unclear how preferences can be meaningfully combined in a real robot where actions, such as the desired direction of motion, are continuous.

Systems such a *contract nets* allow negotiation between agents to make a decision [Smi80]. This is very different from the bidding mechanism described above. With negotiation, the bids an agent makes are influenced by the bids of other agents, while with the bidding described above, there is no interaction among agents (behaviours, goals). Negotiation has been used for the control of multiple robots [Nor92].

2.5 Discussion

Through transient goals an abstract view of robot architecture is provided. From this viewpoint, landmark architectures have been partitioned into two groups: complete reactive architectures and partitioned architectures. Each group has a number of features that are useful in dynamic domains.

In complete reactive architectures, the transient goals of the robot are implicit in the control signals. These architectures can compile down to a circuit level description and can only perform bounded computations, and so are able to respond quickly to changes in the environment.

Partitioned architectures have transient goals explicitly represented as actions. Actions are associated with activity producing modules to abstract away system specific implementation issues and increase modularity. One weakness typical of these architectures is the commitment to arbitrary length plans.

The separation of the selection of transient goals from the frequent generation of control signals in partitioned architectures provides two advantages. First, transient goals can be selected at whatever rate is appropriate for the environment since it is seldom necessary to select transient goals as frequently as control signals. Second, lengthy computations can be performed, while complete reactive architectures are restricted to those that can be done at the same rate as control signals.

It is not appropriate to rank the architectures described in this chapter according to their suitability for dynamic domains, since there is no clear basis for such a comparison. For instance, both the concrete-situated approach and the dynamics of action selection are only partial specifications of robot architectures and results have been limited to software simulations. Shakey, RAP, and ATLANTIS are very similar with only small differences, although ATLANTIS supports some features that recognize the time limited nature of dynamic environments. The utility of the situated automata approach, although it appears limited, is not entirely clear. The subsumption architecture has some good elements, but has not evolved significantly since its inception in the mid-eighties.

Chapter 3
Reactive Deliberation

In this chapter, the main results of this thesis are presented through an architecture called *reactive deliberation.* It combines responsiveness to the environment with intelligent decision making. Even deliberation must be to some extent reactive to changes in the environment. Although the name is an oxymoron, it is consistent with Artificial Intelligence nomenclature (cf. Reactive Planning).

Although reactive deliberation is a partitioned architecture, it shares many of the features of the complete reactive architectures and contains the architectural elements needed in dynamic domains. Reactive deliberation was influenced principally by RAP and the dynamics of action selection, although all of the approaches discussed in the previous chapter have contributed to it in some way.

Reactive deliberation has two related architectural contributions. The first is the partition of the architecture into deliberative and executive layers. The second is the decomposition of the deliberative layer into *behaviours* that represent the goals of the robot. Each of these is motivated and explained in the following sections.

3.1 The Architecture

Reactive deliberation is a partitioned architecture that follows the principles behind Shakey. However, there are many ways to divide and conquer the problem of robot control. The difference between reactive deliberation and other approaches is the level of abstraction at which the split between reasoning and execution monitoring occurs. In this section, an alternative way of selecting this split, one that is suited to dynamic domains, is presented and justified.

Reactive deliberation is partitioned into deliberative and executive layers. The deliberative layer decides what to do and how to do it, while the executive layer interacts with the environment in real-time through a set of *action schemas* that receive run-time parameters from the deliberative layer. The components that form each layer are collectively referred to as the *deliberator* and the *executor* respectively.

A structural model illustrating the partition can be seen in Figure 7. The deliberator is composed of multiple concurrently active behaviours, while the executor has only one action schema enabled at a time. The deliberator and the executor run asynchronously to allow the executor to continuously interact with the world and the deliberator to perform time consuming computations. More detailed information on behaviours and action schemas are provided in the following sections.

The deliberator is responsible for answering the questions: "What to do now?" and "How should it be done?" Believers in the theory of plans-as-communication would argue that these questions can and should be resolved independently [Gat92]. In this case a planner decides what to do based on an abstract world model, while the problem of resolving how each action should be performed is postponed until it is to be executed. In a dynamic

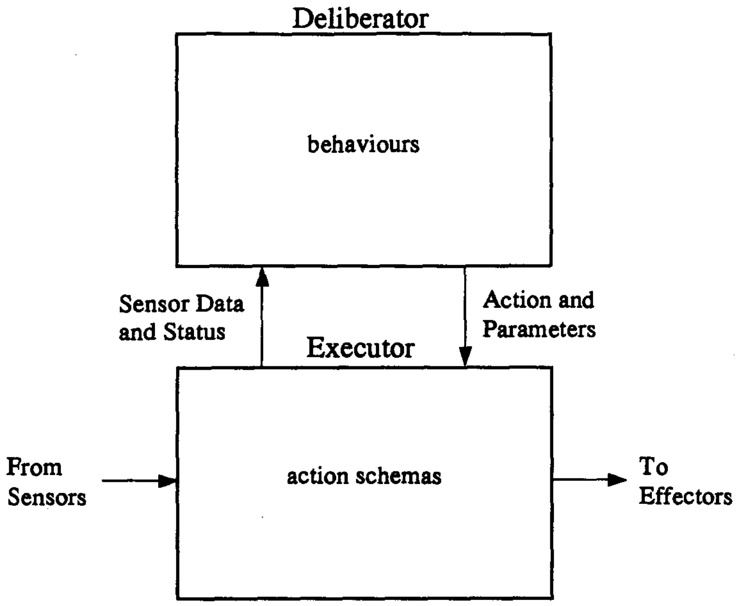

Figure 7: The Architecture of Reactive Deliberation

environment, however, these questions are usually interrelated. Before committing to an action, it is important to verify that the action is both feasible and more appropriate than other actions. Partitioned architectures that follow the planning paradigm check to see if an action is feasible, but not if there is a better action.

Answering the question "How should it be done?" provides information about the utility of an action. For example, detailed planning may show that one action is impossible, while another can be accomplished quickly. This suggests that generating plans at a high level of abstraction may not provide an effective solution for the problem of action selection. Unless all actions of the robot are feasible and the outcomes can be predicted at design time, the question "What to do now?" cannot be intelligently answered without also answering "How should it be done?"

The interface between the two layers is fairly simple, yet reveals important structural features of reactive deliberation. The executor regularly sends filtered sensory data as well as occasional status messages describing events or errors from the enabled schema. The deliberator sends an action with parameters to the executor when the enabled schema has accomplished its objective or an action more appropriate the current situation is found. This is a central feature of the architecture: the deliberator can interrupt the current activity of the executor and provide it with a more appropriate one. Notice also that the deliberator is responsible for generating a *single action,* whereas other partitioned architectures (excepting the dynamics of action selection) generate a *complete plan* (i.e. sequences of actions). This distinction helps focus the deliberative activities on the immediate situation.

3.2 The Executor

The executor is composed of a collection of action schemas. An *action schema* is a robot program that interacts with the environment in real-time to accomplish *specific* actions. Schemas receive run-time parameters from the deliberative layer that fully define the activity. The Action schemas exhibit the same level of complexity as controller modules in RAP and primitive actions in ATLANTIS. They are designed in the spirit of behaviour-based approaches, where each schema is experimentally verified. For example, typical schemas might be: follow path, drive along wall, push box, and pickup block. All the schemas together define the capabilities of the robot; they are independent of the robot's goals.

The deliberator enables a single action schema with a set of run-time parameters that fully defines the activity. Only one action schema is enabled at a time and it interacts with the environment through a tight feedback loop. In the world of real-time control there is no room for time consuming planning algorithms. Computations in action schemas are restricted to those that can keep pace with the environment, so lengthy computations are performed in the deliberator.

The boundary between appropriate and inappropriate computations in the executor is a function of the computing power of a particular system and specific environmental constraints. The idea is that any computations that can be performed within the time constraints of the environment are suitable for use in the executor. Computations which do not meet time constraints are relegated to the deliberator to avoid degrading the ability of the robot to interact in real-time. Regardless of advances in computing power, there will likely be interesting algorithms that do not run in real-time. This suggests that the partition between the executor and the deliberator is indicative of a technology-independent need to segment computations.

3.3 The Deliberator

In this section, the internal specifications of the deliberator module are described. The focus of the deliberator is on an effective mechanism for selecting actions or transient goals in a timely manner. There are a large number of architectural issues to be discussed, so the more peripheral ones have been relegated to the Subsection 3.3.2, Further Issues.

A central feature of reactive deliberation is that the deliberator is composed of modules called *behaviours* that represent the goals of the robot. The notion of a behaviour is used in the sense of Minsky's *mental proto-specialists* [Min86] with some important distinctions. In reactive deliberation, each behaviour computes an action and generates a bid reflecting how suitable it is in the current situation. The most appropriate behaviour, and hence action, is determined in a distributed manner through inter-behaviour bidding. Some examples of behaviours are: clean floor, recharge, deliver mail, and patrol building. Notice that behaviours characterize goals, not actions.

A *behaviour* is a robot program that computes actions that may, if executed, bring about a specific goal. Behaviours compute actions whereas action schemas perform actions. Each behaviour must perform the following: 1) select an action schema, 2) compute run-time parameters for the schema, and 3) generate a bid describing how appropriate the action is.

3.3.1 More Than Just A Theory of Action Selection

The deliberator is responsible for ensuring the robot acts intelligently in its environment. It must decide what the robot should do *now* by selecting the action or transient goal to pursue. Reactive deliberation does this while also addressing the problem of limited time and bounded computational resources.

The deliberator in reactive deliberation is composed of modules that each represent a goal or behaviour. Each behaviour evaluates the world and performs whatever planning is needed to fully describe its associated action. A bid is produced by each behaviour that reflects how beneficial it would be for the behaviour to gain control of the robot. In other words, a bid expresses the expected utility of an action given the current world state. The behaviour with the highest bid gains control of the robot's effectors.

