Enhancements in virtual robotics:

Through simulation of sensors, events and 'pre-emptive' learning.

Patric Tony Eriksson

PhD
The thesis is submitted in partial fulfillment of the requirements for the award

The degree is awarded by the De Montfort University, School of Engineering and Manufacture,
Sponsoring Establishment: University of Skövde

October 1996

Enhancements in virtual robotics

Patric Tony Eriksson

PhD
1996

Abstract

Virtual robotics can be used to dramatically improve the capabilities and performance of industrial robotic systems. Virtual robotics encapsulates graphical off-line programming systems and Computer Aided Robotics (CAR). However current virtual robotic tools suffer from a number of major limitations which severely restrict the ways in which they can be deployed and the performance advantages they offer to the industrial user. The research study focuses on simulation of sensors, programming of event based robotic systems and demonstrates how intelligent robots can be trained adaptive behaviours in virtual environments. Contemporary graphical programming systems for robots can only be used to program limited sections of a robot program, since i) they do not support methods for the simulation of sensors and event detection; ii) they normally use a post-processor to translate programs from a general language to a controller specific language; iii) contemporary robots can not easily adapt to changes in their environments; and iv) robot programs created off-line must be calibrated to adjust to differences between the virtual and real robotic workcells.

The thesis introduces a generic sensor model which can be used to model a variety of sensor types. This model allows virtual sensors to work as independent devices. It is demonstrated that using simulated sensors, event-based robot programs can be created and debugged entirely off-line. Off-line programming of event-based robotic systems demands methods for realistic handling of the communication between independent devices and process. The system must also possess the ability to manage and store information describing status and events in the environment. A blackboard architecture has been used in this research study to store environmental conditions and manage inter-process communication.

Self-learning robots is a possible strategy to allow robots to adapt to environmental changes and to learn from their experience. If suitable learning regimes are developed robots can learn to detect changes between virtual and real environments thus minimising the need for calibration. Most learning is based on experience and this requires experimental data to be fed to the learning system. This thesis demonstrates that robot controllers using artificial neural networks for knowledge acquisition and storage can be 'pre-emptively learnt' in virtual robotic environments using virtual robots and simulated sensors. The controllers are able to generalise from the information acquired by the virtual sensors operating in the virtual environment. Arguably the biggest obstacle to the use of self learning robotic systems in real applications has been the need to train the 'real robots' extensively in the 'real environment'. 'Pre-emptive learning' removes this problem. Furthermore, it is therefore possible to develop and evaluate new learning regimes using virtual robotic systems. This approach provides an opportunity to create a variety of environments and conditions which would be impractical to create in a real environment(due to constraints of time, cost and availability).

Acknowledgements

I would sincerely like to thank Professor Philip Moore for his guidance, advice and support, in particular concerning this thesis, but also for his assistance with other mechatronics related matters. I feel that our relation has grown from a student-supervisor relation to friendship.
I would also like to thank my Head of Department Hans Johansson for his courage to support me and my automation-related ideas during the last eight years. This support has been invaluable for putting this thesis together. Thanks also extend to my Rektor, Lars-Erik Johansson whose support made it possible in getting a 'flying' start with this research study.
I would also like to thank my friend Mikael Bodén for his help with the 'intelligent' parts.
Last but not least I would like to thank Susanne. my dear beloved wife, who has put up with me being absent-minded for long periods during this work.
The thesis is dedicated my grandmother Ina, who always supported my intentions to study.

Table of Contents

List of Figures

List of Tables

Chapter 1 Introduction

The objective of this research study is to determine if sensors can be adequately simulated in graphical off-line programming systems and therefore i) provide better off-line programming capability and ii) train intelligent robots in graphical off-line programming systems using simulated senors as input devices.

This thesis demonstrates how simulation of sensors can be incorporated within three-dimensional computer graphic systems for robot programming. The virtual sensors can be used to extend the capabilities of computer graphic systems to encompass both the programming of event-driven industrial robots and for the 'pre-emptive learning' of robots, which have in-built learning capabilities. A generic sensor model is presented that can be used to create virtual sensors of various types of device. Robot programming in general is described which includes the development of robot programming languages and programming methods, and in particular, off-line programming of robots using 3-D computer graphic tools.

Industrial robotics have made a great impact on the industrial manufacturing process. The hype generated when the first industrial robots were introduced in the early 60s was tremendous. Robots were supposedly able to accomplish highly repetitive tasks without getting bored, they would work in hazardous environments without any health risks and they would be capable of handling heavy objects for hours without getting tired. The robots were predicted to take over almost every manual role in factories and with further development would even undertake jobs in other areas such as power/nuclear industry, military and space operations etc. However, major limitations were soon recognised. These robots were not as capable and intelligent as people believed them to be. They could only perform the sequences they had been taught and they had no abilities to make any decisions of their own. Objects had to be in exact predefined locations as the robots had no ability to detect even the smallest change. "Intelligence" had to be implemented in some way and as a consequence a great deal of research was subsequently started in the area of robot programming. Programming methods have become more sophisticated with the availability of low cost computing platforms with significantly increased processing capability.

Robots manufactured today are fast, have good repeatability and are very robust, however, 'intelligence' of robots is still largely absent. Robots in the main still need highly structured and static environments and are not capable of learning to accommodate changes in the environment or to learn from mistakes. Robots need sensors to obtain information from and about their environments. The sensing capabilities present in robotics today are rudimentary compared to the human sensory capabilities. There are however certain application dependent solutions which appear to be intelligent, in that these systems are capable of following paths that are not predefined, for example seam following in arc-welding.

Robot specific programming issues include; (i) having to deal with a dynamic physical world, and (ii) having the need for easy interaction between operators and robots on the shop floor. Programming languages have evolved from teach-by-showing methods through to task-level programming methods. In teach-by-showing the robot sequence is programmed by physically moving the manipulator to work points and along program paths whilst recording the joint values for later play-back during program execution. The next step in this evolution and the dominate method in industry today are explicit robot programming languages (RPLs), in which the robot programs are textually defined. RPL programs consist of a combination of position definitions, path trajectories and logic statements. In this type of programming method there is no task knowledge built into the programs; these methods are considered as manipulator level languages. Program logic is often defined textually in RPLs and the actual teach-points are defined using the manipulator. The repeatability achieved can be good but the accuracy obtainable is in most cases inadequate. As such, critical workpoints have to be taught on-line using the manipulator. Task and object level programming methods are based on the concept that the programs describe the tasks to be performed and not the explicit robot motions. Knowledge of the tasks and objects are defined and stored in a database.

'Virtual Robotics', which encapsulates graphical off-line programming systems and Computer Aided Robotics (CAR), have emerged as productive and cost effective tools for the design of new robot manipulators, the design of new robot applications and off-line programming of industrial robots. Virtual robotics uses 3-D computer graphics to model the robots and their workcells. Robots and their motions are simulated using virtual robots having similar motion and kinematic properties as the real robots. There are normally libraries with robot models supplied with the virtual robotics tool. Models of objects, fixtures, machine-

tools etc. can normally either be imported from other CAD systems or modelled in the virtual robotics system. Virtual robotics gives the programmer an "image" of the task to program, and a visually represented world model which can be maintained and enhanced progressively unlike the situation when using RPLs where no visual representation is available.

Robots can be programmed off-line without any need to tie up the real robot and its workcell during the programming phase. Program locations are taught within the virtual world and as such are robot independent and can be used by different robot models for evaluation purposes. Program locations can be assigned to objects and move with them, which enables effective evaluation of different workcell layouts. Paths and sequences are debugged and evaluated using 3-D simulations where checks for collisions, cycle duration etc. can be undertaken. The paths and sequences are normally translated, via post-processors, to controller specific code and then down-loaded to the robot controller.

The virtual robots and their associated workcells will not precisely represent the actual kinematics and dimensions of the real workcell. The models have to be calibrated to accurately simulate the real world. Amendments are needed in the robot program after down-loading from the post-processor. These amendments may be for changes in the real workcell that occurred during installation, changes that occur over time, etc. Virtual robotic systems can be used as effective development tools, as a variety of approaches can easily be evaluated, for instance, investigations in dangerous and/or high cost experiments can be conducted without the access to the physical facilities and equipment. Virtual robotics are normally only used by industry to create robot paths and sequences for applications such as arc welding and painting. The inability of current virtual robotic systems to simulate sensors and sensory interaction places major constraints on the development of complete robot programs and the use of virtual robotics in many application areas.

The thesis addresses how virtual robotics can be used to represent the real world more accurately for use with sensor based robotic environments. This dictated a need for sensor simulation. In recognition of this, a generic sensor model is presented, which can be used to model a variety of sensor types, such as proximity and range measurement devices. The objective is to allow the creation of many types of sensor models using the same basic method which enables the characteristics of unique sensor types to be captured. The approach allows sensor

models to be integrated into the virtual robotics environment. To allow off-line programming and debugging of sensor interaction within a workcell the virtual sensor must provide representative characteristics of the real sensor. Using this generic model a number of non-contact sensor types have been created and simulated including proximity, photoelectric and ultrasonic sensors. These virtual sensors have been evaluated by direct comparison with the corresponding real sensors. A number of robotic test cells have been produced and corresponding virtual cells have been generated. Investigations have been conducted whereby both the real and virtual workcells have been directly compared whilst undertaking identical tasks.

The research study investigates how the generic sensor model can be used. For this study event-driven robotics and adaptive learning are two of the more important domains chosen. Event-driven robotics and adaptive robots both potentially allow more flexible robotic systems and production environments to be realised.

To demonstrate the role of simulated sensors in programming and debugging of robot programs having interaction with sensors, an event-driven robotic workcell has been simulated and off-line programmed. In the event-driven robotic workcell the robot's actions are controlled by sensory inputs. To enable off-line programming of robotic tasks which include a high proportion of logic, new robot programming procedures have been created. These new procedures and functions, which are based on Pascal syntax, can be used to drive both the simulated and the real workcell. *It is a central facet of this thesis that the virtual and real environments need to be programmed using the same programming language and underlying control architecture to gain the full benefits of the virtual robotics.* The virtual cells that are modelled and simulated in this research study are created with the virtual robotics tool CimStation.

If we have robots with learning capabilities this provides a major incentive to overcome the problems and deficiencies related to the uncertainties and differences between virtual and real workcells namely, to reduce the need for any workcell calibration. The robots should be able to learn adaptive behaviours within the virtual environment, giving them some priori-knowledge before exposure to the real world, namely, 'pre-emptive learning'. Sensors created with the generic sensor model can be used for learning and training of intelligent

robotic systems in virtual robotic environments. Two different robotic systems have been used to demonstrate the concept of 'pre-emptive learning'. Virtual models of these robotic systems have been created in a virtual robotics system. Artificial neural networks are used for the knowledge acquisition in the robot. These artificial neural networks are trained in the virtual world using virtual sensors for knowledge acquisition. The intelligence within the robot is transferred to the real robotic system.

In summary the thesis is structured as follows:

Chapter 2 discusses programming languages and methods applied in industrial robotics.

Chapter 3 describes graphical programming systems for off-line programming of robots and introduces general modelling and programming techniques.

Chapter 4 presents an overview of self learning robots and the use of artificial neural networks within robotics. Underlying principles of artificial neural networks are introduced and a summary of work on artificial neural networks and robotics is presented.

Chapter 5 describes the tools, equipment and methods used in conducting this research study.

Chapter 6 introduces a method for simulating sensors within graphical programming systems. Sensors of different types have been simulated and their behaviours have been compared to corresponding real sensory devices.

Chapter 7 describes simulation and off-line programming of event-driven robotics. Programming procedures and functions to be used for both a virtual and a real robot are described. An event-driven robotic workcell which has been off-line programmed is described. The robot actions are controlled by events which are detected by sensors. The virtual sensors used in the off-line debugging of the robot programs, were created using a generic sensor model.

Chapter 8 presents 'pre-emptive learning' of autonomous robots. A control architecture capable of learning how to react to different sensor information to accomplish a variety of behaviours is presented. The architecture has been implemented in two different robotic platforms. The controllers are trained, in virtual robotics environment using virtual sensors. The trained controllers are then transferred to the real robots.

Chapter 9 is a discussion of the main points of this thesis, and gives an indication of areas for further investigations.

Finally, chapter 10 presents the authors conclusions drawn from this thesis.

Chapter 2 Robot Programming Methods

This chapter aims to provide an overview of programming methods applied in industrial robotics. Levels of sophistication in different programming languages and methods for programming industrial robots are discussed.

2.1 Robot programming languages, levels and generations

In the definition of an industrial robot, it is stated that an industrial robot is a flexible and reprogrammable manipulator (International Federation of Robotics, IFR; International Organisation for Standardisation, ISO). There has been continuous development of (i) the methods to implement control software in robot systems, and (ii) of levels of sophistication in robot programming languages. There are several problems that make robot programming a difficult task. Programming of robots does not only encounter the problems of traditional computer programs but also the fact that the robot manipulator interacts with its physical environment [Goldman 1985]. This implies that irrespective of the language sophistication (see Figure 1), there is need for a model of this physical environment, for example in the form of locations of objects [Craig 1988].

A robot program is used for two purposes: (i) to define the task for the robot to solve and (ii) to control the robot manipulator as it performs the task. Programming of robots can be classified in two ways; namely (i) where does the programming take place, and (ii) how is the programming performed. The first normally divides into on-line and off-line program development with respect to the access of the robot system. The second is categorised by the level of sophistication of the programming language. The levels of sophistication in robot languages and robot intelligence is illustrated in figure 1[Engelberger 1989],[Nnaji 1993]. Robot programmers can be divided into two categories; (i) shop floor workers that make minor changes and smaller task programs. They do not usually have any formal programming education, and (ii) application and system programmers that create system solutions. These two categories make different demands upon the programming language and programming environment [Craig 1988], [Bergolte 1994], [McKerrow 1991]. The following section of the thesis discusses the development of programming languages and software development environments for industrial robots. It covers the early commercial systems through to some of the

research systems developed for automatic robot programming. There are numerous introductory text books in robotics describing the early evolution of robot programming for example [Craig 1989], [Engelberger 1989], [Bolmsjö 1989], [McKerrow 1991].

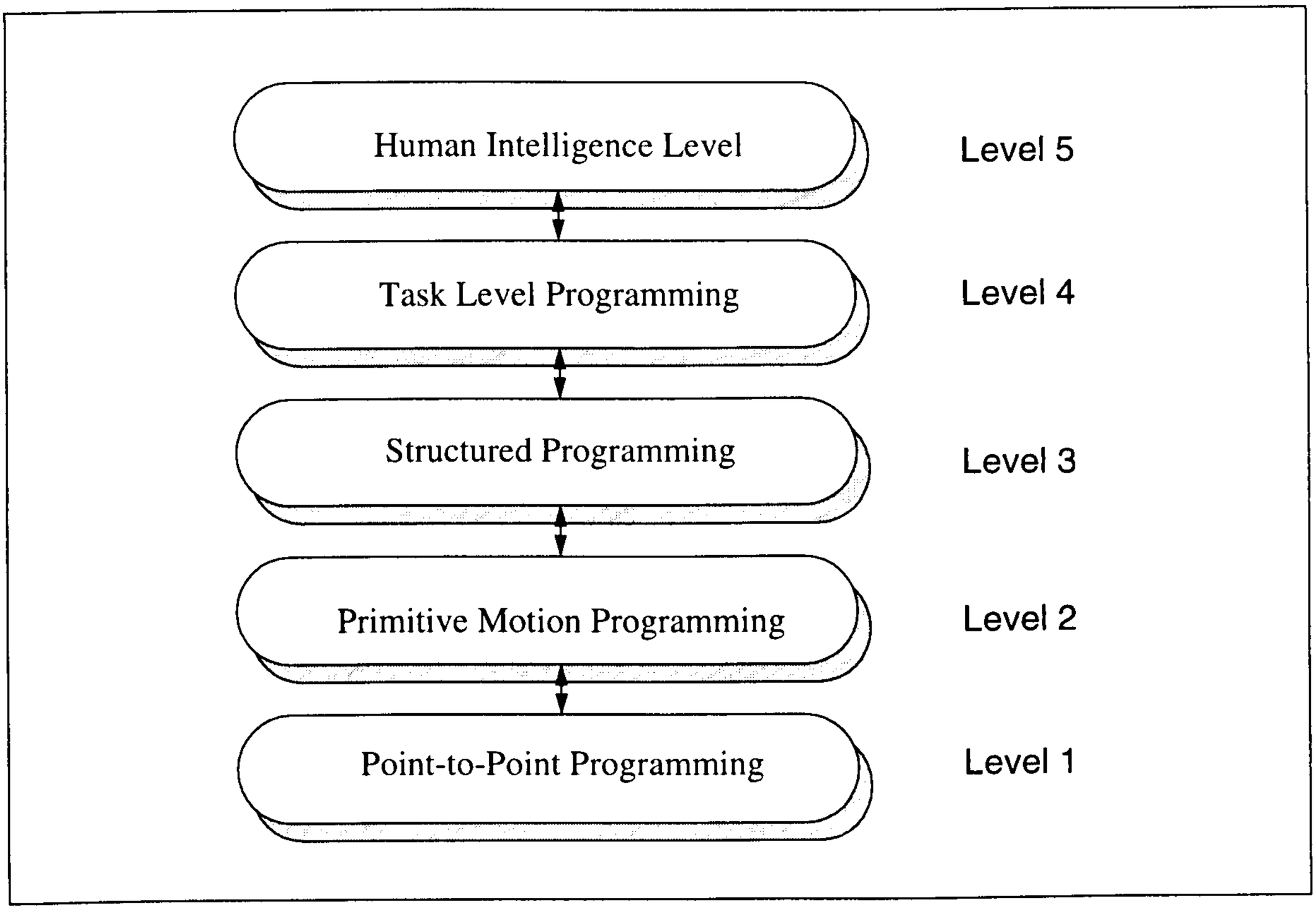


Figure 1: Levels of sophistication in robot languages

2.2 Teach-by-showing

Early industrial robots were programmed by leading the robot through a sequence of discrete workpoints, where the values of each joint were recorded. The recorded workpoints were later played-back for program execution. This method is known as teach-by-showing.
If there was a need to change a workpoint, the whole path needed to be retaught. Teach pendants were introduced to enable the robots to be controlled either by moving each joint explicitly to a desired position, or by moving the robot's end-effector in cartesian co-ordinates. It was also possible to add some simple program logic and to edit positions [Craig 1989]. Teach-by- showing restricts the use of robots to rather simple tasks such as spray painting or spot-welding [Elmaraghy and Rondeau 1991]. Programming robots with languages of this

type does not require an exact model of the robot or its environment, as the deviations, for example in the actual robot links, are already taken into account in the programming phase. The robot is effectively being used to digitise locations in space and the primary requirement of the robot's motion performance, is that of repeatability [Edkins and Smith 1985].

2.3 Explicit robot programming languages

Explicit robot programming is the act of specifying robot executable code in some robot-dependant textual language [Nnaji 1993]. Robot systems had to be able to solve more complex tasks than just pick-and-place, thus demanding that the robots programming language's being capable of maintaining an internal 'world' model. This led to the development of explicit robot programming languages. The robot programs had to be able to read and update sensor information, communicate with peripherals and include logic statements. Almost every robot manufacturer developed their own variant of a programming language [Bergolte 1994], together with languages developed at universities and research institutions, which has led to the existence of over 100 explicit robot programming languages [Nnaji 1993]. Some of the languages are extensions of traditional programming languages such as BASIC and PASCAL and others are completely new programming languages, for example AML from IBM. Common programming languages are: VAL I and II from Unimation Inc., AML (A Manufacturing Language) from IBM, ARLA (Asea Robot Language) and RAPID from ABB, KAREL from GMF. Most of the languages support both off-line and on-line programming (see section 2.3.1).

To simplify the programming of robots from different manufacturers there have been attempts to create a standard language. The first attempt was induced by NAM, German Committee for standardisation in mechanical engineering and was called IRDATA (Industrial Robot Data). The development was started in 1980 and was adopted as a German national standard (DIN 66313). The next step was the IRL (Industrial Robot Language) language whose first part was completed in 1992 and became a German national standard (DIN 66312) in 1993. The international standardisation organisation (ISO) has started work on ICR (Intermediate Code for Robots) and PLR (Programming Language for Robots). Bergolte describes the IRL language [Bergolte 1994]. IRL was designed as a general purpose language with specific enhancements for the programming of industrial robots. IRL has a similar

syntax to PASCAL and was designed to meet the demands of the two categories of programmers. A minimised version of IRL, for applications which need only limited functionality, called the Worker Subset, is provided for unskilled programmers.

2.3.1 On-line and off-line programming

Robot programming can be classified with reference to where the actual programming is performed. This approach divides robot programming into on-line and off-line programming.

On-line programming is usually performed via a teach pendant. The teach pendant allows the programmer to interact with the manipulator to develop the robot program. The program commands and logic instructions, for example, gripper, register and jump instructions, are generated through menus and the actual workpoints are taught by moving the robot to the desired position and recording it to a file.

Off-line programming takes place in a separate computer using a text editor. The program is compiled and then transferred to the robot controller. Workpoints can either (i) be typed in as cartesian coordinates (position and orientation) or (ii) be taught with the actual robot and stored in a file. The first approach relies on the accuracy of the robot. Accuracy is the precision with which a manipulator can attain a computed point [Craig 1989]. The second approach relies on the repeatability of the robot. The repeatability specifies how precisely a robot manipulator can return to a taught point. Accuracy is generally much worse then repeatability and varies from manipulator to manipulator [Craig 1989]. Tables 1 and 2 provide a comparison between the two programming methods.

Table 1: Off-line programming

Good features	Weaknesses
- Provides the programmer with the possibility to create well structured and well documented code. - The code can be optimized. - The robot and its peripherals can still be in production during the programming phase.	- The application engineer needs to be a skilled computer programmer. - Efficient off-line programming development needs an extensive world model. -Relies on the accuracy of the robot manipulator.

Table 2: On-line programming

Good features	Weaknesses
- On-line programming is easier for 'non-programmers', as it is more visual. The programming takes place in the actual task environment, which creates less demands for a world model. - There are no problems in translating geometry data from a world model to the real world. - The functionality of the program can easily be verified.	- The robot and its peripherals are out of production during the programming and debugging phase. -The shop-floor does not provide an efficient environment for program development. -Is difficult to create well structured and well documented code.

2.3.2 Error-recovery and event detection

As the complexity of robot workcells increases, the robot systems capability to detect errors and handle them intelligently becomes more important. This implies that the system requires: (i) the capability to interpret sensory data, and (ii) the robots should preferably be able to reason about the world state and how to recover from possible errors. Robots generally have quite limited sensing and reasoning capabilities, thus making error detection difficult. In order to detect an error, a robot program must contain some explicit tests.

Internal manipulator errors are normally monitored by the robot controller, where the error detection is built into the operating system. Internal sensors are monitored for missing signals for example torque sensors on the actuators are monitored for detecting overload and collisions and if something fails the robot action is normally stopped and an error is reported.

External errors such as missing or defective parts etc. must be controlled by the task program. All possible errors and situations can not be taken into account in the robot program. Most errors are normally avoided by building very robust and stable equipment and forcing parts to exact locations. This is both very expensive and can lead to inflexibility. Formal methods for verification of program flow have emerged, for example Petri-nets and state-graphs, thus enabling some level of evaluation of the process and thereby indicating some possible sources of error that may occur. These methods should allow the robot programmers to implement event detection and error-recovery procedures at these critical parts of the program sequence. Event detection and formal verification methods are discussed in section 7.4.

2.4 Task and object level programming

Task and object level programming languages are attempts to construct methods where the robots are programmed with a more 'human'-like instruction repertoire. In task level programming the program is constructed by sets of operations on objects or sets of tasks to be performed. This is akin to explicitly telling the robot what to do, instead of describing each working point. Table 3 gives examples of task and object level program statements.

Table 3: Example of program instructions in task level programming

```
Grab Bolt;
Insert Bolt in Housing;
Load Grinder;
etc.
```

The explicit robot motions and actions are calculated from these task commands. This requires a world model describing objects, robots, peripherals etc. The world model must contain information such as:

- Geometric descriptions of all objects and machines in the task environment.

- Physical descriptions of all objects, for example mass and inertia.
- Kinematic descriptions of all linkages.
- Output and accuracies of all sensory modalities.

World knowledge can be represented using CAD systems and the dynamic properties of the world can be dealt with using sensors [Nnaji 1993].

There are several programming systems reported in literature, for example RAPT, AUTOPASS and RALPH. AUTOPASS (designed by IBM) enables the programmer to use english-like commands, such as PLACE and INSERT. There is no intelligent reasoning system as such the user has to decide in which order operations take place [Lieberman and Wesley 1977]. RAPT is a language developed at the University of Edinburgh [Ambler and Popplestone 1983] and RAPT provides a more complete and explicit description of objects. Relations between objects are described. The user specifies actions to move the objects in order to obtain some desired state. RAPT was derived from the APT language for NC-machines. As a complete CAD model is not incorporated the system is not capable of collision detection, which makes program verification a difficult task. The RALPH language [Nnaji 1993] is built around a CAD modeller which makes reasoning about features possible. Task level programming systems are still very much in the domain of the laboratory research and there are very few commercial installations.

2.5 Graphical programming systems

Virtual robotic systems or off-line programming systems (OLPs) as they are referred to in older literature, are application programs running on graphic workstations for the interactive design, programming and simulation of automated manufacturing systems [Craig 1989]. They can be used to carry out design for manufacturability studies including conceptual design, visualisation, animation, evaluation and full facility emulation without tying up the physical manufacturing resources. Using accurate functional models of factory equipment, manufacturing engineers can simulate the operation of automation systems (including robots, conveyors etc.) and generate or verify actual task programs for the equipment in the workcell. Three-dimensional CAD models are used to represent the robots, parts, machinery and other

equipment. Simulation offers significant advantages over traditional hard prototyping in its ability to create, store and retrieve complete libraries of robots, peripheral devices, ancillary equipment, human models and entire workcell layouts, whilst safeguarding equipment [Bien 1992]. Virtual robotic systems are thoroughly discussed in section 3.0

2.6 Summary

Robot programming started with teach-by-show methods using the actual manipulator for describing the tasks. Programming was tedious and time consuming and there were limitations in programming logic statements. Explicit robot programming languages are the dominate programming method currently used by industry. Often a mix of off-line and on-line programming is used. The robot manipulator is normally used when defining work point locations. There are a great number of explicit robot programming languages. A standard language for robot programming would be beneficial to industry. Task and object level languages are methods which allow easier programming and should reduce the actual programming time. Task and object level languages are still very much at the research stage. Virtual robotic systems have emerged as development tools that can be used to reduce development lead-times and minimise the interruptions of the production process.

2.7 References

[Ambler and Popplestone 1983] Ambler A. and Popplestone R., *RAPT: A language for specifying robot manipulations*. Robotic Technology 1983 pp125-141.

[Bergolte 1994] Borgolte U., *The new German standard robot programming language: IRL.* Proceedings 25th International Symposium on Industrial Robots 1994.

[Bien 1992] Bien C., *Simulation a necessity in safety engineering*.Robotics World, December 1992, pp 22-26.

[Bolmsjö 1989] Bolmsjö G., *Industriell robotteknik*. Studentliteratur 1989, ISBN 91-44-28512-4.

[Craig 1988] Craig J.J., *Issues in the Design of Off-line Programming Systems* Robotics Research, MIT Press 1988 ISBN 0-262-02272-9.

[Craig 1989] Craig J.J., *Introduction to Robotics, Mechanics and Control*. Addison Wesley 1989, ISBN 0-201-09528-9.

[Edkins and Smith 1985] Edkins M. and Smith C.R.T. *The practical problems involved in off-line programming a robot from C.A.D. system.* Robots and automated manufacture 1995, ISBN 0-86341 053-7.

[Elmaraghy and Rondeau 1991] Elmaraghy A. and Rondeau J.M. *Automated planning and Programming environments for robots*. Robotics 1992 ,Vol. 10 pp 75-82.

[Engelberger 1989]. Engelberger J.F., *Robotics in service*. Kogan Page 1989, ISBN 1-85091-358-7.

[Goldman 1985] Goldman R., Design of Interactive Manipulator Programming Environment. UMI Researsch Press 1985, Ann Arbour, Michigan .

[Lieberman and Wesley 1977] Lieberman L. and Wesley M., *AUTOPASS: an automatic programming system for computer controlled mechanical assembly*. IBM Journal of Research and Development, 21, 1977.

[McKerrow 1991] McKerrow P.J., *Introduction to Robotics*. Addison-Wesley 1991, ISBN 0-201-18240-8.

[Nnaji 1993] Nnaji B.O., *Theory of Automatic Robot Assembly and Programming*. 1993; ISBN 0-412-39310-7.

Chapter 3 Graphical Programming Systems

Graphical off-line programming (OLP) systems have emerged to enable the development of robot programs without access to the robot itself (figure 2 illustrates a virtual robotic work-cell). An off-line programming (OLP) system can be seen as a robot programming language which has been sufficiently extended, generally by means of computer graphics [Craig 1989].

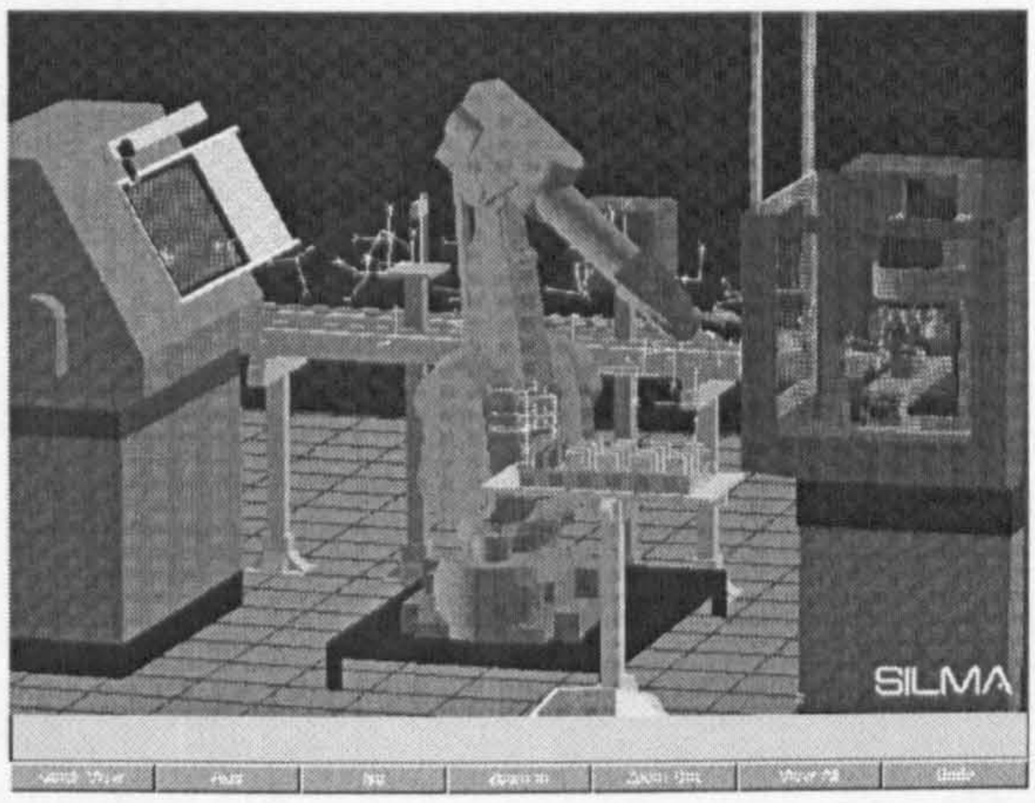


Figure 2: Virtual robotics workcell

3.1 Objectives in creating graphical programming environments

Graphical off-line programming systems (virtual robotics) provide a vital link between CAD/CAM systems and the equipment on the factory floor. Geometric data is derived from shape models of work pieces, tools etc. with support of CAD-functions [Edkins and Smith 1985], [Bernhardt et al. 1991]. Models can either be constructed in the CAD-system built into the virtual robotics system or imported from a proprietary CAD system via some standard interface, for example IGES or STEP [CimStation 1], [Mayr and Held 1989], [Tarnoff et al. 1992], [WorkSpace 1].

Virtual robotic systems should serve as the natural growth path from explicit programming systems to task level systems. Additionally virtual robotic systems provide an important foundation for research and development of task level systems [Craig 1988], [Dillman and

Huck 1986]. Off-line programming on a graphic workstation frees robots for productive use and greatly reduces disruptions to production schedules. Potential damage to equipment, tools and/or fixtures due to programming errors causing collisions or unwanted moves can be avoided. Virtual robotic systems help to shorten the design-to-manufacturing cycle by enabling users to identify and correct programming errors before they reach the factory floor. As one of the most time consuming elements of robot programming is debugging [Smith 1992], the use of simulation environments will reduce the time from idea to the generation of safe and executable code significantly. Juha Renfors et al. describe how they used off-line programming to implement a welding application at Bronto Skylift Ltd. which produced significant economic benefits [Renfors et al. 1993]. On-line programming time was 3.5 weeks in two shifts to accomplish 4.5 hours of welding time. The robot was out of production 3.5 weeks. Using off-line programming it took 56 hours of programming and modelling time in the virtual robotics system. The robot was out of production for a total of 12 hours for the purpose of program down-loading and final commissioning. Simulation makes it possible for engineers to: (i) evaluate workcells without any visual obstructions; (ii) select the most appropriate robots, tools and ancillary equipment; and (iii) determine the optimal layout for each component [Bien 1992]. By bringing various forms of information together at a single reference point, multiple levels of decision-makers, such as integrators, machine operators, maintenance engineers, human-factor engineers, design engineers and process engineers, can collectively participate in the entire workcell design process [Bien 1992]. Virtual robotics can be used as well in the design and development phase of new robot manipulators [Craig 1988].

The typical phases when working with virtual robotics/OLP systems can be seen in figure 3. When the preliminary layout is ready, the user teaches robot workpoints in the virtual environment. This can, depending on the system, be done in several ways. Workpoints can be taught by moving the robot with a simulated teach pendant, or by giving the exact cartesian coordinates. Some systems provide the user with a visual teaching coordinate frame to assist in the creation of coordinate frames that can be used as workpoints for the robot. The visual teaching coordinate frame can be moved in the environment by using a pointing device such as a mouse or a light pen. The taught coordinate frames can in some systems be attached to objects and then follow the objects as they are moved within the workcell. The taught frames are workpoints with which the robots should align its TCP (Tool Centre Point) during

program execution. Visual coordinate frames can be edited by means of translations and rotations. This aids the process of developing smooth paths, appropriate tool orientations etc.

The layout planning in general starts with the modelling of peripheral equipment, tools and objects to be manipulated. Some of these objects can hopefully be retrieved from an external CAD system. Common robots are usually available in libraries provided by the simulation software vendor. The next step is to design the layout. Components are placed in the virtual workcell, workpoints are checked for reach and different motion sequences can be investigated to obtain optimal solutions. There are systems that provide optimized robot placements [CimStation 1]. The placement optimization can be applied for reach, cycle time etc.

Having the appropriate workplace layout the robots and other programmable equipment are programmed. Programming languages and operating systems used will be discussed in detail in section 3.4. The programs are executed and a simulation is created. Using the graphical simulation and the animation sequence, the correctness of paths and possible collisions can be observed. When error free and optimized code has been developed the virtual workcell must be calibrated to replicate the dimensions and positions of the real factory set-up [Bernhardt et al. 1991], [Craig 1992]. The code generated is typically then translated through a post-processor and transferred to the robot controller for program execution.

3.2 Modelling and graphic techniques

A central feature of the user interface of virtual robotic systems is the computer graphics view of the robot and its environment. This requires the robot and all peripherals to be modelled as three dimensional objects [Craig 1988]. Not only are geometric definitions needed but also functional and kinematic definitions must be included in the model of objects and workcells. The modelling can be done either in a CAD module integrated into the virtual robotics system or in an external CAD system. Multiple representations of a spatial shape are generally required as different demands such as graphical speed, kinematic coupling, accurate surface representation etc. need different methods for optimal description and performance.