Each bid is an estimate of the expected utility that is based on the current state of the world as well as the results of planning. Currently, the criteria for generating the bids are hand coded and tuned so that the most appropriate behaviour is active in each situation. This approach requires the designer of a system to explicitly state the conditions under which certain behaviours are suitable and favourable. A simplified version of this appears in all the complete reactive architectures. For example, the concrete-situated approach uses binary preference relations to establish an ordering of proposers or actions.

In a real robot there is more to the problem of action selection than just deciding what to do. In dynamic environments, a robot needs to quickly decide what to do and how to do it. The deliberator must keep pace with changes in the environment to produce intelligent behaviour. Each behaviour is responsible for computing a bid and planning the action. Fixed computational resources need to be distributed among the behaviours, since it is typically the case that there is too much computation to be done.

In reactive deliberation behaviours are not always active. Assuming the architecture is implemented on a standard serial processor (which may be time sharing with the controller or on a separate processor), the order of activation of behaviours must be scheduled. At any given time there will be a ruling behaviour that has control of the robot executor. It is desirable to schedule the ruling behaviour more frequently, since it may want to modify the current action to reflect changes in the environment. Another observation is that behaviours whose bids will be lower than the active behaviour's do not need to plan their actions. This is useful since planning an action is more computationally expensive than generating a bid.

The exact mechanism for distributing computational resources is left unspecified as it is strongly dependent on the real-time requirements of the system, the number of behaviours, and the resources need by each behaviour. However, the basic principle is to divide the available computational resources among the behaviours such that the ruling behaviour receives more resources. This allows behaviours that perform minimal computations to respond quickly, while those that perform lengthy computations will respond slowly. It might be appropriate to allocate resources according to the importance and needs of each behaviour, but there are no provisions for this in the current implementation. There is no perfect architectural solution to the problem of limited computational resources. If the computations slow, then the robot will be slow too. The only possible solutions are to get more powerful computers or better algorithms.

Reactive deliberation is a good approach for robot domains that do not have a large number of goals (and hence behaviours). Most robots built to date have few capabilities and can only accomplish a small number of goals. It is clear that reactive deliberation may become cumbersome if there are a large number of behaviours. On the other hand, mechanisms could be developed that might, for instance, switch banks of behaviours off in situations where they would never gain control.

3.3.2 Further Issues

In this subsection a number of issues peripheral to reactive deliberation are discussed. These are: starvation and stability, planning, multiple resources, distributed versus central decision making, learning, and multiple robots.

Starvation and Stability There are two issues that must be addressed by any arbitration mechanism: starvation and stability. Starvation occurs when a behaviour should gain control but does not. This can be avoided by tuning the bids generated by each behaviour. For instance "impatient" behaviours might have bids that increase with time until the behaviour gains control. A system is unstable if it alternates between two behaviours without accomplishing the goal of either one. Stability can be accomplished through appropriate bidding. Behaviours that have terminating actions bias their bid upward to reflect the fact that they are closer to completing the action. This prevents the system from alternating between two different behaviours without accomplishing either objective. Nonterminating behaviours may have to incorporate a "boredom" component in bidding so that if they have control for a long period of time, they get bored and reduce their bid to allow other behaviours to gain control.

Use of Plans The use of plans is permitted within the deliberator and may be of great use guiding the activities of the robot. For instance, a behaviour might include a planner and a plan monitor that would bid against other behaviours on the basis of the appropriateness of the current plan step in the current situation. However, plans are inappropriate when used as an abstract robot program simply handed over to an executor that follows it to the exclusion of alternate actions. It is widely recognized that robots need to operate in a dynamic environments that are unpredictable and full of uncertainty [Mac93]. An exception to traditional plans is *universal plans* [Sch87] where actions are described for all possible world states; however, this is not a tractable solution.

Multiple Resources In the above description, all the behaviours compete for control of the entire robot: there is only a single resource. This can be extended to multiple independent resources, such as motion, sensor and manipulator control, where each behaviour produces a bid for each resource that it needs. If control were assumed only when all the desired resources were available, the problem of deadlock could be avoided, although starvation must again be solved by proper tuning of the system.

Modularity and Concurrency Reactive deliberation is a distributed mechanism for action selection. Each behaviour is responsible for evaluating the world according to its own criteria. There is no central decision maker that evaluates the world and decides the best course of action. One feature of reactive deliberation is the ability to add new behaviours without modifying the bidding criteria of the established system; a new behaviour must, of course, be tuned to be compatible with existing ones. Also, the behaviours are independent, so they can have different representations and approaches to generating actions. For instance, a behaviour could incorporate a traditional planner to generate complex activities. Another major advantage is that behaviours can be run concurrently on different processors, thus improving the response time of the system.

A Possible Unitary Framework Reactive deliberation need not be distributed. It could be implemented as a unitary system that consults a database to obtain bidding criteria and the goal of each behaviour. Unfortunately a unitary system forces commitment to one representation for deliberative activities, whereas independent behaviours allow multiple representations and distributed processing. However, one advantage to using a unitary system is that a single representation system is easier to design and test.

Learning It is possible to implement a learning system that would generate appropriate bids, rather than use the current method of tuning the bids by hand. Given a fixed set of inputs describing important domain attributes and the results of planning, appropriate bids could be learned. For example, neural networks are good for approximating nonlinear multivariable functions. However, the training set may be difficult to generate for dynamic domains.

Inter-robot Cooperation Reactive deliberation can be extended to multiple robots, where each robot would broadcast its intended actions and a bid which estimates the appropriateness of that action. Robots whose actions are in conflict will reevaluate their decision to include the bid information of other robots. Robots with lower bids than other robots will lower their internal bid for that action, since that action is not a good one in the current situation. This should result in another behaviour gaining control of the robot. It is not clear, however, that such a mechanism will lead to effective inter-robot cooperation.

3.4 Discussion

Reactive deliberation contains a consistent set of architectural elements needed in dynamic domains. It was developed to investigate some of the problems with producing real-time intelligent behaviour in dynamic environments. Reactive deliberation makes three contributions to robot architecture.

A new split between reasoning and control is needed because the questions "What to do now?" and "How should it be done?" are interrelated. The partition between reasoning and control advocated in reactive deliberation makes the deliberator responsible for answering these questions. The executor is restricted to bounded-time computations to provide a tight feedback loop with the environment. Reactive deliberation recognizes that (low level) robot control is an on-line activity, while action selection and planning can be off-line activities without degradation in performance. Complete reactive architectures and constraint nets are essentially on-line and do not take advantage of the ability to perform off-line computations. Other partitioned architectures will in general have greater lag since detailed planning is postponed until the action is to be executed.

The transient goals of a robot need to be evaluated at a rate commensurate with changes in the environment. This is an important claim since other partitioned architectures (except the dynamics of action selection) do not meet this criterion. In reactive deliberation, the restriction of attention to the next action allows the deliberator focus on the best action *now*.

Goal-oriented *behaviours* are a useful abstraction that allow effective goal arbitration and sharing of scarce computational resources. Goal arbitration is accomplished through a distributed bidding mechanism that reflects the expected utility of the actions of the robot, while computational resources are shared among behaviours.

Table 2 on page 29 compares reactive deliberation with the partitioned architectures. The dynamics of action selection serves a similar role as reactive deliberation, but uses the spreading of energy in an activation network to select actions, rather than utility minded behaviours. The traditional planning-based architectures are quite different since they consider plans rather than individual actions. This bias prevents them from reacting to changes in the environment.

Reactive deliberation is not a panacea for robotic woes. A further disclaimer is that it is an *incomplete* robot architecture. Its focus is on the issues related to dynamic domains, ignoring a number of important robotic issues. The largest omissions are on sensing issues and on world modeling. Sensor processing, fusion, and the development of maps and world models are really hard problems. This work ignores them to focus on the problem of real-time intelligent behaviour in dynamic environments.

Chapter 4
Playing Soccer: A Real-World Experiment

This chapter links theory to practice describing a robot controller that has been constructed using reactive deliberation. The controller has been designed so that the robot may compete with another robot in a game of soccer. Soccer-playing was chosen as a domain for experimentation since it requires real-time interaction with a dynamic environment and involves inter-robot competition.

4.1 The Dynamite Testbed

The Dynamite testbed, which has grown out of the Dynamo (Dynamics and Mobile Robots) project at UBC [BKL+93; BKM+93], consists of a fleet of radio controlled vehicles that receive commands from a remote computer. Robot position and orientation is determined using off-board visual sensing. The testbed allows autonomous robots to be controlled with relative ease from an off-board computer. The testbed is intended for robotics experiments, so an effort was made to simplify the visual tracking of robots and the programmer interface while retaining the challenges faced by real robots.

4.1.1 The Soccer Field

The mobile robot bases are commercially available radio controlled vehicles. Two 1/24 scale racing-cars are used in the experiment. (There are six racing cars in the lab altogether). Each is 22 cm long, 8 cm wide, and 4 cm high excluding the antenna. These cars have driven under computer control at speeds of 140 cm/s, well below the maximum speed of 6 m/s. The soccer field is 244 cm by 122 cm in size. Two cars and a ball are shown their environment in Figure 8. The ball is the small object in the middle of the image; each of the cars is fitted with two circular colour markers allowing the vision system to identify their position and orientation. The robots are neither as flexible nor as competent as human soccer players. As a result, the environment is modified in two ways. First, there is a wall around the soccer field which prevents the ball (and the players!) from going out of bounds. Second, there are barriers to prevent the ball from getting trapped in the corners. Since these are Canadian robots, it is not unreasonable for the soccer field to be shaped like an ice hockey rink.