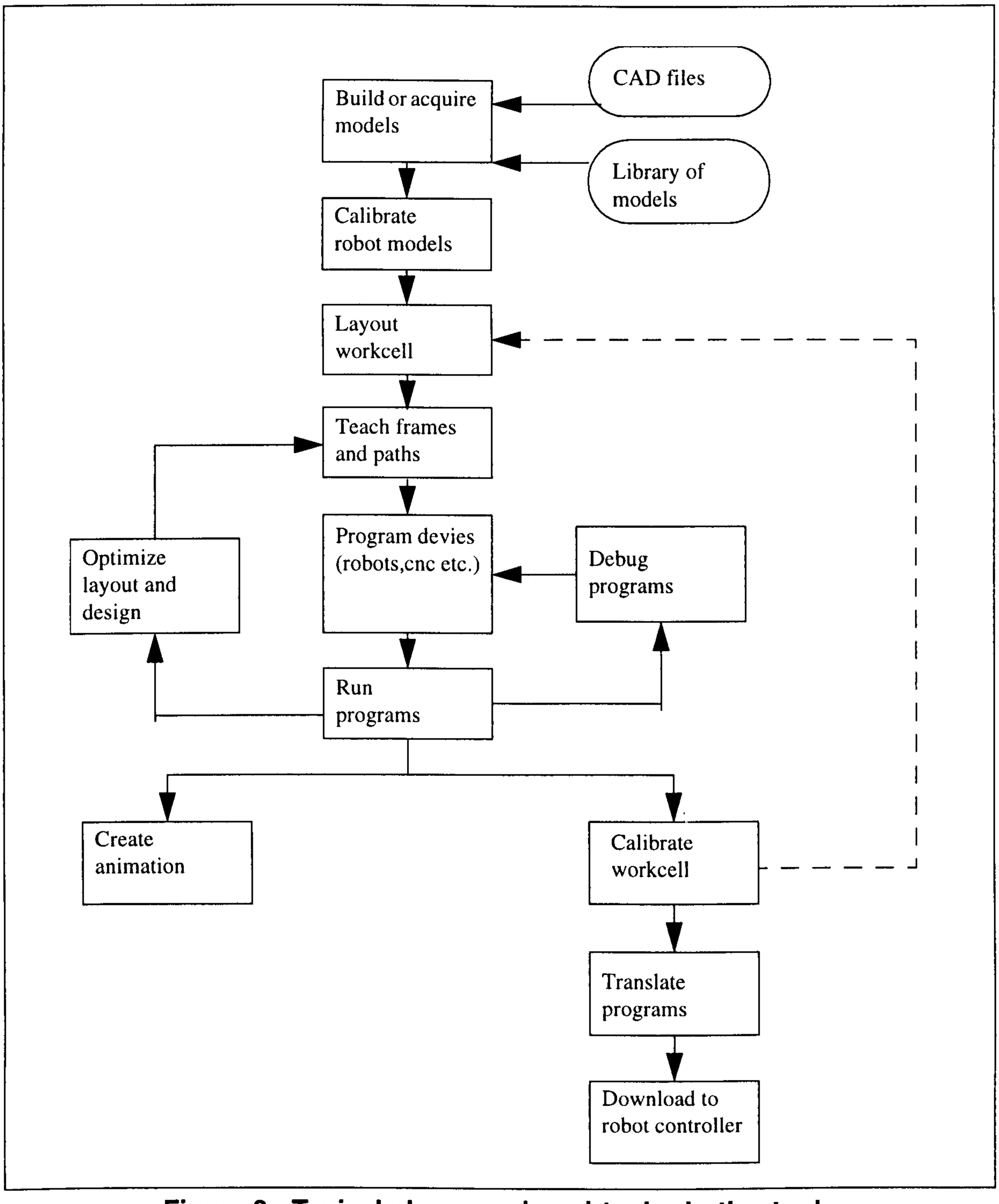


Figure 3: Typical phases using virtual robotics tools

The robots and their working environments must be considered as a unit consisting of active and passive objects. This requires a model representation of the objects that differs from conventional CAD models, as the model must contain more than just the geometry of an object. The CAD data structure must be extended with non-geometric data such as kinematic properties, functional relationships between objects etc. [Dillman and Huck 1986], [Dai 1989].

Theveneau and Pasquier describe a model representation that both contains a numerical representation for the display of the object and a symbolic representation to be used for reasoning [Theveneau and Pasquier 1988]. There are several CAD modelling techniques used in robot simulation reported in the literature. To enhance the systems animation performance two models can be used (i) an exact analytical description of a surface and (ii) an alternative method that creates a faster graphics, for example a faceted representation can be used for faster animations. The interaction should occur with the true representation even though the screen displays a simplified surface [Craig 1988]. Some simpler systems running on computers with limited graphic capabilities and the earlier systems reported use wireframe models, while other systems use shaded rendering techniques for displaying faceted surfaces or solid modelling techniques. The emerging product description standard STEP (STandard for the Exchange of Product Model Data) allows incorporation of manipulator information, such as kinematics, into the CAD data structure [Wapler and Neugenbauer 1994]. This should enable easy exchange of robot models between different virtual robotics platforms and CAD systems.

3.2.1 CAD modelling techniques

To allow realistic and thorough investigations, three-dimensional representations of the robots and their environments are required. There are several methods and algorithms for describing and displaying of three-dimensional geometry. Three-dimensional CAD objects can be modelled as wireframe, surface or solid models. These methods differ in construction, their properties, display speed, accuracy etc. Contemporary virtual robotic systems uses one or several of these methods.

Wireframe modelling is the simplest and fastest of these methods. Wireframe models only contain curve descriptions. 3-D objects modelled with wireframe techniques consist of curves that join 3-D points. Curves can be described with several mathematical methods, for example with the B-spline, Bezier and NURBS algorithms.

Bezier curves are constructed using control points and a control polynomial to approximate the curve. The grade level of the polynomial gives the amount of control points to be used. The use of Bezier curves gives fast calculations and is attractive for interactive work.

B- Spline curves are also constructed using control points and a control polynomial but they

can have an arbitrary number of control points as long as the number is at least one more than the grade level of the control polynomial. The control points can be given local weights where more exact approximations are needed.

NURBS -curves (Non-Uniform Rational B-Splines) do, in contrast to the other described representations allow non-uniform parameter intervals. This, as the control polynomial has more parameters, gives more degrees of freedom. For the mathematical descriptions of curve representations see any text in CAD mathematics for example [Bartels et. al. 1987], [Bezier 1986], [McMahon and Brown 1993] and [Rooney and Steadman 1993]. Wireframe descriptions do not provide sufficient information about objects for robot applications where the surface of an object is of importance, which is the case in for example spray painting and polishing.

Techniques based on surface models can be used to provide more thorough information about objects than is available when using wireframe techniques. Common methods for describing surfaces are the curve methods extended to accommodate surface descriptions [McMahon and Brown 1993],[Rooney and Steadman 1993]. *Bezier surfaces* as with Bezier curves are constructed with control points. The control polynomials used for the curves are substituted with characteristic polyhedrons. *B-spline surfaces* (as with B-Spline curves) give the possibility of local control of points, the control polynomials used for the B-Spline curves are substituted with characteristic polyhedrons to provide surface information. *NURBS-Surfaces* are extended in the same way and allow description of complex surfaces with one surface description where Bezier and B-spline representations would require multiple surfaces to approximate the desired surface.

The alternative modelling technique is based on solids. Solids can be constructed with Constructive Solid Geometry (CSG) or sweeping techniques and they are used for volume representation [Dai 1989], [Theveneau and Pasquier1988]. CSG takes primitive shapes and combine them as models using a class of set-operations. Primitive shapes such as spheres, cylinders cubes, prisms, cones etc. are used. The class of set-operations are normally: regularised set union ($\cup$); intersection ($\cap$); and difference(-). The model is represented as a binary tree in which the leaf nodes represent three dimensional basic components and the branch nodes represent the set operations.

Virtual robotic systems normally allow the user to switch between surface and wireframe display techniques, to speed up performance some systems automatically switch to wire-frame mode while changing the view of a model.

3.2.2 Collision detection

Collision detection is a powerful technique within virtual robotic tools. This is a feature used for debugging and evaluation of paths, design of fixtures and checks for reach. Collision detection is used to avoid failures when executing off-line generated robot code. Collision detection can also be used for design and evaluation of object and task level programming. This can be useful for example in automatic programming for robotic assembly. As controlled contact between objects is a desired state in assembly then collision detection has potential application in this domain.

Exact collision detection for general 3-D solids is quite a difficult problem, whereas collision detection for the same models in facetted form is somewhat more tractable [Stobart and Dailly 1985], [Goldenberger and McQuilian 1991], [Craig 1989]. To speed up collision detection the objects that are checked for intersection are bounded with boxes that surround the entire object. Intersections between these bounding volumes are then checked. If two or more bounding volumes intersect, the intersection between the exact volumes are checked and if there is an intersection between the true volumes the collision detection algorithm reports a collision. Some systems allow collision tolerances to be defined, this reduces the need for checking collisions between exact geometries and speeds up the process. The use of hardware graphics can speed up the collision detection as it provides built in functions for calculation of intersections between objects.

3.3 World modelling

Models for robots, fixtures, grippers, feeders, parts etc. are assembled into a model describing the environment and properties such as geometric relations between objects etc. The virtual robotic workcell or factory set-up is referred to as the 'world'.

The virtual 'world' is considered to be a model. The world model is a binary tree of objects with a 'null'-model as the root [Stobart and Dailly 1985], [Goldenberger and McQuilian 1991], [Craig 1989]. If a parent object is moved then child objects are moved with it. Objects can be affixed to other objects. Objects that are affixed to an object will move with the object to which it is affixed. Affixment can be permanent or temporal. Examples of permanent affixments/adoption are robot links that are affixed to each other in a chain, an example of a temporal affixment is when an object is gripped by a robot. Figure 4 shows a relation tree for a world model.

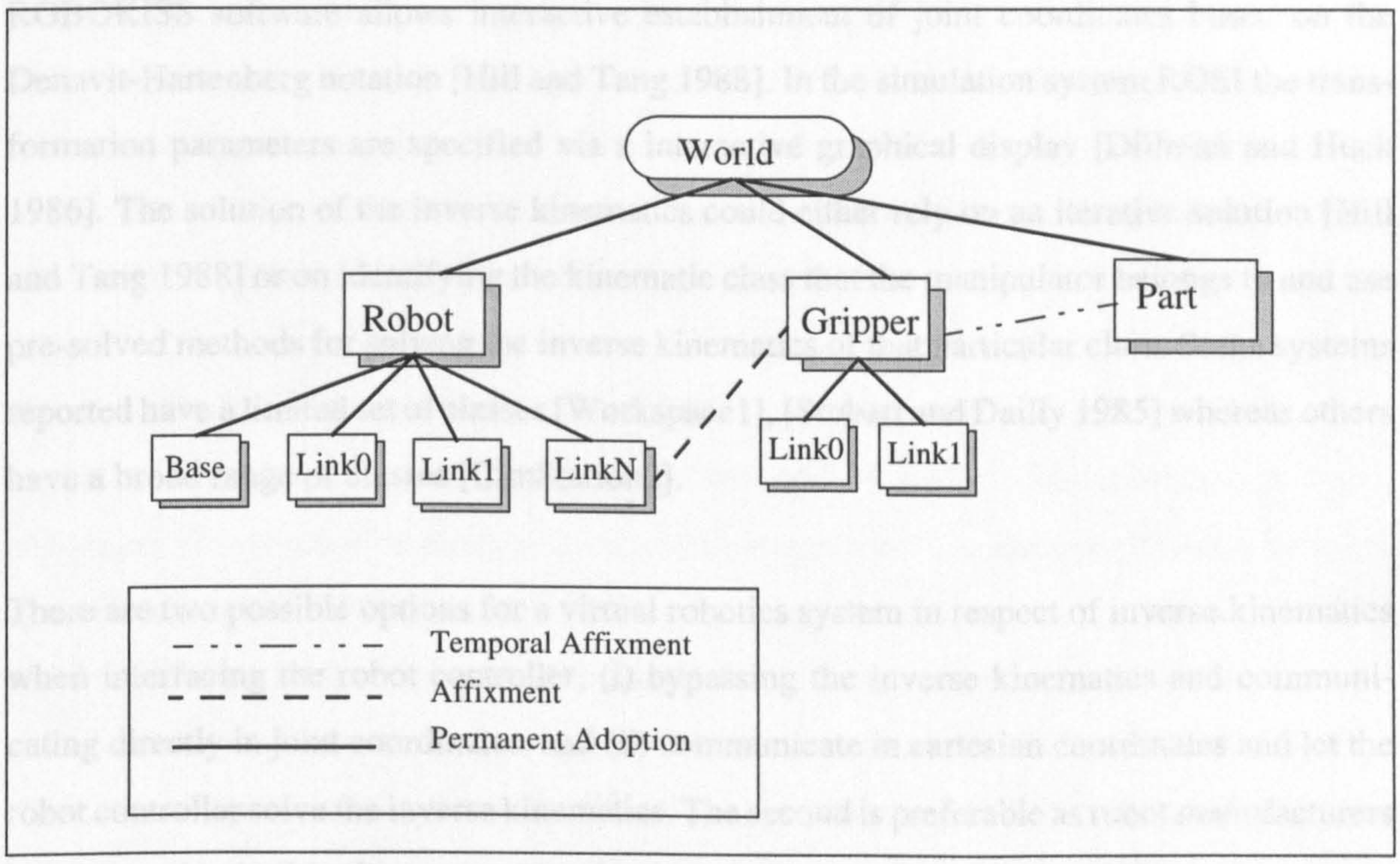


Figure 4: Relation tree of a world model

3.3.1 Robot modelling

To form a virtual robot the geometrical definitions (CAD geometry must be constructed into kinematic entities describing the relations between links, velocities, accelerations and other manipulator characteristics. A robot model is normally constructed using a dedicated CAD object class which combines both the geometric descriptions of robot links and the kinematic description. The robot model is formed into a binary tree of links. The joint coordinates are described and each joint is assigned kinematic properties such as joint limits, speed, acceleration, control algorithms etc.

Virtual robotics packages have different ways of preparing a model to be a robot object. CimStation [Cimstation2] requires the top node model to be a null model, whose name will constitute the robot name. For a robot with N-joints, the model must be comprised of N+1 subparts named LINK0, LINK1, LINK2,....,LINKN, where these objects must be nodes of the null model, as illustrated in figure 4. The coordinate frame of each link must have its Z-axis parallel to the joint axis and the origin must lie on the joint axis. Each link is then assigned limits and motion parameters, such as control planner, speed, acceleration, sampling rate under cartesian motion etc. The position and orientation of the coordinate-frames for links are crucial for the solution of forward and inverse kinematics of the manipulator. The ROBOKISS software allows interactive establishment of joint coordinates based on the Denavit-Hartenberg notation [Hill and Tang 1988]. In the simulation system ROSI the transformation parameters are specified via a interactive graphical display [Dillman and Huck 1986]. The solution of the inverse kinematics could either rely on an iterative solution [Hill and Tang 1988] or on identifying the kinematic class that the manipulator belongs to and use pre-solved methods for solving the inverse kinematics of that particular class. Some systems reported have a limited set of classes [Workspace1], [Stobart and Dailly 1985] whereas others have a broad range of classes [CimStation2].

There are two possible options for a virtual robotics system in respect of inverse kinematics when interfacing the robot controller, (i) bypassing the inverse kinematics and communicating directly in joint coordinates, and (ii) communicate in cartesian coordinates and let the robot controller solve the inverse kinematics. The second is preferable as robot manufacturers have started to build robot arm calibration into their controllers [Craig 1988].

To be able to simulate the robot and its actions several details of the inverse kinematic solution have to be emulated. The kinematic solutions used by the robot manufacturers are difficult to obtain and the kinematic solutions used for the virtual robots are normally estimations. What distinguishes a demonstration and animation system from a good planning and off-line programming system is how good the motion planning and monitoring algorithms of the manipulator are modelled. The system has to tell the user about constraints, such as singularities and how the robot will perform an orientation interpolation etc. [Angermüller et al. 1989].

Path planners and dynamic algorithms vary considerably from one robot manufacturer to another [Craig 1988]. These algorithms must normally be estimated by the designer of the virtual robot as they are often difficult to obtain from the robot manufacturers. The dynamics of robots is critical when optimizing paths and speed. Overloading a robot manipulator could result in disastrous consequences, as the robot might take an incorrect trajectory.

Research on how to simulate manipulator dynamics in graphical environments has been conducted by several research groups [Lee at al.1995], [Ravani 1988], [Bullinger et al. 1989], [Zomaya 1992], [Wloka 1989]. Most of this research has focused on how to accurately simulate 'real' dynamics. This is useful when designing new manipulators and new manipulator controllers, or when the actual dynamic solutions are available. Virtual robotic systems normally provide some dynamic simulation capability. CimStation [CimStation 3] allows the user to enter dynamic attributes such as the location of the centre of mass of the link; an inertia matrix for the link; total mass of the link; viscose friction coefficients for the proximal joint and the coulomb friction level for the proximal joint.

The Realistic Robot Simulation initiative (RRS), which includes both robot manufacturers, virtual robotics vendors and robot users, has come up with a Black-Box specification which will allow virtual robotics vendors to use manipulator specific solutions provided by robot manufacturers. The robot manufacturer does not have to reveal any company secrets such as algorithms etc. with this approach [Bernhardt 1994]. Exact inverse kinematic solutions and manipulator specific dynamic algorithms can be included in the RRS Black-Box. RRS specifies the interfaces in and out from the Black-Box enabling the virtual robots to be controlled with the same algorithms as the real robots. This ensures that the robot models created are accurate and any evaluations of a particular application undertaken in the virtual environment will be accurate with respect to chosen work paths, estimated cycle times, etc. Virtual robotics tool vendors may in the future perhaps even receive the exact geometric dimensions, in the form of 3-D CAD drawings, together with the RRS definitions. This will allow the virtual robots to accurately emulate their corresponding real robots.

3.4 Operating systems and programming languages

Virtual robotics tools are implemented on a variety of computer operating systems. The virtual robotics system often incorporates a special design programming language for pro-

gramming of robots and simulations.

As robot workcells often contain individually controlled devices such as feeders, material handling systems, co-operating robots etc., the virtual robotics system should allow the simulation of concurrent activities. As a basis for this the underlying environment in which the system is implemented need to be a multi-tasking system [Craig 1988]. Some systems, such as CimStation, IGRIP, ROBCAD and GRASP, are based on the UNIX operating system, which is a true multi-tasking operating system.While other systems, such as WorkSpace, AutoMatos and Moses, are based on DOS where the multi-processing has to be simulated.

Some systems use a universal language for programming both the robots and the simulation. These languages are normally specific to the virtual robotics tool. Programs written in a universal language are then translated to the native language of the robot controller. The translation is made by post-processors, which normally use "dictionaries" for command translation. There is a need for a post-processor for every type of robot controller, and these post-processors must normally be updated when the controller software is updated. The use of a universal language allows greater flexibility as the same code (robot program) can be used to evaluate the performance of different robots to accomplish a specific task. The translation does, as all language translations, create problems and is an important issue of research [Craig1988].

Other systems use the native language of the real robots to program the virtual robots. There is a need for simulation and animation commands, for example to allow virtual objects to move with the robot when they have been gripped. These commands are system specific commands and must be added to the robot program to enable proper simulation. The simulation specific commands are normally treated as comments when down-loading to the controller.The use of the native language eliminates the problems of translation from one language to another, but restricts the flexibility of the system.

Systems that provide a powerful general language can serve as research platforms for the future research of task level programming [Craig1988] [Dillman and Huck 1986].

3.5 Application specific modules in virtual robotic tools

Some virtual robotic systems provide application modules to simplify program generation in certain task domains, such as spot welding, painting, arc welding and coordinate measurement machines. These application modules contain expert knowledge for the specific application. The user is guided and helped in choosing application parameters. Some application modules can use geometrical entities for automatic generation of robot paths. These systems normally ask the user to specify task parameters, for example in arc welding, the current, desired thickness of seams etc. The shape of a CAD object can then be used to automatically generate welding paths. The control points on the generated path can then be checked by moving the virtual robot with the welding gun to the control points. Positions on a path can if necessary be re-oriented, and extra control points can be added. The path can in some systems be attached to the object and will then move around with it.

In spray painting applications parameters such as, paint viscosity, nozzle diameter, air-pressure, desired paint thickness etc. are defined by the user. The user indicates which surface that is to be coated with paint, by graphically picking it, and the system will then automatically generate the paths needed for obtaining the desired paint coverage and thickness. The surface is in some systems visually "covered" with paint during the simulation, allowing the user to check the thickness at certain positions on the object. Speed and other parameters of the paint path can then be changed to obtain the desired result.

Using application specific modules, such as arc-welding and painting, allows the user to access expert knowledge in several domains. It saves time as it can be very time consuming to generate and debug long motion sequences. It allows different layouts to be evaluated for cycle time etc. without any re-programming and re-definition of work points.

3.6 Calibration of virtual robots and workcells

Although virtual robotics can offer significant advantages as described above, the growth in industrial users of such systems has been relatively slow. This is partly due to the fact that, programs developed in a virtual robotics system can not reliably represent the real world without the use of calibration procedures. This is due to differences between the CAD-models

in the simulated workcell and the real world. The simulated robot workcell does not represent precisely the actual robot kinematics and workcell geometry. Deviations due to mechanical and manufacturing deficiencies in the robot and the workplace and manufacturing process variability are unlikely to be accounted for in the simulated system [Craig 1992]. To overcome such problems a more accurate representation of the robot and its immediate environment is required within the workcell simulation system. The robot model differences can be accommodated by correctly adjusting the parametric representation of the robot kinematic model determined through direct calibration of the robot.

In [Shröer and Bernhardt 1992] the authors describe a calibration procedure that is based on a robot model which includes significant deterministic sources of pose deviations, within which the geometric-kinematic model of robot motions are extended to include the effects of elastic deformations and gear parameters. For calibration purposes, the robot is considered as a stationary system in which the input values are the joint encoder values and the output values are the position and orientation of the TCP. The procedure allows the determination of geometric parameters of the rigid body system such as zero position errors of the joints; link lengths; joint axis misalignments; transmission and coupling factors of the gears; gear and link elasticity as well as gear eccentricity. The known data of the robot, such as number of joints; the joint workspace; the matrix of transmission and coupling coefficients; masses and centre of gravity of the links etc. are entered into the system. The kinematic configuration (the chain of rotational and translational joints) can be entered as a simple polygon model. Using this information the system calculates which robot poses are needed to allow the determination of the deviations. The number of poses can be between 40 and 200. The actual poses are measured with a non-contact theodolite system. The actual poses are than compared with the calculated TCP poses and the deviations are calculated. The result is a set of parameters for each individual robot.

In [Owens 1994] the calibration system RoboTrack is described. RoboTrack uses three cables mounted on encoders. The cables are mounted to the TCP of the robot and the encoders are mounted in a triangle, within the robot workcell. The virtual robotics system WorkSpace has a direct interface to the RoboTrack system. The robot is moved to 25 different poses throughout its working range. The robot poses are recorded using the teach-pendant of the robot and WorkSpace records an x,y,z measurement of the TCPs position. The software than

compares the file created with the teach-pendant, with the data recorded by the virtual robotics system. The deviations calculated in this comparison are used to update the simulated robot model. Errors which may be detected with the RoboTrack and WorkSpace system include: the zero position of each joint; the length of each link and the distance offset at each link; the compliance of each joint; and the x,y,z tool offset from the mounting plate to the TCP.

The calibration package provided for CimStation can adjust parameters such as tool offset; joint zero location; actuator parameters; link lengths and offsets. The package can use several kinds of input data to serve different kinds of measuring systems [Craig 1992]. The simplest way of calibrating is to have a calibration tool with known properties mounted on the robot. Move the tool to a calibration point, record the joint values and then move to the same point with a different arm configuration and for each configuration store the joint values. At least three configurations for each calibration point are needed. The calibration point should be placed throughout the working range of the robot. The positions used in the off-line generated program are then updated using these calibration parameters.

However, there are not only differences in the representation of the robots, there are also differences between simulated objects and their location and the actual objects and their real location in the workcell. There may also be differences in the shape of objects in virtual and real world. In order to overcome these deficiencies the location and orientation of the workpieces must be determined and the trajectory of the robot modified before the manufacturing process begins. This is called workcell calibration. Some applications such as arc-welding involve many robot workpoints relative to a rigid object fastened in a fixture. The actual position and orientation of the fixture can be measured by moving the robot to three locations on the fixture and then recording the robot position. These data can then be used to update all robot poses of for example the welding path. This method can be used if the demands for the accuracy is limited. If high accuracy is needed a measurement system of the kind used for robot calibration is needed [Duffau and Kehoe 1991]. The RoboTrak system described earlier can be used to measure the position of parts in the workcell. One way to perform workcell calibration is to use sensors, mounted on the robot arm or within the workcell, to locate and predetermine critical features of the workpieces in 3-D [Kehoe, et al. 1991].

Workcell differences and uncertainties are more difficult to deal with as they are introduced both during the design and production phases. The system should be able to adapt to changes as they appear, therefore having to constantly monitor the environment to detect changes and to adjust to them.

3.7 Sensor simulation

Contemporary robot systems are not good at handling uncertainties in their environment and as such this problem has provided the basis for much robotics research. Robot programs rely on exactly defined conditions. Sensors can be used to guide the robot through paths enabling robots to adjust to changes this is for example done in commercial seam tracking systems for arc welding. There are several research groups [Kugelmann et al. 1994], [Tarnoff et al. 1992], [Wahrburg and Papperitz 1995], [Hamura and Kataoka 1995] that have investigated having sensors mounted to the robots for real time adaption to object positions etc.

A substantial part of an industrial robot application program is dedicated to checking for errors and to preventing failure. A typical approach is to use sensors to check for errors and pre-programmed error recovery procedures [Smith and Gini 1992] for example sensors monitor feeders to detect parts in place and if a signal is missing the robot moves to a waiting positions etc. It is difficult to predict all possible errors thus hard to create the appropriate error recovery mechanisms [Smith and Gini 1992]. These aspects must also be considered when robot programs are created off-line using a virtual robotic tool. If complete programs are to be created and simulated off-line, not just motion sequences for arc-welding etc., these events must also be possible to simulate. These facts dictate that *it is necessary to enable simulation of sensors in robot software development tools*, such as virtual robotic systems, to allow the programmers to address the issues related to sensor interaction at the planning and development stage of a system, otherwise will much of the robot program be left for on-line evaluation.

Important research issues related to virtual robotics are simulation of sensor based robots and static and dynamic process simulation [Nitzan 1990]. Sensors must be used to compensate for errors in the workstation world model and inaccuracies associated with position control

of robots. An effective off-line programming environment must be capable of generating sensor programs based on environmental models [Tarnoff et al. 1992]. Thus, creating more realistic simulation and animation software for sensor-based robots has emerged as one of the important research issues in the domain of robotics and automation [Chen et al. 1991].

There are a number of research groups working in the domain of sensor simulation in graphical robot simulators. In [Bruyninckx et al. 1992] the authors explain how a three dimensional simulator was used where the operator, via a contact force simulator can, interactively analyse and program compliant motions. Tarnoff et al. have a force/position model incorporated into their virtual robotics environment [Tarnoff et al. 1992]. Chen et al. simulate proximity, laser and vision in their simulation environment [Chen et al. 1994]. A ray-tracing algorithm was used for the simulation of sensors and binary values used for representing the output from their proximity sensors and eight bit values for representing the distance from the laser sensor. The main conclusion was that a simulator in sensor-driven robotic systems must incorporate simulation of sensory information feedback. Wybrow and Wykes have shown that it is possible to extract information from CAD-models to train an ultrasonic object recognition system [Wybrow and Wykes 1992] . A simplified transducer response cone was modelled from measured data and used to simulate how the waves from the transducer propagate. In [Crane 1992] the author describes how a Z-buffer technique was used to simulate laser and ultrasonic sensors. The detection range was modelled as a volume, and for the laser sensor simulation the geometry of a box was used for simulating the detection range. For the ultrasonic sensor a perspective viewing volume was used to simulate the conical nature of the ultrasonic detection range. The volume was checked for intersections with other objects. If any intersection occured, the data buffer was traced to get the distance to the object. The author used five sensors in the simulations and there was no apparent reduction in the image generation rate which implies that the use of appropriate graphic hardware will make sensor simulation possible without adversely slowing the simulation speed.

In [Freund et al. 1994] the authors describe the simulation system COSIMIR which provides some facilities for sensor simulation. Information about the material properties was incorporated into the CAD models, making it possible for the simulation system to determine if the object is detectable by a certain type of sensor. Sensors were simulated with ray-tracing algorithms and checks for interference between the sensor ray and obstacles.

Kugelmann et al. uses a 3-D simulation and planning system together with a robot manipulator mounted on an AGV and with a vision system mounted to the manipulator [Kugelmann et al. 1994]. The sensory system is used to provide information of the present state, the planning system then investigates future possible states by simulating the robotic system and its sensor. The real sensory system provides the simulation and planning system with data about objects positions in real time. They simulate the vision system in the 3-D simulation environment and the simulated vision system is used for supporting the planning algorithms. When the task has been planned, a program is created and down-loaded to the real robot controller for execution. The virtual camera simulates the camera after edge detection, which means that light problems etc. are not taken into account. The geometrical descriptions used to represent objects is crude, which may effect the realism of the detection.

Hirtzinger et al. describe a "sensor based teaching-by-showing" approach for tele-manipulators [Hirtzinger et al. 1994]. The manipulator is graphically guided through the task in graphically simulated situations, the relevant nominal situations are stored for later recall and reference during real-time execution. It is suggested that instead of calibrating the robot, tele-sensor-programming can provide the real robot with simulated sensory data that refers to relative positions between the gripper and environment, thus compensating for any kind of inaccuracies in the relative positions of the robot and real world.

3.8 Virtual reality and robotics

The use of virtual reality tools (VR) for robotics is an emerging technology and a growing research area. Virtual reality is mostly used in the robotics field in conjunction with teleoperation and in some cases workcell design. This section briefly describes virtual reality and some of the research in the domain of virtual reality and robotics.

Virtual reality allows the user to participate in a synthetic environment and mediates a 3-D "feeling". Conventional computer graphic systems put the user as an external observer, looking at the synthetic environment.

The following definition of virtual reality can be found in [Gigante 1993].

Definition of virtual reality:

The illusion of participation in a synthetic environment rather than external observation of such an environment.VR relies on three-dimensional(3-D), stereoscopic, headtracked displays, hand/ body tracking and binaural sound.VR is an immersive, multi-sensory experience.

Tele-presence is one of the main research areas in the VR community. Examples of Tele-presence environments include the tele-operation of robots in hazardous or remote environments [Takahasi and Sakai 1991], [Piguet et al. 1995], [Hirzinger et al. 1994], [Pook and Ballard 1995]. Here the visual system is coupled to remote cameras that track head and eye movements so that the environment can be observed as being at the remote location. Hand and body motions can be coupled to a manipulator in the same environment to enable the robot to move with the operator's movements.

Virtual reality is reliant on the following technologies [Gigante 1993]:

1. Real-time 3-D computer graphics.
2. Wide-angle stereoscopic displays.
3. Viewer(head) tracking.
4. Hand and gesture tracking.

A VR system typically consists of: a 3-D computer graphic workstation to generate the synthetic scene; a head mounted display(HMD) that provides stereo imaging and head tracking; and motion input from data gloves that register both arm and finger motions.

Technical demands on VR are [Gigante 1993]:

- rapid update rates, preferably at least 30 frames per second.
- short lag times between input from input devices to actuation in the virtual scene.
- secondary visual cues like shadows and textures.
- motion and force feedback.

As feedback needs to be immediate, time is the biggest constraint on VR [Gigante 1993]. This constraint reduces the level of 'image/ picture like' quality that can be presented. High resolution rendered images that can be seen on high performance graphic computers can not be viewed with proprietary VR tools.

In [Milgram et al. 1995] a VR system is described for remote control of robot manipulators. The system is designed for environments that are only partly modelled, this means there are parts of the world that are unknown and unstructured. The scene of the real environment is provided by stereo video cameras. A computer is used to generate stereo graphics of the real scene. The operator can add virtual 3-D graphical objects to the scene. Virtual workpoints and paths can be added to the scene and they can use a virtual robot manipulator. This approach allows the operator to perform and evaluate robot manipulations with virtual objects and robots in a real scene. The demand for a detailed model of the environment is lowered as the visual background coincides with the actual remote worksite. The system is not intended to be used in structured environments with repetitive sequences such as factories and it does not provide off-line programming. "The intelligence in our approach remains with the human component of the system, we endeavour to provide her with the tools for carrying out required data acquisition operations" [Milgram et al. 1995].

The virtual reality workstation VR4RobotS developed at IPA, Germany permits three application areas for industrial robotics: i) simulation for industrial application planning; ii) off-line programming and teaching; and iii) teleoperation of robots [Flaig et al. 1994]. The system uses a 6-D trackball, a dataglove and a head mounted display for interaction with the VR system. To supply the operator with the necessary image update speed, a specialised computer was developed consisting of a parallel transputer network which enables robot motion to be calculated independently from the graphics tasks. The off-line programming part of VR4RobotS is a conventional off-line programming system with the distinction that the operator "touches" virtual menus through the dataglove. Hand movements done with the dataglove can be tracked and transferred to robot paths. This approach needs data reduction algorithms to produce a path which contains a suitable amount of via points. It seems that the benefits of using this system instead of conventional 3-D virtual robotics system is that it allows a better evaluation from a geometrical point of view.

Takahasi and Sakai in [Takahasi and Sakai 1991] propose a robot teaching method in which the operator, using a dataglove, demonstrates the task in a virtual environment. The robot is taught by using a kind of "hand sign language". The dataglove can separate 30 different hand movements. The system uses the colour attribute of CAD-models to distinguish between objects. The real world is then observed by a colour CCD camera and objects are distinguished by means of colour. The authors approach and objective was to be able to teach

robots without any programming experience and that humans normally are taught from teach-by-showing. Their approach is interesting from a virtual reality perspective but it does not match the trend of task and object level programming, and it also ignores the fact that humans are trained mainly from written instructions.

VEVI (Virtual Environment Vehicle Interface) developed by NASA is a virtual reality system for direct tele-operation and supervisory control of robotic vehicles and manipulators [Piguet et al. 1995]. VEVI is developed to run on high performance computer systems to allow operation of aerospace and sub-sea robots. The system is designed to allow multiple users to access the same remote environment through separate virtual reality interfaces networked together. It is pointed out [Piguet et al. 1995] that the frame rate is crucial for a realistic interaction with virtual environments. The frame rate should at least be 10 frames per second. VEVI allows incorporation of data sampled from real sensors such as visual, tactile and proximity to be incorporated into the virtual environment. The system is not intended to be a programming tool for industrial robots, although it does contain data types for creation of virtual robot manipulators.

Pook and Ballard reports research where an industrial robot is used for tele-manipulation [Pook and Ballard 1995]. The robot is interfaced and controlled with VR tools such as a virtual reality stereoscopic helmet and a dextrous hand master. The robot is directed via hand signs made by the operator. Robot actions that needs servo control such as motions are pre-programmed. The system does not allow creation of new robot programmes.

3.9 Summary

Virtual robotics can enhance productivity both when designing new robot workcells and when re-programming existing robotic installations. The technology is used by industry mostly for design of workcells and for programming of complex robot paths. There are problems related to inaccuracies between the virtual and real robots and workcells which dictate a need for calibration. Programs are normally written in a language specific to the virtual robotic system and then translated through post-processors to a robot controller specific language, which creates serious limitations as to the capability to off-line generate and debug program logic. When a controller specific language is used in the virtual robotic systems it puts constraints on the productivity when designing new installations. Some virtual robotic systems reported in the literature are summarised in appendix A.

A significant part of robot programs in industry, deals with sensory interaction and interpretation. Contemporary virtual robotic systems do not allow simulation of sensors, which put constraints on the type of application they are able to off-line program.

Most of the research in the domain of virtual reality and robotics is done in the field of tele-operations, with the exception of the work at IPA, Germany [Flaig et al.1994]. Very little work has been done on program creation for industrial robots with VR tools. The use of VR equipment such as head mounted displays together with virtual robotic systems, could be used to allow better visualisation and evaluation of layouts and program sequences. One of the common conclusions reported is that the frame rate is crucial for a satisfactory interaction with the virtual environment. This currently imposes a reduction on the resolution of the 3-D world and models and could significantly reduce the benefits gained from using high resolution graphics in virtual robotics tools.