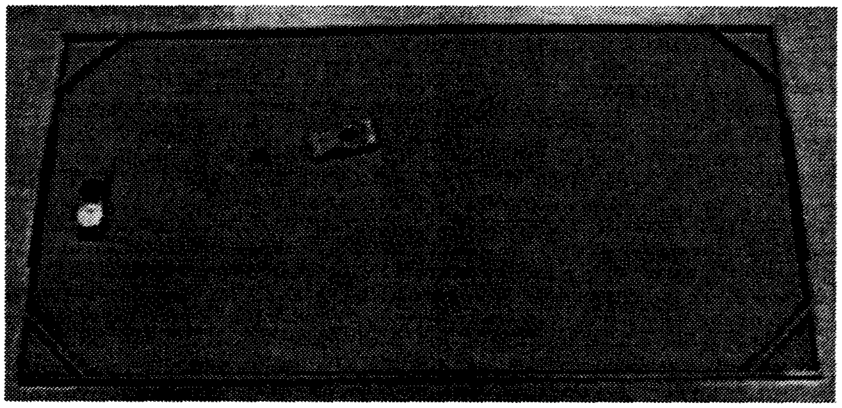

Figure 8: Robot Players Inhabiting the Soccer Field

4.1.2 System Overview

The hardware used in this system is shown in Figure 9. There is a single colour camera mounted in a fixed position above the soccer field (Figure 8 shows the image of the soccer field viewed from the camera). The video output of the camera is transmitted to special-purpose video processing hardware named the *DataCube* in Figure 9. The DataCube is a dataflow computer which has been programmed to classify image pixels into different colour classes at video rate. This information is transmitted to a network of transputers which form a MIMD computer. Additional vision processing is performed on the transputers to find the position, in screen coordinates, of the centroid of each coloured blob and to transform these positions from screen to world coordinates. The entire vision subsystem is called the Vision Engine [LBaJL91]. The Vision Engine produces the absolute position and of all the objects on the soccer field. The orientation is also produced for the cars, but not the ball. This is done at 60 Hz with an accuracy in position of a few millimeters. For further information on the Vision Engine and the vision software, refer to Appendix A.

The reasoning and control components of a vehicle can be implemented on any number of transputers out of the available pool. Each vehicle is controlled by a user program running on two transputer nodes: one for the deliberator and one for the executor. An arbitrary number of nodes, labeled 1 to n in Figure 9, can be used in parallel to control n independent vehicles. The movement of all vehicles is controlled through a radio transmitter attached to a single transputer node. Commands are transmitted to the vehicles at a rate of 50 Hz. System users are able to install their own planning and control routines and simply link in to the vision and transmitter systems. Each user program is independent and does not effect the operation of others. User programs can cooperate with one another by communicating via message passing.

4.1.3 User Interface

One of the advantages of this environment is a clean user interface. The user program

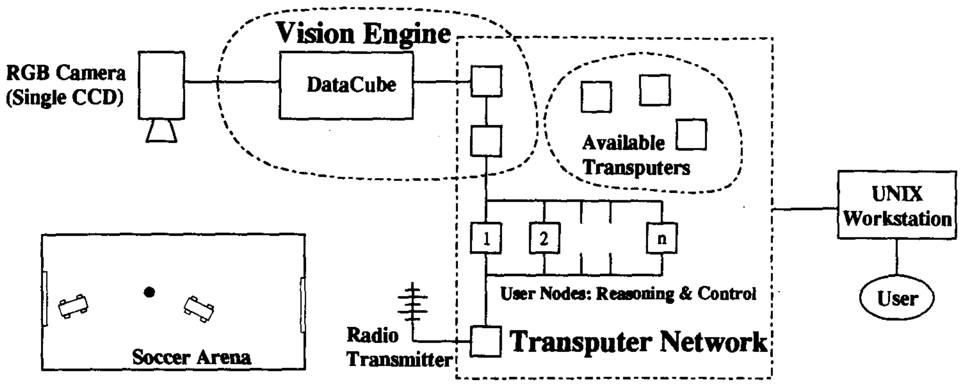

Figure 9: Hardware Setup

which is to perform reasoning, planning and control is shielded from the some of the complexities in the world. The input to the user program is a sequence of vectors describing the locations of all the objects. The output from the user program is the sequence of control signals sent to the vehicle. The control signals are *throttle* and *steering angle*. These control the power given to the motors and the angle of the front wheels.

This interface is of significant benefit to the user. It allows one to focus on making the robot "do the right thing" instead of worrying about implementation details. It is possible to create this interface since there is a robust vision system and a reconfigurable network of off-board computers. Another benefit this abstraction provides is that user programs can be connected to either a simulator or the real system. The simulator is described fully in Appendix B. It has proved to be an invaluable tool in developing robot programs.

4.1.4 System Latency

The vision system runs at the video standard of 60 Hz (τ = 16.7 ms). The rate of our commercial R/C equipment is 50 Hz (τ = 20 ms). The rates do not match and there is no mechanism for providing synchronization. This problem is ignored by the action schema and in the internal model of the vehicle dynamics. The rate mismatch problem causes an increase in the latency of the servos of between 0 and 20 ms. It is possible for control signals to be overwritten in the output buffer, however this will have no significant impact because control signals typically change smoothly except at separated discontinuities. The overall system latency (including rate mismatch latency) is estimated at 140 ms, but this is mainly due to mechanical servo-travel time. As a result, ignoring the problem has only a small effect on the controllability of the robots.

4.1.5 Soccer with Monster Trucks

1/10 scale radio controlled trucks as well as the 1/24 scale racing cars have been used to play soccer. In Figure 10 a racing car, Zeno, is in the foreground and a truck, Heraclitus, is in the background. (The names are inspired by Monty Python's soccer-playing philosophers sketch. Zeno and Heraclitus were two Greek philosophers particularly concerned with dynamic worlds.) A 12 inch (30 cm) ruler is also visible. The trucks were the original vehicles used in soccer-playing experiments. The field used with the truck was 10 ft. (3 m) by 10 ft. which is too small because the truck's minimum turning circle is about 7 ft. In such a constricted environment, navigation was difficult and resulted in odd-looking path plans. The Dynamite testbed, in contrast, has a width of about 2.5 turning circles and a length of twice that. There is more than enough room to play and experiment in.

4.1.6 Non-holonomic Constraints

As mentioned earlier, the robots used in the Dynamite testbed are radio controlled cars. These robots cannot translate and rotate freely in the plane, unlike the omnidirectional robots typically used in indoor mobile robot experiments, such as Shakey. Car-like robots are subject to *non-holonomic* kinematic constraints that limits the change in a robot's direction of motion [Lat91]. A holonomic constraint is a constraint that is a function of only the current configuration (position and orientation) and time. Non-holonomic constraints are constraints

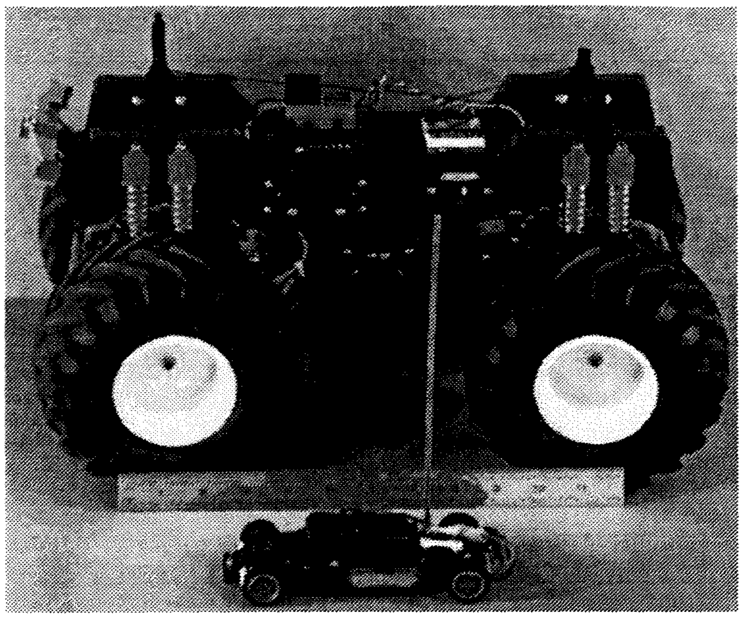

Figure 10: Two Dynamo Vehicles: Zeno (front) and Heraclitus (rear)

with higher order (with respect to time) terms such as velocity that cannot be integrated out. In practical terms, this means that cars cannot move sideways and have motion that is bounded by minimum turning radius circles. Non-holonomic constraints can alternatively be interpreted as constraints that do not reduce the size of the configuration space of the robot.

The non-holonomic constraints affect the problem of robot control. Since motion is constrained, it is no longer possible to move in the direction of the desired destination. Although motion can be accomplished without motion planning, there is a stronger need for planning than with omnidirectional robots. Real life problems such as parallel parking are connected with motion planning. Cars need to move backwards when parallel parking close to the curb, just as the soccer-playing robots drive backwards to make a shot when the ball is very close to a wall.

4.1.7 Design Decisions

The Dynamite testbed has been engineered so that some important issues in the "real world" are ignored. This reflects a careful research strategy where the problems faced in dynamic domains have been examined to the exclusion of other problems.

The vision system is simple, but this thesis is not about a new approach to vision, nor about building robots to roam around offices or solve other industrial problems. A single off-board camera allows us to track at least six (and possibly more) robots at a time in a fixed area, and that is all that is needed for experiments with multiple robots. Off-board computation is used because it is easier to modify software and the few conceptual gains to be made from engineering an on-board system are not relevant to this thesis.

In Section 2.1.1 a number of characteristics of robotic domains were introduced. The Dynamite testbed is a structured, simple environment. (i.e. not unstructured and complex.) The problem of sensor ambiguity has been engineered away. Coloured blobs are mapped into car and ball location and the locations of the goals and the walls of the field are known a priori in the fixed world coordinates. The camera is calibrated to a coordinate system that is consistent with the fixed locations. The goal markings discernible in Figure 8 are for the benefit of humans only; it has no effect on the vision system. The two important characteristics that do apply to soccer-playing are that it is unpredictable and dynamic.

Although the complexity of the testbed has been minimized, the experiments explore a challenging dynamic world. The cars are able to drive safely at scale speeds upwards of 80 km/h in a confined space. These are autonomous mobile robots that function in a near-real world. Robots playing soccer sounds like fun, but in no way is this a toy experiment.

4.1.8 Importance of the testbed to this thesis

An important disclaimer: the author did not build this testbed. Although he has put a significant amount of time and energy into it, others have contributed far more. This thesis should be evaluated on the usefulness of reactive deliberation and not on the construction of the Dynamite testbed. The testbed was, however, necessary for testing reactive deliberation and its *use* is an integral part of this thesis.