Intelligent robots with sensing capabilities should reduce the need for workcell calibration, as they would be capable of detecting changes in the environment. If virtual sensors are used in virtual robotic systems intelligent robots could be trained, using the virtual sensors, to detect differences in the environment. Initially these robots could be trained in the virtual environment and then exposed to the real environment where adaption would continue.

3.10 References

[Angermüller et al. 1989] Angermüller G., Niedermayr E. and Roth N., *Off-line Programming and Simulation of Flexible assembly*. Assembly Automation 1989, Vol. 9 pp 97-102.

[Bartels et al. 1987] Bartels R.H, Beatty J.C. and Barsky B.A., *An introduction, Splines for use in computer graphics & geometric modeling.* Morgan Kaufman publishing 1987, ISBN 0-934613-27-3.

[Bernhardt 1994] Bernhardt R., IPK Berlin. *Realistic Robot Simulation (RRS) Initiative.* Technical Description 1994.

[Bernhardt et al. 1991] Bernhardt R., et al., *Execution of Off-line generated and simulated application programs*. Mechatronics & Robotics, 1 1991 IOS press.

[Bezier 1986] Bezier P., *The mathematical basis of the UNISURF CAD system.* Butterworths & Co. Ltd. 1986, ISBN 0-408-22175-5.

[Bien 1992] Bien C., *Simulation a necessity in safety engineering*. Robotics World, December 1992, pp 22-26.

[Bruyninckx et al. 1992] Bruyninckx H., et al., *A CAD-based contact force simulator as a learning tool for compliant motions*. Intelligent control IEEE conference 1992, pp287-292.

[Bullinger et al. 1989] Bullinger H.J., Menges R. and Warschat J., *GROSS- Graphic Robot Simulation System.* CAD/CAM, Robotics and Factories of the future, Vol. III, Springer Verlag 1989 ISBN 0-387-51134-0.

[Chen et al. 1991] Chen C., Trivedi M.M.,Bidlack C. and Lassiter N., *An Environment for Simulation and Animation of Sensor-based Robots*. Application of Artificial Intelligence 1X (1991) vol. 1468 pp 354-366.

[Chen et al .1994] Chen C.,Trivedi M.M. and Bidlack C., *Simulation and Animation of Sensor-Driven Robots*. IEEE transaction on robotics and automation, Vol. 10, No. 5, 1994.

[CimStation 1] *CimStation Users Manual.* Silma Inc., Cupertino, CA.

[CimStation 2] *Meta-Kinematics Users Manual.* Silma Inc., Cupertino, CA.

[CimStation 3] *Dynamics User's Manual.* Silma Inc. 1995, Cupertino, CA.

[Craig 1988] Craig J.J., *Issues in the Design of Off-line Programming Systems.* Robotics research, MIT Press 1988, ISBN 0-262-02272-9.

[Craig 1989] Craig J.J., *Introduction to Robotics, Mechanics and Control.* Addison Wesley, 1989, ISBN 0-201-09528-9.

[Craig 1992] Craig J.J., *Robot Calibration Facilitates Off-line Programming*. Robotics World, March 1992,Vol 10, part 1, pp 24-25.

[Crane 1992] Crane III, C.D., *A computer graphics based approach to range sensor simulation*. Application of Artificial Intelligence X, machine vision and Robotics, 1992, pp 306-312.

[Dai 1989] Dai F., *Modelling of robot workcells for a programming and simulation system*. System modelling and simulation, Elsevier 1989.

[Dillman and Huck 1986] Dillman R. and Huck M., *A Software System for the simulation of Robot Based Manufacturing Processes*. Robotics (2) 1986, pp 3-18.

[Duffau and Kehoe 1991] Duffau B. and Kehoe A. 1991. *Overcoming Robot Workcell Uncertainties*. An Overview of ESPRIT Project CIM-SEARCH 5272.

[Edkins and Smith 1985] Edkins M. and Smith C.R.T., *The practical problems involved in off-line programming a robot from C.A.D. system.* Robots and automated manufacture 1995, ISBN 0-86341 053-7.

[Flaig et al. 1994] Flaig T. , Neugebaur J-G. and Wapler M., *Virtual Reality for Robot Simulation and Off-line Programming*. Fraunhofer-Institute for Manufacturing Engineering and Automation (IPA) Stuttgart, Germany 1994.

[Freund et al. 1994] Freund E. , Rossman J., Uthoff J. van der Valk U., *Towards Realistic Simulation of Robotic Workcells*. 1994 International Conf. on Intelligent Robots and Systems. ISBN 0-7803-1934-6.

[Gigante 1993] Gigante M. A., *Virtual Reality: Definitions History and Applications*. Virtual Reality Systems, pp 3-14, Academic Press 1993, ISBN 0-12-227748-1.

[Goldenberger and McQuilian 1991] Goldenberger A.A. and McQuilian F.J. ; *Geometric Uncertainty in Off-Line Programming of Robot Manipulators for Assembly Tasks*. Journal of dynamic systems, measurement and control 1991, Vol. 113 pp 329-334.

[Hamura and Kataoka 1995] Hamura M. and Kataoka M., T*he Arc Welding Robot System with AI function.* Proceedings 26th Int. Symposium on Industrial Robots 1995, ISBN 1-86058-000-9.

[Hill and Tang 1988] Hill J. L. andTang S., *Kinematic Simulation of Robotic Systems*. CAD/CAM Robotics and factories of the future 1988, Vol. 3 ISBN 0-387-51134.

[Hirzinger et al. 1994] Hirzinger G., Gombert B. , Dietrich J. and Shi J., *Transferring space robot technologies into terrestrial applications*. Proceedings 25th International Symposium on Industrial Robots 1994, ISBN 0-85298-939-3.

[Kehoe et al. 1991] Kehoe A., et al., *Calibration Concepts for Off-line programming of robotic based manufacturing systems*. Mechatronics & Robotics 1, 1991 IOS press.

[Kugelmann et al. 1994] Kugelmann D. , Reinhart G. and Milberg J., *Autonomous Robotics handling Applying Sensor Systems and 3D simulation*. Proceedings Int. Conf. on Robotics and Automation 1994, pp196-201.

[Lee et al.1995] Lee P. U. , Ruspini D. C. and Khatib O., *Dynamic Simulation of Interactive Robotic Environment.* Robotics laboratory, Stanford University 1995, CA, USA.

[Mayr and Held 1989] Mayr H. and Held M., *SMART: A Universal System for the Simulation of Machining and Robot Tasks*. Computer Application in Production and Engineering, Elsevier 1989.

[McMahon and Brown 1993] McMahon C. and Brown J., *CAD CAM From principles to practice*. Addison-Wesley 1993, ISBN 0-201-56502-1.

[Milgram et al. 1995] Milgram P. , Rastogi A. and Grodski J. J., *Telerobotics Control Using Augmented Reality*. Proceedings 4th IEEE RO-MAN'95, 0-7703-2904-x/95 1995 IEEE.

[Nitzan 1990] Nitzan D., *Encyclopedia of Artificial Intelligence, chapter robotics*. pp 923-944, New York, NY, John Wiley & Sons, 1990.

[Owens 1994] Owens J., *RoboTrak: Calibration on a shoestring*. Industrial Robot. Vol.21 No.&, 1994 pp10-13.

[Piguet et al. 1995] Piguet L. , Fong T. , Hine B. , Hontalas P. and Nygren E., *VEVI: A Virtual Reality Tool For Robot Planetary Explorations*. NASA Ames Research Center 1995, MS 269-3, Moffett Field CA 94301,USA.

[Pook and Ballard 1995] Pook P. K. and Ballard D. H., *Remote Teleassistance*. pp 944-949. Proceedings IEEE Int. Conference on Robotics and Automation 1995. ISBN 0-7803-165-6 .

[Ravani 1988] Ravani B., *World modeling for CAD based robot programming and simulation.* CAD Based Programming for Sensory Robots, Springer-Verlag 1988.

[Renfors et al. 1993] Renfors J., Aalto H. and Jokela M., *Accurate off-line programming for a 12 dof robot cell at Bronto skylift Ltd.* Proceedings Robotikdagar 2-3 Juni 1993, Linköping ISBN 91-7871-136-3.

[Rooney and Steadman 1993] Rooney J. and Steadman P.,*Computer aided design.* Pittman publishing 1993, ISBN 0-273-02672-0.

[Shröer and Bernhardt 1992] Shröer K. and Bernhardt R., *Program execution.* Integration of robots into CIM 1992. ISBN 0-412-37140-5.

[Smith 1992] Smith M.G., *An Environment for More Easily Programming a Robot.* Proceedings of the 1992 IEEE International Conference on Robotics and Automation, Nice, France May 1992 pp 10-16.

[Smith and Gini 1992] Smith R. and Gini M., *Error management for robot programming.* Journal of Intelligent Manufacturing 1992, volume 3, pp 59-73.

[Stobart and Dailly 1985] Stobart R.K. and Dailly C., *The use of simulation in the off-line programming of robots.* Robots and automated manufacture, Peter Peregrinus 1985 ISBN 0-86341 053 7.

[Takahasi and Sakai 1991] Takahasi T. and Sakai T., *Reality.* Proceedings Int. Workshop on Intelligent Robots and Systems 1991, pp 1583-1588, IEEE Cat. No. 91TH0375-6.

[Tarnoff et al. 1992] Tarnoff N. , et al., *Graphical Simulation for Sensor Based Robot programming.* Journal of Intelligent and Robotics System 5, 49-62 1992.

[Theveneau and Pasquier1988] Theveneau P. and Pasquier M., *A Geometric Modeller for an Automatic Robot Programming system.* CAD Based programming of Sensory Robots, Springer Verlag 1988.

[Wapler and Neugenbauer 1994] Wapler M. and Neugebaur J-G., *Improving the integration of Off-line programming and Robotic Simulation Systems with STEP*. Proceedings 25th International Symposium on Industrial Robots, 1994.

[Wahrburg and Papperitz 1995] Wahrburg and Papperitz. *Application of Enhanced Sensor-based Robots to Improve the Automation of Grinding and Deburring Tasks.* Proceedings 26th International Symposium on Industrial Robots 1995, ISBN 1-86058-000-9.

[Wloka 1989] Wloka D. W., *Efficient Calculation of Generalised Forces in a Robot Simulation Environment*. European simulation congress 3, Edingburg 1989, pp 595-601.

[WorkSpace 1] *WorkSpace Users Manual.* Robot Simulations, Newcastle Upon Tyne U.K.

[Wybrow and Wykes 1992] Wybrow and Wykes. *An Ultrasonic system for object recognition in a manufacturing environment.* Journal of Intelligent Manufacturing (1992) 3, pp 163-172.

[Zomaya 1992] Zomaya A. Y., *Modelling and Simulation of Robot Manipulators*. World Scientific Series in Robotics and automated systems, Vol. 8, ISBN 981-02-1043-4.

Chapter 4 Self Learning Robots

Contemporary robots can only work satisfactorily in rigidly structured environments and do not easily tolerate uncertainties or variability in the workspace [ELmaraghy and Rondeau 1991]. To be able to handle uncertainties adaptive robot systems are needed. Adaptive/intelligent behaviour of robots can either be achieved by implementing conventional control strategies or by using robot learning regimes.

4.1 Intelligent robots

An intelligent system can be defined as [Albus 1991]:

> A system having the ability to act appropriately in uncertain environments, where appropriate action is that which increases the probability of success, and success is the achievement of behaviour sub-goals that support the system's ultimate goal. Both the criteria and the system's ultimate goal are defined external to the intelligent system. For an intelligent machine system, the goals and success criteria are typically defined by designers, programmers and operators.

Five different paradigms for machine learning have emerged namely: connectionist (artificial neural networks) learning methods; genetic algorithm and classifier systems; empirical methods for inducting rules and decision trees; analytic learning methods; and case-based approaches to learning [Schlimmer and Langley 1992]. The intelligent process includes four elements [Albus 1991], [Brooks 1986]: (i) sensor processing; (ii) a world model; (iii) value judgement; and (iv) behaviour generation.

Robot learning can be an effective strategy to adopt where; (i) knowledge is difficult to capture in a conventional program; and (ii) unknown information and changing operational environments [Connell and Mahadevan 1993]. The goal of robot learning is to prepare the robot to deal with unforeseen situations and circumstances within its environment [Brooks and Mataric 1993]. The problem most often addressed by both connectionist and symbolic learning systems is the inductive acquisition of concepts from examples. Most learning is based on experience, and this requires a representation for the experimental input given to the learning system. The connectionist approach and the backpropagation algorithm is

chosen in this study as its performance compared to symbolic learning systems is: (i) significantly better on data sets containing numerical data; (ii) better in terms of classification on new examples though it takes a longer training time; and (iii) better when examples are noisy or incompletely specified [Shavlik et al. 1991]. This fits the requirements for industrial robotics.

4.2 Artificial neural networks

Artificial neural networks (ANNs) are systems that consist of a network of interconnected units called artificial neurons. The ideas come from the study of biological nervous systems and neurons. Artificial neural networks are also known as connectionist computing and parallel distributed processing. The ANN produce an output pattern when given an input pattern. The network is trained by changing the relations between connections in the network to produce a desired output pattern to a given input pattern.

4.2.1 Basic principles of artificial neural networks

The elementary element of an ANN is the artificial neuron. The first model is the neuron proposed by McCulloch and Pitts, which combined neurophysiology and mathematical logic to model the neuron as a binary discrete time element. The function of the McCulloch-Pitts neuron is as follows, refer to figure 5 and equation 1. The neuron has one or several input connections (i1-iX), these input connections simulate the synapses of a real neuron. Each input connection has an attached weight (w1-wX). When the sum of all input connections multiplied by their weights reach a threshold value, the neuron 'fires', that is it gives a logical one on its output connection(O). The relationship between the input and the output is called the transfer function (f(x)). Transfer functions other than the hard-threshold function proposed by McCulloch and Pitts can be used to enable a generalised function of artificial neurons, a commonly used transfer function is the sigmoid function which allows non-linearly separable functions to be implemented.

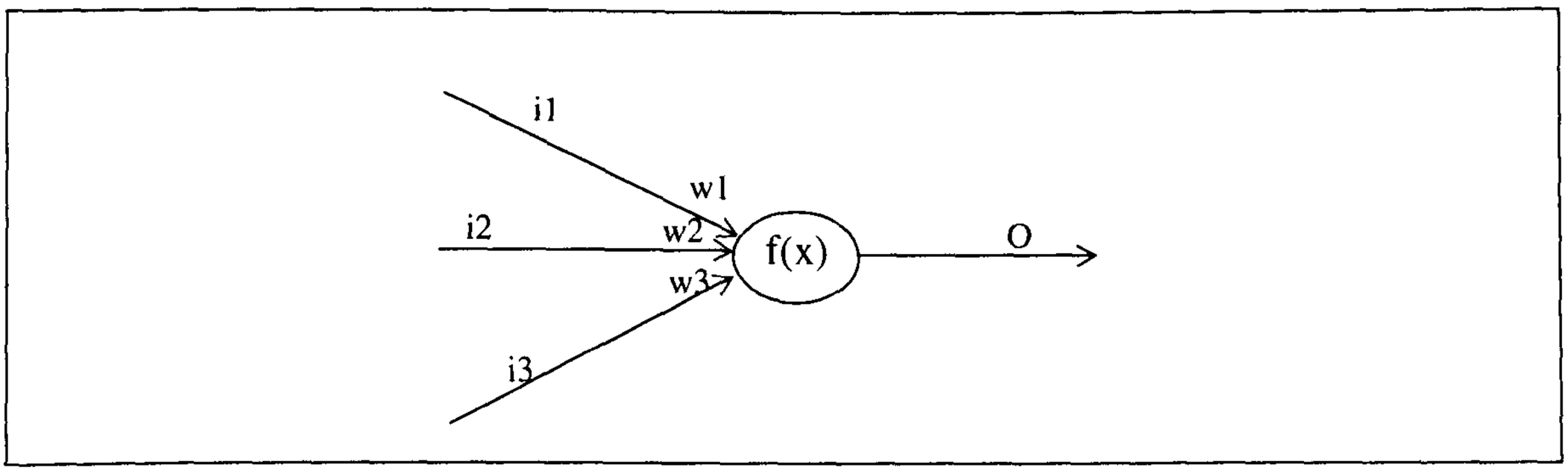


Figure 5: The McCulloch-Pitts neuron

$$O_j = f\left(\sum_{x=0}^{y} i_x w_x\right)$$

Equation 1: The output O from a single neuron(j)

4.2.2 Network structures

There has been and still is a considerable amount of research in the domain on how to organise neurons into network structures. One of the most widely used structures is the multi-layered perceptron (an example is illustrated in figure 6), where the artificial neurons are arranged into layers. A multi layered perceptron is usually constructed of an input layer, n intermediate or hidden layers (n=0-~) and an output layer. The number of hidden layers(n) and the number of neurons in each layer, depends on the learning requirements. If the output of any layer only goes to subsequent layers and not to earlier layers, the layout is called a multi layered feed forward network. If artificial neurons send their output back to artificial neurons in preceding layers, the network is called a feedback network or recurrent network. By arranging artificial neurons into layers, implementation of non-linear separable functions is made possible. Other network structures used are Hopfield networks and Kohonen networks [Eberhart and Dobbins 1990].

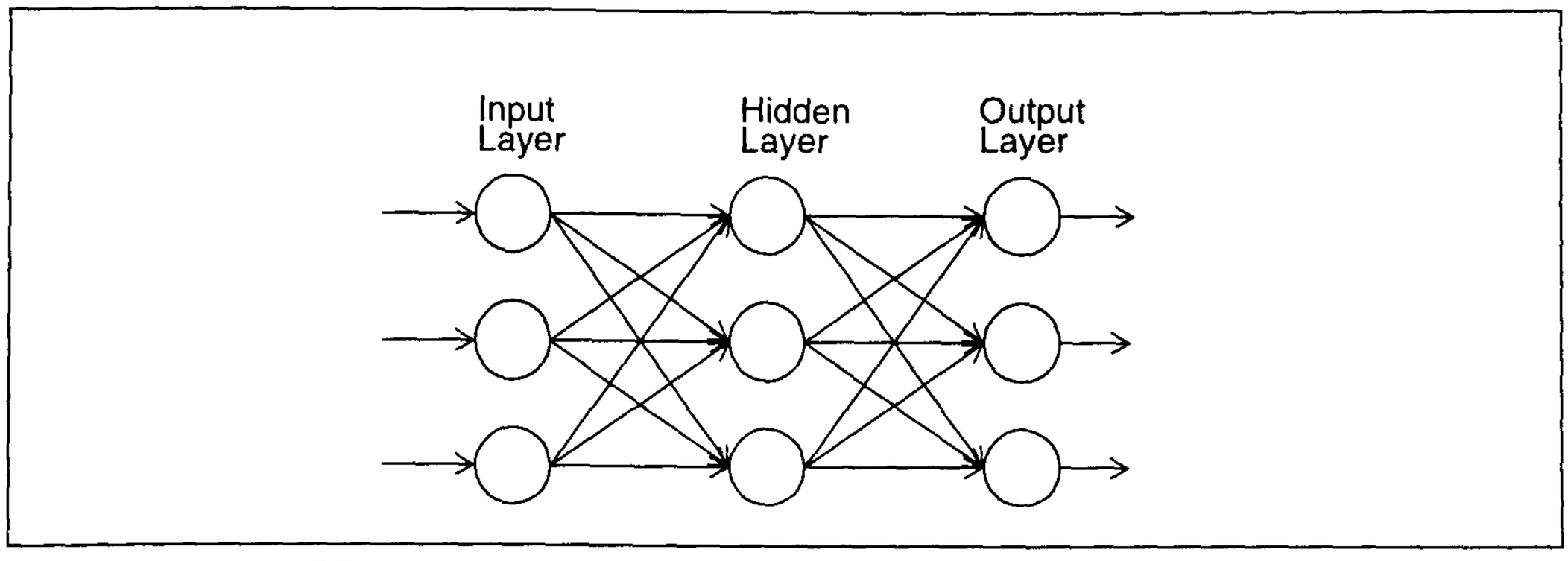


Figure 6: Multi-layered artificial neural network

4.2.3 Learning in artificial neural networks

The ANN is trained by using sets of matching input and output patterns. Coefficient weighting adjustment, termed learning in the neural network case, is an attempt to adjust network weights such that for the same input pattern the network provides, as near as possible, the same corresponding output pattern.The input pattern is presented and propagated through the network. The output from the output layer is compared with the output pattern that corresponds to the input pattern propagated. The difference is calculated and used to adjust the connection weights (equation 2). Learning algorithms are generally based on the minimisation of an error or energy function E(W), where W is the matrix of weights to be adjusted. Gradient descent algorithms are widely used as error correction functions [Warwick 1996]. The length of the search step (often called the learning rate) $\eta(k)$ controls the amount of learning needed for a neural network to converge. A higher value creates faster learning but could create a nervous(unstable) system. A smaller value results in a longer training time but gives better stability. The learning rate can be set to a high value at the beginning of the learning phase and then decreased as the learning proceeds.

There are several error correction rules described in the literature, one of the most widely used is the backpropagation method. Backpropagation involves making corrections to the weights between the last-but-one layer and the last layer first, then using the calculations involved in these corrections as the basis for calculating the corrections for the next layer, seen from the back of the architecture, until the input layer is reached.

The backpropagation method is based on the steepest-descent method to arrive at a minimum of the mean squared error cost function.

$$W(k+1) = W(k) + \eta(k)d(k)$$

W(k) is the weight at step k.

d(k) is the search direction.

n(k) is the length of the search step

$$d(k) = \Delta E(W(k))$$

Δ indicates the derivate, or gradient of the error function.

Equation 2: Weight updating

4.2.4 Artificial neural networks and their applications in robotics

There are historical parallels between the study of artificial neural networks and robots. Artificial neural networks are suitable for robotic control, where there are parameters not precisely known such as physical dimensions, joint friction etc. [Bekey and Goldberg 1993]. The two main research areas in the field of robotics and artificial neural networks are: (i) the use of neural networks for low-level behaviour control, such as obstacle avoidance, for mobile robots [Nemzow and McGonicle 1993], [Sekiguchi et al. 1992], [Chesters and Hayes 1994], [Biewald 1996]. In mobile robotics, neural networks are usually used to map sensor readings to correct actions. The input layer is usually feed with information from various sensors and trained to map these sensor readings to an appropriate motion control; and (ii) kinematic and dynamic control for robot manipulators [Aylor et al. 1992], [Wu et al. 1993], [Li and Zeng 1993], [Warwick 1996], [Zalzala 1996], where the neural network is used to replace or support conventional control algorithms. Figure 7 illustrates how the inverse kinematics of a six-axis manipulator could be solved by using a layered network, where pairs of cartesian coordinates (of the robots TCP) and the corresponding joint values are used for training. These pairs could be constructed by moving the TCP to known positions and reading the actual joint values. The ANN should then be fed with the cartesian coordinates as input

and the output from the network should be the corresponding joint values. By using this approach manufacturing deficiencies could be incorporated in the inverse kinematic function.

Artificial neural networks are appropriate for processing complex sensory information to generate robot actions, as they are trained by adjusting weights, when presented examples of start states and corresponding goal states. These are especially appropriate where the exact analytical function for mapping of start states to goal states is difficult or impossible to find.

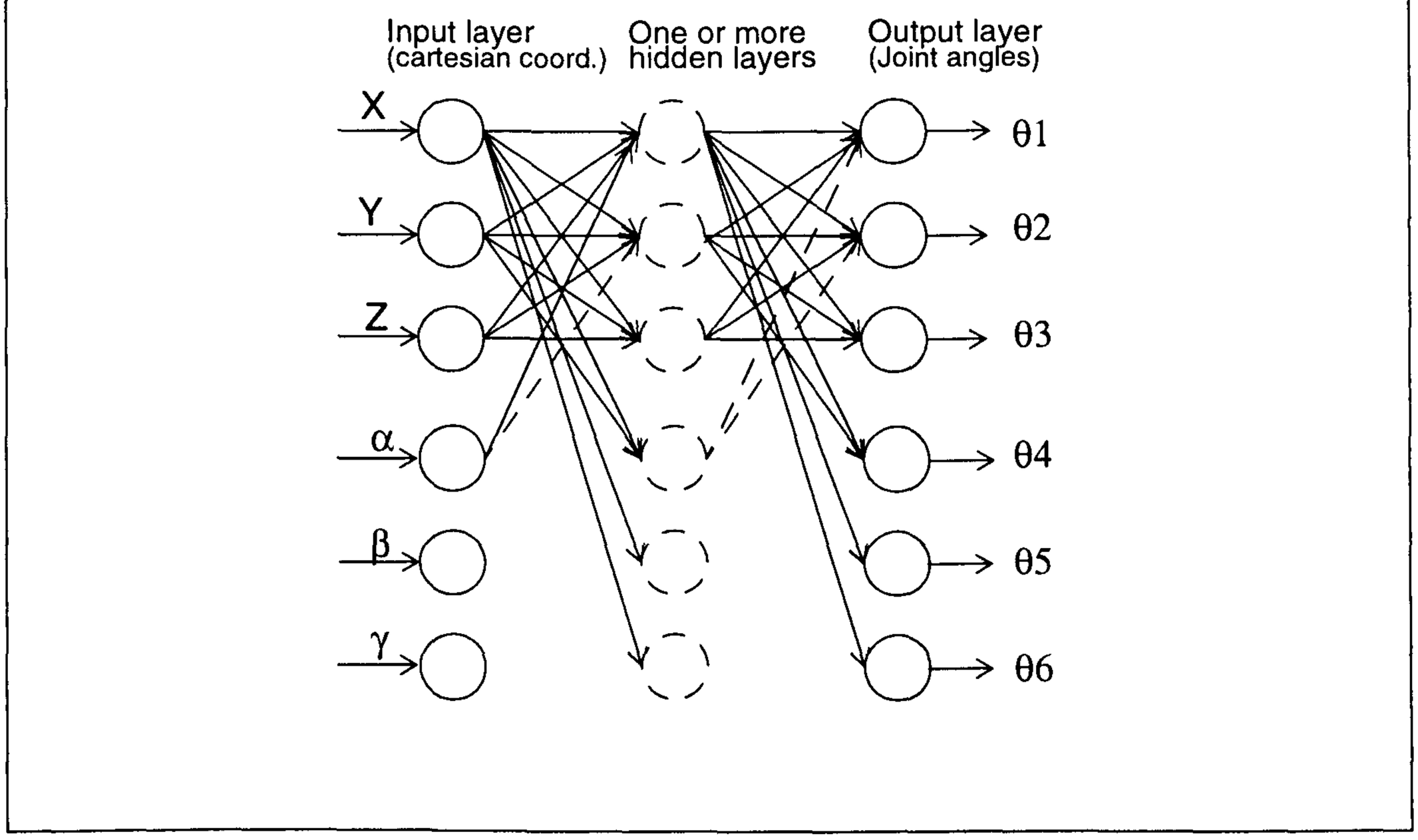


Figure 7: Network for solving inverse kinematics of robot manipulators

This section discusses some of the significant work done in the area of robotics and artificial neural networks, it is not an attempt to cover the complete research area. Much of the work in the field of neural networks is said to be targeted towards robotics but much of the work presented is far removed from the field of industrial robots or even physically working machines.

Biewald presents research in artificial neural networks for navigation of mobile robots, based around a non-holonomic robot vehicle and man-made environments [Biewald 1996]. Man

made environments can be classified as simple semi-regular environments.

Biewald's controller is divided into sets of artificial neural networks that are dedicated for different tasks in the overall navigation goal. Local navigation is performed by multi-layered perceptrons which are trained using the backpropagation algorithm. In the approach, separate neural controllers are used for each basic action. This does, according to the author, reduce the complexity and time of learning. During training one neural network that emulates the vehicle kinematics and one neural network that emulates the sensors are used to train the controller neural networks. The author tries to avoid the use of a global artificial coordinate system. In the early phase of the training, an internal state vector consisting of the vehicles position and orientation is used. This state vector is not presented to the neural controller as it is only used by the vehicle emulator. The system uses a symbolic map which is a network of places and paths. The nodes in the network represent places and the connections are the paths. Local navigation tasks learnt are: learning to keep distance; learning to follow a wall; and learning to turn and go straight ahead on cross-roads. Places are classified according to the sensory data obtained at certain positions in the environment (views). The environmental description is constructed with paths(connections) between places (nodes). Planning of routes between places is done through conventional AI search techniques.

Nehmzow uses a connectionist associative memory to control the low level behaviour of a mobile robot [Nehmzow 1992]. The connectionist associative memory stores the effective associations between input signals and motor actions. The approach is to train the neural controller on one task at the time, starting with the simplest. After the first task is learnt, another task to be learnt is added and so on. The research shows that the same neural network can be trained to make distinctions between different mappings of sensor input vectors and motor actions without forgetting previous learnt knowledge.

Zalzala [Zalzala 1996] has designed a real-time controller for motion control of manipulators using multi-layered networks where adaption for environmental changes is accommodated using a backpropagation algorithm. The controller architecture consists of two artificial neural network based controllers, one for solving the dynamic control and a second for solving the inverse jacobian. The work is applied to a PUMA six degrees of freedom manipulator. The dynamic controller consists of three individual artificial neural networks; (i) one dealing with the acceleration related term of the dynamic equation; (ii) a second taking care

of the velocity related term; and (iii) the third solving the gravity term. Inputs are formed from three sets of values, each with six parameters (one for each joint) namely; (i) position of each joint; (ii) velocity of each joint; and (iii) acceleration of each joint. The velocity network is fed with the sets of position and velocity values. The acceleration network is fed with the sets of position and acceleration values and the gravity network is fed with the set of position values. The networks are all fed with the same input vector but each have different internal architectures. The output from the controller, six nodes, is formed from the summation of the outputs from each individual controller network. Problems reported in the study are that there are no analytical models of how to design the networks and establish the initial weights and learning rate. This must be done based on experience [Niklasson 1996]. Zalzala suggests that the use of an initial model of the manipulator within the neural controller would give a starting point and could reduce the learning time needed.

Van der Smagt has shown that it is possible to control a robot manipulator for on-line grasping of objects with unknown dimensions using monocular vision and an artificial neural network [Van der Smagt1996]. Instead of using stereo vision the time derivate of the visual field is used to obtain depth information. The neural network is trained to deal with the on-line motion planning for the grasping operation from gross-motion to fine-motion planning. The neural network takes the velocity obtained from the visual observation as well as the current robot position, velocity and acceleration. The output of the network consists of joint accelerations for each individual joint.

Further research on robotics and artificial neural networks such as coordinate transformations for solving the kinematics of robot manipulators is described in [Aylor et al. 1992]; Robot inverse kinematics and trajectory planning is described in [Wu et al. 1993], [Li and Zeng 1993] and mobile robot control is described in [Nehmzow and McGonicle 1994], [Sekiguchi et al. 1992], [Chesters and Hayes 1994].

4.3 Summary

Research on artificial neural networks has been closely connected to research on autonomous systems, where robotics can be seen as a sub-branch. An intelligent sensory-based robotic system is expected to operate independently and in real-time. Thus the controller must be

able to generalise its operation to include any new situations with a minimum of extra learning and the associated information processing must be fast. Most research on self learning robots has focused on the use of artificial neural networks for learning and imitation of intelligent behaviours, or use artificial neural networks as part of the overall system solution. Much of the research on artificial neural networks use "robots" for evaluation of theories. These "robots" are usually holonomic robots with no kinematic or physical constraints. They have "intelligent" sensors which are able to detect "food", "humans", "doors", etc. Most of the research has been carried out in 2-D simulators where the robot is a circle or a point. Whilst others, as described above, use real robot manipulators to learn actual motion control.

Neural networks are well suited for intelligent robotic control as they:

- can flexibly and arbitrarily map non-linear functions.
- map interactions and cross-couplings readily whilst incorporating many inputs and outputs.
- can be trained off-line with on line tuning or entirely trained on-line.
- are inherently parallel processing devices.
- are noise tolerant and are able to work even if there is information loss in input data.

Neural controllers can cut the computational complexity for manipulator control, through their inherent parallel processing capability. They also provide a learning scheme through which adaption to any changes in the environment can be accommodated. It is useful to give the neural controller some initial knowledge, through simulations, before performing the fine-tuning on the real robots [Biewald 1996].

The use of realistic 3-D simulation of robots, sensors and the environment would provide an incentive to allow more realistic evaluation of new theories in the field of neural networks, furthermore, it could also be used for the first training phases of robots using intelligent controllers before they are exposed to the real environment for fine tuning.

4.4 References

[Albus 1991] Albus J.S. *Outline for a theory of intelligence*. IEEE Transactions on Systems, Man and Cybernetics, 1991, pp 473-509.

[Aylor et al. 1992] Aylor S., Rabelo L. and Alkptekin S., *Artificial Neural Networks for Robotics Coordinate Transformation*. Computers in Engineering Vol. 22, No 4, pp. 481-493, 1992.

[Bekey and Goldberg 1993] Bekey G. and Goldberg K., *Neural Networks in Robotics*, 1993 ISBN 0-7923-9268-X.

[Biewald 1996] Biewald R., *A neural network controller for the navigation and obstacle avoidance of a mobile robot.* Neural Networks for Robotic Control edited by A.M.S Zalzala and A.S Morris, Prentice Hall 1996, ISBN 0-13-19892-0.

[Brooks 1986] Brooks R.A., *A Robust Layered Control System For A Mobile Robot*. IEEE Journal of Robotics and Automation 1986, Vol.2, No.1 pp 14-23.

[Brooks and Mataric 1993] Brooks R.A. and Mataric M. J., *Real Robots, Real Learning Problems*. Robot learning, Kluwer Academic Publisher 1993.

[Chesters and Hayes 1994] Chesters W. and Hayes G., *Connectionist Environment Modelling in a Real Robot*. Department of Artificial Intelligence, University of Edinburgh.

[Connell and Mahadevan 1993] Connell J.H. and Mahadevan S., *Introduction to robot learning*. Robot Learning, Kluwer Academic Publisher 1993.

[Eberhart and Dobbins 1990] Eberhart R. and Dobbins R., *Neural Network PC Tools*. Academic Press 1990, ISBN 0-12-228640-5.

[ELmaraghy and Rondeau 1991] Elmaraghy and Rondeau J.M., *Automated planning and Programming environments for robots*. Robotics 1992, Vol. 10 pp 75-82.

[Li and Zeng 1993] LI Y. and Zeng N., *A Neural Network Based Inverse Kinematics Solution in Robotics*. Neural Networks in Robotics, 1993 ISBN 0-7923-9268-X.

[Nehmzow and McGonicle 1994] Nehmzow U. and McGonicle B., *Achieving Rapid Adaptions in Robots by Means of External Tuition*. From animals to Animates 3, MIT Press 1994.

[Nehmzow 1992] Nehmzow U., *Experiments in Competence Acquisition for Autonomous Mobile Robots*. Ph D Thesis University of Edinburgh 1992.

[Niklasson 1996] Niklasson L., PhD artificial neural networks, Interview 1996, Connectionist research group, University of Skövde, Sweden.

[Schlimmer and Langley 1992] Schlimmer J. C. and Langley P., *Encyclopedia of Artificial Intelligence*. 1992, 0-471-50307-X pp. 785-805.

[Sekiguchi et al. 1992] Sekiguchi M , Nagata S. and Asakawa K., *Behaviour control for a mobile robot by structured neural network*. Advanced Robotics, vol 6 No. 2, pp 215-230 1992.

[Shavlik et al. 1991] Shavlik J.W., Mooney R.J. and Towell G.G., *Symbolic and Neural Learning Algorithms: An Experimental Comparision*. Machine Learning, 1991, vol. 6, pp 111-143.

[Van der Smagt 1996] Van der Smagt P., *A robot arm is neurally controlled using monocular feedback*. Self Learning Robots, IEE, Savoy Place London 1996.

[Warwick 1996] Warwick K., *An overviewof neural networks in control applications*. Neural Networks for Robotic Control, edited by A.M.S Zalzala and A.S Morris, Prentice Hall 1996, ISBN 0-13-19892-0.