4.2 The Soccer Controller Implemented in Reactive Deliberation

4.2.1 Overview

Figure 11 shows the soccer-playing controller based on reactive deliberation. The Executor box lists the action schemas, while the Deliberator box lists the behaviours. The behaviours are responsible for accomplishing objectives, whereas the action schemas are primitive operations or activities that the robot is capable of performing. For example, the *go home line* behaviour plans a path from the current position to a position in front of the robot's home goal, while the *follow path* schema causes the robot to track a specific path trajectory. The details of behaviours and action schemas will be explained further in the following subsections.

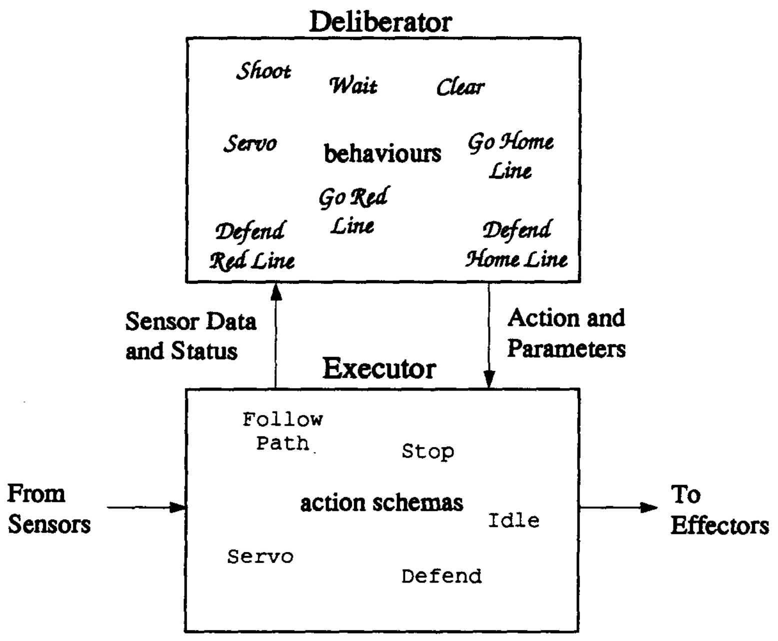

Figure 11: The Reactive Deliberation Controller

The deliberator and executor each run on their own transputer nodes (processors) and communicate through message passing. The executor sends position and status information to the deliberator at 60 Hz. The deliberator asynchronously updates the current action of the executor when appropriate.

The controller has been implemented in C. (This is the only programming language our lab supports on the transputer network.) It has been constructed over a period of a year, yet only an estimated 3 or 4 person-months have gone into design, coding, and testing. The various parts of the controller total more than 5000 lines of code. This figure excludes the simulator and vision system, since they were written by other members of the laboratory.

4.2.2 Executor

Table 3 shows the action schemas that have been written for the executor. Only one action schema is enabled at a time. The enabled schema sets the robots control outputs, throttle and steering angle, and sends messages to the active behaviour in the deliberator when it is having problems. The enabled schema can also transfer control to other schemas. For instance, the *follow path* schema transfers control to the *stop* schema when it has reached the end of its path. Similarly, the *stop* schema transfers control to the *idle* schema when the robot has stopped moving.

The *follow path* schema is by far the most complex one, since the robot is subject to non-holonomic constraints. This schema follows the path (that consists of circular arcs and straight line segments — see Appendix C) to within a certain tolerance measured in absolute position and heading errors. An example of a planned path and the actual path followed is shown in Figure 12. The path plot is for the center of the car. The box in the middle indicates the dimensions of the robot.

Action Schema	**Parameters**	**Function**
Follow Path	Path Plan.	Follow the path trajectory while minimizing tracking error.
Servo	none.	Drive at ball until it is struck (provided this will advance the ball).
Defend	Offset of defense line.	Stay between the ball and the net and intercept incoming balls.
Stop	none.	Reduce the speed of the car to zero.
Idle	none.	Do nothing.

Table 3 Action Schema for Soccer-Playing

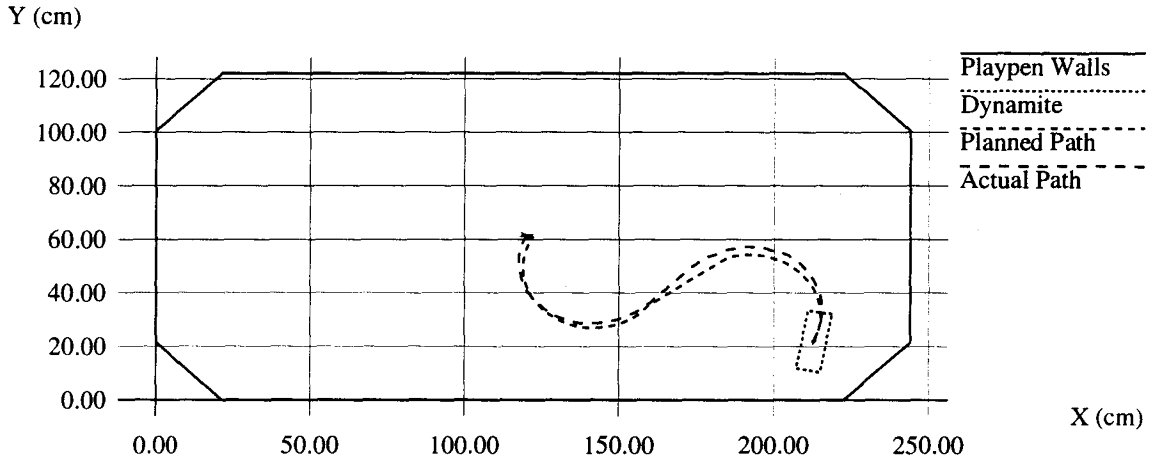

Figure 12: Path Plot for the Car-like Dynamite

Even the *servo* schema is atypical of simple behavioural routines. This schema tries to servo the robot into the ball. However, rather than just driving straight at the ball, this routine servos the robot to a future location of the ball that is predicted using an internal model of the ball's dynamics.

The *defend* schema alternates between two modes. Normally, the robot stays between the ball and the center of the net. However, if the projected motion of the ball will carry it past the end of the robot, the robot moves to intercept it in an effort to keep the ball away from the net.

The levels of performance in control that are demonstrated in this thesis are due principally to the use of feed-forward control. At low speeds, feed-forward control is not needed, and a simple fuzzy logic controller is sufficient for accurate control of the robot. However, at higher speeds (1 m/s), simple control routines are not adequate. The future position and velocity of the robot is predicted using information about the latency in the sensing and control and a model of the robot dynamics. Feed-forward control uses the predicted state of the robot to determine the control signals, rather than the current state.

There are three executor subsystems that run independently of the enabled action schema. Alternatively, one could think of these routines as being instantiated in each schema. The sensor processing subsystem receives position data for all the objects in soccer field from the vision engine at 60 Hz. A least-squares approximation is used to estimate the velocity of the ball and the cars. The communications subsystem updates the planner with position and velocity data at 60 Hz. The collision detection subsystem automatically transfers control to the stop schema to prevent collisions with other robots.

4.2.3 Deliberator

The deliberator implemented using reactive deliberation does not contain a traditional planner. Instead, the deliberator is composed of behaviours, listed in Table 4, that represent the goals of the robot. The action schema invoked by each behaviour is listed in the second column. It is possible for a behaviour to select different schemas, although this was found not to be needed. The last column identifies the goal of the behaviour. Each behaviour is responsible for generating a bid and planning an action. However, in this implementation, planning is limited to motion planning.

Behaviour	Action Schema	Goal
Shoot	Follow Path	Shoot the ball into the net (either directly or off a wall).
Clear	Follow Path	Clear the ball from home end of the soccer field
Servo at Ball	Servo	Advance the ball towards the opponent's goal.
Wait	Idle	Wait for an opportunity to do something.
Go Red Line	Follow Path	Drive to the center of the soccer field.
Defend Red Line	Defend	Play forward defence.
Go Home Line	Follow Path	Drive to the home net.
Defend Home Line	Defend	Play goalie.

Table 4 Soccer-Playing Behaviours in Reactive Deliberation

Behaviours generate bids in the range [0,10] (floating point). The bids are simple algebraic formulas that are easily computed. Each behaviour has a basic bid that is modified through the addition and subtraction of weighted factors that depend on the environment and the results of motion planning. Complex conditions in the environment, such as the distance of the ball from one's home goal, are converted to factors in the range [0,1]. One exception is path plans that are converted to factors based on the log of the expected travel time.

The *shoot* behaviour generates bids in a typical manner. It has a base bid that is modified with environment factors and planning results. Some of the factors include: the speed and heading of the ball, the Euclidean distance to the ball, the Euclidean distance of the other car to the ball, and the speed of the other car. Since *shoot* plans and follows a path, its bid is biased upwards when it has begun following a path (this is to provide stability in the bidding process).

The bids used here are crude estimates of the expected utility of the actions proposed by each behaviour. The utility of each action depends on the internal importance of the goal (the base bid), environmental conditions (factors), and the results of planning (log factors). The factors are combined with weights so that bids are an estimate (according to the designer) of utility.

The system currently alternates between the ruling behaviour and other behaviours. There is no time limit (in theory) on the computations, although in the implementation all the behaviours are restricted to bounded computations that do not last longer than 300 ms. This is a maximum, and many behaviours do not perform planning when their bids are too low, so there is little need for a more complex scheduling scheme.

The non-holonomic constraints on the car-like robots make motion planning difficult, since it is not possible to move the robot in an arbitrary direction. An algorithm based on Jumps [FW88] is used to provide fast motion planning. A number of paths are generated and the lowest cost (in terms of time) is selected. Extensive details can be found in Appendix C.

4.3 The Experiment

In this section, a series of soccer-playing experiments that are used to evaluate reactive deliberation are described. Different versions of the controller will be compared to establish the utility of the architecture.