[Wu et al. 1993] Wu C.M., Jiang B.C. and Wu C.H., Using Neural Networks For Robot Positioning Control. Robotics and Computer Integrated Manufacturing, Vol. 10 No. 3 pp 153-168, 1993.

[Zalzala 1996] Zalzala A.M.S., *Model-based adaptive neural structures for robotic control.* Neural Networks for Robotic Control, edited by A.M.S Zalzala and A.S Morris, Prentice Hall 1996, ISBN 0-13-19892-0.

Bibliography

Arbib M.A. and Buhman J. *Neural Networks* . Encyclopedia of Artificial Intelligence, 1992, 0-471-50307-X pp. 1016-1060.

Bekey G and Goldberg K. *Neural Networks in Robotics.* 1993, ISBN 0-7923-9268-X.

Bharath R and Drosen J. *Neural Network Computing*. McGraw-Hill 1994, ISBN 0-8306-4523-3.

Eberhart R and Dobbins R. Neural Network PC Tools. Acedemic Press 1990, ISBN 0-12-228640-5.

Picton P. *Introduction To Neural Networks*. Macmillan 1994, ISBN 0-333-61832-7.

Chapter 5 Research Tools Used

To investigate simulation of sensors, use of virtual sensors for robot programming and 'pre-emptive learning' of robots a variety of platforms were used. This chapter provide an overview of the major systems used in the research study. Figure 8 is an illustration of the platforms and the interaction between systems.

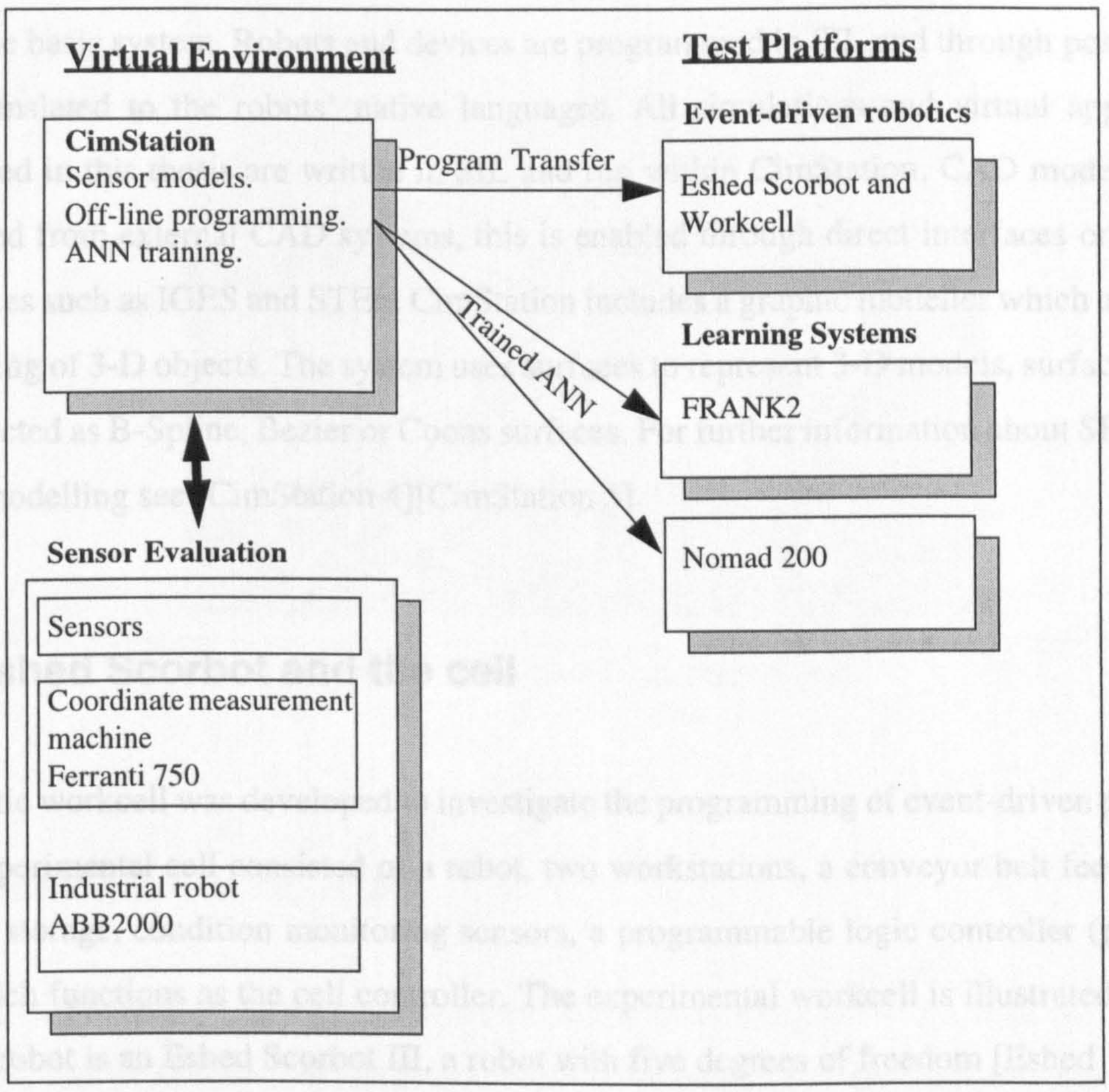


Figure 8: Experimental platforms used

5.1 CimStation

The virtual robotics tool CimStation was used as a development and simulation platform. CimStation is a graphical simulation and off-line programming system for devices that can be described as kinematics chains [CimStation 1]. The system consist of a graphic kernel into

which modules can be incorporated. There are modules for basic robotics; coordinate measuring machines; cnc-machines; and robotic tasks such as spray painting, spot-welding and arc welding. The system runs under UNIX on multiple hardware platforms. The system provides a programming language called SIL. SIL was designed for graphical simulation of kinematic devices. SIL has similarities with Pascal with features from both LISP and C added. SIL is an object oriented language. Code can either run interpreted or compiled. The language is a multi-process language and allows inter-process communication. CimStation itself was created using SIL. The kernel of the system is accessible for the user, enabling the user to rewrite the system's behaviour. The system enables the user to create menus and shells to run upon the basic system. Robots and devices are programmed in SIL and through post-processors translated to the robots' native languages. All simulations and virtual applications described in this thesis are written in SIL and run within CimStation. CAD models can be imported from external CAD systems, this is enabled through direct interfaces or standard interfaces such as IGES and STEP. CimStation includes a graphic modeller which allows the modelling of 3-D objects. The system uses surfaces to represent 3-D models, surfaces can be constructed as B-Spline, Bezier or Coons surfaces. For further information about SIL and the CAD modelling see [CimStation 4][CimStation 5].

5.2 Eshed Scorbot and the cell

A robotic workcell was developed to investigate the programming of event-driven programs. The experimental cell consisted of a robot, two workstations, a conveyor belt feeder, component storage, condition monitoring sensors, a programmable logic controller (plc) and a PC which functions as the cell controller. The experimental workcell is illustrated in figure 9. The robot is an Eshed Scorbot III, a robot with five degrees of freedom [Eshed 1]. A procedure and function library, for control of the Eshed Scorbot III, has been created in the Pascal programming language. These procedures and functions have been implemented in the cell controller PC to provide command and control functions for operating the robot with respect to both motions and auxiliary functions. This allows the robot to be programmed in Pascal (the native language of the robot controller is not used in this research study). The cell controller PC is used for programming and control of the robot and the control of the entire workcell. A virtual model of the robot workcell has been created in the virtual robotics

system, CimStation. The same procedures and functions are made available for use within the virtual robotics system's programming environment. The syntax of the languages in the virtual and the real environment is the same, this being desirable when creating event-driven robot programs off-line as it ensures limited amendments before down-loading to the controller. The workcell contains a variety of sensor types (inductive, optical and ultrasonic) with which the robot interacts. The sensors detect events all of which the robot interacts with.

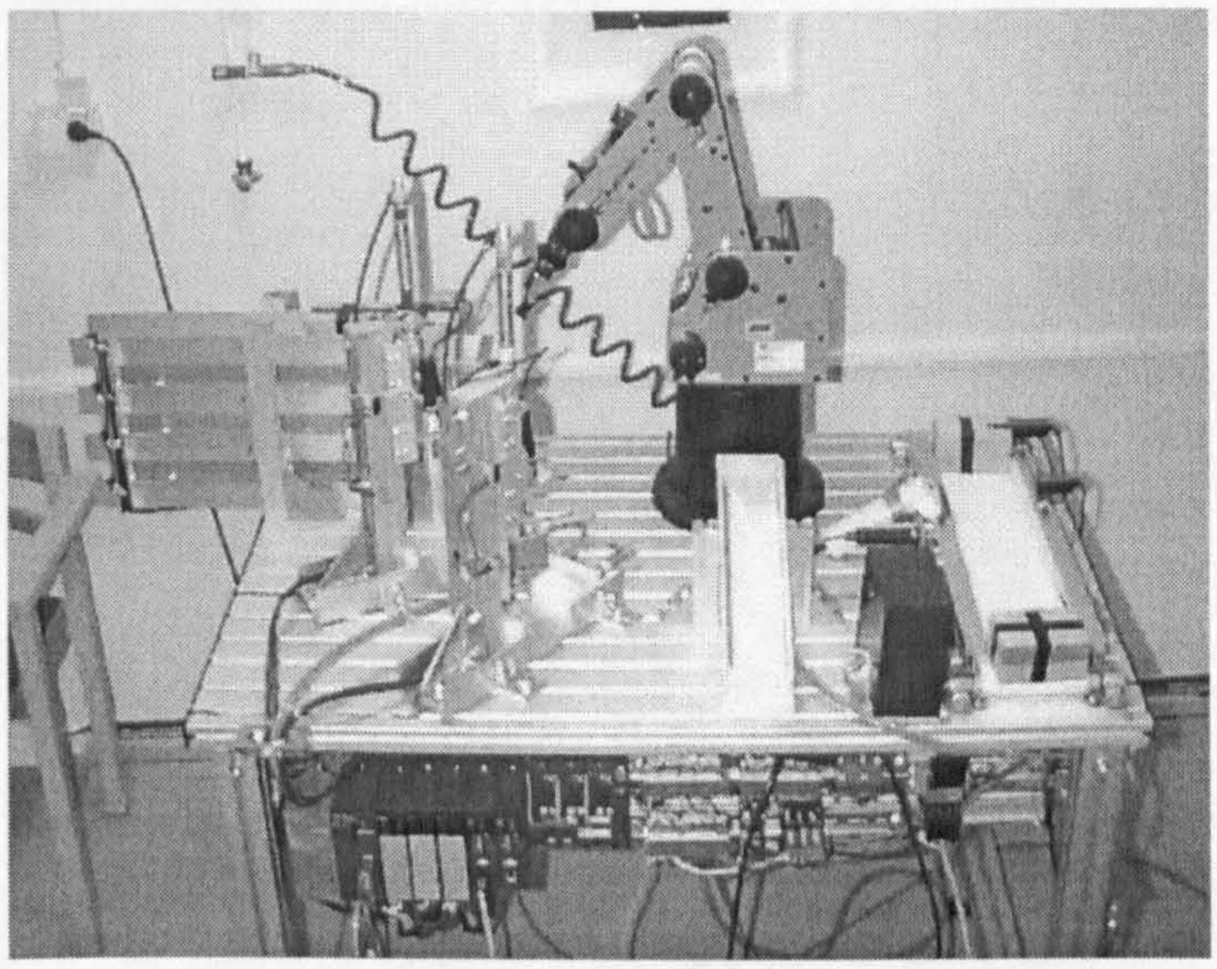


Figure 9: Event-driven robot cell

5.3 Frank2, a mobile robot

The Frank2 robot, from TAG Ltd.(figure 10), has four sensor pods with a Polaroid ultrasonic transducer, an infrared sensor and a bumper sensor in each sensor pod. It has two individually controlled motors and a 486 computer onboard. The Polaroid ultrasonic transducers have a detection range between 150-2000 mm and the infrared sensors have a detection range between 0-150 mm. The robot can be programmed in any Intel-based programming language that enables direct access to the data bus. The sensors and motors are accessed and controlled through a PC-30 card which includes A/D and D/A converters and digital I/O lines [TAG 1]. In this research study MS-Dos and Borland Turbo Pascal have been used to implement motor control and sensor interfacing procedures. The control architecture described in section 8.1

has also been implemented in Pascal.

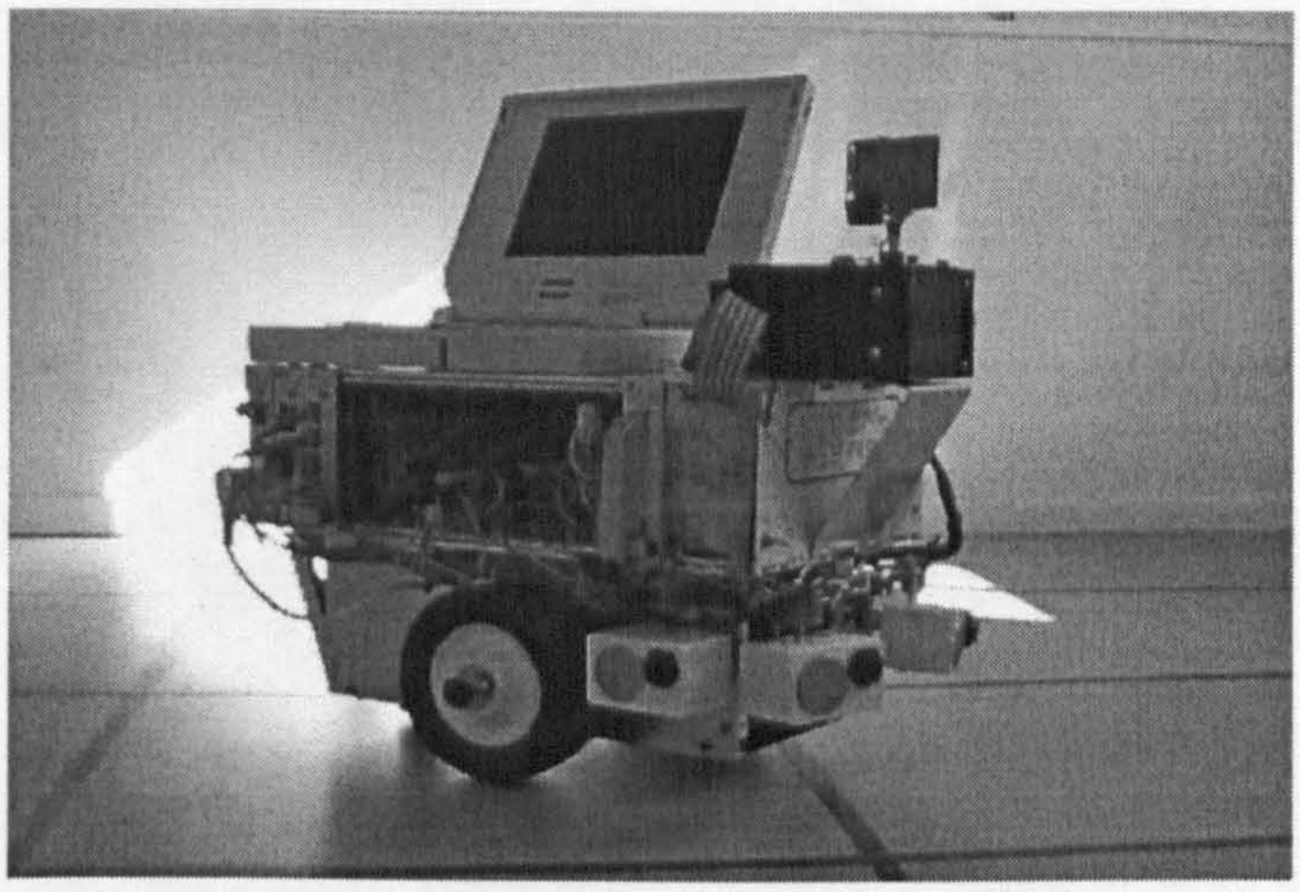


Figure 10: FRANK2 mobile robot

5.4 Nomad 200 robot

The Nomad 200 robot (figure 11) from Nomadic Technologies used in this research study, is equipped with ultrasonic sensors, infra-red sensors, bumber sensors and a structured light vision system. The robot has a Pentium computer on-board and is controlled through wireless ethernet by a server program running on a Unix workstation. The robot is programmed in the C-language and basic functions for controlling the robot, such as motion commands and sensor readings, are provided with the system. The ultrasonic sensor ring consisting of 16 Polaroid sensors driven by a Polaroid 6500 ranging board giving the sensors a detection range between 431 and 6477 mm and a beam width of 25 degrees. [Nomadic 1].

Figure 11: A Nomad 200 robot

5.5 References

[CimStation 1] *CimStation Users Manual*. Silma Inc., Cupertino, CA.

[CimStation 4] *Developers Guide*. Silma Inc., Cupertino CA, 1994.

[CimStation 5] *Command Reference*. Silma Inc., Cupertino CA, 1994.

[Eshed 1] Scorbot-ER III User's Manual, Eshed Robotec 1992, Tel Aviv, Israel.

[Nomadic 1] *Nomad 200 Users Manual*, Nomadic Technologies 1994, USA.

[TAG 1] Frank2 Mobile Robot Research Platform ,User Guide and Reference, TAG 1994, Alnwick, Northumberland, Uk.

Chapter 6 Sensor Simulation

The objectives for using simulation of sensors in graphical programming systems are discussed. A method using a generic sensor model for simulation of sensors, is presented. Models, simulations and experimental validation of photoelectric, proximity and ultrasonic sensors are presented. The generic model as implemented, is limited to the modelling of non-contact sensors such as proximity and range measurement devices. If sensor with more 'intelligence and complexity are to be implemented the corresponding 'intelligence' must be added to the sensors control program. Vision systems are not considered in this research study.

6.1 Objectives for sensor simulation

To be able to simulate interaction with objects in a realistic way, virtual robotic systems must provide sensor simulation capabilities. For off-line programming and debugging of robot software for event-driven robotic systems, sensor based programming and the ability to simulate sensor behaviour within the virtual environment is necessary. The use of sensor simulation capabilities can also enable off-line development and simulation of sensor responsive control strategies. Robots using learning regimes, such as artificial neural networks can use the simulated sensors as input in the training phase to learn about error recovery reasoning. If uncertainties in an object's location are incorporated in the workcell description then robots using learning strategies (using the simulated sensors) should be able to learn to adapt to changes in the environment. If appropriate learning strategies can be developed this could result in a reduction of the need for workcell calibration. For realistic interaction between robots and sensors and direct use of code and knowledge developed in graphical simulation environments, the output signals from the simulated sensor devices must have operational characteristics which are similar to the real devices (namely the detection range and output format). To enable the rapid generation of 'sensor models' for sensor devices of different types, a generic sensor model (described in section 6.2) has been created. The generic sensor model can be used to produce models for simulation of sensors such as inductive proximity sensors, photoelectric sensors and ultrasonic transducers.

6.2 Generic sensor model

The generic sensor model, as shown in figure 12, is divided into two parts: (i) The geometric definition which describes the sensor's detection range and the device geometry; and (ii) the functional definition which describes the operational characteristics, such as detection rate and output format. Each virtual sensor has its own associated control program, this enables the sensors to act as independent objects within the simulated workcell. The sensors geometric description of range and device body are created using a CAD system. Information about the nature(geometry) of the detection range can either be retrieved from the manufacturers data sheets and/or by experimental verification. The sensor object is created with a null object as the parent object, where all geometric objects, such as range volume, trace-lines, sensor body etc., are adopted (put as children) by the parent object (figure 13).

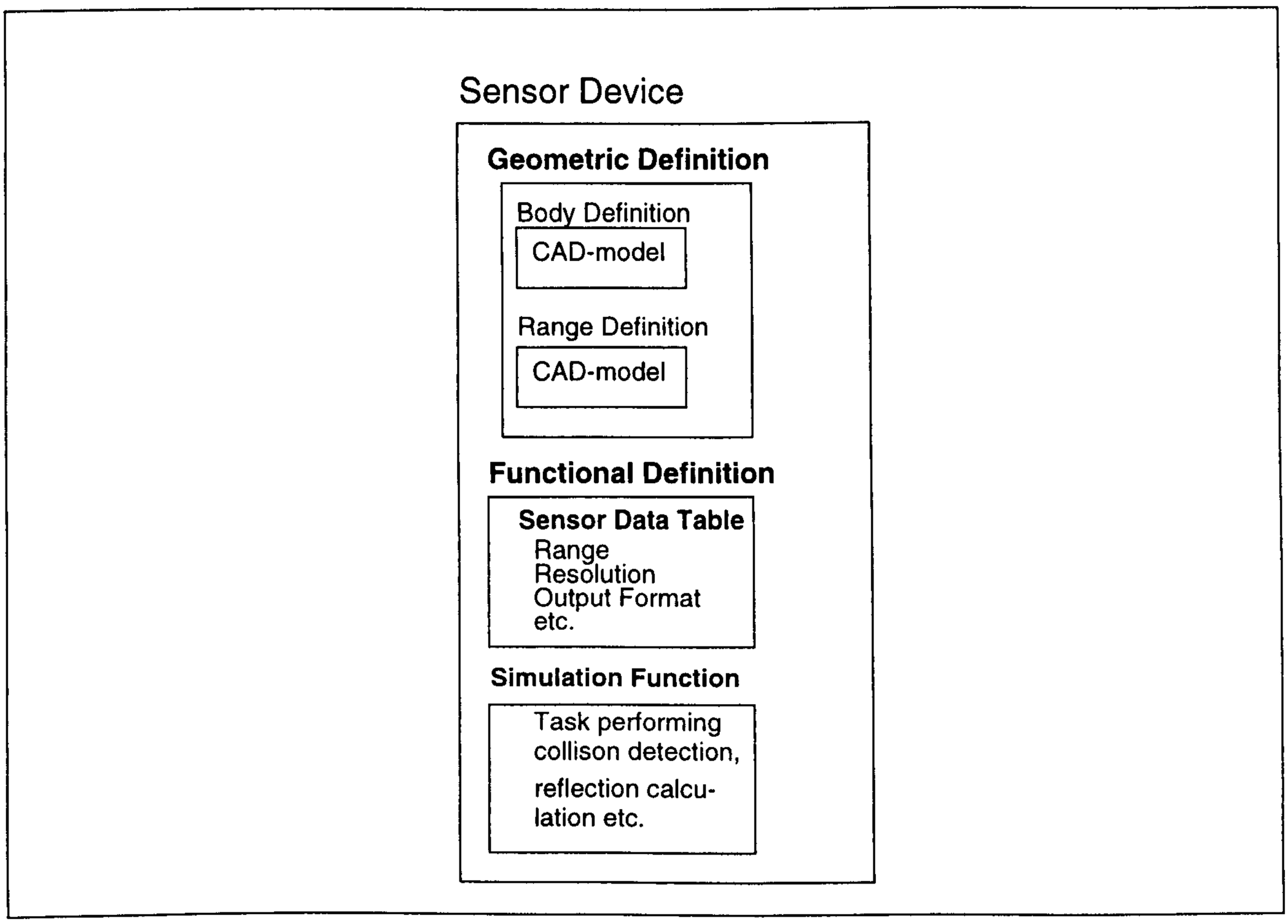


Figure 12: The generic sensor model

A *template-file and 'general program blocks'* are used for creating the sensors functional descriptions, see section 6.2.1.1. By identifying the nature of the sensory device, for example device type, output format and number of trace-lines, the appropriate 'program blocks' can

be selected. The 'program blocks' selected are assembled within a sensor control program, see section 6.2.2. The template file holds variables which need to be initialised and linked to the geometric definitions of the sensor object. General sensor detection is utilised by performing collision detection between the CAD model representing the detection range and the rest of the virtual environment. Depending upon the sensor type simulated, the virtual sensor has one or more trace-lines associated with it, these trace-lines are used for the calculation of distances and angles between the sensory device and detected surfaces.The phase, when working with the generic sensor model to create virtual sensors, is illustrated in figure 14.

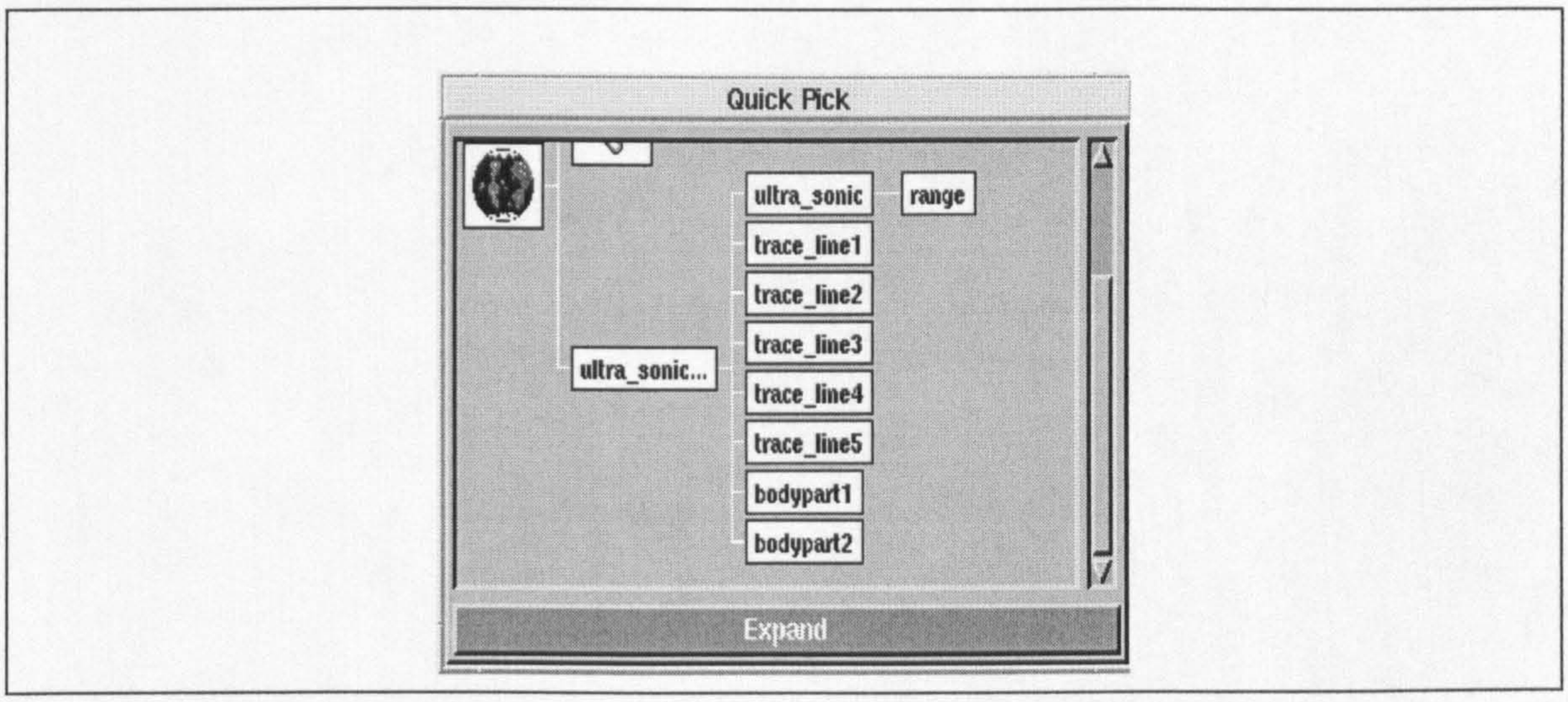


Figure 13: Object structure for sensor object

6.2.1 Using the template and assigning parameters

This section describes how a sensor object is created. It describes the 'general program blocks', variables/identifiers in the template file and illustrates the process used when creating a sensor object in Cimstation. Each simulated sensory device has an associated sensor control function, which performs the collision detection between the simulated detection range and the environment. If the simulated sensor is of the range measurement type, the distance and angle to the detected object are also calculated by this control function. The sensor control function is associated with a sensor by assignment of parameters, such as the object representing the simulated range, trace-lines and default output.

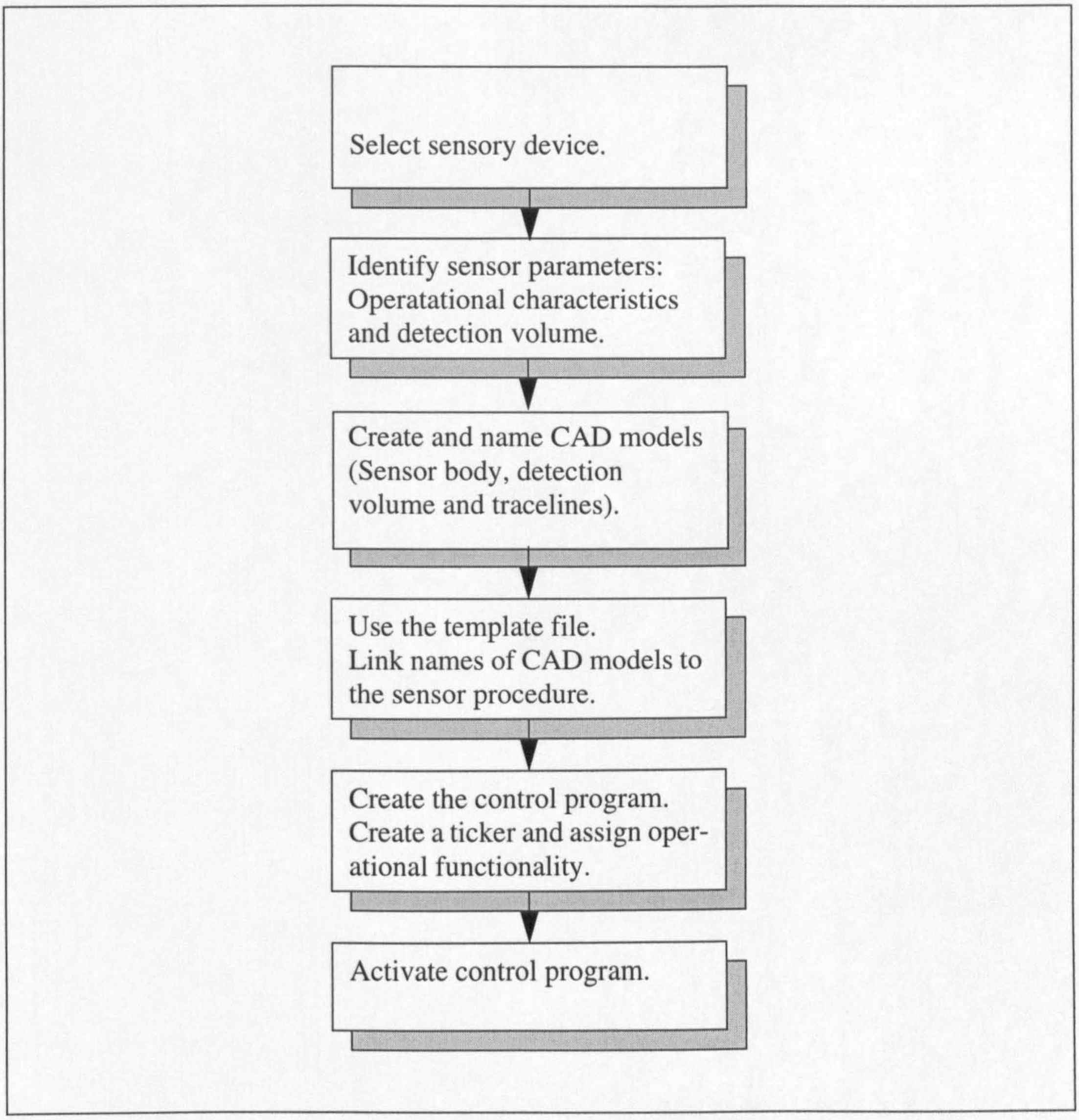


Figure 14: The process of creating a virtual sensor

Variables and identifiers used when creating a sensor control program are:

Sensor Name: should be the same as the name of the 'geometric' sensor object.

Number of trace-lines: this variable is used to identify the number of trace-line identifiers needed.

Device type identifier: used to indicate the device type (proximity or range).

Output format identifier: describes the format of the sensor output i.e. boolean, integer or real.

Update rate: this variable determine the 'speed' of the sensor control program i.e. the interval

with which the sensor updates it's output.

Default output: identifies the value a sensor delivers if it is out of detection range.

Identifiers used by the sensor control program of which some need to be linked to the 'geometric' sensor parts:

Detection range identifier: a variable representing the detection range and which is linked to the CAD object representing the detection volume.

Trace-line identifiers: if the sensor type simulated is a range sensor, one or several variables representing trace-lines are assigned to the CAD lines representing the trace-lines. Trace-lines are used for calculation of distance and angle to the detected objects (see section 6.2.2)

6.2.1.1 Functions, Procedures and 'Program blocks' used to assemble a sensor control program and to 'complete' a sensor object

This section describes the functions, procedures and 'program blocks' (parts of programs) used to construct the simulation function of a sensor object. Using the generic sensor model 'program blocks' are assembled to a sensor control program. Depending on the sensor characteristics to simulate different combination of 'blocks' are used. The functions, procedures and 'program blocks' are described in more detail in section 6.2.2.

Function is_hit():Boolean: Check for collisions and reports true if any collision occurs. This function is used for all device types

Function reflect (Detected_object, line, treshhold_value) :real: Calculates the angle between a traceline and a detected surface. The function returns a value of type real, the code can be extended with code to handle distortion which may occur near the treshold value. This function is used with range measurement sensors.

Check range for collision (range_object) : This block identifies if the sensors range object is colliding with any objects (detection). This block is used for all device types.

Calculate distance (trace_line): This block calculates the distance from a trace-lines origin to a detected object. This block is used with range measurement sensors and is duplicated for each trace-line used.

Calculate minimum distance: This block compare the distances calculated for the trace-lines and reports the minimum distance.

Function Report_Sensor : real: This function is created by assembling the above blocks and functions. A function is created for each individual sensor object.

Function OutPutValue (distance:real): type: This function calculates the 'realistic' output value for the device simulated. The body of the function must be manually created for each device.

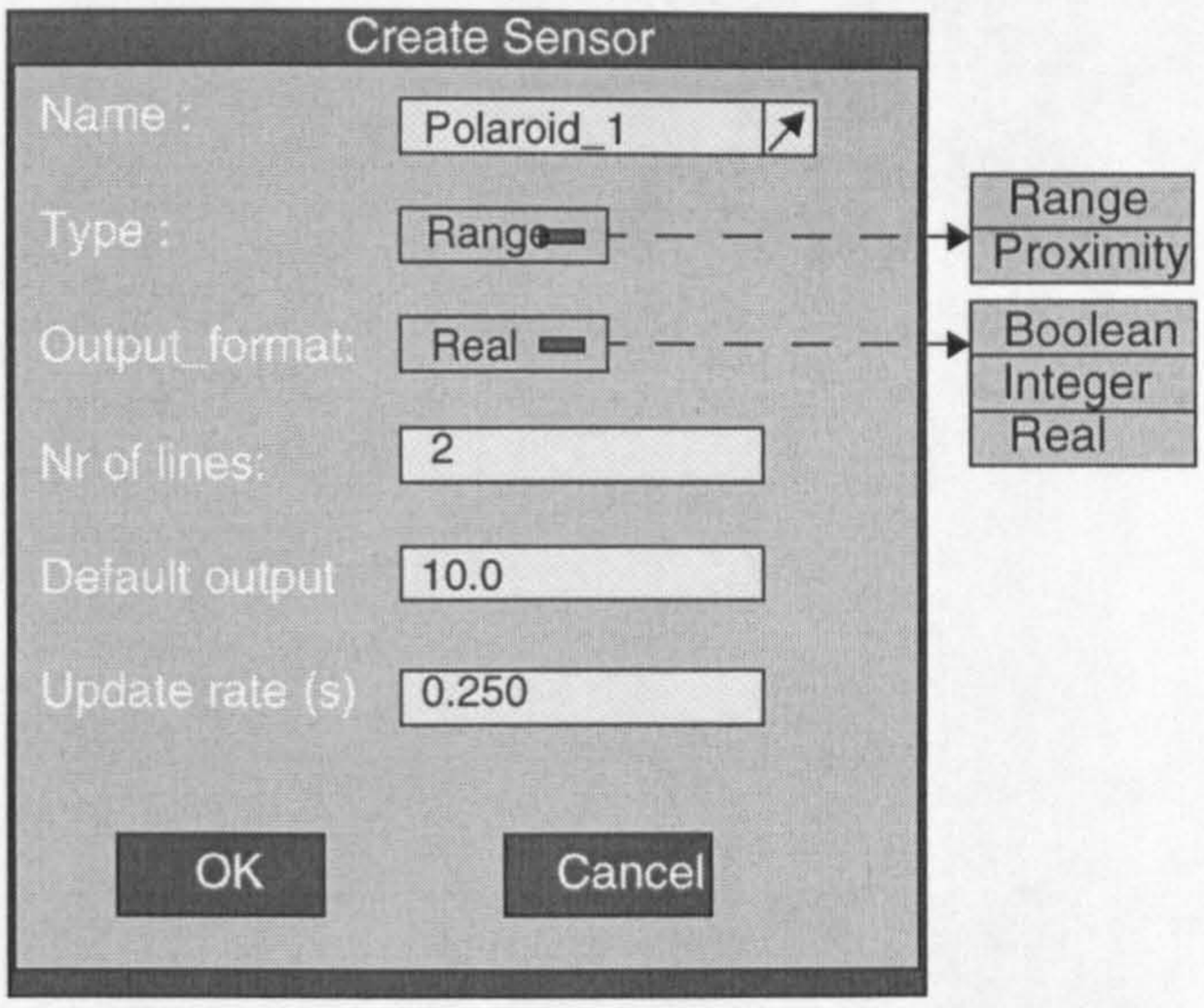
Procedure Sensor_Sample: This is the complete sensor procedure, which performs one sensor sample. This procedure contains the Report_Sensor and OutPutValue functions. There is one procedure for each sensor.

Ticker Sensor_control: This program makes the sensor_sample procedure an independent process which is executed with the interval identified with *Update rate* variable. There is one program for each sensor.

6.2.1.2 Interactive creation of sensor objects

The following section illustrates the interactive creation of a sensor object in CimStation using the generic sensor model. The first step in creating a sensor object is to model the sensors geometric entities. A 'parent' object entitled with the name of the sensor object is first created. The sensors detection range should be modelled with a geometry that is representative of the actual detection range. Information about the actual detection range/volume can be retrieved from manufacturer data sheets and/or experimental verification. The trace-lines are modelled as lines. The sensors body is modelled with a representative geometry. All objects are made as children to the parent object. Once these entities have been modelled the following steps can be executed. (This section illustrates how the process can be performed interactively through menus, these steps can also be done manually)

Step 1



Step 2

With the information from the form the following variables are created (these variables are used by the sensors control program).

Polaroid_1_Value :Real; (The variable is given the type selected)

Polaroid_1_Range :Shape;

Polaroid_1_Trace-Line1: Line;

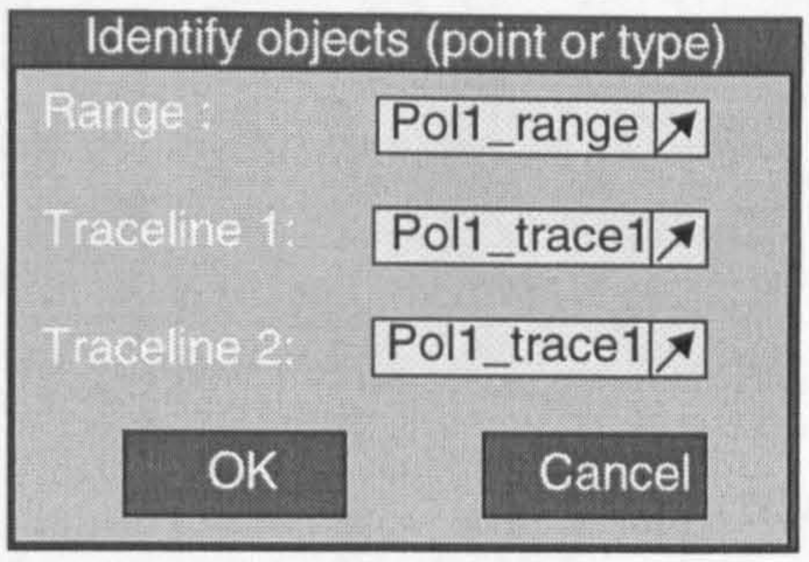
Polaroid_1_Trace-Line2: Line;

Polaroid_1_Update_rate == 0.250;

Polaroid_1_Default_Value == 10.0;

Step 3

The CAD objects that should be linked to the variables created should now be identified. These objects are already created and exists in the graphical environment



Step 4

The objects selected are then linked to the variables

Polaroid_1_Range :=wlkup('POLAROID_1/POL1_RANGE');

Polaroid_1_Trace-Line1:= wlkup('POLAROID_1/POL1_TRACE1');

Polaroid_1_Trace-Line2:= wlkup('POLAROID_1/POL1_TRACE2');

Step 5

Using the information above the appropriate program blocks can be selected and assembled into a control program.

6.2.2 Functional description of the generic sensor model

The implementation of the generic sensor model in the virtual robotics system CimStation is described in this section. Firstly a general description of the implementation is given, followed by a pseudo-code implementation. An example implementation in SIL is given in appendix C.

The *is_hit* function and the *Check if the range is colliding* 'block' checks for intersections between the detection volume object and other objects in the environment. When collisions

are detected, the buffer containing intersected objects is traced, to determine if the range object is colliding with any objects. If the simulated sensor is a range measurement device the surface, of the detected object, which is closest to the sensor, is identified. This part is performed by the *Calculate distance* 'block'. Each trace-lines point of intersection with this surface is calculated. These intersection points, are given in world coordinates and they have to be transposed into a position defined in the trace-lines coordinate frame to enable distance calculations with respect to the sensory devices, as illustrated in figure 15. The distance from the trace-lines origin (which should lie on the sensors origin) to the intersection point is calculated. Equation 3 takes the homogenous transforms describing each intersection point's location in the world coordinate frame and transposes them to the sensor's reference frame. The distances of the intersection points and the trace-lines coordinate frames are then calculated using equation 4. These calculations are performed for each trace-line.

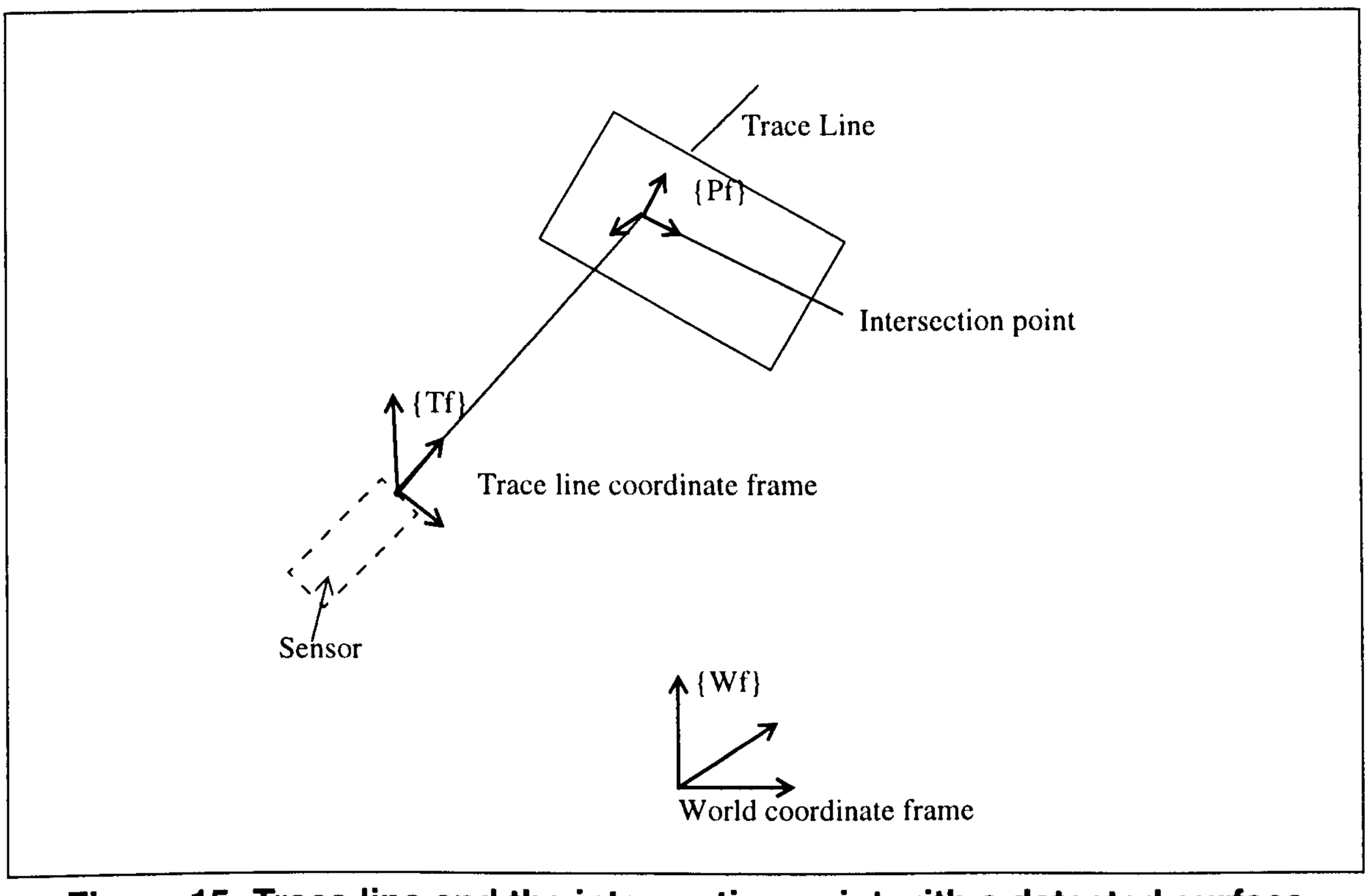


Figure 15: Trace line and the intersection point with a detected surface

$$^{Tf}T_{Pf} = {}^{Wf}T_{Tf}^{-1} \times {}^{Wf}T_{Pf}$$

Equation 3: Coordinate transform for an intersection point to a sensor's coordinate frame

$$\text{Distance} = \sqrt{XC^2 + YC^2 + ZC^2}$$

XC,YC,ZC: The Composants, on respective axes of the trace-lines coordinate frame, of the origin of the intersection point's coordinate frame.

Equation 4: Distance calculation to an intersection point

If the angle of a reflecting surface, with respect to the sensor, is critical for the sensor measurements, the angle between the detected surface and the sensor with its trace-lines has to be calculated. This is the case for example with ultrasonic and laser sensors, where the sound or light can be reflected away from the sensor if the angle is too acute. The normal [**N**] to the detected surface at the intersection point must be determined as shown in figure 16. The direction [**D**] of the trace-line with respect to the normal is then calculated. The direction vector is calculated by transformation of the trace-lines homogenous transform described in world coordinates to the homogenous transform of the normal. The generic sensor model calculates the angle (α) using equation 5. The intersection angle is calculated for each trace-line.

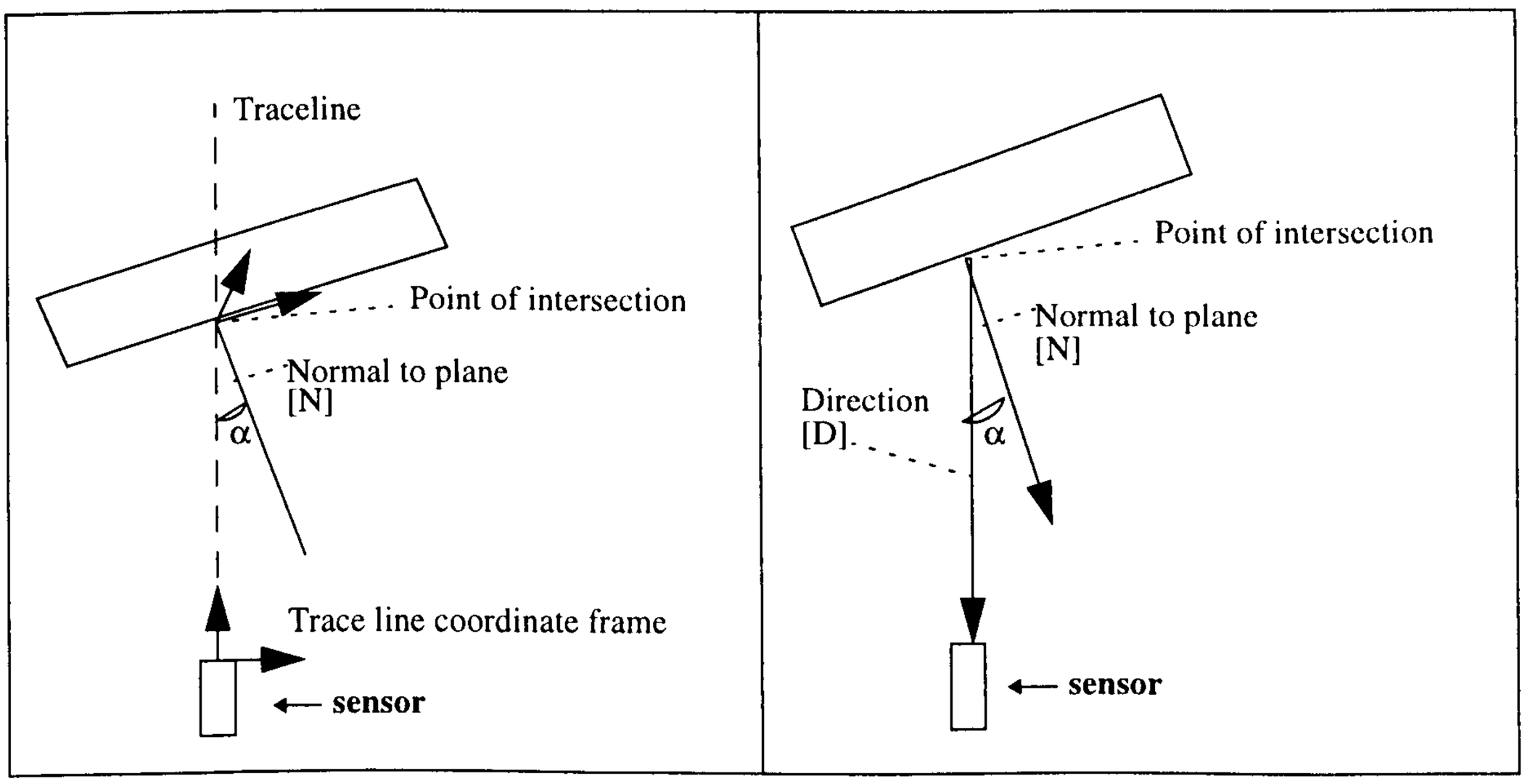


Figure 16: Calculation of the normal at an intersection point and the intersection angle

$$\cos\alpha = \frac{N \times D}{|D| \times |N|} = \frac{DX \times NX + DY \times NY + DZ \times NZ}{\sqrt{DX^2 + DY^2 + DZ^2} + \sqrt{NX^2 + NY^2 + NZ^2}}$$

Equation 5: Calculation of intersection angle between a trace-line and a detected object (see figure 16 for reference)

The complete sensor simulation is performed by a procedure that uses the collision detection and if appropriate the distance and angle calculation functions (Equation 3, 4 and 5).

It passes the distance calculated, to an output transformation function. The output transformation function transforms the measured distance to the output format used by the particular sensor simulated, for example if the sensor gives an 8-bit binary value and the sensor is linear over the measurement range, the distance is converted to an 8-bit value, as in table 4. Depending on the sensor simulated the intersection angle is used differently. For some sensors is it used as a hard thresh-hold value (sensor is detecting /not detecting) while with other sensors it outputs distorted signals between certain reflection angles. Figure 17 shows schematically the working principle of a simulated sensor object.

Table 4: Example of an output format function transforming distance to an 8-bit value

```
/* Min. detection 0 mm */
/* Max. detection 500 mm */
Function OutputValue(distance:real): Integer;
Begin
    OutputValue:=roundoff(255*distance/500)
End;
```

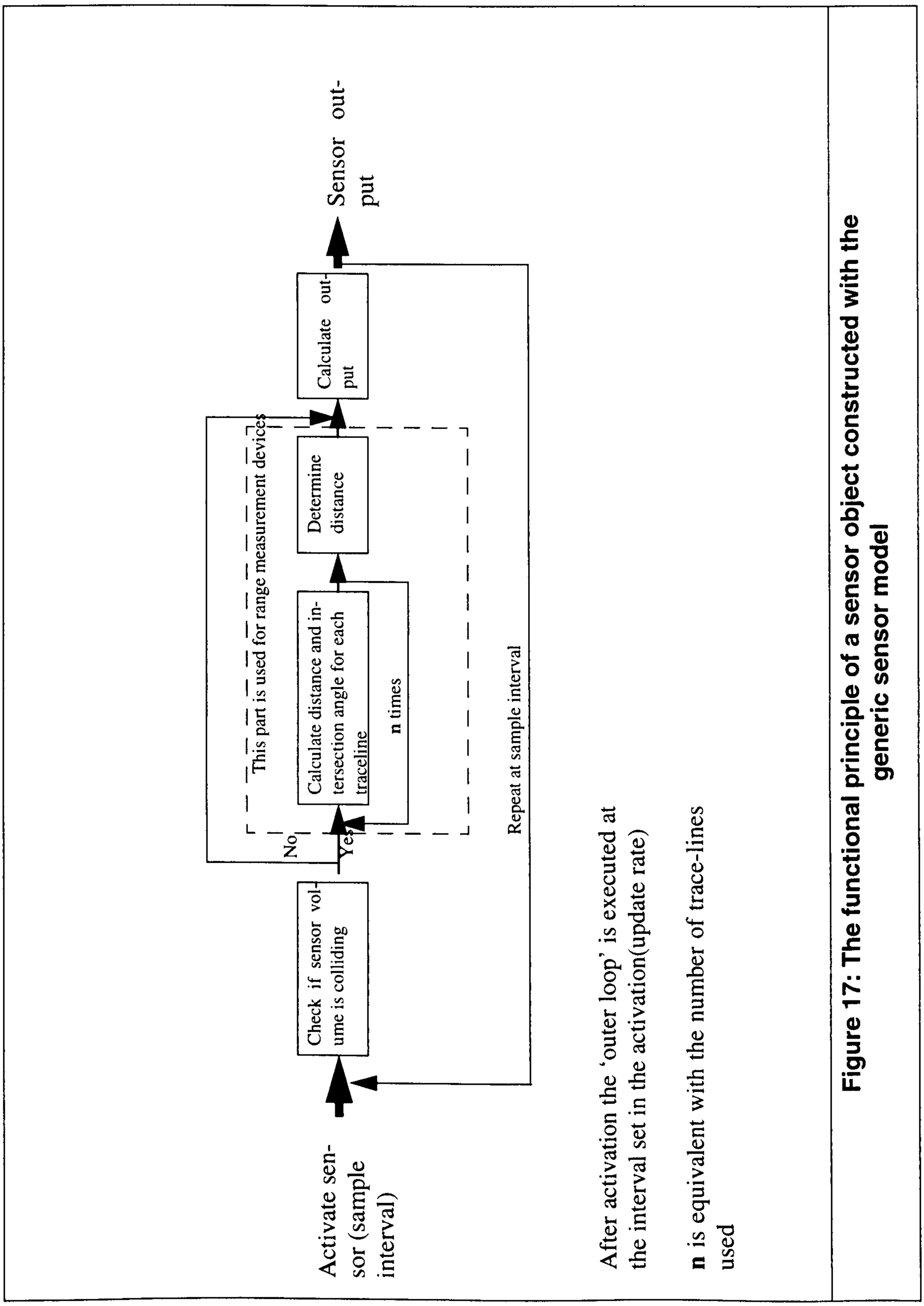


Figure 17: The functional principle of a sensor object constructed with the generic sensor model

The following section describes the implementation of a sensor. The separate 'program' blocks are described and then a report_sensor function using the 'program blocks' and a sample_ sensor for a range sensor are described. The implementation is described with 'pseud-code'. An implementation of a sensor in SIL is presented in Appendix C.

Table 5: 'Program blocks' to be used in the construction of a sensor control program

Check if the range is colliding (range_object)
If any collision **(using the is_hit function)**
For list of colliding objects do
if range in pair of colliding objects then
report range detection

Calculate distance (trace_line). *This block is used by range mesasurement sensors and is duplicated for each trace-line used.*
Check if the trace-line is colliding with the same objects as the range
Trace list of object to get the closest shape
Determine intersection point and transform to sensor coordinates
Calculate the distance to the intersection point for the particular trace-line
Calculate if any reflection **(using the reflect function)**

Calculate minimum distance
Compare distances calculated for the trace-lines
If no collision then report default-value

Table 6: A report sensor function built from 'program blocks'. The function is 'pseudo version' of a report sensor procedure, some parts are compulsory whilst others (in italics) are sensor type dependent.

Function report_sensor(range: shape, defualt_output :real, range_name : string): real
Begin
  Detection:=is_hit();
  If detection then do
    Check if the range is colliding (range_object)
    Calculate distance(trace_line1)
    Calculate distance(trace_line2)
    ....
    Calculate distance(trace_lineN)
    Calculate minimum distance (if proximity set distance to 1.0)
  If Detection then report_sensor:=minimum_distance
  Else Detection:=Default_value;
End

Procedure sensor_sample
Var Output:real
Begin
  Output:= **report_sensor(...)**
  Sensor_value:=**OutPutValue**(Output)
End;

The sensor control procedures are set to be independent processes, which is accomplished using the ticker facility provided in CimStation. The processes are set with a time argument that represents the 'sampling' rate of the actual sensor simulated. This ensures that the sensor will act independently from any associated simulated robot program (see table 7 and table 8). Each sensor process is activated and deactivated separately and independently from other sensors and processes as illustrated in table 9.

Table 7: Sensor control procedure

```
Procedure sample_sonic();
Begin
        DISTANCE:=Report_Sensor(Rangeshape, Traceline1,9999,
        'ULTRA_SONIC/RANGE', 'ULTRA_SONIC/TRACE_LINE1');
        SONIC_DISTANCE:=OutPutValue(DISTANCE);
End
```

Table 8: Making the sensor control procedure an independent process with a sample interval of 350 ms

```
Sample_Intervall==0.350;
Simulate_sonic == mk_ticker(mk_application("sample_sonic,
emptylist(universal)), Sample_Intevall);
```

Table 9: General description of a cell control program which activates sensors and starts robots

```
Process Cell_control();
....
Begin
     Activate(Simulate_Sonic); /* start ultrasonic sensor
     Activate(Simulate_Prox1); /* start proximity sensor.
     Robot_program(); /*start robot program
     .
     .
     .
     DeActivate(Simulate_Sonic);/* stop ultrasonic sensor
     DeActivate(Simulate_Prox1);/* stop proximity sensor
.
End;
```

6.3 Simulation of proximity sensors

The most widely used sensors in robotic manufacturing facilities are proximity devices [Bolmsjö 1992]. These sensory devices give binary outputs when an object is within the sensor's detection region. These sensors work with many different physical principles and include variants based on inductive, capacitive, optical, pneumatic, acoustic and magnetic mechanisms. Proximity devices are typically used to detect objects arriving on conveyors, parts at grip positions in feeders, objects in the gripper etc.

The following experimental set-up was arranged to evaluate the signal response from simulated proximity sensors:

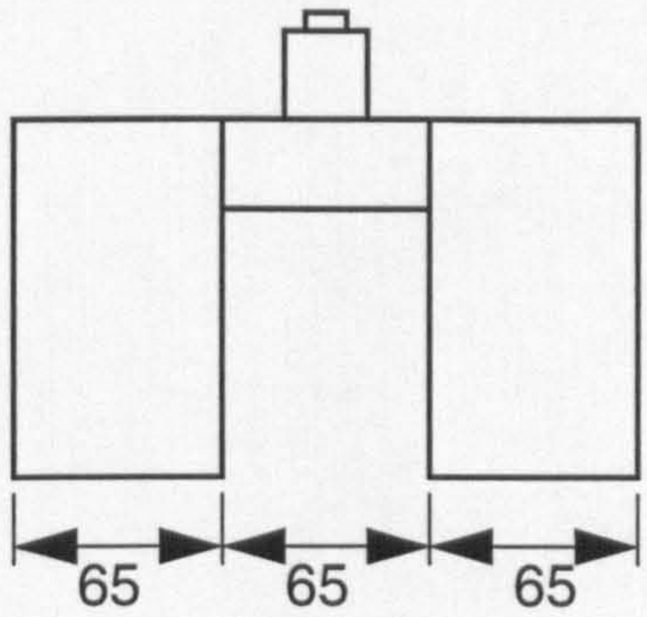
The sensors were mounted on a test rig. A tool with know dimensions (as illustrated in figure 17) was mounted to the end plate of an ABB IRB2000 robot. The robot was moved with known speed along a known path in front of the sensors while the sensors were sampled into a computer. The robot was moved at different speeds and the sensors were sampled at different sampling rates. A virtual workcell for simulation of the experimental workcell was

created in CimStation (as illustrated in figure 19) and the same experiments where conducted in the virtual robot cell. Table 10 shows the control program used for sampling the sensors in the virtual environment. The sensors where activated in the start up sequence of the control program. As the sampling in the real workcell was conducted with a separate computing device, a process which operated separate to the virtual cell's control program was created. This process sampled the virtual sensors and logged the samples to a file, which emulated the sampling conditions in the real workcell. Inductive and photo-electric sensors were evaluated (see next two sections).

The sensors actual detection ranges were measured. The measurements were conducted using a CNC machine-tool. The sensors were mounted on the x-axis of the machine-tool and a calibration tool was mounted in the machine-tool. The calibration tool was moved in front of the sensor devices while monitoring the signals from them. The coordinates were read from the CNC-controller and these were used to calculate the actual detection volumes.

Table 10: SIL program sequence for sampling a virtual sensor

```
Activate(omron_proximity); /* start sensor
Start("Sample_Sensor); /* start sampling program
Start("Robot_Path); /* start robot program
Stop(Sample_Sensor);
Deactive(omron_proximity);
```



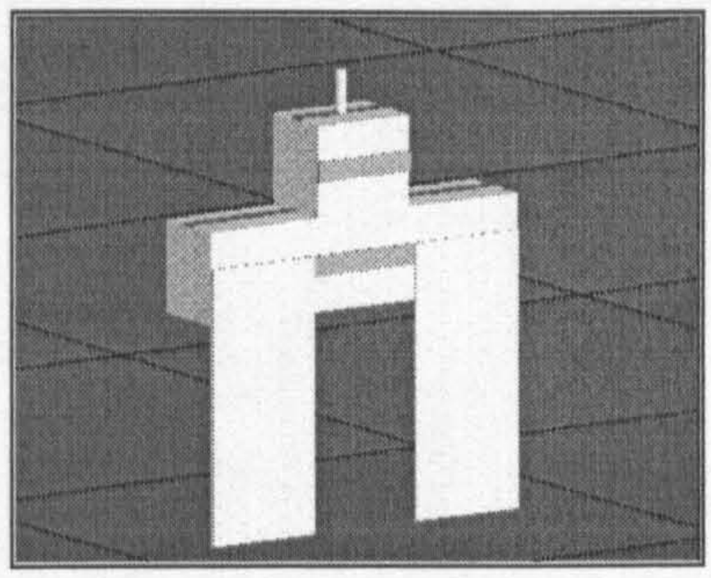


Figure 18: Virtual test tool and dimensions of the component

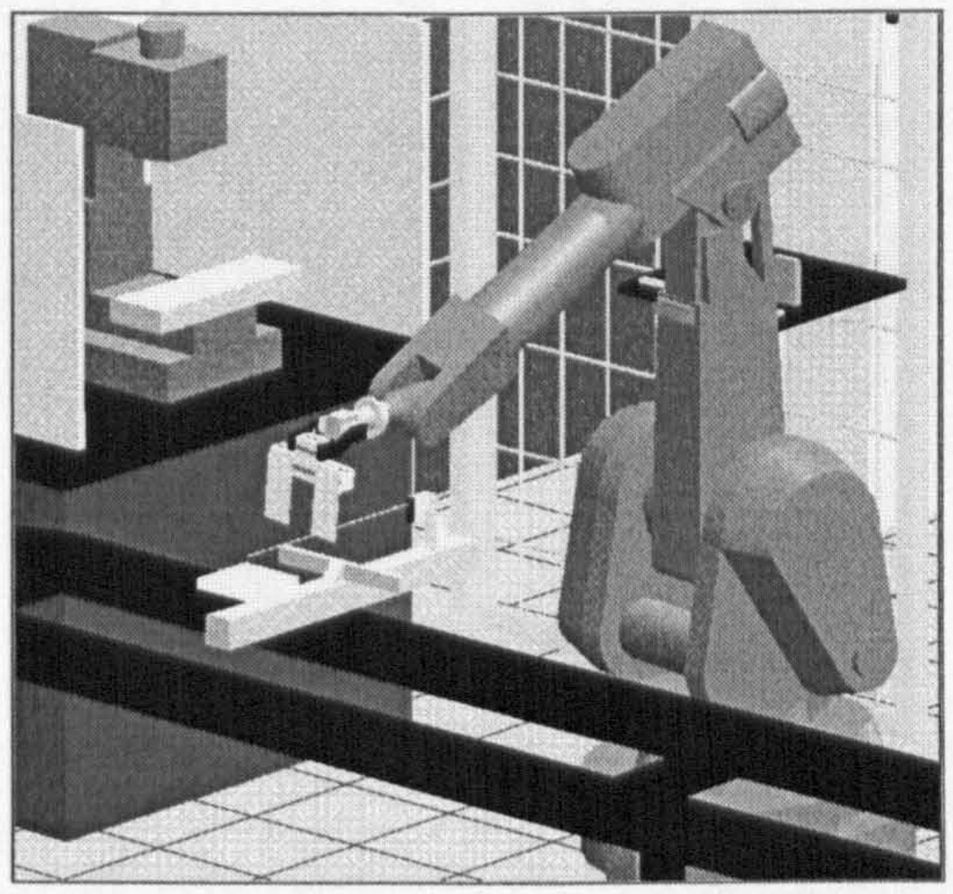


Figure 19: Virtual test set-up for sensor evaluation

6.3.1 Simulation and validation of a virtual photoelectric sensor

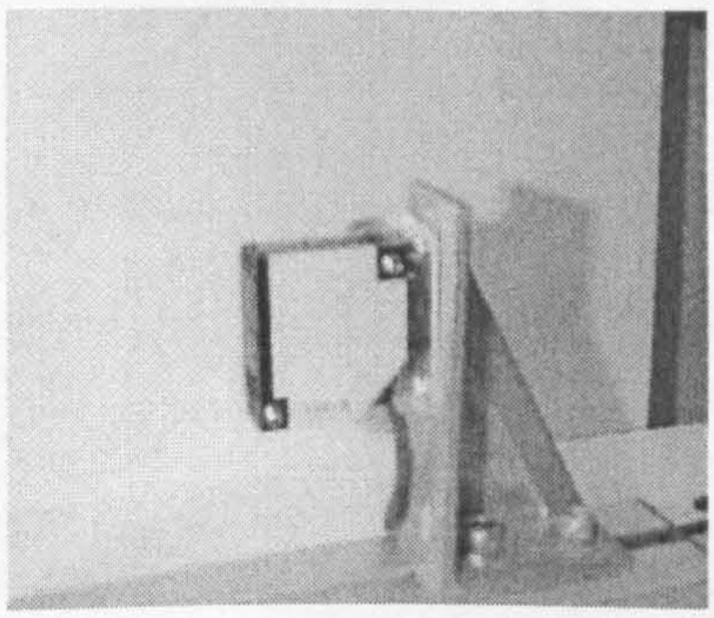
The simulated photoelectric sensor is shown in figure 20. Information about sensor characteristics were taken from the instruction manual provided by the manufacturer [Omron 1], the actual detection range of the real sensor was measured and a cylinder was used to simulate the detection volume of the device. To emulate the output from the real sensor, which is 0.0 V for no detection and 5.0 V for detection, the corresponding simulated outputs were 0.0 and 5.0. Samplings were conducted at three different robot speeds 24mm/s, 42 mm/s and 88mm/s, with a sampling rate of 2 Hz for 24mm/s and 4 Hz for all other speeds. The sampling speed

was set to provide at least 3 detections for each side of the test tool without giving long periods of 'no-detection', which would make comparisons difficult as there would be long intervals of no-detection when the test tool moves away from the sensor. A suitable ratio for speed/sampling was determined after trying several different ratios experimentally. For each setting five samplings were conducted. For each sample the test tool was moved along the path eight times. Figure 21 A-F shows parts of samples from real and virtual sensors. The diagrams show 70 samples taken from the 'middle' of the sampling files. The simulated sensor's correctness was verified by comparing the sets of detection-non detection when the test object passed through the detection range of the sensor. The averages of samples the sets of detection-non detection are shown in table 11. As the difference is less than one sample the virtual sensor is considered to adequately simulate the real sensor. Differences in the samples originates from the relatively slow sampling frequency and higher robot speeds, these differences decreases as the ratio sampling-speed increases. Differences between individual samples can be observed, these differences are similar in both real and virtual samples.

Robot speed Sampling rate	24 mm/s 2 Hz sample	42 mm/s 4 Hz sample	88 mm/s 4Hz sample
Real samples	18.625	18.875	9.25
Virtual samples	18.75	18.375	9.5

Table 11: Comparison of the average number of samples in the sets of detection-non detection when sampling the photoelectric sensor. The average is calculated from one sampling sequence in the real and the virtual environment

It can be observed by comparing samples from real and virtual workcells that there is an increased difference between actual robot speed and simulated speed when the speed set for the robot motion is increased. The virtual robot motion is at nominal speed, whilst the real robot does not reach the nominal speed due to acceleration and deceleration times. These differences can be seen by observing figure 21 E and F. The amount of samples taken while the robot is at an end position differs significantly between samplings made by the real and virtual sampling processes. Here a RRS module, as described in section 3.3.1, should enhance the simulation of the robot motion.