The deliberator is capable of performing at a number of levels of ability that reflect the incremental development of the controller. The simplest version alternates between offensive and defensive behaviours without any evaluation of the environment. The next gives control to the deliberator only when the car is stationary and the *idle* action schema is active. The final version exhibits the concurrent deliberation and execution that characterize reactive deliberation.

The purpose of the soccer playing robot is (not surprisingly) to play soccer. Success in soccer is measured by the number of soccer games won compared to the number lost. The primary objective in soccer is to win the game. Although real soccer players may have secondary goal such as avoiding injury and personally scoring as many goals as possible, these will not be addressed. The way to win a game is to score more goals than the other team. This involves scoring goals on the other team and preventing goals from being scored against one's own team.

Soccer is a dynamic game; the locations of the ball and the robots are constantly changing. To be successful in this domain, the robot must be able to accomplish a collection of meaningful tasks and not just exhibit "cute" behaviours. In soccer, there is an easily identified measure of success — the score at the end of a soccer game. The competitive nature of soccer allows us to compare the adequacy of the robot players either with other autonomous robots or with robots under direct control of a human being. Perhaps a better description for the soccer domain is *adversarial*, rather than dynamic, since the opposing player is working directly against the objectives of the robot.

The three controllers compete for the LCI Cup (Laboratory for Computational Intelligence Cup). The LCI Cup is a round-robin tournament where each controller plays the others in a game of soccer. Presumably the fully implemented controller will be victorious, however, differences in the levels of play may indicate the relative importance of the different architectural elements. Part of the tournament includes games where controllers play copies of themselves! The purpose of these games is to get an idea of how random the games are.

The soccer games are one-on-one tournaments between robots. With only one robot on each team, the focus is on dynamic domains and not on multi-robot cooperation. Each games is ten minutes in duration, with five minute halves. At halftime the robot chassis are switched to prevent a bias based on differences in the mechanical components. There is only one game played between each pair of controllers since the length of each game is arbitrary. For example, the total difference in score in two one minute games would be the same as with one two minute game. The game duration was arbitrarily set at ten minutes, and this proved to result in sufficiently high scores to allow differentiation between controllers.

One aberration in the testing procedure is that the ball was *not* centered after each goal. There is no external process that referees the game and tells the robots when a goal has officially been scored and when to resume play. As a result, the robots are continuously playing soccer: each half-game is five minutes of continuous play. To compensate for one robot repeatedly scoring goals, a robot that had been scored upon had to clear the ball from it's end before any more goals were counted. This greatly simplified the testing procedure.

Soccer games with human beings have not been included in the LCI Cup. Humans are at a great disadvantage playing soccer through a radio-controlled car since it is difficult to achieve the accurate control needed to successfully shoot the ball. Games between the middle version of the controller and many individuals have been held. In these informal games the computer usually won, however, a skilled and practised human could certainly win on a regular basis.

Chapter 5
Results and Evaluation

5.1 The LCI Cup

The winner of the 1993 LCI Cup is the fully implemented reactive deliberation controller. Table 5 shows the scores for all the games. The score for the controller in the row is the first number, and the score for the controller in the column is the second. (e.g. 11 - 1 means that the reactive deliberation controller scored 11 goals while the no-wit controller scored only 1.)

Controller	**No-wit**	**Half-wit**	**Reactive Deliberation**
Reactive Deliberation	11 - 1	7 - 4	3 - 3
Half-wit	6 - 3	5 - 5	
No-wit	8 - 2		

Table 5 Final Scores in the LCI Cup

The *no-wit* controller does not select goals based on the state of the world, but simply alternates between offensive and defensive behaviours. The *half-wit* controller gives control to the deliberator only when the car is stationary and the *idle* action schema is active. This occurs either when a schema has terminated or when a set maximum time is exceeded.

The *reactive deliberation* controller performs concurrent deliberation and execution, as is intended of the architecture.

There is an element of chance in these soccer games: the scores are a result of a complex set of interactions between the robots and their environment. These results are partially repeatable because the same general results will emerge, but the actual scores will be different. For a better estimate of the results, the duration of the soccer game could be extended.

The rank of the controllers from best to worst is: reactive deliberation, half-wit, and no-wit. This ranking is probably reliable since the better controllers scored nearly twice as many goals (7–4 and 6–3 are the scores) as the controller ranked beneath it. The results of the games played with the same controller indicate that the better two controllers (reactive deliberation, half-wit) generate fairly constant performance, while the no-wit controller produces somewhat random performance. The scores (5–5 and 3–3) should be interpreted as close scores, rather than identical. They really do not show the underlying randomness that *is* present as might be shown by a listing of when the goals were scored. The score 8–2 in the no-wit vs. no-wit game is a result of the almost random playing strategy of that controller.

5.2 Discussion

The difference in score between the reactive deliberation and half-wit controllers is significant. The only difference between these two controllers are that reactive deliberation considers alternate actions all the time, while the half-wit controller only when an action schema terminates. The reactive deliberation controller selects transient goals as frequently as possible and can interrupt actions. The half-wit controller is like the traditional planning-based architectures: alternate actions are considered only when the current action has terminated. This is evidence that the frequent evaluation of transient goals is critical to success in dynamic worlds. This is not a surprising result — what is surprising is that architectures such as Shakey and RAP do not include provisions for this.

The level of performance that the robots were able to achieve is partially due to the use of internal world models. As was mentioned in the last chapter, an internal model of the dynamics of the robot is used to provide feed-forward control. This is not a superfluous element; it really is necessary for the robots to operate at speeds of 1 m/s. Brooks argues that "the world is its own best model" and that internal models are inappropriate [Bro91]. Experiences with soccer-playing robots suggest that Brooks' slogan is wrong.

If internal models are necessary, then how can the complete reactive architectures be suitable for dynamic domains? The subsumption architecture prohibits the use of explicit internal models, thus limiting the knowledge a designer can encode in a robot. The concrete-situated approach focuses on building patterns of interaction with the environment based on the way the world is *now*. To use feed-forward control, a history of control signals is needed. The situated automata approach only has internal state in the perception part of the controller, so one of the perception outputs could be the predicted location/state of the robot. Of the three architectures, only the situated automata is not ruled out on the basis of internal model.

The performance of the robot players is largely a function of the selection of transient goals. The bidding mechanism has been fine-tuned through an iteration cycle with observations of soccer games followed by incremental changes to the behaviours. A useful abstraction that helps with this is the *routines* of action from the concrete-situated approach. The idea here is that a pattern of activity such as clear the ball, shoot, defend red line, . . . In the case of soccer-playing, the construction of successful robots does involve careful attention to patterns of activity. This is an emergent result of the soccer-playing experiments.

In conclusion, the reactive deliberation controller plays a nice, although not flawless, game of soccer. The competitive nature of soccer places very strict time constraints on the robots and allows different controllers to be easily compared. The dynamic and unpredictable nature of one-on-one robot soccer favours approaches that are concerned with the immediate situation and reactive deliberation takes advantage of this.

Chapter 6
Conclusions

6.1 Summary and Contributions

The purpose of this thesis is to investigate the structures needed in robot architectures for dynamic environments. The architectures surveyed were argued to be inadequate for dynamic domains. The weakness of the partitioned architectures such as RAP is due to the commitment to arbitrary length plans and the infrequent evaluation of transient goals. Complete reactive architectures such as the subsumption architecture are limited by the lack of internal models and restricted computational structures.

Reactive deliberation has been designed to present a set of structural elements needed in dynamic domains. The main features of the architecture are the following:

- Reactive deliberation is partitioned into an executive layer and a deliberative layer.
- The executive layer interacts with the environment in real-time through a set of *action schemas* that receive run-time parameters from the deliberative layer.

- The deliberative layer is composed of goal-oriented modules called *behaviours* that select transient goals (actions) in a distributed manner through utility estimates.
- Behaviours provide modularity in the design of robot controllers and allow effective goal arbitration as well as the distribution of limited computational resources.

The theoretical contributions of reactive deliberation to the design philosophy of robot architecture for dynamic environments are the following:

- The transient goals (actions) of a robot need to be evaluated at a rate commensurate with changes in the environment.
- A new split between reasoning and control is needed since action selection cannot be suitably determined independently of detailed planning.
- Goal-oriented *behaviours* are a useful abstraction.

Soccer is a demanding dynamic domain; the locations of the ball and the robots are constantly changing. A controller based on reactive deliberation has been implemented to allow robots to compete in one-on-one games of soccer. Current functionality includes motion planning, ball shooting and playing goal. The robots can drive under accurate control at speeds up to 1 m/s, while simultaneously considering alternate actions.

In a soccer tournament called the LCI Cup, the effectiveness of the controller has been demonstrated. Specifically, the importance of modifying goals in response to changes in the environment has been shown. Further, the results suggests that the architectural elements in reactive deliberation are sufficient for real-time intelligent control in dynamic environments.

It has been postulated that reactive deliberation is nothing more than a kludge that has been hacked together solely for the purpose of soccer-playing robots. Although the soccer domain was the impetus for the development of reactive deliberation, the guiding principles of its design were drawn from other architectures. The extensive survey of other architectures contained in Chapter 2 stands as testimony of the attention that has been paid to previous work in this area. Reactive deliberation is not a kludge.

One criticism that has been made of this work is that the soccer domain is too reactive. Many tasks in the real world do not require that an agent be highly reactive to changes in the environment. For such cases reactive deliberation is irrelevant. The usefulness of reactive deliberation is in those tasks where a robot must be reactive to changes in the environment.

6.2 Future Work

Reactive deliberation is an architecture that focuses on the problems associated with dynamic domains. One possible extension to reactive deliberation is to address problems such as sensing and world modeling that have been hitherto ignored in this thesis. Perhaps more important than this is the need to test reactive deliberation with robots operating in other environments.

The cornerstone of reactive deliberation is action selection through behaviours that report utility estimates. One extension would be to develop a more formal mechanism for estimating the utility of actions in robots. Alternatively, attention could be focused learning mechanisms and this would preclude the need for a more formal mechanism. With either extension, good abstractions are needed to minimize complexity, yet there are few theories on how to find good abstractions. Finally, reactive deliberation could be extended to multiple vehicles to test the appropriateness of local utility estimates with multiple robots.

The problem of robot control in dynamic domains is still not fully solved. The outstanding problem specific to this area is that of scarce computational resources. The desire to respond rapidly to changes in the environment is in conflict with the need to assimilate new information and plan alternate actions. The complete reactive architectures use limited computational models and sacrifice the assimilation of information, while the partitioned architectures typically sacrifice rapid responses. The solution taken in reactive deliberation is to focus attention on the immediate situation to perform only directly relevant computations. In general, the computations that are appropriate with fixed computational resources are a function of the environment and the tasks the robot must achieve. More attention is needed to fully explore this trade-off.

The assessment of robotics architectures is difficult since they describe a design process. This thesis has demonstrated that reactive deliberation is a plausible architecture and its advantages over other approaches have been argued. Just as reactive deliberation has been influenced by previous architectures, future architectures may be influenced by reactive deliberation. Reactive deliberation is an incomplete architecture specification that is focussed on dynamic worlds; no claim has been made that it is the ultimate robot architecture. Rather, robot architectures should be seen as a succession of improvements.

The experiments performed in this thesis only compare versions of a reactive deliberation controller with itself. The experiments should be extended so that reactive deliberation can compete with other architectures and with a human operator. Since motor control of the robots is difficult for human beings, a user interface could be constructed where the human does the deliberation and the computer does the path planning and control. This would pit the intelligence of reactive deliberation against the intelligence of a human operator and provide a better measure of the effectiveness of this architecture.

Appendix A
The Vision Subsystem

The Vision Engine is able to track the world space position of the centroids of coloured targets placed on the robots at 60Hz using the off-board camera. Figure 13 shows the pipelined setup of the hardware. There are four stages in converting from RGB video to (x,y,θ) configurations in the world: (1) classification into regions (on the DataCube), (2) finding the moments of the convex regions (on the MAXTRAN), (3) converting from (row,column) to (x,y) for each region, and (4) grouping regions into logical objects (i.e. (x,y) $\rightarrow$ (x,y,θ)).

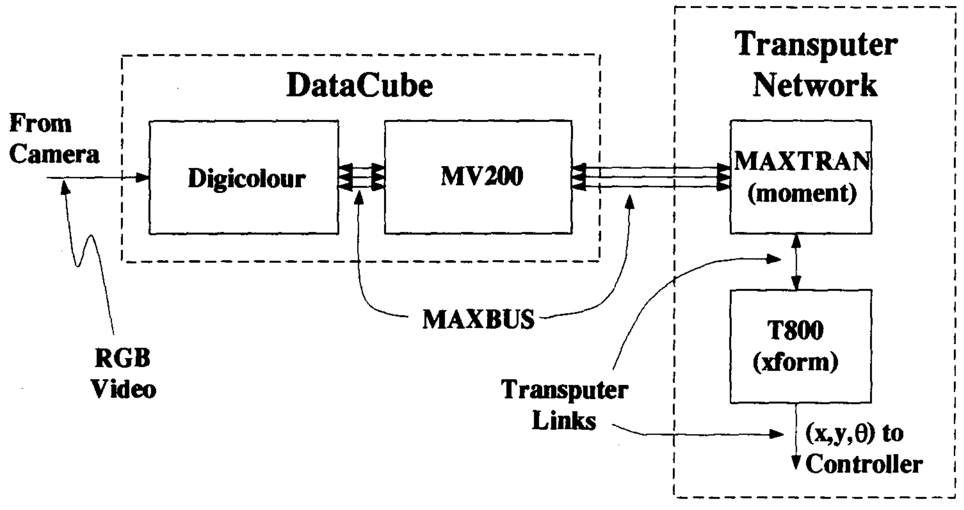

Figure 13: The Vision Engine

The colour video camera is placed in a fixed position so that the entire soccer field/workspace is visible, as is shown in Figure 14. The video is fed into the Datacube hardware which is programmed to preprocess the image to simplify and speed up the centroid calculations. The Digicolour converts the analog video signal to a digital format. The MV200 converts the incoming colour pixels from RGB colour space into HSV colour space and then classifies them as belonging to either background or a designated colour class. An example of the processed output is shown in Figure 15. The coloured blobs for the car and the ball show up as black in the figure, while the background is classified into black and white depending on the value of the image point. This stream of classified pixels is then run-length encoded and passed onto the transputers for further processing. The mapping of stage 1 is: video $\rightarrow$ blobs.

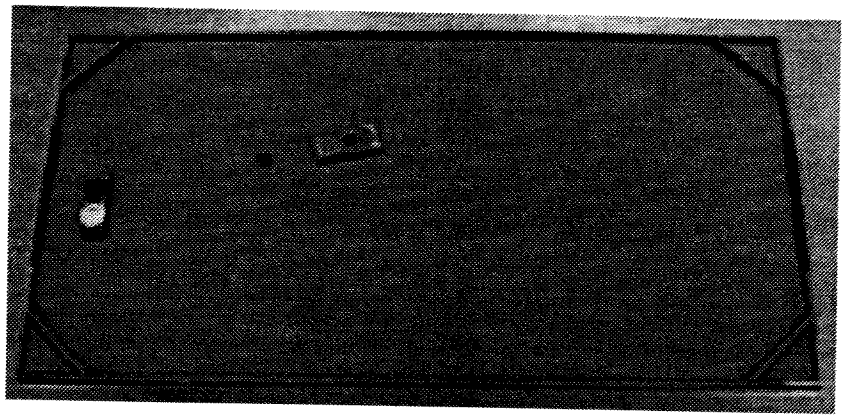

Figure 14: Output from the real camera

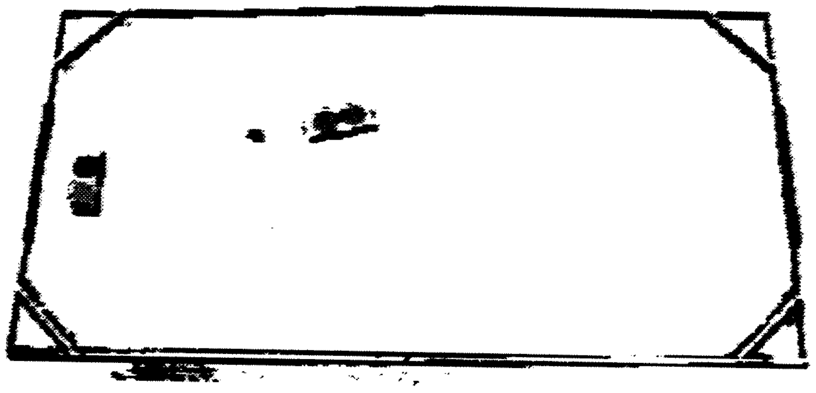

Figure 15: Processed output from the DataCube

The *moment* program on the MAXTRAN transputer examines the incoming processed image and finds the centroid, in screen coordinates, of each disjoint, connected region. The mapping of stage 2 is: blobs $\rightarrow$ (row,column) coordinates of each blob.

The *xform* program performs stages 3 and 4 of the transformation on a single transputer node. In stage 3, xform corrects for radial lens distortion and finds the corresponding point in world space which satisfies both the perspective projection for the camera's view position and the constraint that all points must lie in a known plane (parallel to the floor). The mapping of stage 3 is (for each blob): (row,column) $\rightarrow$ (x,y).

Once the world space position of each region has been found, it is necessary to map these onto targets. In the current implementation there will be only a single target of each colour class visible, and the region with the most pixels for each class is assumed to be the target. Then, since each robot has two targets on it, the position and orientation of the robot can be found directly from the position of its targets. The accuracy in position is on the order of a few millimeters. The mapping of stage 4 is: (x,y) blobs $\rightarrow$ (x,y,θ) logical vehicles.

The robustness of the system varies considerably. The theory is as follows. By painting the targets using fluorescent paint, and then only accepting highly saturated colours of the appropriate hue, the pixel classification can be made with almost perfect accuracy. A further advantage of picking fluorescent colours is that it is less sensitive to changes in the colour of the lighting. In reality, Stewart and I have spent hours twiddling around with the boundaries in hue, saturation, and value for the colour classes. Even so, the system can produce incorrect classifications under certain conditions[2] that result in catastrophic errors. However, once these problems have been ironed out and the system is in a stable state, it works *really* well.

The system is quite fast, for tracking with video cameras, and has low latency. The Datacube preprocessing only introduces a few milliseconds of delay relative to the camera, and the moment program typically needs only 2ms of processing time to find the region's centroids (for a normal image this processing can be done while the image is being transferred to the Maxtran). The remaining calculations are fairly trivial and require at most 1ms. Once the cost of transmission is included, the entire latency of the Vision Engine is about 5ms in contrast with a separation of 16.7ms between video frames.

[2] Such as someone wearing blue jeans walking by.

Appendix B
The Simulator

The simulator has proved invaluable in the development of the robot controller described in this thesis. Although a significant amount of work is required to build a simulator, it is well worth the time spent. It is easier to use the simulator than the real testbed for the following reasons: (1) real hardware breaks down, and is unavailable from time to time; (2) you may have to timeshare physical resources such as computer hardware with other users; (3) the simulator can be used from any workstation, and multiple people can simultaneously be working; (4) when your controller goes haywire, you don't wreck your carefully constructed robot; (5) code can be tested quickly without booting up the real system (which takes time).

The version of the simulator that was used in the development of this thesis, supported a single car in an infinite plane. A ball, walls, and another car, could be hallucinated by the controller so that fairly extensive testing could be done even with a minimal simulator. All that the simulator supported was a coarse model of the vehicle's dynamics. Graphics were not used since there was only one car and a xgraph plot can be used to illustrate both the planned path and the path traversed. (See Figure 12 on page 55.) Even this minimal setup provided enough facilities to do a lot of debugging.