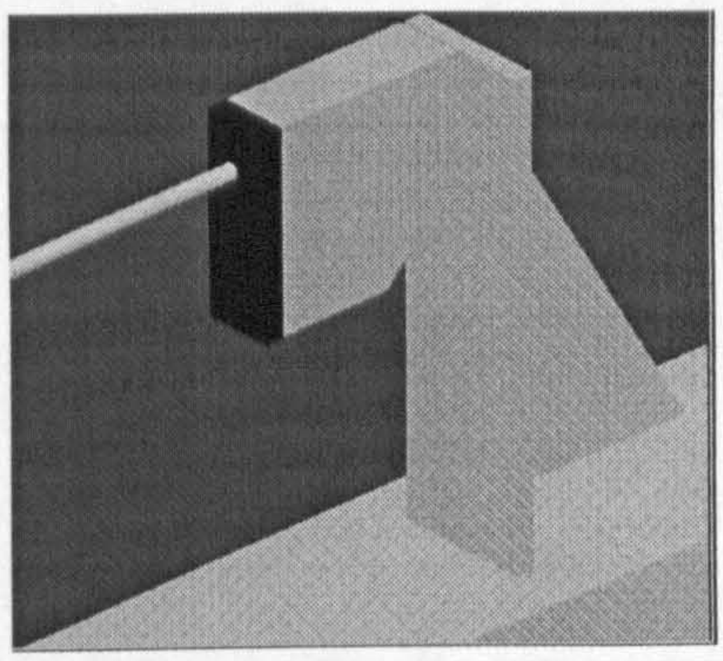
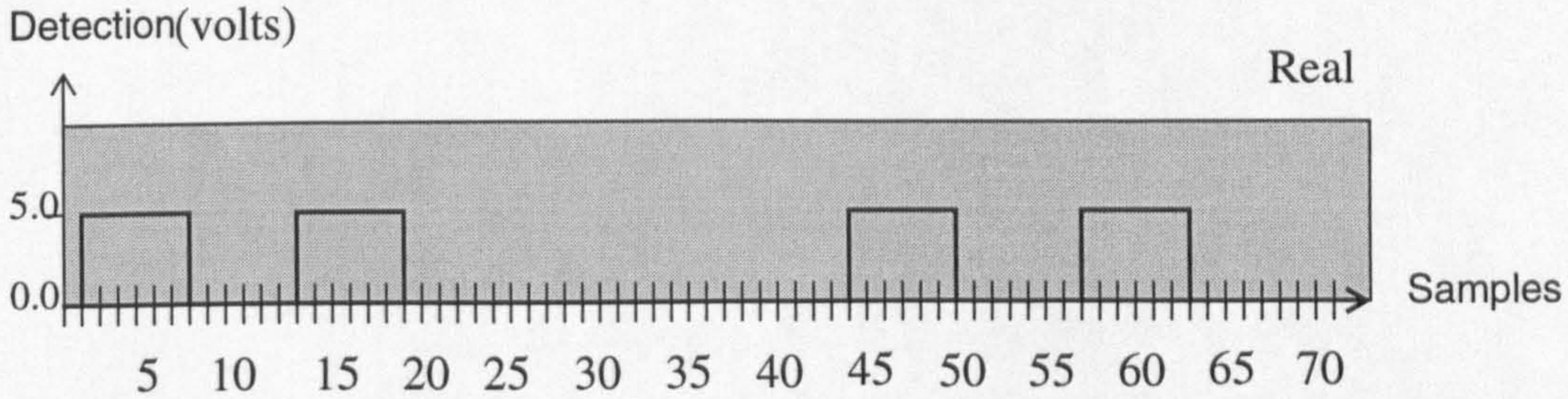
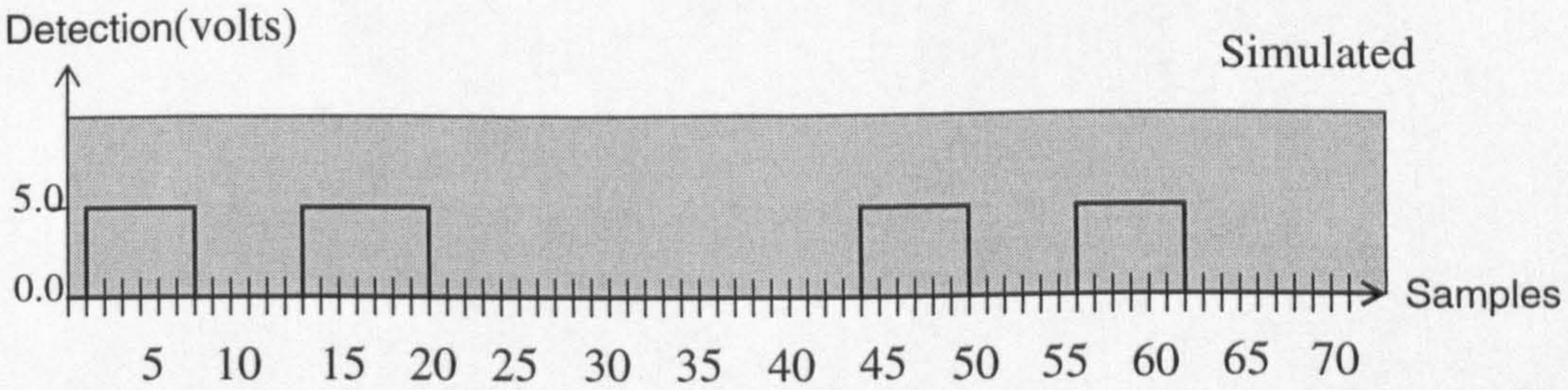


Figure 20: Real and virtual photoelectric sensors

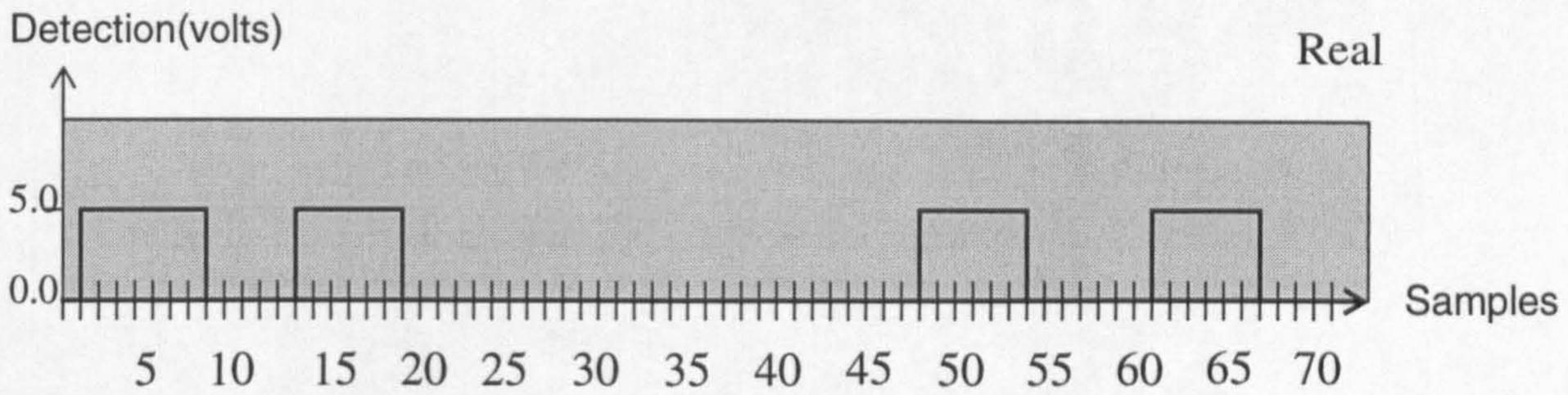


A) Omron photoelectric sensor, robot speed 24mm/s, sampling rate 2Hz.

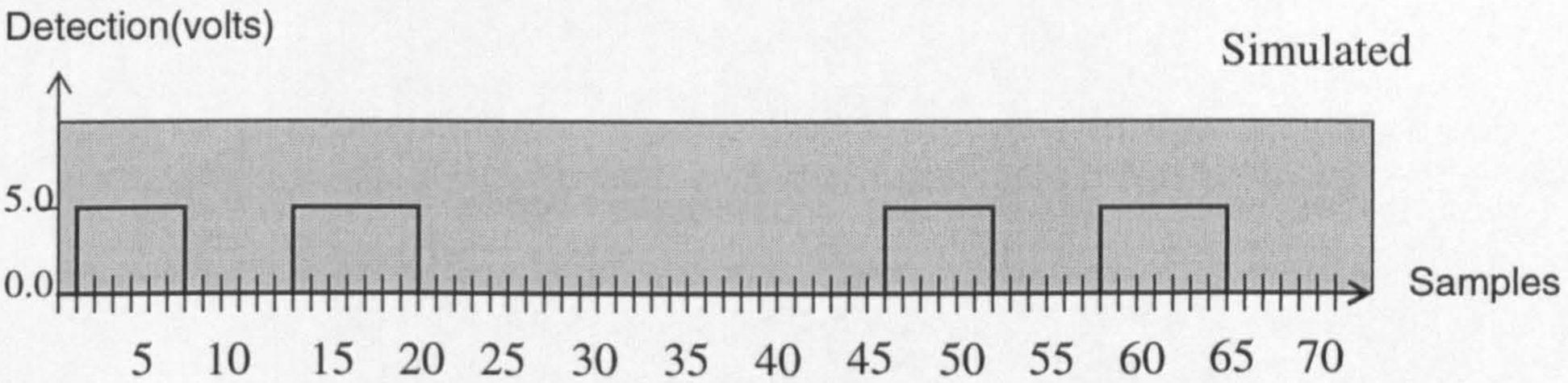


B) Simulated photoelectric sensor, robot speed 24mm/s, sampling rate 2Hz.

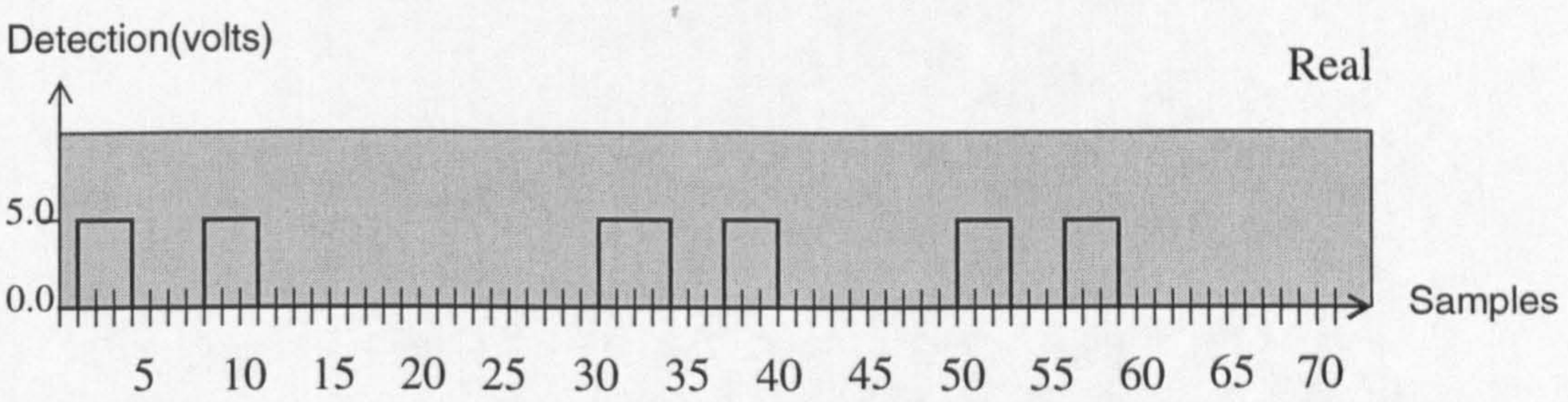
Figure 21: Simulated and real samples of photoelectric sensor



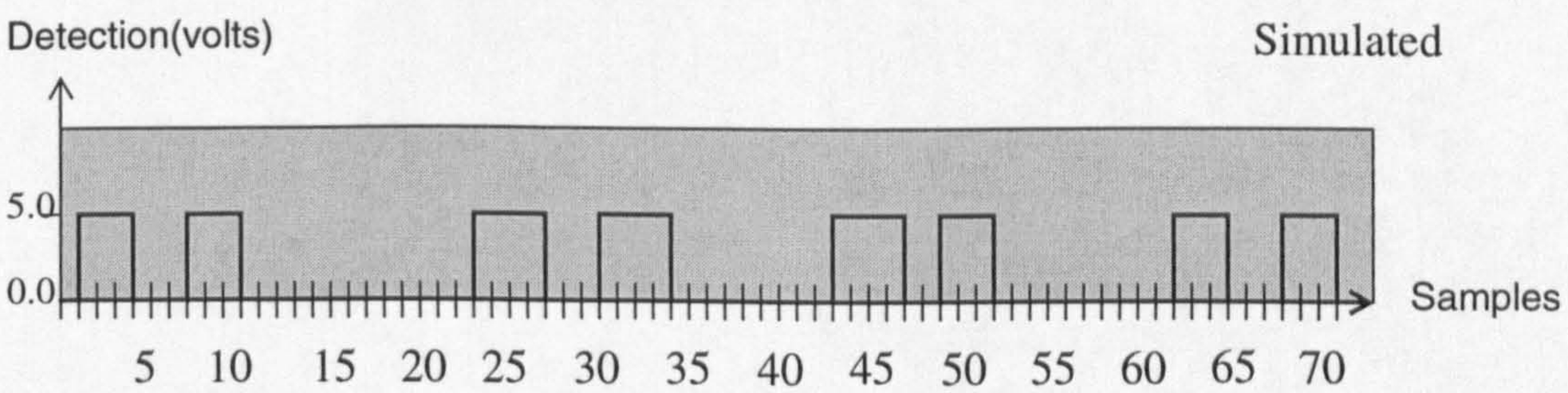
C) Omron photoelectric sensor, robot speed 42mm/s, sampling rate 4Hz.



D) Simulated photoelectric sensor, robot speed 42mm/s, sampling rate 4Hz.



E) Omron photoelectric sensor, robot speed 88mm/s, sampling rate 4Hz.



F) Simulated photoelectric sensor, robot speed 88mm/s, sampling rate 4Hz.

Figure 21: Simulated and real samples of photoelectric sensor

6.3.2 Simulation and validation of a virtual inductive sensor

Inductive sensory devices can be used to detect metallic objects. The real and virtual proximity sensors are shown in figure 22. The sensor is an Omron inductive proximity sensor. Information about sensor characteristics was taken from the data sheets provided by the manufacturer [Omron 2] and the actual detection volume was measured and then simulated as a cylinder. To emulate the outputs of 0.0 V for no detection and 5.0 V for detection from the real sensor the corresponding output from the simulated sensor was 0.0 and 5.0.

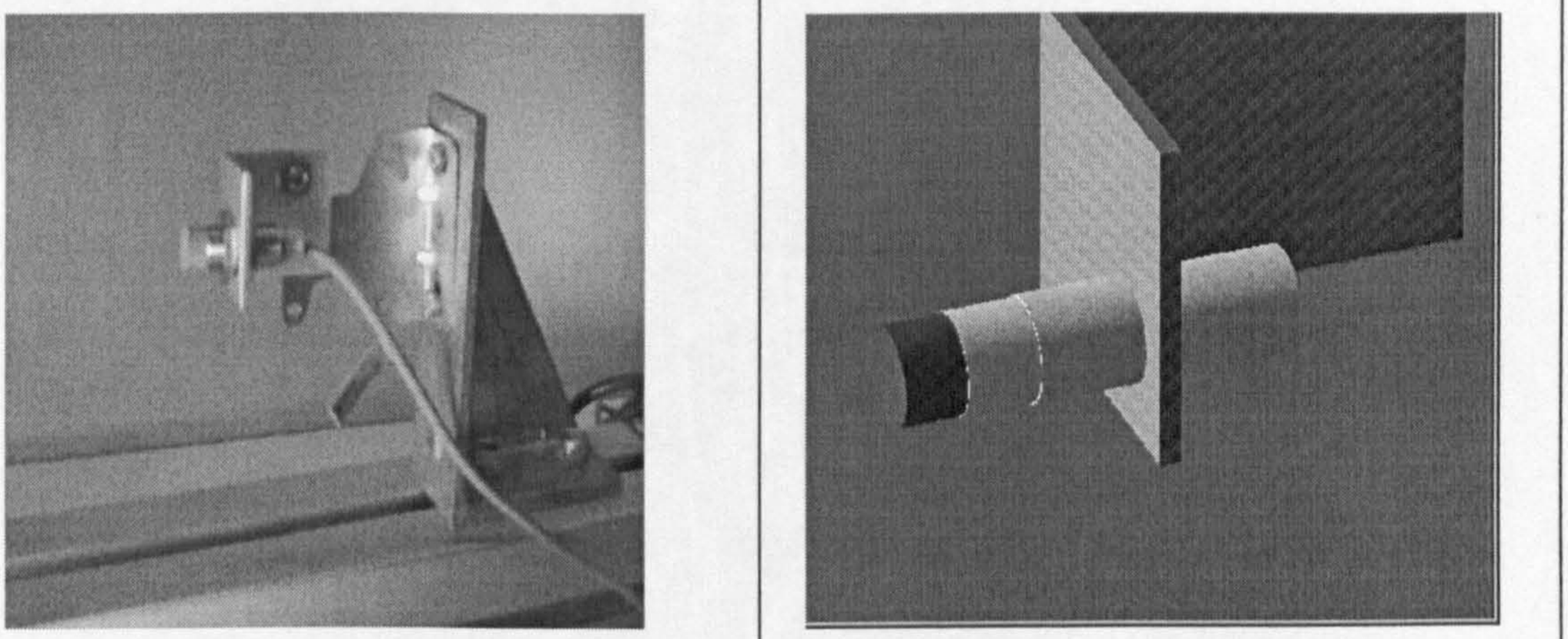


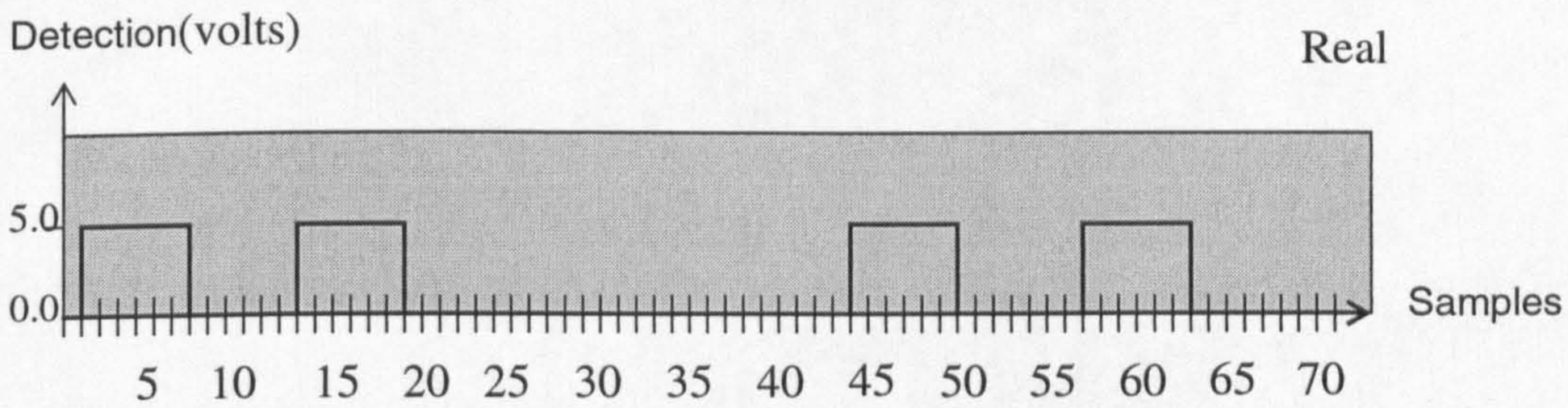
Figure 22: Real and virtual inductive sensors

Samplings were conducted at three different robot speeds 24mm/s, 42 mm/s and 88mm/s with a sampling rate of 2 Hz for 24mm/s and 4 Hz for the other speeds. The sampling speed was set according to the same reasoning as with the photoelectric sensor. For each setting five samples were conducted. For each sample the test tool was moved along the path eight times. Figure 23A-F shows samples from real and virtual sensors. The diagrams shows 70 samples taken in the 'middle' of the sampling files. The simulated sensor's correctness was verified by comparing the sets of detection-non detection when the test object passed through the detection range of the sensor. The average of samples in the sets of detection-non detection are shown in table 12. As the difference is less than one sample the virtual sensor is considered to adequately simulate the real sensor. The virtual inductive sensor satisfactory emulates the real sensor. As with the photoelectric sensor differences between samples can be observed

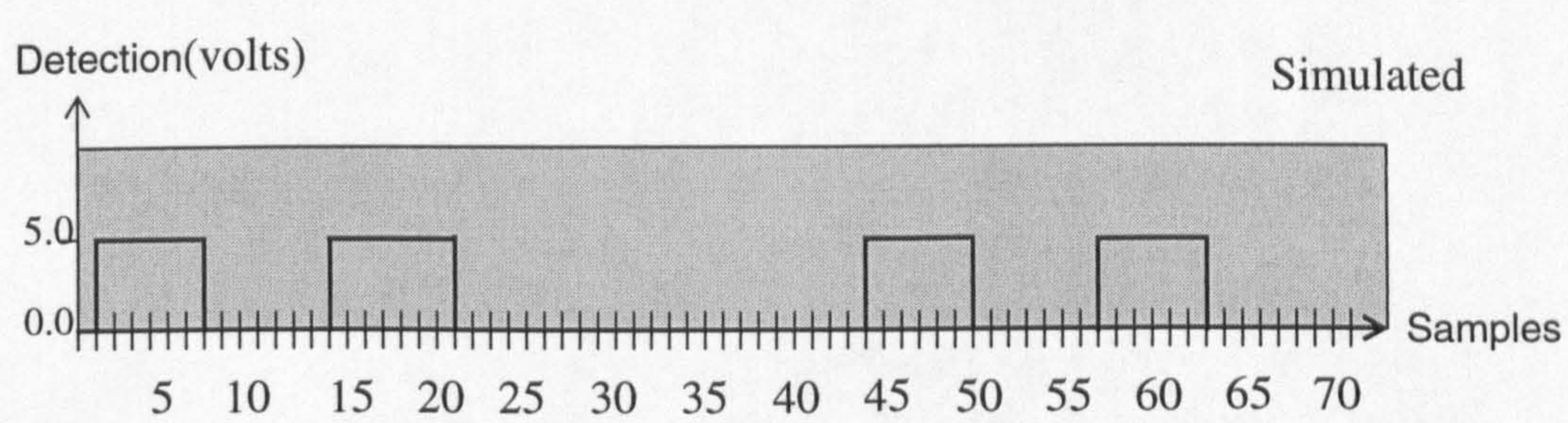
.

Robot speed Sampling rate	24 mm/s 2 Hz sample	42 mm/s 4 Hz sample	88 mm/s 4Hz sample
Real samples	18.5	18.25	9.625
Virtual samples	18.875	18.5	9.625

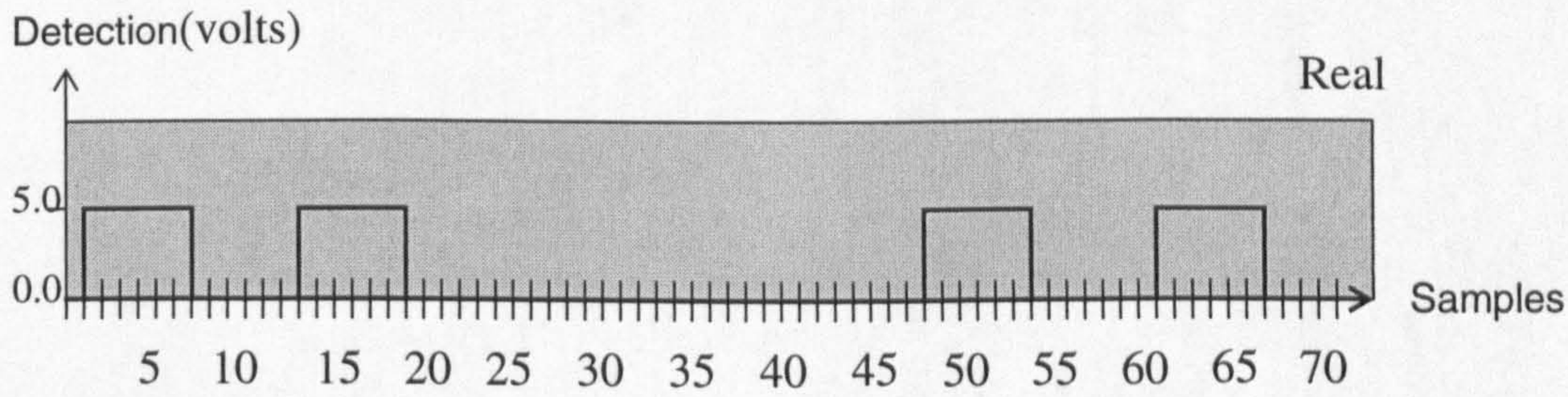
Table 12: Comparison of the average number of samples in the sets of detection-non detection when sampling the inductive sensor. The average is calculated from one sampling sequence in real and virtual environment



A) Omron inductive sensor, robot speed 24mm/s, sampling rate 2Hz.

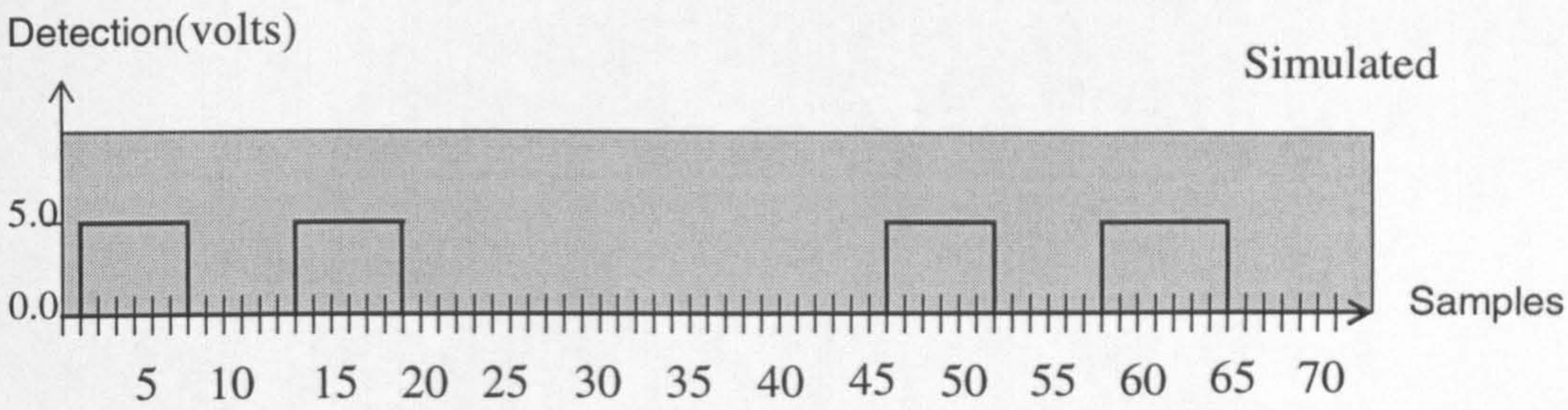


B) Simulated inductive sensor, robot speed 24mm/s, sampling rate 2Hz.

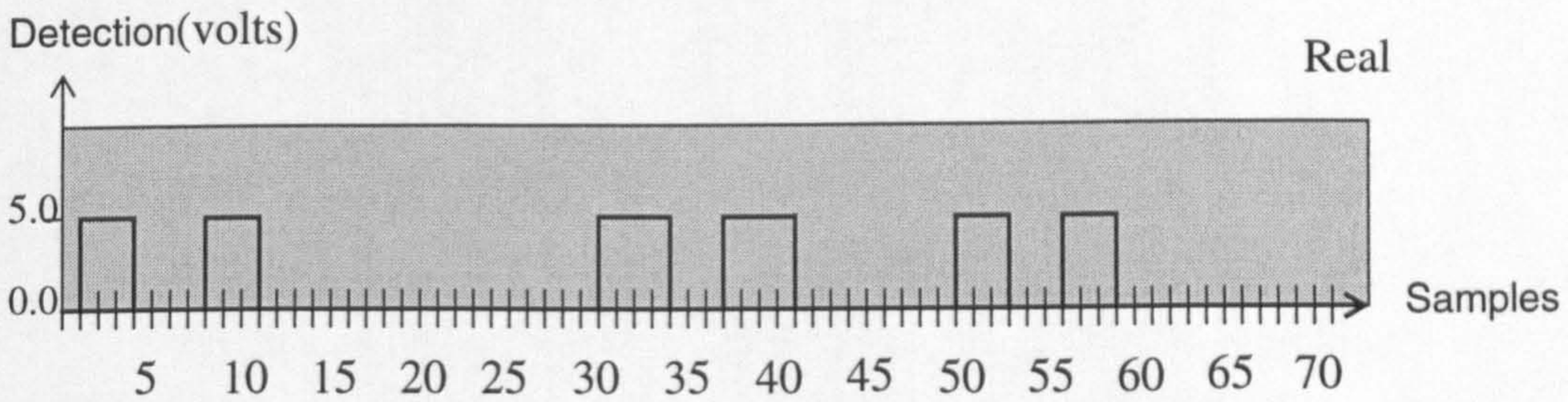


C) Omron inductive sensor, robot speed 42mm/s, sampling rate 4Hz.

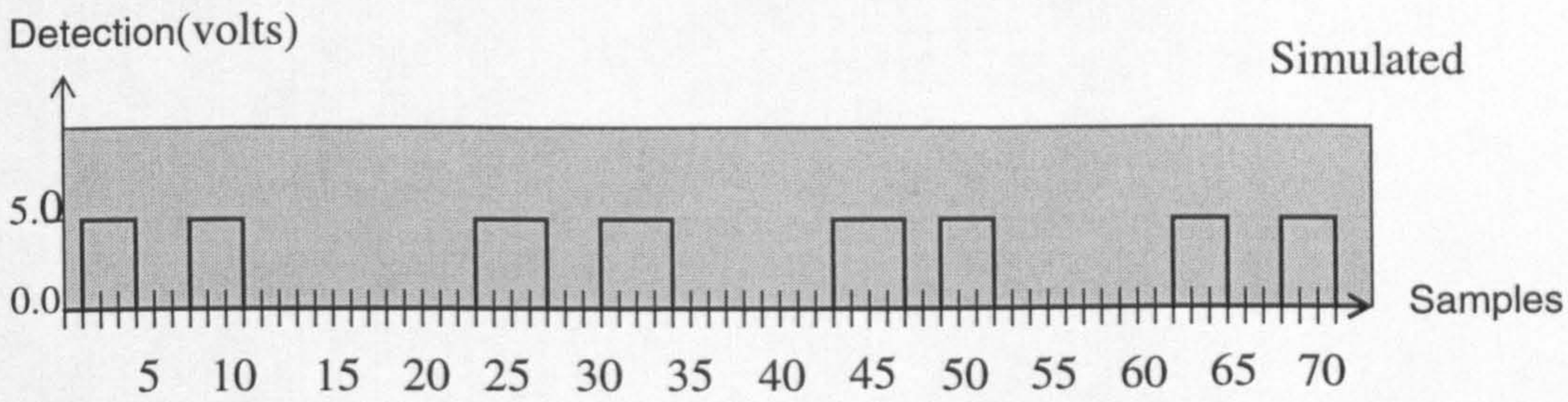
Figure 23: Simulated and real samples of inductive proximity sensor



D) Simulated inductive sensor, robot speed 42mm/s, sampling rate 4Hz.



E) Omron inductive sensor, robot speed 88mm/s, sampling rate 4Hz.



F) Simulated inductive sensor, robot speed 88mm/s, sampling rate 4Hz.

Figure 23: Simulated and real samples of inductive proximity sensor

6.4 Simulation and validation of virtual ultrasonic sensors

The use of ultrasonic sensors in robotics has gained considerable interest in the research community [Blazevic et al. 1991], [Leonard and Durrant-Whyte 1992], [Pomeroy et al. 1986], [Vietze 1992], especially for use with autonomous mobile robots. Most ultrasonic sensors work on the time of flight (TOF) principle, that is, a sound wave is sent out and the time for

the wave to return after reflecting from an object is measured. From the time for the sound to return (echo) the distance to the reflecting object is calculated. Ultrasonic transducers work in a similar manner to the neural sensing system of bats. Bats have been observed being capable of navigating with relatively high accuracy using this principle. This has led to an interest in using ultrasonic transducers for research on robot navigation, as ultrasound is not affected by light conditions etc.

Ultrasonic transducers from Honeywell and Polaroid have been simulated. The sensors were modelled using information provided by the manufacturers [Honeywell 1], [TAG 1] and through direct measurements. The detection beams of the transducers were determined through measurements. The transducers detection volumes were simulated using CAD objects with the shape of a cone, this simplification of the sound 'beam' is sufficient according to [Nnaji 1993]. Figure 24 illustrates the real and virtual Honeywell ultra sonic sensor. The virtual sensor uses a detection cone and five trace-lines to calculate the minimum distance to a detected object. When the detection cone has detected a collision, the distance from each trace line's coordinate frame to its intersection point with the object is calculated. The distances calculated are compared and the shortest distance is used as the output from the sensors. To simulate the characteristics of sound reflecting off objects and away from ultrasonic sensors, the angle between the trace line and the detected surface is calculated. The angle is calculated to decide if there will be reflection back to the transducer or not. The Honeywell ultrasonic sensor has an update rate of 0.350 sec, as such, the simulated sensor control program is set to perform measurements at this rate. The initialisation of the simulated Honeywell ultrasonic sensor is illustrated in table 13.

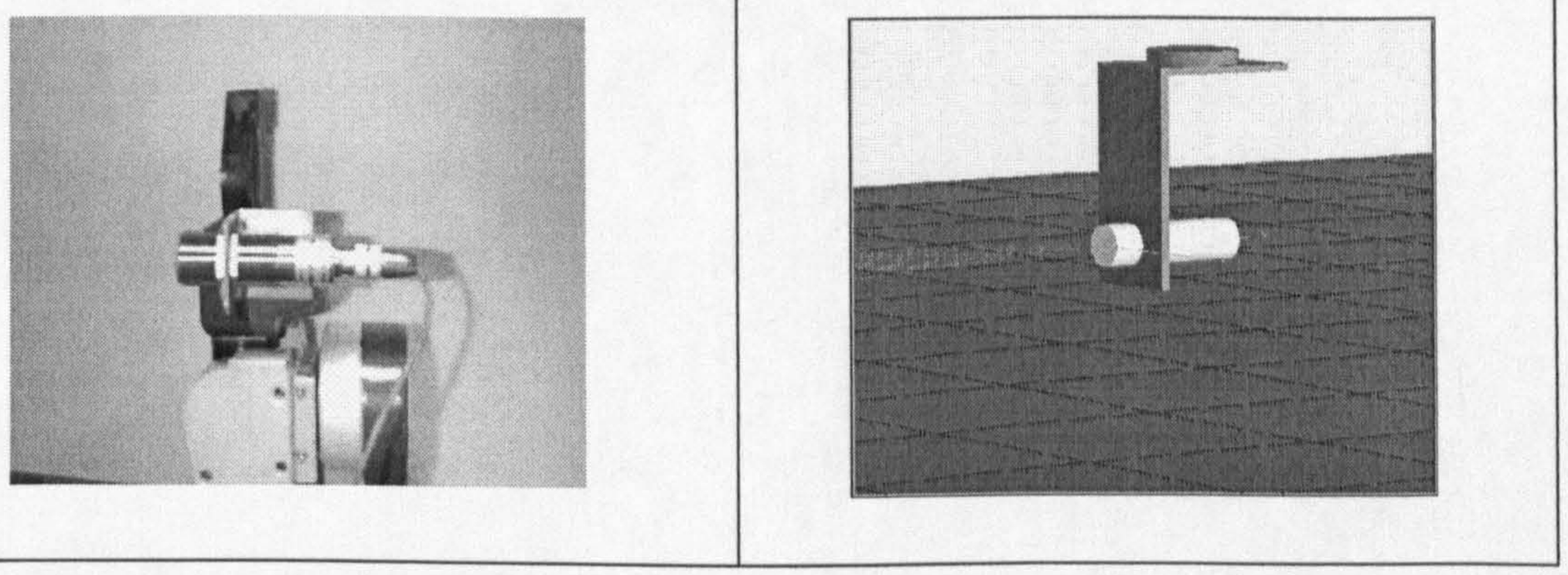


Figure 24: Real and virtual Honeywell ultrasonic transducers

Table 13: Creation and activation of virtual Honeywell ultrasonic transducer

```
Update_rate==0.350; {this is the sample rate}
Honeywell_sonic==mk_ticker(mk_application("report_honeywellsonic,empty
list(universal)),update_rate);

Activate(Honeywell_sonic);
.............
Deactivate(Honeywell_sonic);
```

The following experiments where conducted to evaluate the characteristics of the Honeywell ultrasonic transducer:

The real sensor was mounted on a Coordinate Measurement Machine (CMM). An object, as shown in figure 26, was mounted on the table of the CMM and its exact location was measured. The CMM had the ultrasonic sensor mounted to its head and was moved along a programmed path in front of the object. The sensor signal was sampled by a computer during the motion. The same experimental set-up was modelled in CimStation, see figure 25. A model of the CMM used (a Ferranti CMM 750) had to be created as it was not available in the robot library. Data provided by Ferranti was used for the construction of the virtual machine. The simulated sensor was sampled while moving along the same path in the virtual cell. The comparison of the sampled signals can be seen in figure 27. Differences between real and simulated sensors can be observed when the sensors encounters the corners. In most industrial robotic applications the sensors are mounted to avoid such situations as the feedback from the transducer is uncertain in such situations.

.

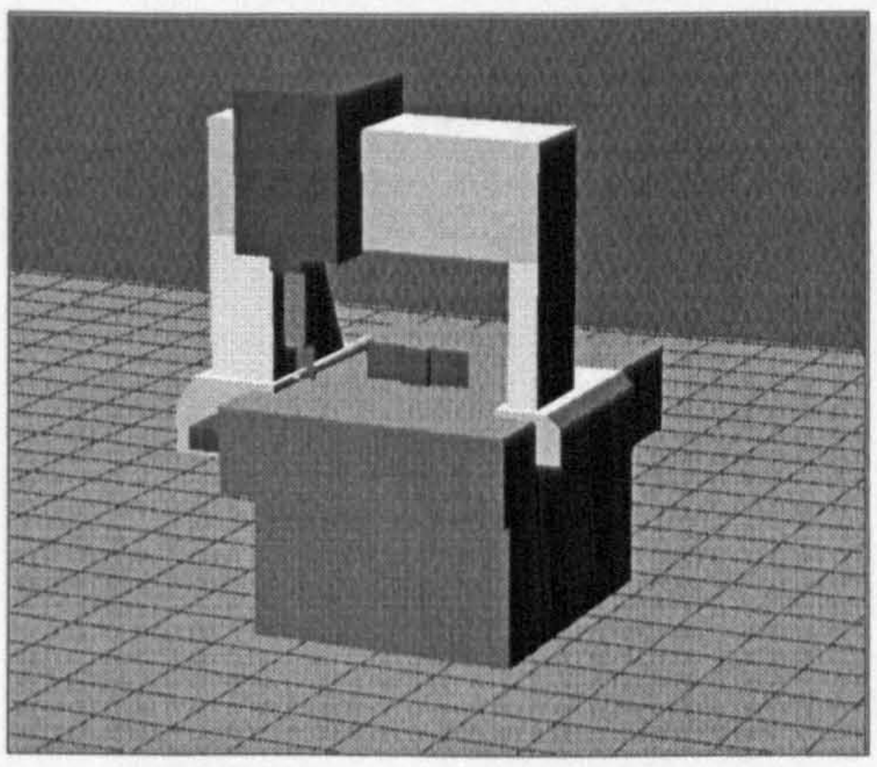


Figure 25: Experimental set-up of a virtual coordinate measuring machine with ultrasonic transducer

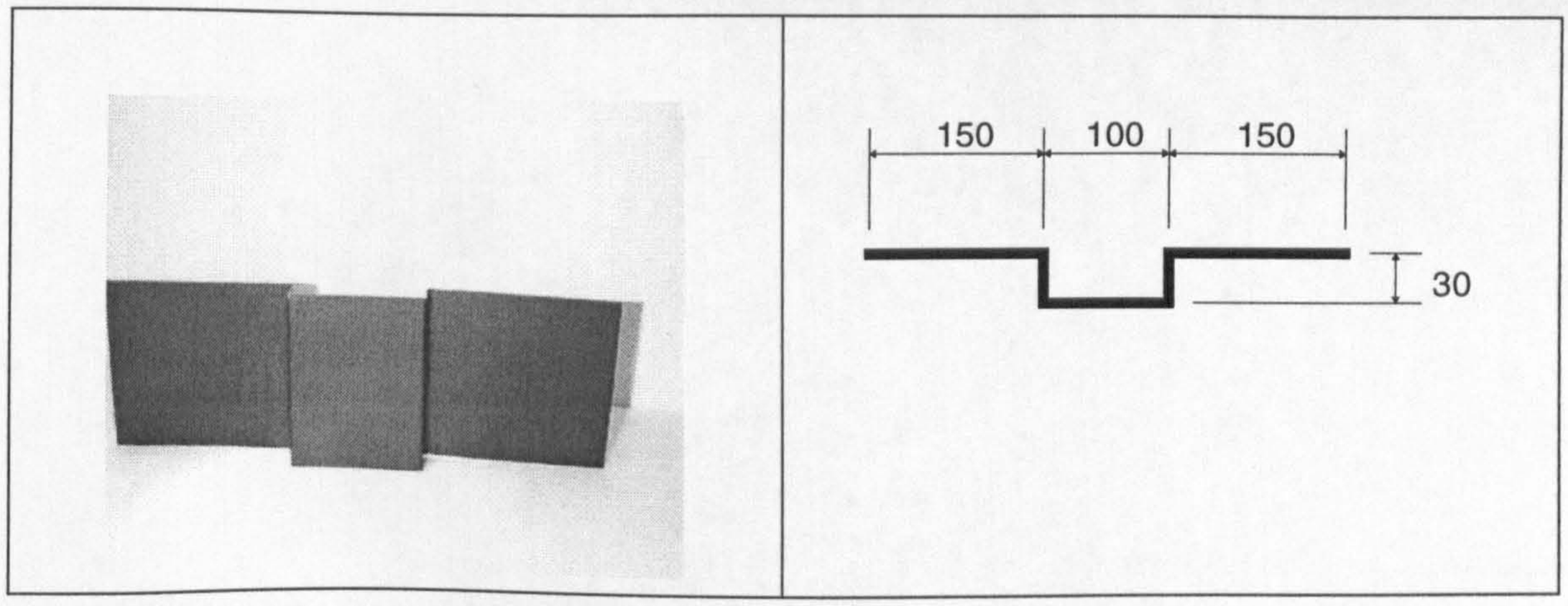


Figure 26: Test object used for evaluation of responses from ultrasonic transducers

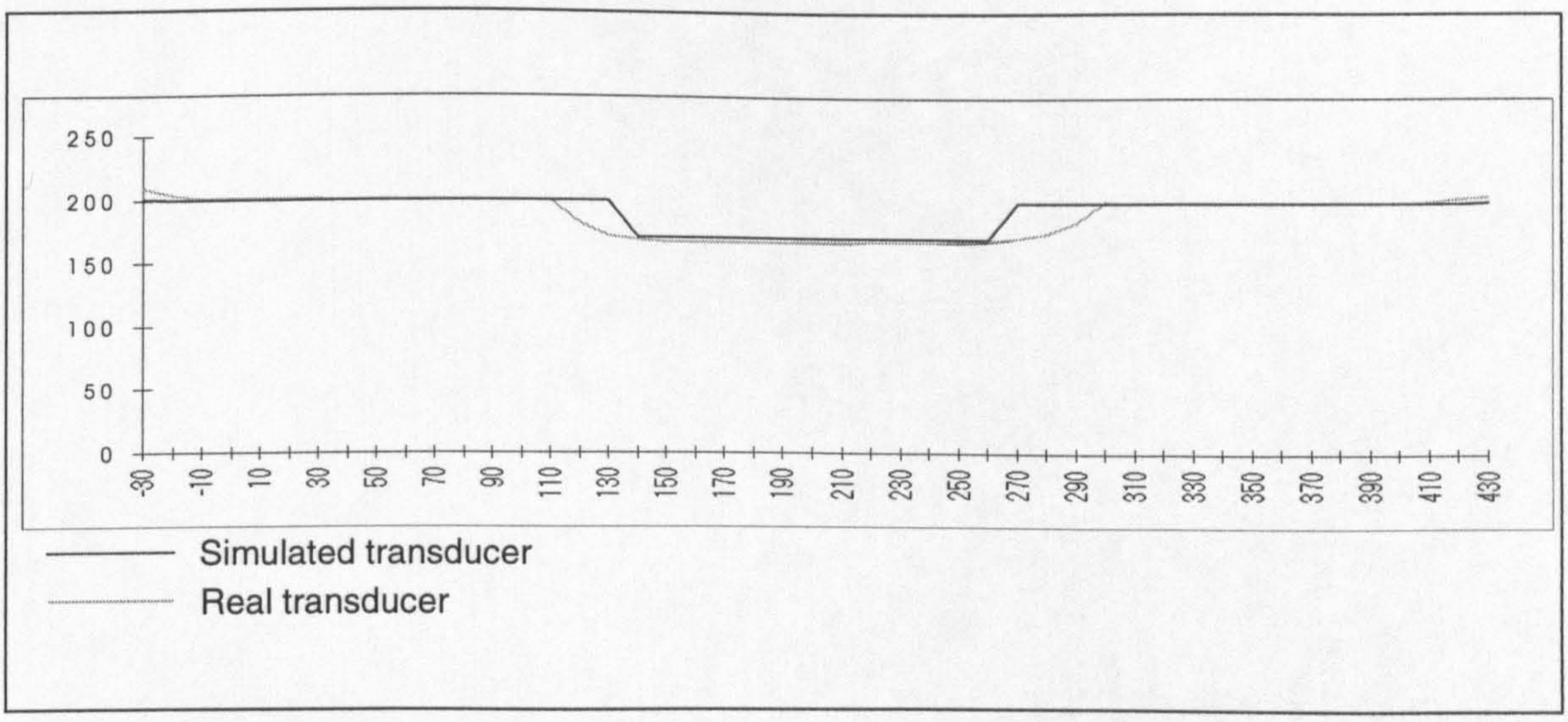


Figure 27: Data from virtual and real ultrasonic sensor contrasted

A simulated Polaroid transducer object, as shown in figure 28, was constructed in a similar way, but with a wider and shorter cone for simulating the detection volume. The output from the Polaroid transducer is a non-linear function giving values between 0 and 10 volt as shown in figure 29. A transfer function (table 14) was used to simulate the output from the transducer. The Polaroid transducer is used in the experiments with the FRANK2 mobile robot described in section 8.2.3.

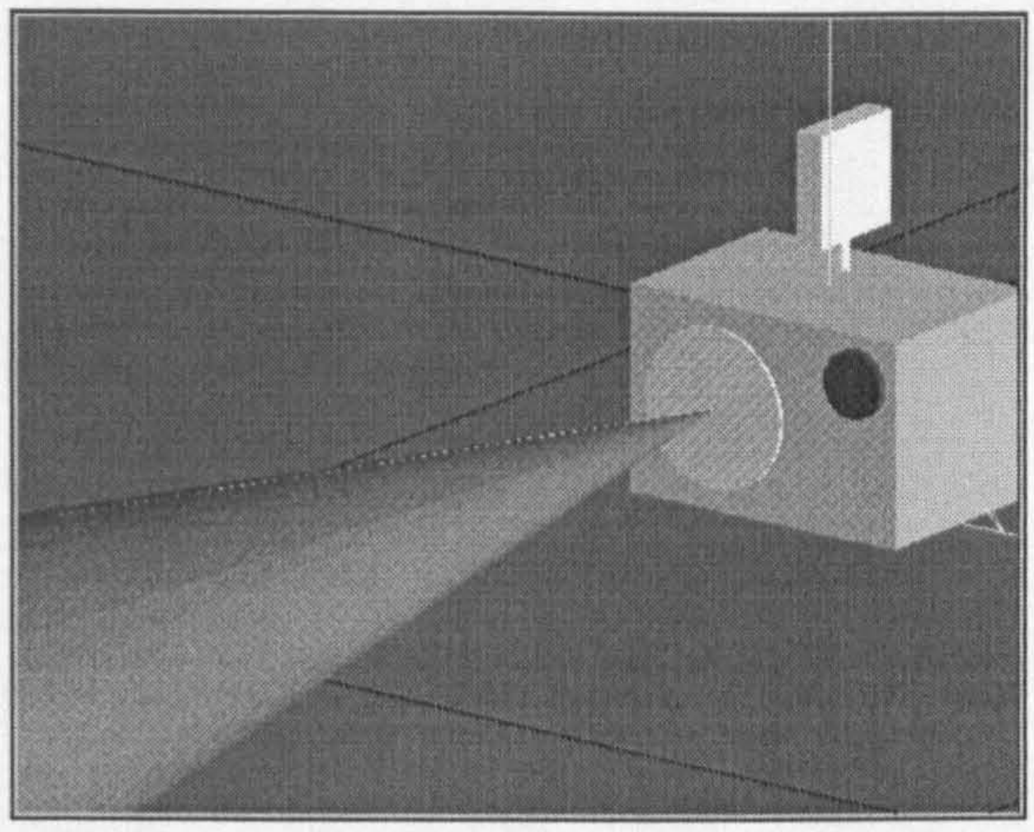


Figure 28: Virtual polaroid ultrasonic sensor

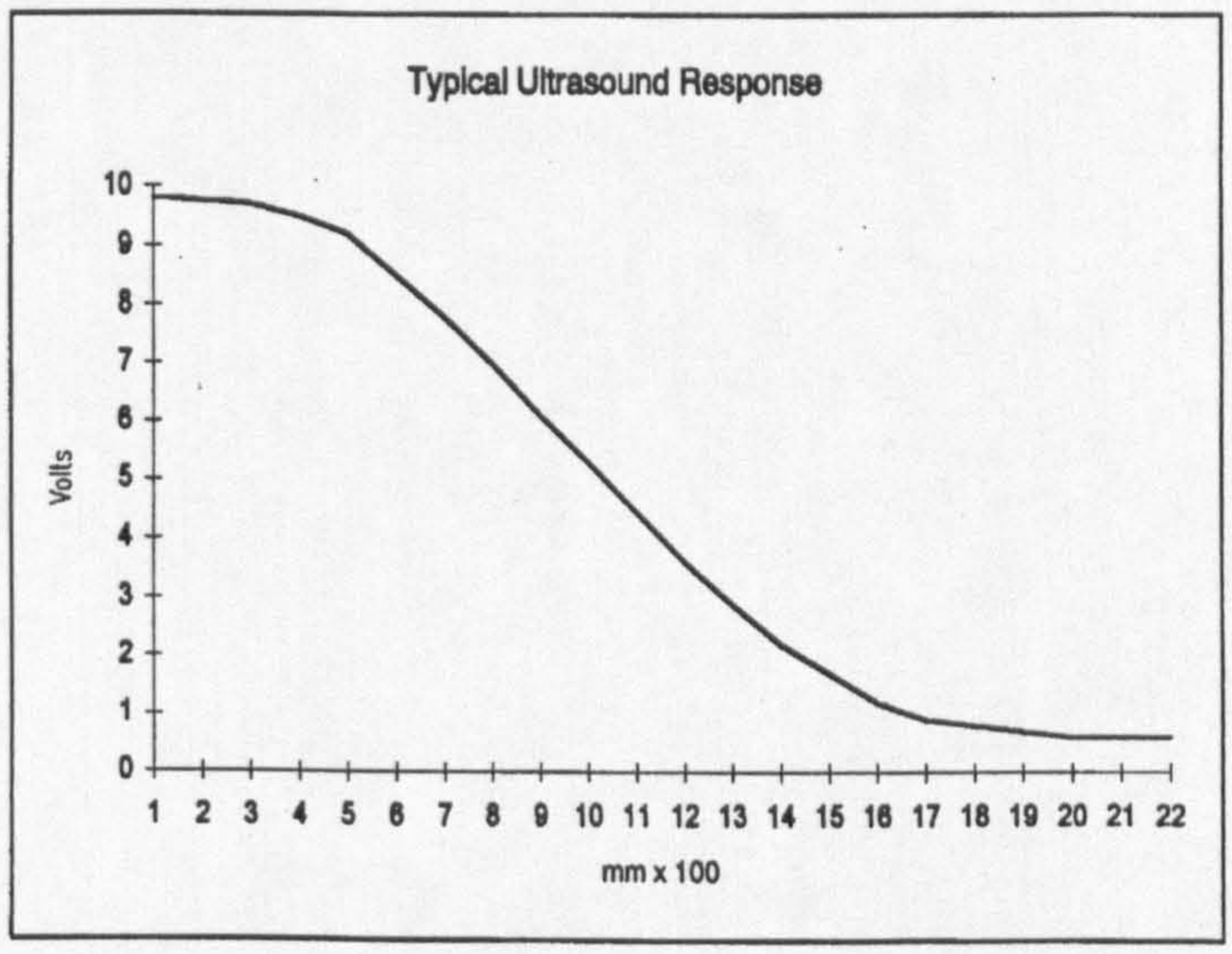


Figure 29: Output from a polaroid ultrasonic sensor pod from TAG,Source: Technology Applications Group. *Sensor interface module*. Alnwick, UK 1992.

Table 14: Output format function for the virtual Honeywell ultrasonic transducer

```
Function Outputformat(distance:real):real;
Begin
   Outputformat:=roundoff(4095)*((2.718**((1-Distance)/200.0)**2))-1));
End;
```

6.5 Summary

A generic sensor model has been suggested that can be used for the simulation of a variety of sensor types. Comparisons between experimental results from simulated and real sensors shows that the characteristics from the simulated sensor sufficiently emulates the characteristics of the sensors needed for normal industrial robot applications. Sensor types such as inductive, photo-electric and ultrasonic transducers have been experimentally validated. The generic sensor model consists of a geometric and a functional description. The geometric description is created using CAD software. Using output format functions the virtual sensors emulate the output format from the real sensors, thus enabling the use of identical robot program instructions for handling sensory information in both virtual and real robots. A generic procedure and template is used for the creation of the functional description. This arrangement provides a quick method for creating new sensor objects and their incorporation into a virtual robot workcell.

6.6 References

[Blazevic et al. 1991] Blazevic P., Delaplace S., Fontaine J.G. and Rabit J., *Mobile robot using ultrasonic sensors: study of a degraded mode*. Robotica 1992, Vol. 9, pp 365-370.

[Bolmsjö 1989] Bolmsjö G., Industriell robotteknik. Studentliteratur 1989, ISBN 91-44-28512-4.

[Honeywell 1] Honeywell Europe. *Data sheet ultrasonic sensor 942-B3A-2D-1C1*. Belgium 1993.

[Leonard and Durrant-Whyte 1992] Leonard J.J. and Durrant-Whyte H.F., *Directed sonar Sensing for Mobile Robot Navigation*. Kluwer 1992, ISBN 0-7923-9242-6.

[Nnaji 1993] Nnaji B.O., Theory of Automatic Robot Assembly and Programming. 1993; ISBN 0-412-39310-7.

[Omron 1] *Type E3JK Photoelectric switch, Instruction manual*. Omron Corporation 1990.

[Omron 2] *Model TL-X-__-__E Proximity Switch, Instruction sheet*. Omron Corporation 1990.

[Pomeroy et al. 1986] Pomeroy S.C., Dixon H.J., Wybrow M.D. and Knight J.A.G., *Ultrasonic Distance Measuring and Imaging Systems for Industrial Robots*. Robot Sensors, pp261-270, Springer Verlag 1986, ISBN 0-948507-02-0.

[TAG 1] Technology Applications Group. *Sensor interface module*. Alnwick, UK 1992.

[Vietze 1992] Vietze L., *Ultrasonic Modelling*. Advances in Control Systems and Signal Processing, Contributions to Autonomous Mobile Systems., Vieweg 1992, ISBN 3-528-06383-1.

Chapter 7 Event-driven Robotics

Industry is being driven by market demands to provide more product variants and to produce direct to customer order rather than to stock. The lifetime of products is getting progressively shorter and the production facilities must be able to easily adjust to new variants and even entire new products without major reconfiguration and reprogramming of the manufacturing equipment. To enable true flexible manufacturing environments to be realised the workcell and robot control should preferably be controlled by events rather than by pre-programmed sequences. The robot sequences to be executed should be determined, for example by the presence of an actual object. Physical events are detected by sensors and the robot program should interact with these sensor events. Examples of event-driven robotics are flexible assembly cells capable of assembling different product variants in the order they arrive from previous production units (or where multiple products are being assembled concurrently). Some car manufacturers, for example Volvo, are now producing to customer order and the production is not scheduled to produce a certain quantity of a specific variant. The cars are produced in the order the customer orders are placed.

These methods for production scheduling creates new demands upon the programming environments and languages used for programming computer controlled production equipment such as robotic workcells etc. IT must be possible to simulate most of the processes, such as machine interaction, sensing etc.as new products and variants must be introduced with a minimal disruption to production. To enable off-line generation of robot tasks (other than pure motion sequences) tools for debugging and analysis of the synchronisation of robot programs and their interaction with other devices are necessary. Simulation and programming of multiple devices working together must be possible.

The objective of this chapter is to present the off-line programming of sensor event-driven robot applications. Further more, the generic sensor model presented in chapter 6, can be used to generate satisfactory models of sensors to be used in off-line creation and debugging of robot programs. This chapter describes how a robot workcell has been programmed to react to sensory information. The programs have been generated off-line, simulated and debugged with a virtual robotics tool. The sensors in the workcell have been simulated using the generic

sensor model described in section 6.2. The programs were then down-loaded to the real cell and, with minor amendments to the syntax, run on the real equipment.

7.1 Simulation of event-driven robotics

This is a short survey of literature presented in the area of graphical simulation and programming of event-driven robot workcells.

Mahajan et al. describe an event simulation tool called QUEST [Mahajan et al. 1993]. This system allows simulation of production sequences. The system has its own programming language (SCL) where control logic can be programmed to be triggered by events. These events are created by other processes, not by sensors. Only the "cell control" program is created and simulated, programs for machines such as robots and machine-tools are not created and simulated. Rather than programming and simulating the behaviour of individual machines, the machines in the simulation environment are given predefined values which are used for the simulation of the process time for each machine. SCL programs used in the simulation can, with a software package (PLCLINK), be translated to PLC code [Lahti 1995]. The translated PLC programs conform to the of IEC1131-3 standard. The translator only translates program logic, any signalling between devices, used in the SCL program, is omitted in the translation. According to [Lahti 1995] "signals work slightly different in QUEST simulation than in real life".

In [Okano et al. 1988] a system for off-line programming of multiple robots and synchronisation of devices is described. The system does not allow any direct signalling between robots. All signals must pass through a simulated PLC, which is used for synchronisation. The system does not allow any simulation of sensors. The system described in [Meijer and Hertzberger 1988] is a system that allows off-line programming of exception handling in robot programs, either letting the programmer provide the system with sensor data manually or using recorded data from real sensors. This system does not cater for the simulation of sensors.

Chung et al. have created a programming and simulation environment for event based robot-served flexible manufacturing cells (FMCPS) [Chung et. al. 1993]. In their approach there is no formal main robot program, the task is divided into sub-processes. Scheduling of the workcell is performed by a dedicated cell controller which is also used to activate the robot

to perform different tasks (called events by Chung et al.), such as load machine, pick objects, etc. Messages are passed between programs using the 'pipe' function available in UNIX. They do not simulate sensors or events detected by sensors, all events are generated by the simulation scheduler. In their approach each robot event is connected to an object's coordinate frame, such as the base coordinate frame of a machine-tool. The robot events are defined relative to the object's coordinate frames and not relative the robot's coordinate frame as this allow greater flexibility in the design phase of the system.

To enable the programmer to be able to generate and debug the overall behaviour of a workcell, the programming environment must allow simulation of sensors to detect environmental changes such as parts coming/not coming, measuring objects in real time, etc. As event orders can not be fully predicted, they must be simulated to enable the debugging of robot sequences and to prevent dead-lock situations caused by event orders not predicted when the program code was first developed. The programming language used must be more general- purpose than traditional industrial programming languages and preferably be object oriented as this should enable re-use of program code and more easily accommodates event-driven robot programs. It is desirable that the simulation environment is transparent to the robot controller, as this allows the off-line generated program to be down loaded without any post-processing [Meijer et al. 1992].

7.2 Experimental event-driven robotic workcell

A robotic workcell was developed, described in section 5.2, to investigate the design and off-line programming of event-driven systems. The experimental workcell is illustrated in figure 9. A procedure and function library was produced within the cell controller PC to provide command and control functions for operating the robot with respect to both motions and auxiliary functions. The cell controller is used to by-pass the robot controller and its integral language. The control system (ACL) of the Eshed Scorbot is normally programmed in a BASIC style language which does not allow great flexibility. The robot programs must be programmed in a dedicated programming environment and the files are not accessible outside this environment, thus preventing the generation of programs with a virtual robotics tool. One of the objectives in developing the new robot Pascal extensions, ScorPas, was to allow the development of sophisticated sensor interaction and evaluation tasks in the robot controller.

The ScorPas procedures and functions use the ACL controller to read the values of the robots internal sensors, such as encoders, and to command each individual actuator. I/O lines are connected to the ACL controller and procedures are implemented in ScorPas to read and set these I/O lines. The control of robot motions and actions is done entirely in ScorPas, allowing development of complex robot programs, exploiting the inherent flexibility of Pascal.

A virtual model of the robot workcell was generated in the virtual robotics system, CimStation as shown in figure 30. The same procedures and functions that drive the real cell are made available for use within the virtual robotics system's programming environment. The syntax of the languages used is almost the same in both the simulation model and the cell controller, which is desirable when creating true event-driven robot programs which do not require much amendment before down-loading from the virtual environment to the real controller. The workcell contains a variety of sensor types (for example inductive, optical and ultrasonic) with which the robot interacts. The sensors detect events which are used to determine the interaction with the robot.

7.2.1 Off-line programming of event based systems

Virtual robotic tools should enable simulation and debugging of the functionality of robot programs capable of handling events and exceptions. There should not be any need for significant changes to the program structure (before down-loading and execution on the actual robot). If the programs need significant amendment before down-loading the operational functionality of the program can not be trusted. The simulation environment should preferably be transparent to the robot controller and programs should be possible to down-load without any further processing. This allows the program, once sufficiently tested in the simulation, to be down-loaded to the actual robot controller and executed on-line.[Meijer et al. 1992].

A procedure and function library was created for the Pascal programming language for controlling an Eshed Scorbot III. The use of a "standard" high-level language allows complex program logic to be readily implemented and is more suitable for event-driven robot tasks than traditional RPLs, typically supplied by robot vendors. It is worth noting the efforts towards a 'standardised' language (IRL), for robot programming made in Germany. IRL was

designed as a general purpose language with the addition of commands, types and procedures specific for the domain of robot programming and control. IRL was designed with a syntax similar to PASCAL (this was discussed in detail in section 2.2). Table 13 and 14 illustrate explicit robot procedures that have been developed for use in controlling the Eshed Scorbot 3 robot from a PC using the Pascal programming language. Programs for event based systems must include 'reasoning capabilities' in order to be able to handle complex event sequences. This 'reasoning capability' is implemented through the use of logic statements and variables containing environmental information. In the Pascal programming language the 'reasoning capability' can be implemented through statements such as: while-do; if-then-else; case-of etc., and with the use of variables which are assigned information such as the status of external sensory data etc.

Table 15: ScorPas Motion Commands

```
procedure MoveXyz(x,y,z,pitch,roll: real);
{ Move robot to specified position by giving x, y and z in mm, pitch and
roll in degrees.}

procedure MoveX(offset: real);
{ Move robot in x-direction by specified offset in mm.}

procedure MoveY(offset: real);
{ Move robot in y-direction by specified offset in mm.}

procedure MoveZ(offset: real);
{ Move robot in z-direction by specified offset in mm.}

procedure Speed(speed: byte);
{ Set speed value of all motors at specified speed (speed: 1 to 9).}

procedure StopMotor(axisNum: byte);

procedure GoPuls(steps: integerTypeArray);
{ Move robot to position by specified pulses (array 1-5).}
```

Table 16: ScorPas auxillary commands

```
procedure Gripper(status: boolean);
{ Operates gripper to open (status = gOpen) or close (status = gClose.}

procedure SetOutputNo(outputNum: byte);

function InputStatus(inputNum: byte):boolean;
{ Check if specified input(inputNum: 1 to 8) is on. If so Inputstatus is
true.}

function GetInputValue:integer;
{ Reads ACL inputs 1 to 8 as binary code with LSB as input number one.
  Returned value in GetInputValue as a decimal value.}

function WaitInterrupt(intCode: string; intTime: word):boolean;
{ Wait for specified interrupt code or when specified time runs out.
inTime specified in ticks (18.2 ticks = 1 second).
```

7.3 Programming and simulation

The virtual robotics system CimStation has been used for off-line generation, simulation and evaluation of programs for the event-driven robot workcell described in section 5.2. CimStation has been extended with robot programming procedures and functions having the same syntax as the procedures and functions developed in ScorPas. The procedures and functions have been implemented in CimStation using the SIL programming language. Internal CimStation commands for programming robot motions, gripper operation etc. are "hidden" in the procedures and functions. The procedure and function calls in SIL have the same syntax as the corresponding ScorPas commands. As SIL has a syntax similar to Pascal and is object oriented, it is possible to create the same programming facilities and constraints in the virtual environment as experienced in the real robot workcell. Equipment in real workcells are typically controlled by individual controllers working "independently" from each other. This equipment normally communicates only at discrete points. To be able to simulate event-driven robot tasks, is it therefore necessary for the simulation environment to provide multi-process/task simulation capabilities with realistic timings of events and motions.

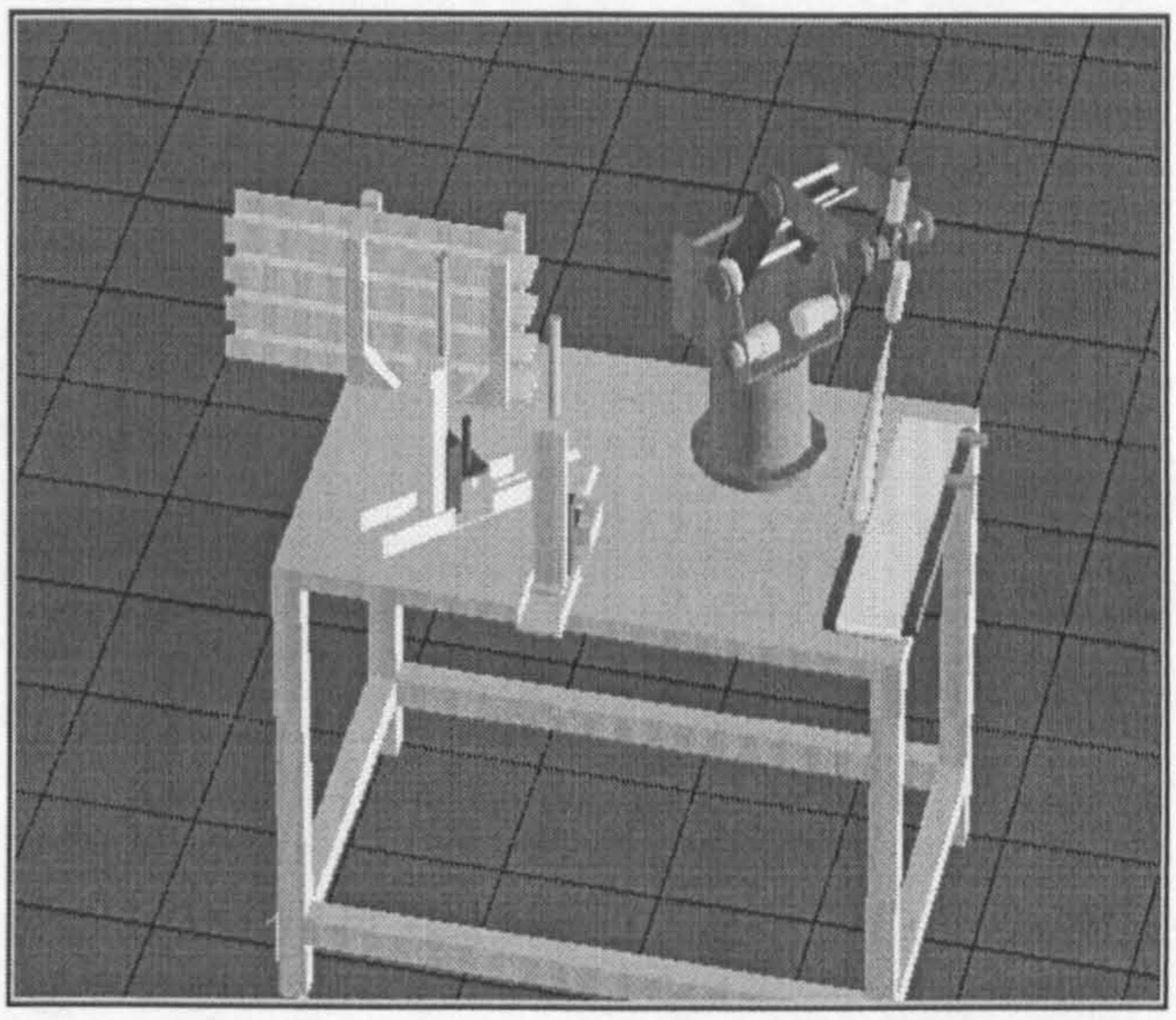


Figure 30: Virtual event-driven robot cell

7.3.1 Experiments: An event-driven workcell task to be off-line programmed

To investigate the possibility of using a virtual robotics tool for off-line programming and debugging of robot programs for use in event-driven robotic workcells (which includes events that are detected by sensors and events that are generated by other machines) the following experiments were conducted:

A task for the event-driven workcell was defined. The programs for the robot and other equipment were generated using CimStation extended with the ScorPas procedures and functions. The virtual sensors were created using the generic sensor model. The robot programs were generated and after debugging transferred and run on the real workcell for evaluation.

Definition of task

Figure 30 illustrates the configuration of the test workcell. Parts arrive at random on the conveyor belt. The main objective is to ensure that the machines are processing parts if parts are available. The simulation consists of four different programs which run concurrently, plus the control programs for the virtual sensors. One program for the robot; one program for the conveyor belt and one program for each machine-tool. The aim for each program is to accomplish a defined task, as described below.

The conveyor belt control program: The conveyor belt program moves parts along the belt towards the sensors. The conveyor belt is monitored by two proximity sensors (sensor 1 and sensor 2 in figure 30). The conveyor belt continuously feeds parts until sensor 2 is activated. Sensor 1 gives a signal to the robot when a part is approaching on the conveyor. The virtual sensors were modelled using the generic sensor model. Each sensor has an associated "behaviour control" program. The sensors have to be "initialised" by activating their control programs, before the cell can start to operate.

The robot control program: The robot control program monitors the sensors on the conveyor belt to ensure that parts are picked up as quickly as possible and loaded into an available machine. When the sensor detects an incoming part it gives a signal to the robot to move to a waiting position and then the pick up of the part can proceed when the sensor detects the part at the pick position (determined by sensor2). The part is then loaded to an available machine. When there are no parts on the conveyor, the robot monitors the machines. It then unloads a machine as soon as it has finished machining a part. Loading a machine has a higher priority than to unloading a machine. The robot program retains the status of the machines.

The machine-tools control programs: The machine-tools are 3-axis machines, with two translation and one rotation axis. The machine-tool begins machining upon receiving a signal from an input line from the robot. When the machine has finished its task, it signals on an output line to the robot that it is ready. The control program for each virtual machine-tool is a SIL process. Each machine-tool is modelled in CimStation. The SIL process actuates the machine-tool, controls the process time of the machine-tool and controls the I/O functions.

The workcell task described is an example of a robot program where it is difficult to identify all event sequences and logical states. These types of tasks can generate deadlock situations in their execution. The robot can take unpredictable actions as sensors can generate unforeseen states. The amount of states of the workcell have to be taken into account and the complexity of the cell increases exponentially as sensors and event-driven robot procedures are added. The relatively simple task described here needed several programming, simulation and debugging cycles before satisfactory completion could be achieved. The time needed to complete a successful on-line generated program would be much longer as there is a need to physically reset the workcell before each test sequence. Table 15 to 17 show a flowchart of

the robot control program.