The visual display that the latest version of the simulator generates is shown in Figure 16. The simulator supports dynamic models for the cars and the ball. In the image two cars can be seen (an arbitrary number of cars can be used) in the soccer field with a semi-visible ball between the cars. Collisions between the ball, cars, and walls are supported as well, so soccer games can be conducted in the simulation.

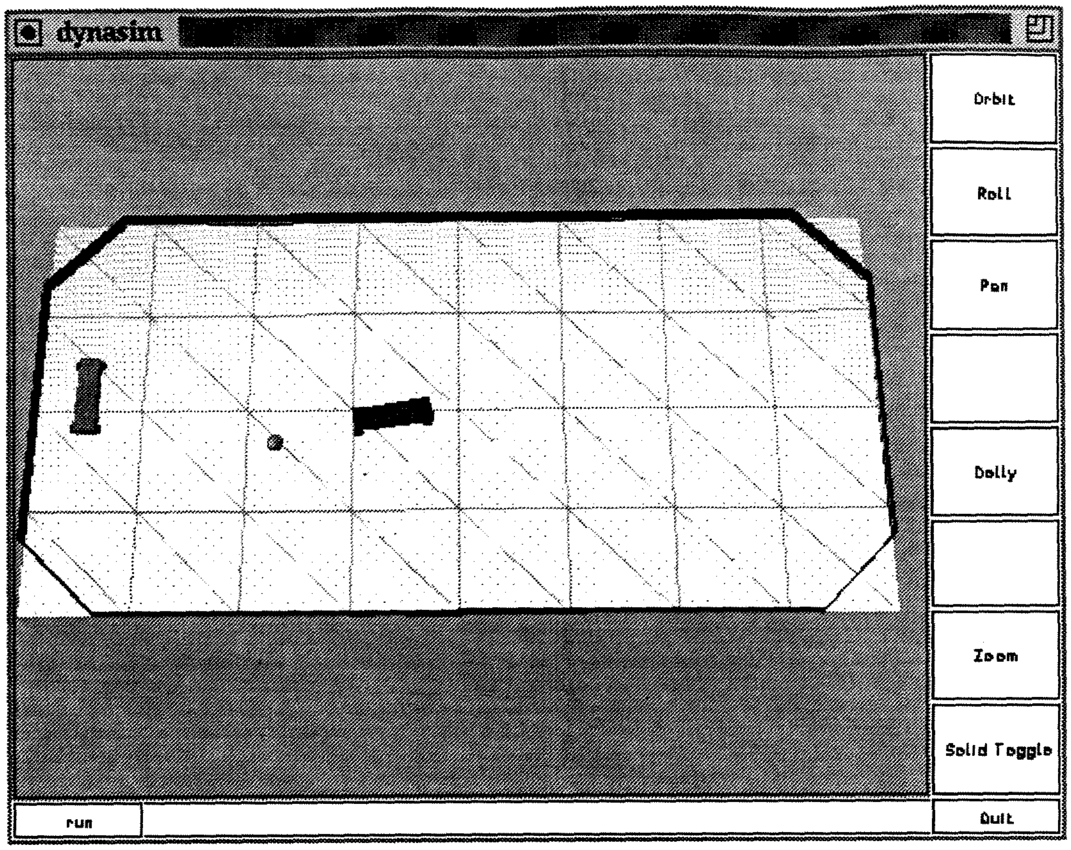

Figure 16: The Dynamite Simulator with Two Cars and a Ball

Appendix C
Path Planning

C.1 Introduction

Fast generation of motion plans is an essential part of a robot operating in a dynamic environment. The problem of planning for a car-like robot that is capable of travelling in forward and reverse is considered. Car-like robots are subject to non-holonomic kinematic constraints that limit the change in a robot's direction of motion.

Recently, there has been considerable interest in motion planning with non-holonomic constraints [JC89; RS90; BCL91; BLJ91], however, much of the work fails to meet all three criteria: 1) fast execution, 2) efficient motion plans (i.e. forward as well as reverse motion must be provided), and 3) planner can handle obstacles. There is no algorithm that satisfies all three criteria. My approach sacrifices optimality and completeness for speed, by considering a restricted class of motion plans.

The basic problem is: given an initial configuration $c_i = (x_i, y_i, \theta_i)$ in $\mathbf{R}^2 \times \mathbf{S}^1$, find a path to a final configuration $c_f = (x_f, y_f, \theta_f)$. A configuration describes the position and orientation of a robot. A car-like robot can move forward and backward, but not sideways. The rate of change of orientation with respect to path length is limited by the maximum turning angle of the wheels. The result is that the motion is bounded by a minimum turning radius circle.

The following section looks at metrics for comparing motion planning algorithms. This is followed by a survey of previous work that has been divided into two sections: one for motion planning without obstacles and one with obstacles. The algorithm used in this thesis is presented in Section C.5.

C.2 A Metric for Motion Planners

Previous work in motion planning under non-holonomic constraints has focussed on developing algorithms that compute optimal paths that are the lowest cost according to some definition of cost. Cost (cost1) is defined as the path length plus a penalty for each change in the direction of motion.

In robotics applications the *real cost* is a function of the time used to compute the path, time used following the path, and the energy used following the path. The cost defined above captures these notions relevant to the output of a path planner. The time needed to compute the path is the measure that is used to compare different approaches.

The *path length* is the total distance that the turning center of the car travels. Note that the average distance each wheel travels could also be used, but this will not, in general, produce the same costs. A *change in the direction of motion* occurs when the car changes from forward to reverse motion or vice versa. Changing direction is not desirable, since it takes time and requires energy. A penalty can be computed in units of distance that is representative of the cost of changing direction.

There are two other possible cost estimates: cost2 is just the path length, and cost3 is path length + penalty for changes in direction of motion + penalty for changing steering or curvature. Many planning algorithms use cost2, although cost1 and cost3 are more accurate. The *penalty for changing curvature* accounts for the limited rate at which the wheels of a vehicle can be turned.

Although cost3 is realistic for robots that will use these planning algorithms, the effect of changing curvature is ignored in literature. One exception is [Eln93] where tangential and normal accelerations along the path are considered. The jump method that is extended in this thesis also ignores this cost. The justification is that ignoring the penalty for changing curvature greatly simplifies motion planning. Also, an appropriately designed low-level controller can compensate for the problem.

The difference between cost2 and cost1 is simply a matter of counting the number of reversals in motion. An optimal path using cost2 is the shortest path in units of distance. An optimal path using cost1 can be thought of as an optimal trade-off between distance, time, and energy consumed. In this paper, cost1 will be used to evaluate the approach presented.

All three of the proposed cost functions implicitly account for the energy and time required to follow a path plan. The functions do not, however, account for the time required to plan a path. It is hard to evaluate this cost for two reasons. One, a robot may need to compare multiple paths before selecting one. Two, it is difficult to compare algorithms, since the approaches are varied.

C.3 Motion Planning in an Infinite Plane

In this section, motion planning for a robot in an infinite plane will be considered since this is easier than motion planning in an arena or with obstacles. For instance, optimal paths under cost2 can be easily generated.

A set of optimal paths for a car that cannot change direction is described in [Dub57]. Since reversals are not allowed, cost2 is used and optimal means shortest. An optimal path consists of 3 subpaths that are either circular arcs (C) or straight line segments (S). Such paths are of the form CSC or CCC. The circular arcs have curvature inversely proportional to the radius of the minimum turning radius circle. There are 6 possible combinations of segments that can result in an optimal path, however there is no explicit formula for this path. The only way to find the optimal path is to try each one of the 6 combinations.

The results of [Dub57] were extended in [RS90] to a car that can change direction. The results are analogous. Here an optimal path consists of 5 subpaths, where subpaths can be zero in length. The paths are of the from CCSCC. There are 48 paths that must be considered to determine an optimal path. This method for computing paths is expensive even in the absence of obstacles. The paper describes a method for determining the shortest path using cost2. It is not clear that cost1, which is a preferable measure, can be incorporated easily, since paths that are currently discarded may be optimal.

[FW88] created the notion of jumps, where a *jump* is the concatenation of 3 subpaths: a circular arc, a straight line segment and another circular arc (or CSC). The circular arcs are part of a minimum turning radius circle. [Lat91] has used jumps for motion planning with a car that cannot change direction. In such a case, there are at most four possible jumps between two configurations. An example of such paths are shown in Figure 17. Each path is labeled with two letters. The first letter corresponds to the orientation of initial circle, and the second to the final circle. For instance, "RL" refers to the path where the minimum turning radius circle to the *right* of the initial configuration is chosen as well as the circle to the *left* of the final configuration. The algorithm presented in this paper is an extension to this approach.

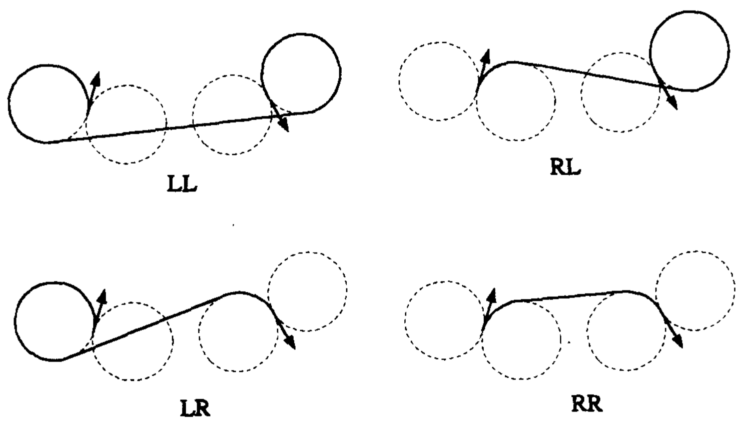

Figure 17: Jump paths with unidirectional motion

C.4 Motion Planning in a Convex Arena

In this section, we consider the problem of motion planning where the robot is located in a convex arena. It is assumed that the arena is the only static obstacle. There may be other moving objects, but they will not be considered as it is assumed that their motion is unpredictable and that collision detection can be considered an on-line problem that is outside the scope of this thesis.

[RS90] have a method for computing optimal paths in an infinite plane that has not been extended to motion planning with obstacles. A motion planner that produces optimal paths in the presence of obstacle has yet to be created. None of the following methods are optimal.

Using the original jump notation for a unidirectional car, [FW88] propose an algorithm for deciding whether there exists feasible path by considering contact points with obstacles. This approach is not sufficient for the problem under consideration since it does not consider paths with reversals.

Geometric motion planners produce path plans that do not in general obey non-holonomic constraints. [BLJ91] discuss an iterative motion planner that runs as a post-processing stage after any geometric planner. The iterative planner modifies the path within a tolerance bound to provide a path that does obey non-holonomic constraints. This path will be locally optimal, but there is nothing to ensure that it is globally optimal. Further, since the two planners are disjoint there is little evidence to suggest that efficient paths will be produced.

[JC89] approach the problem of motion planning by discretizing configuration space and building a directed search graph. Path planning can be done by searching this graph. The results presented are for generating paths without reversals, although it can be extended to include paths with reversals at the expense of doubling the branching factor in the graph. The main problem with this approach is that a large number of possible paths are computed and discarded. It is too slow. It is interesting to note that the performance of this algorithm will improve as complexity increases and free space is reduced.

[MC92] describes a method for planning that uses skeletons (safe path segments). A path consists of three subpaths: a path from the initial configuration to a skeleton, a concatenation of skeletons, and a path form the final skeleton to the final configuration. The approach is based on generating a table lookup for the shortest feasible path distance between two points in configuration space that can then be used to compute skeletons. This approach, although complex, seems to be practical for a robot operating in an environment with many obstacles. The approach, however, does not meet our earlier requirement of fast execution.

C.5 A Fast Motion Planner

The approach used in this thesis, called *jumps+*, is an extension of the jumps approach to a car that can change direction. If there were no restrictions placed on motion there would be 64 paths to be considered. (4 pairs of circles) $\times$ (4 inter-circle tangents) $\times$ (2 directions to reach the tangent on each circle)^2 = 64. The paths generated will not always be optimal. If an optimal path is of the form CSC (Circle arc, line Segment, Circle arc), it will be found using jumps+. There are some optimal paths that cannot be found, such as CCC and CCSCC paths.

To shoot or drive into a ball in soccer, the robot can only approach its final configuration in one direction. As a result, only 32 of the 64 possible paths are of interest. A simple approach is used where all 32 possible paths are generated, and then tested for collisions in order of increasing length. This is not very inefficient, and could be greatly improved by interleaving path generation with collision tests.

This approach is reasonable for a convex arena. However, the planner is flawed because it a) can only find some optimal paths, but not all of them and b) fails to produce any path plan with certain initial and final configurations. It does, however, perform much less computation than any of the other path planning approaches, by considering only a restricted set of paths.

Possible extensions include the addition of static and moving obstacles. However, increasing the number of obstacles may not be feasible, since this will further deteriorate the chances that a CLC path will exist between any two configurations. The problem of motion planning with moving obstacles could potentially be solved under this approach since it is fast and could respond in real-time.

Bibliography

[AC87] Philip Agre and David Chapman. Pengi: An implementation of a theory of activity. In *Proceedings of AAAI-87*, 1987.

[AC90] Philip Agre and David Chapman. What are plans for? In Pattie Maes, editor, *Designing Autonomous Agents: Theory and Practice from Biology to Engineering and Back*, pages 17–34. M.I.T. Press, 1990.

[Alb81] James Albus. *Brians, behaviour, and robotics*. BYTE Publications, 1981.

[Asi90] Isaac Asimov. *Robot Visions*. Peguin, 1990.

[BCL91] J-D Boissonnat, A. Cerezo, and J. Leblond. Shortest paths of bounded curvature in the plane. Technical Report No. 1503, I.N.R.I.A., 1991.

[BKL$^+$93] R. Barman, S. Kingdon, J. Little, A. K. Mackworth, D.K. Pai, M. Sahota, H. Wilkinson, and Y. Zhang. Dynamo: real-time experiments with multpile mobile robots. In *Proceedings of Intelligent Vehicles Symposium — Tokyo*, 1993.

[BKM$^+$93] R.A. Barman, S.J. Kingdon, A.K. Mackworth, D.K. Pai, M.K. Sahota, H. Wilkinson, and Y. Zhang. Dynamite: A testbed for multpile mobile robots. In *Proceedings IJCAI Workshop on Dynamically Interacting Robots*, 1993.

[BLJ91] Andre Bellaiche, Jean-Paul Laumond, and Paul Jacobs. Controllability of car-like robots and complexity of the motion planning problem with non-holonomic constraints. In *Proceedings of the International Symposium on Intelligent Robotics*, 1991.

[Bro86] Rodney A. Brooks. A robust layered control system for a mobile robot. *IEEE Journal of Robotics and Automation*, RA-2:14–23, 1986.

[Bro90] Rodney A. Brooks. Elephants don't play chess. In Pattie Maes, editor, *Designing Autonomous Agents: Theory and Practice from Biology to Engineering and Back*, pages 3–15. M.I.T. Press, 1990.

[Bro91] Rodney A. Brooks. Intelligence without reason. Technical Report 1293, M.I.T. A.I. Lab, 1991.

[Cha91] David Chapman. *Vision, Instruction, and Action*. MIT Press, 1991.

[DB88] Thomas Dean and Mark Boddy. An analysis of time dependent planning. In *Proceedings of AAAI-88*, 1988.

[Dub57] L.E. Dubins. On curves of minimal length with a constraint on average curvature and with prescribed initial and terminal positions and tangents. *American Journal of Math*, 1957.

[Eln93] A. Elnagar. Piecewise smooth and safe trajectory planning for mobile robots. In *Proceedings of the International Conference on Intelligent Robotics and Systems*, 1993. (Forthcoming).

[Fir89] R. James Firby. *Adaptive Execution in Complex Dynamic Worlds*. PhD thesis, Yale University, 1989.

[Fir92] R. James Firby. Building symbolic primitives with continuous control routines. In *First International Conference on Artificial Intelligence Planning Systems*, 1992.

[FW88] S. Fortune and G. Wilfong. Planning constrained motion. In *Proceedings of the Fourth Symposium on Computational Geometry*, 1988.

[Gat92] Erann Gat. Integrating planning and reacting in a heterogeneous asynchronous architecture for controlling real-world mobile robots. In *Proceedings of AAAI-92*, 1992.

[JC89] P. Jacobs and J. Canny. Planning smooth paths for mobile robots. In *In STOCS, Chicago*, pages p. 445–459. Association for Computing Machinery, 1989.

[Kae90] Leslie Pack Kaelbling. An architecture for intelligent reactive systems. In *Readings in Planning*. Morgan Kaufman, 1990. Originally in Reasoning about Actions and Plans (Morgan Kaufman), 1987.

[KD89] Keiji Kanazawa and Thomas Dean. A model for projection and action. In *Proceedings of the Eleventh International Joint Conference on Artificial Intelligence*, 1989.

[KR90] Leslie Pack Kaelbling and Stanley J Rosenschein. Action and planning in embedded agents. In Pattie Maes, editor, *Designing Autonomous Agents: Theory and Practice from Biology to Engineering and Back*, pages 35–48. M.I.T. Press, 1990.

[Kub92] Claus Ronald Kube. Collective robotic intelligence: A control theory for robot populations. Master's thesis, University of Alberta, 1992.

[Lat91] Jean-Claude Latombe. *Robot Motion Planning*. Kluwer, 1991.

[LBaJL91] J. Little, R. Barman, and S. Kingdon anf J. Lu. Computational architectures for responsive vision: the vision engine. In *Proceedings of Computer Architectures for Machine Perception*, 1991. Paris.

[Mac93] Alan Mackworth. On seeing robots. In A. Basu and X. Li, editors, *Computer Vision: Systems, Theory, and Applications*. World Scientific Press, 1993.

[Mae89] Pattie Maes. The dynamics of action selection. In *Proceedings of the Eleventh International Joint Conference on Artificial Intelligence*, 1989.

[Mae90] Pattie Maes. Situated agents can have goals. In Pattie Maes, editor, *Designing Autonomous Agents: Theory and Practice from Biology to Engineering and Back*, pages 49–70. M.I.T. Press, 1990.

[MC92] B. Mirtich and J. Canny. Using skeletons for nonholonomic path planning among obstacles. In *Proceedings of the 1992 IEEE International Conference on Robotics and Automation*, 1992.

[Min86] Marvin Minsky. *The Society of Mind*. Simon & Schuster Inc., 1986.

[MP92] Zohar Manna and Amir Pnueli. *The temporal logic of reactive and concurrent systems*. Springer-Verlag, 1992.

[Nil84] Nils Nilsson. Shakey the robot. Technical report, SRI International, 1984. Collection of Earlier Technical Reports.

[Nor92] Fabrice Noreils. An architecture for cooperative and autonomous mobile robots. In *Proceedings of the 1992 IEEE International Conference on Robotics and Automation*, 1992.

[RS90] J.A. Reeds and L.A. Shepp. Optimal paths for a car that goes both forwards and backwards. *Pacific Journal of Mathematics*, 1990.

[Sch87] Marcel Schoppers. Universal plans for reactive robots in unpredictable environments. In *Proceedings of the Tenth International Joint Conference on Artificial Intelligence*, 1987.

[Smi80] G. Smith. The contract net protocol: High-level communication and control in a distributed problem solver. *IEEE Transactions on Computing*, 29(12), 1980.

[Tyr93] Toby Tyrrell. *Computational Mechanisms for Action Selection*. PhD thesis, Edinburgh University, 1993.

[ZM92a] Ying Zhang and Alan Mackworth. Constraint nets: A semantic model for real-time embedded systems. Technical Report TR 92-10, University of British Columbia, 1992.

[ZM92b] Ying Zhang and Alan Mackworth. Will the robot do the right thing? Technical Report TR 92-31, University of British Columbia, 1992.

[ZM93] Ying Zhang and Alan Mackworth. Design and analysis of embedded real-time systems: An elevator case study. Technical Report TR 93-4, University of British Columbia, 1993.