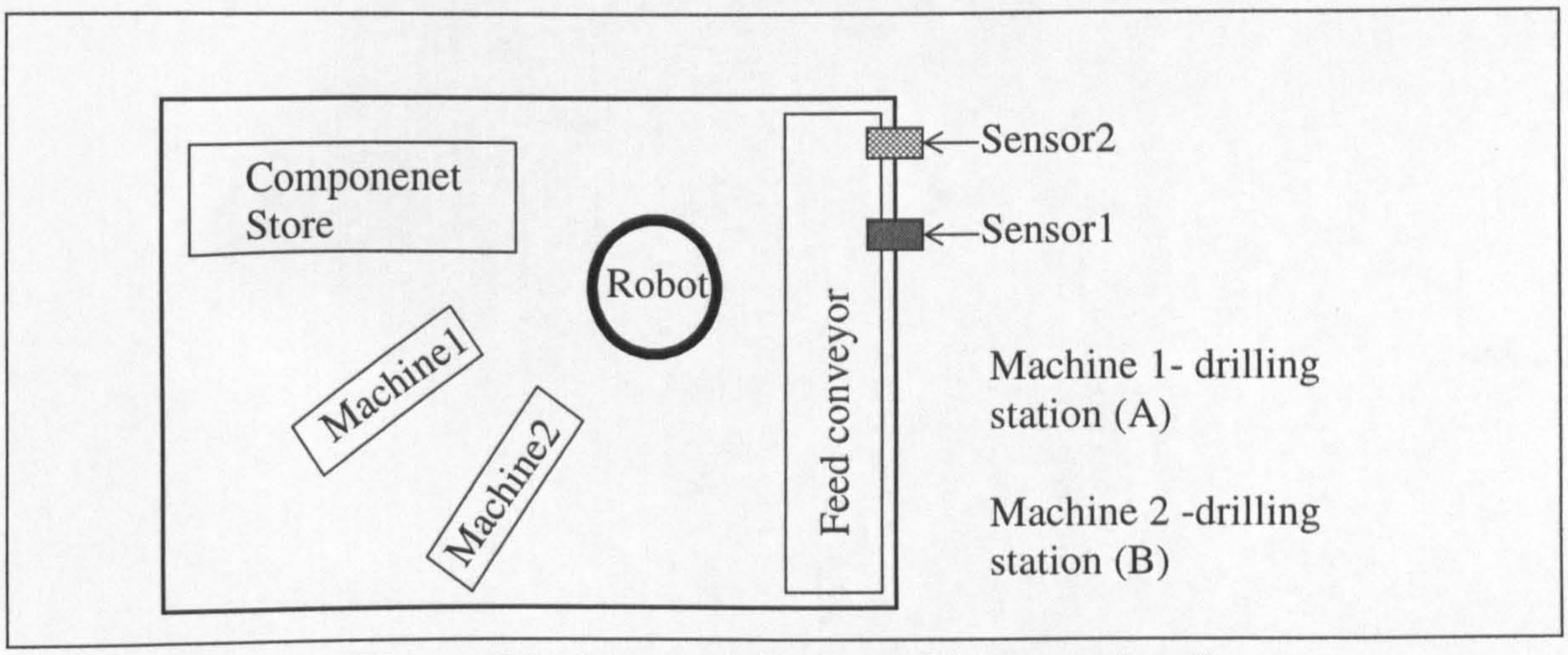


Figure 31: Layout of event-driven workcell

Table 17: Flowchart of the event-driven robot program

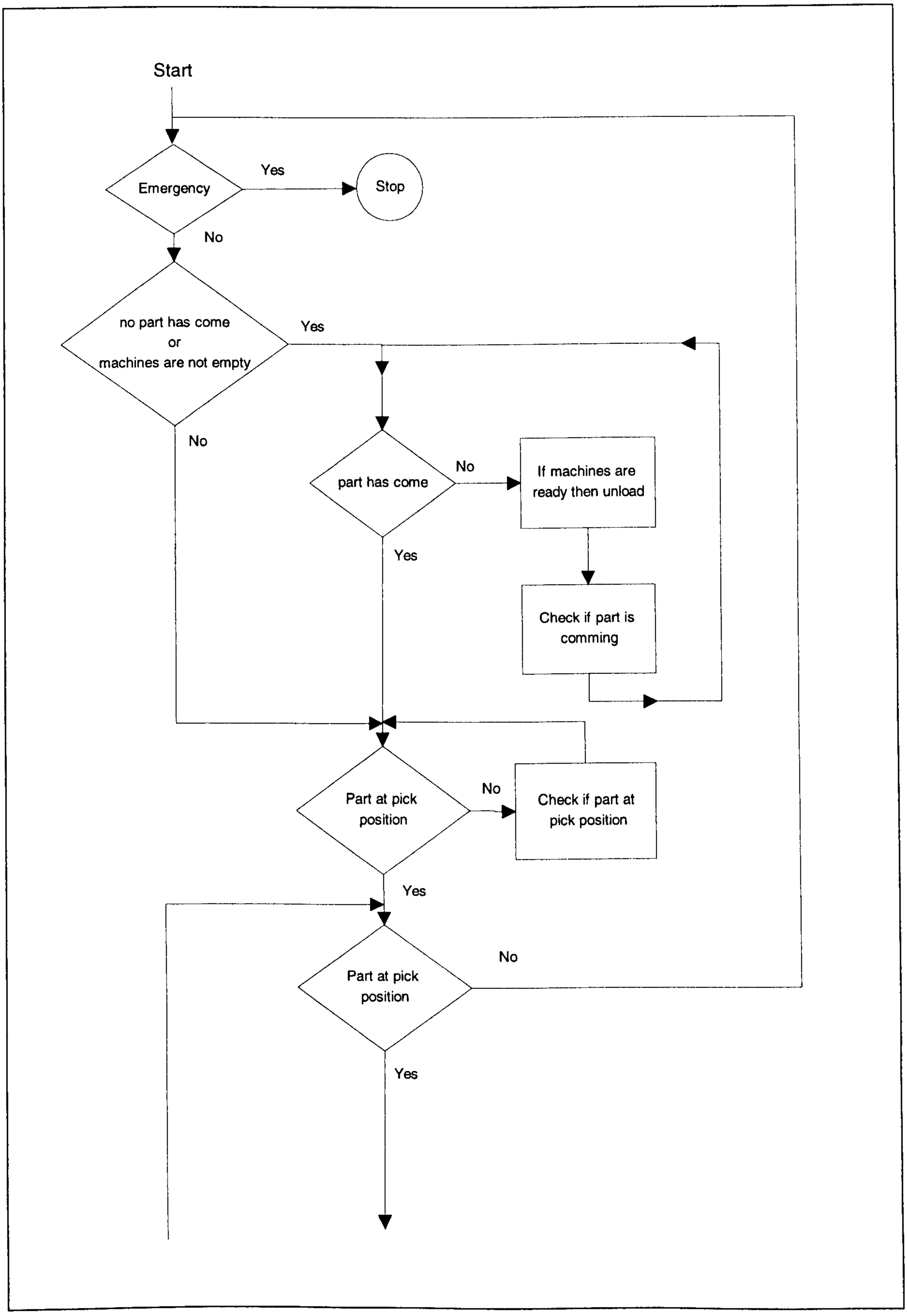


Table 18: Flowchart of the event-driven robot program continuing from Table 15

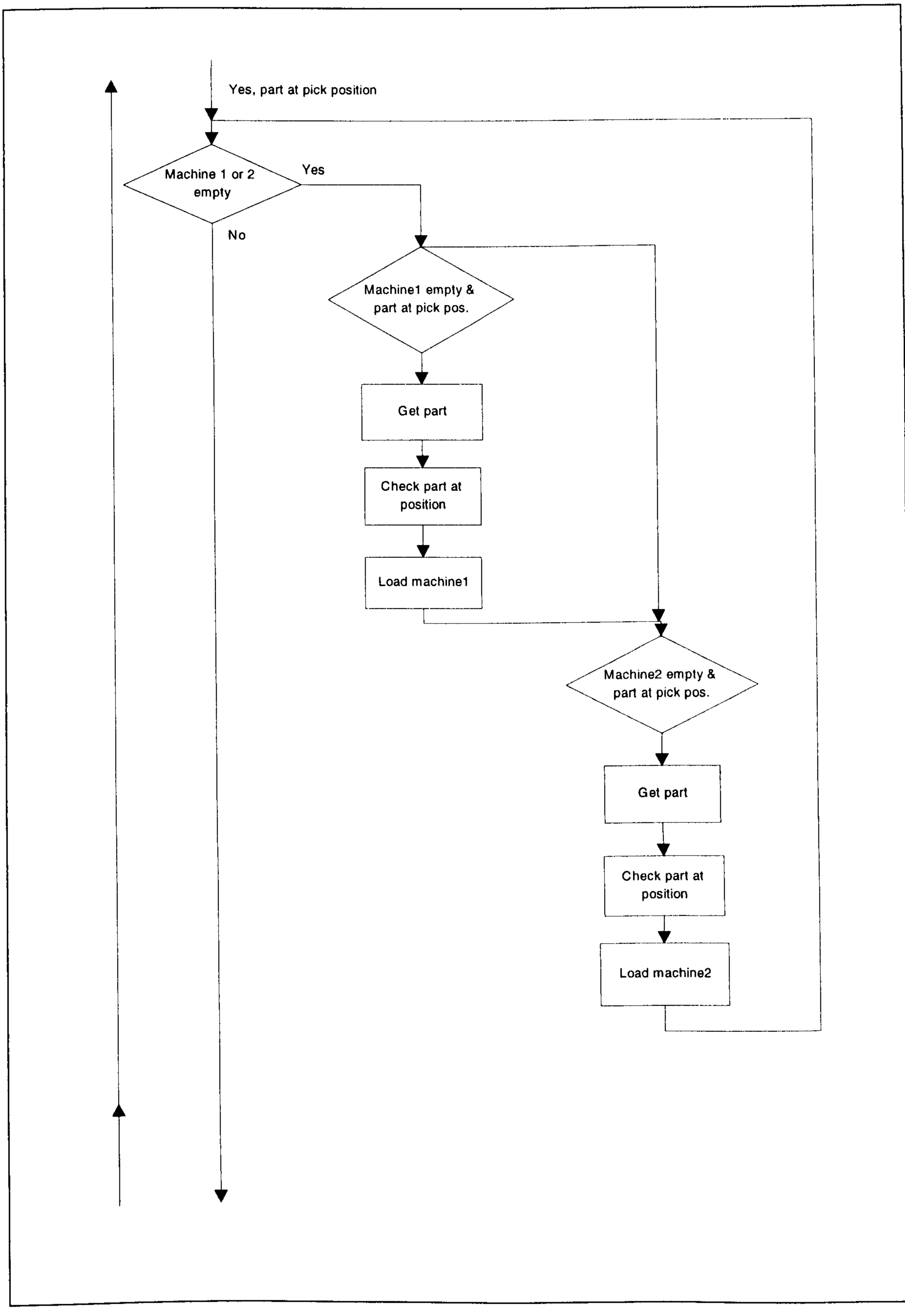
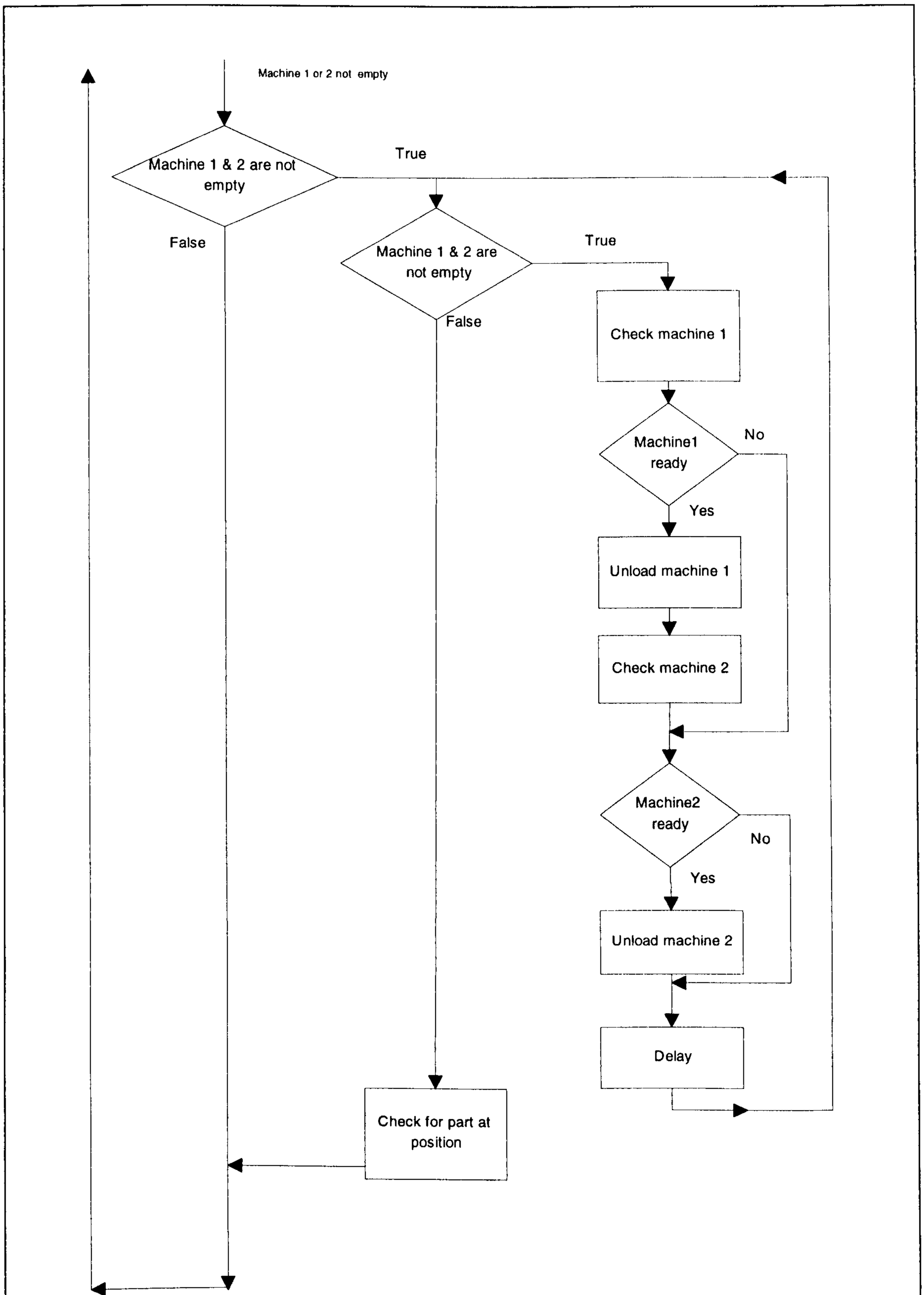


Table 19: Flowchart of the event-driven robot program continuing from Table 16



7.3.2 Transferring of programs from a virtual to a real workcell

Off-line programming of event-driven robot programs with virtual robotic tools using universal programming languages is not an option with contemporary virtual robotics tools. It is difficult to transfer program logic from one programming language to another, when there are no post-processors available for translating the program logic into a robot programming language.

To enable successful off-line programming and debugging of sensor dependent robot program code, as much as possible of the program syntax of the real robot controller needs to be available in the virtual environment. The main procedure of the program code created and simulated off-line to drive the virtual workcell to complete the task described is illustrated in table 20. The code for the real workcell (suitably amended to suit the robot controller) is illustrated in table 21. In real workcells there are likely to be many computing devices running several programs in parallel, such as PLCs, robot controllers and industrial PCs. These computing units will communicate with, for example I/O and network protocols. This communication will be physical electrical communication which is difficult to truly emulate in the virtual environment. The communication between independent processes must be performed via shared variables and memory that retains the states of for example I/O lines. A blackboard architecture as illustrated in figure 32, holding variables for process states, has been used here. All processes can access the information on this blackboard. This approach generates a need for program sequences that are only needed in the simulated world and not in the real cell.

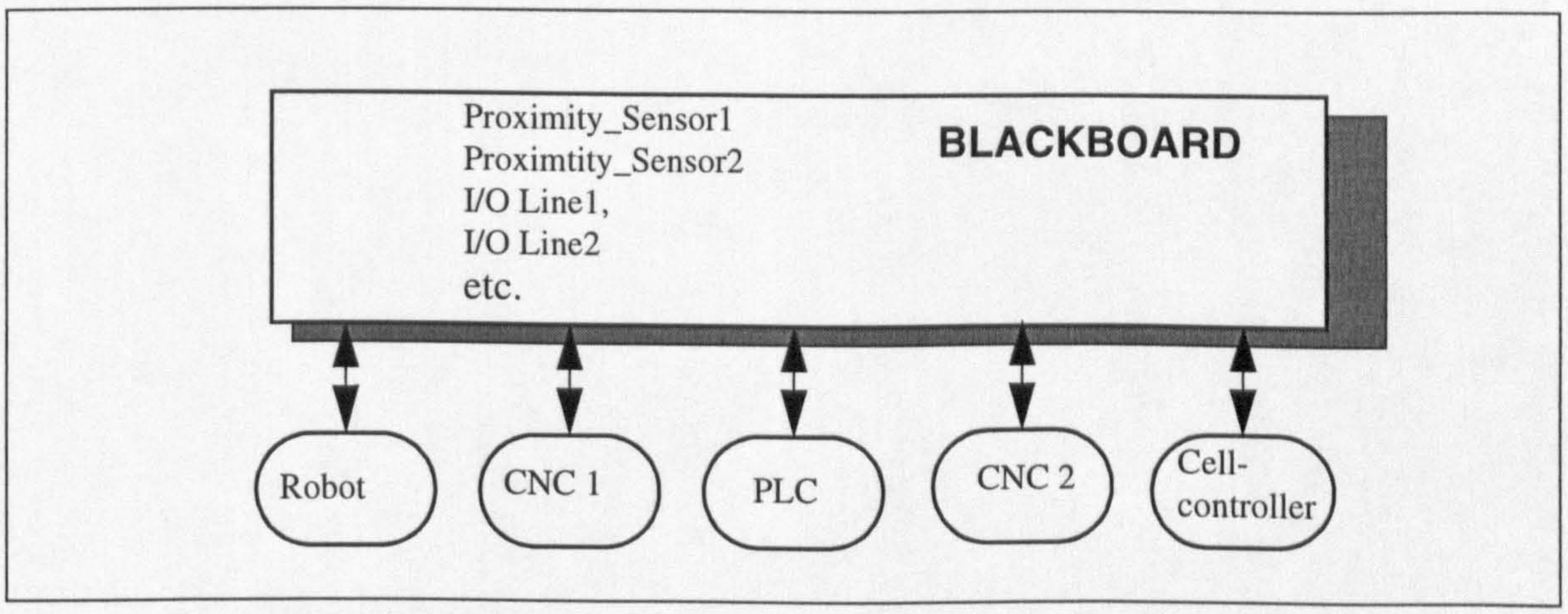


Figure 32: Blackboard architecture used in virtual cell control

Table 20: SIL code used in CimStation for the event-driven workcell

```
Process Mainprogram();
Begin
    init();
    While emergency=false do
    Begin
        movexyz(initpos);
        delay(1.0);
        Check_Part_Comming();
        If ((Part_has_come=false) or (machine1_empty=false) or
        (machine2_empty=false)) Then
        Begin
            While Part_has_come=false do
            Begin
                Check_Machine1();
                Check_Machine2();
                IF ((machine1_Ready=true) and (machine1_empty=false)) then unload_machine1();
                IF ((machine2_Ready=true) and (machine2_empty=false)) then unload_machine2();
                Check_Part_Comming();
                delay(1.0);
        End;
End;
{If a part is detected on the conveyor the robot waits until it gets to the grip position}
movexyz(pos1);
While (Part_has_Come=True) do
Begin
    Check_Part_At_Position();
    If Part_At_Position=True then Begin Part_has_Come:=false; End;
    delay(1.0);
End;
While Part_At_Position=True  do
Begin
    If ((machine1_empty=True) or (machine2_empty=True)) then
    Begin
        If  ((Part_At_Position=True) and (machine1_empty=True)) then
        Begin
            Get_Object();
            Check_Part_At_Position();
            Load_Machine1();
        End;
        If ((Part_At_Position=True) and (machine2_empty=True)) then
        Begin
            Get_Object();
            Check_Part_At_Position();
            Load_Machine2();
        End;
    End;
    If ((machine1_empty=false) and (machine2_empty=false)) Then
    Begin
        While ((machine1_empty=false) AND (machine2_empty=false)) do
        Begin
            Check_Machine1();
            IF machine1_Ready=true then
            Begin
                unload_machine1();
            End;
            Check_Machine2();
            IF machine2_Ready=true then
            Begin
                unload_machine2();
            End;
            Delay(1.0);
        End;
    End;
    Check_Part_At_Position();
    End;
End;
End;
```

Table 21: ScorPas code for the event-driven workcell

```
Procedure Main_prog;
Var emergency:Boolean;
Begin
    init;
    While emergency=False do
    Begin
        MoveXyz(160,262.66,236.94,-55.92,2.39); {Init}
        speed(5);
         Waitpos;
        Check_Part_Comming;
        If ((Part_Has_Come=False) or (Machine1_Empty=False) or
        (Machine2_Empty=False)) Then
        Begin
            While Part_has_come=false do
            Begin
                Check_Machine1;
                Check_Machine2;
                IF ((Machine1_Ready=True) and (Machine1_Empty=False)) Then Unload_Machine1;
                IF ((Machine2_Ready=True) and (Machine2_Empty=False)) Then Unload_Machine2;
                Check_Part_Comming;
                delay(1000);
        End;
End;
MoveXyz(-51.14,262.66,236.94,-55.92,2.39);    {wait}
speed(5);
Waitpos;
While part_has_come=True do
Begin
    Check_Part_At_Position;
    If Part_At_Position=True then Part_Has_come:=False;
    Delay(10);
End;
While  Part_At_Position=True do
Begin
    If ((Machine1_Empty=True) or (Machine2_Empty=True)) Then
    Begin
        If ((Part_At_Position=True) and (Machine1_Empty=True)) Then
        Begin
            Get_Object;
            Check_Part_At_Position;
            Load_Machine1;
        End;
        If ((Part_At_Position=True) and (Machine2_Empty=True)) Then
        Begin
            Get_Object;
            Check_Part_At_Position;
            Load_Machine2;
        End;
End;
If ((Machine1_Empty=False) AND (Machine2_Empty=False)) Then
Begin
    While  ((Machine1_Empty=False) AND (Machine2_Empty=False)) DO
    Begin
        Check_Machine1;
        If Machine1_Ready=True Then
        Begin
            Unload_Machine1;
        End;
        Check_Machine2;
        If Machine2_Ready=True Then
        Begin
            Unload_Machine2;
        End;
        Delay(1000);
        writeln('Waiting for machines to get ready');
    End;
    Check_Part_At_Position;
                End;
            End;
        End;
    End;
```

Most of the code for manipulating blackboard variables can be hidden in the SIL procedures to minimise the need for amendments of the off-line generated code. Table 20 shows an extract of the code needed for manipulating the blackboard variables. This approach makes it possible to use almost identical robot program procedures for the virtual and real robots, where the only differences are syntactic in character (as illustrated in table 23and table 24). To simulate the parallel processing occurring in separate computing devices, as it is accomplished in reality, demands fast computers for running the virtual world. The computer running the simulation should simulate the execution of each device program at the same speed as it is executed in reality.

Table 22: Code for handling Blackboard parameters. The procedure simulates the input status function which is implemented in ScorPas.

```
Function InputStatus(nr : integer):boolean;
begin
   InputStatus:=False;
   case nr of
     1: InputStatus:=Prox1_Value;
     2: InputStatus:=Prox2_Value;
     3: InputStatus:= machine1_running;
     4: InputStatus:= machine1_ready;
     5: InputStatus:= machine2_running;
     6: InputStatus:= machine2_ready;
end;
end;
```

Table 23: SIL procedure to check if a part is arriving on the conveyor.

```
Procedure Check_Part_Comming();
Begin
    Part_Comming:=InputStatus(1);
    If Part_Comming=True Then Part_has_come:=True;
    delay(0.2);
End;
```

Table 24: Corresponding ScorPas code (table 21), checking if a part is arriving on the conveyor

```
Procedure Check_Part_Comming;
Begin
    Part_Comming:=InputStatus(1);
    If Part_Comming=True Then Part_has_come:=True;
    delay(200);
End;
```

7.3.3 Extended workcell program

To demonstrate that the use of the generic sensor model makes it possible to easily expand an existing workcell with more sensors, thus creating a more complex environment and providing for more flexible behaviours, the task described in the previous section was expanded to include decisions based on sensor information. A sensor is used to distinguish between objects of different dimensions.

A virtual ultrasonic transducer was mounted onto the virtual Scorbot (as depicted in figure 30). The sensor was a model of the Honeywell ultrasonic transducer described in section 6.4. Parts with two different heights were fed on the conveyor (part 1 and 2). Parts of type1 could only be machined in machine 1 and parts of type 2 in machine 2. The ultrasonic transducer was used to detect the height of the parts. Based on the information from the sensor, the robot makes a decision as to which machine to load and to check whether that machine is available or not. The new sensor was easily added to the virtual workcell, using the generic sensor model. The robot task program was rewritten to suit the new demands. The sensors "behaviour control" programs all work independently of each other. The proximity sensors update the information on the blackboard every 0.1 sec and the ultrasonic transducer updates its value every 0.350 sec. The update rates of the proximity sensors are set to meet the requirements needed for monitoring the conveyor belt. The update rate of the ultrasonic transducer is set according to the specification from the manufacturer. If the control function of the simulated sensors are set to very short intervals (for example one microsecond) the execution speed of the simulation will be slow. The update rate of each sensor should be set with respect to the demand from the particular application and to the monitoring frequency set in the robot control program, i.e there is normally no need for updating a sensor at 100 Hz., if its monitored with 0.5 Hz., The update rates should be set to values which do not jeopardise the realism of the simulation.

This task, using a sensor's range information, has similarities with the use of the search function implemented in ABB's ARLA programming language, where a sensor is connected to the controller and usually mounted at the gripper. The robots can be programmed to move to a "start" position and then to move in one direction until a predefined threshold value from the sensor is reached. This is a function extensively used at Volvo's car body plant in

Gothenburg Sweden, where for example the robots in a welding line are programmed off-line with a virtual robotics tool. The search function is used to compensate for errors still present after calibration and for time dependent process variances. Search functions of this type can not be programmed off-line and simulated as the systems currently available do not provide functions for the simulation of sensors [Axelsson 1995]. Search functions and other sensor related functions must be added to the robot programs after the application has been simulated, debugged and post-processed. Hence, this exposes untested program logic to the factory floor. Table 23 illustrates some of the robot code for sensors and search functions. The code is from an industrial application where some parts of the installation were programmed off-line with a virtual robotics system, but the parts which include the logic and sensor interaction had to be added manually after the simulation exercise.

Table 25: Robot program from an industrial application using sensor information to control motion

```
PROGRAM 300  /*Get Part from Pallet

10    TCP 1
20    RECT COORD
30    JUMP TO 540 IF R41 > 0 * R42 > 0
40    JUMP TO 290 IF R41 > 0
50    JUMP TO 70 IF R42 > 0
60    JUMP TO 290 IF R11 = 1
70    POS V=100.0% C1 #58
80    POS V=100.0% STORE POSITION 10
90    JUMP TO 120 IF R12 = 1
100   POS V=100.0% STORE POSITION 2
110   LET R12 = 1
120   POS V=100.0% PATH POSITION 2 OFFSET X=0.0 Y=0.0 Z=0.0
130   POS V=4.0% FINEC SEARCH S1 #59
140   LET R11 = 1
150   JUMP TO 60 IF INP 1 = 0
160   POS V=4.0% STORE POSITION 2
170   LET R7 = R3
...
...
...
Ingenjörsfirma
Evert Johansson AB
1995 -03-02 Joakim Janneson
```

7.4 Event based systems and verification methods

In a CIM (Computer Integrated Manufacturing) environment, an attempt is made to automate the craftsmanship of the operator, that is, computers are not only programmed to control indi-

vidual machines they are also programmed to handle events occurring in and between machines. The tasks for this automated operator is, in the first place, to handle errors at the operational level of the production process [Meijer et al. 1992], for example when parts are missing at a certain operation, re-scheduling due to machine break-down etc. This imposes needs for programming and analysing tools for event based CIM environments. Most programming methods for event-driven robotic workcells reported in the literature are related to task-level programming. In task-level programming the events are of an "abstract nature" such as planning of a sequence when certain objects are available with error recovery targeted to plan actions when an object is missing or faulty. Some events are not considered such as: a robot is awaiting signals from several sensors and its action depends on which sensor signal is received. What will happen if the robot program receives two sensor signals simultaneously? Which action will take priority?

Error detection with respect to robot motions and internal manipulator states is normally well catered for by the robot controllers, where system programmers build in monitoring functions into the manipulator control software. Internal sensors are monitored for missing signals, for example, torque sensors on the actuators are monitored for detecting overload and collisions. Error detection in program logic and errors in the workcell have to be provided by the task program. The following features are typical for event-driven tasks like flexible assembly [Meijer et al. 1992]:

- Several desired states can be supported at each program step.
- Sensing instructions can be prescribed in the assigned program.
- The program´s execution flow is dependent of the result of the sensor readings.

A flexible robot workcell which handles events and recovers from errors must be capable of detecting deviations of environment values from their expected values. Uncertainty in the environment model involves both the position of objects and their tolerances (as discussed in section 3.6). Event-based systems can be analysed with different formal methods such as state/precedence graphs and Petri-nets. The development and evaluation of exception and error handling strategies can be supported using such methods. In precedence graphs, subtasks are represented by nodes and the dependency between two tasks is presented by an arc, connecting the two nodes. The dependency arc indicates that a node is not executed before the node/nodes connected at the top end/ends of the arc/arcs are successfully performed.

Meijer and Herzberger have created a method for exception handling based on precedence graphs, EHM (Exception Handling Model) [Meijer and Herzberger 1988]. The method is based on the assumption that a representation of the task structure or task intent is available. A list of imaginable exception situations is created and to each situation a set of recovery strategies is connected. The EHM contains a mechanism for selecting and performing the recovery plan. Jacak and Rozenblit have implemented a system for simulation and programming of robotic tasks that are sequential events [Jacak and Rozenblit 1992]. The methodology for verification of the robot task is based on precedence graphs. This approach is based on the robot being programmed in a task level programming language. The method is used to verify that the tasks are performed in the correct order. The approach is purely for sequential events and does not enable concurrency.

Petri-nets introduced by Adam Petri [Petri 1962] is a method for the modelling and simulation of dynamic systems. Particular features of petri-nets are their ability of handle parallelism and concurrency. A standard Petri-net is composed of four parts: a set of places (P), a set of transitions(T), an input function (f) and an output function (O).

N=(P,T,f,O)

P={P1,...,Pn} a set of n places

T={T1,...,Tn} a set of n transitions

f(p,t)= is an input function that defines directed arcs from input places to transitions.

O(p,t)= is an output function that defines directed arcs from transitions to places.

The execution of Petri-nets is controlled by a distribution of tokens (illustrated in figure 33). If a token exists at each input to a transition, that transition is triggered and it puts a token out to the next point, all tokens at the input are then removed. In a manufacturing verification, objects can be represented by tokens, locations by places and operations by transitions [Remboldt et al. 1993]. In it's original form time is not represented in Petri-nets. To enable Petri-nets to handle an extended range of entities, for example time, external control and different classes, there have been several extensions to the original Petri-net model. Classes of petri-net for example are condition/event Petri nets with time transitions (CEP) and Colour Petri Nets.

Time of operations can be modelled in the Petri-net by using places as operations [Remboldt et al. 1993].

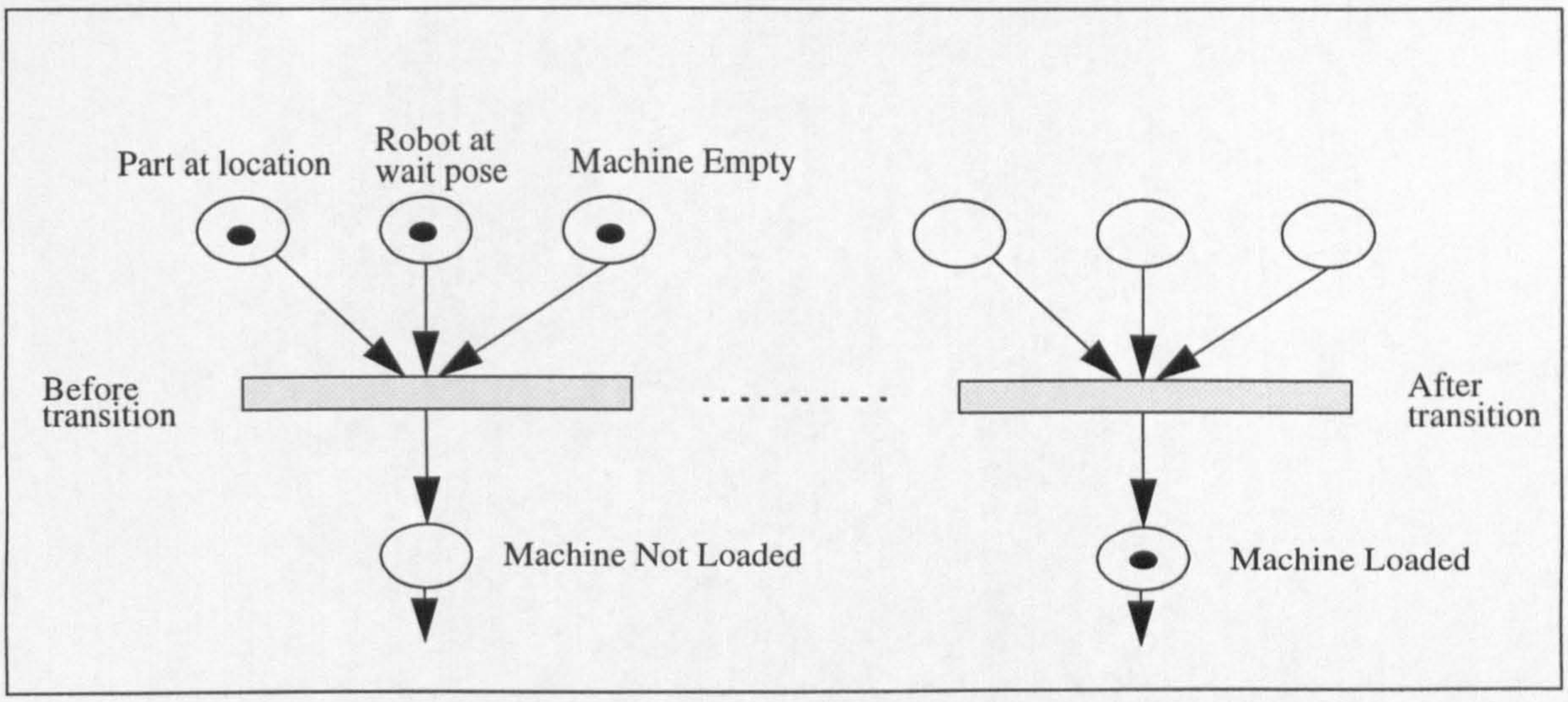


Figure 33: Part of Petri-net: When the robot is at the waiting position and a part has arrived on the conveyor and the machine is empty. The machine will be loaded after the transition.

Petri-nets have been used to verify scheduling of operations in flexible robotic manufacturing systems and robotic assembly [McCarragher 1993], [Remboldt et al. 1993], [Yeung and Moore 1996]. They are generally used for detection of deadlocks such as: will all sub-parts of an assembly be available and arrive in the correct order to an assembly operation or will some parts be delayed or missing? Are all sub-tasks performed in the correct sequence? McCarragher uses Petri-nets to adapt robot motions in robotic assembly [McCarragher 1993]. Petri-nets are used to adapt to uncertainties in objects' locations and geometry. The adaption made in McCarragher's approach involves changes to the robot's motion velocities. To be able to adapt the robot's motion a fifth set of controls, which are discrete controls (B), are added to the Petri-net model. The approach is purely theoretical and needs a complete knowledge of the workspace geometry. The methods could be helpful for off-line evaluation of task-level programming systems. Yeung and Moore presents a methodology where colour Petri-nets are used for on-line scheduling and control of flexible assembly systems [Yeung and Moore 1996]. In their approach a Virtual Colour Petri-net, which is a copy of the real controller Colour Petri-net, is used to evaluate possible operational conditions. With this built-in pre-knowledge situations such as dead-lock can be avoided.

7.5 Summary

Off-line programming of robot programs at the task level needs simulation of sensors as the system should be able to generate actions based on an environmental model. Task-level programming is described in section 2.4. Task-level programmed systems should include exception handling strategies [Meijer and Hertzberger 1988]. For complete off-line generation of task level programs, there is a need to simulate sensors for detection of simulated environmental changes and events. It is essential during the debugging phase that the programmer is not in control of events external to the program, for example simulation of signals from other machines, sensor signals and input from keyboards etc. The programmer can not foresee all possible situations and states and therefore should not be controlling signals manually during the debugging phase (for example by connecting input signals to the keyboard or other input devices). The programmer will most likely set sensors and other events in the order that they where intended to occur when creating the programs. The experiments show that it is possible to create event-driven robot programs where the events are actually generated from simulated sensors and not just from machine-interaction. The virtual sensors were created with the generic sensor model, which allows fast generation of sensor models with different characteristics and properties.

To be able to generate robot programs consisting of complex logic statements, which are needed when creating a production unit with more autonomous behaviour than traditional production lines, is it important that the languages used for programming virtual and real equipment have a similar syntax. It is preferable that the virtual and real robots are programmed with the same programming language. Translating programs from one language to another, using post-processors, is not always straight forward. This is particularly a problem if the programs include logic and reasoning. Contemporary virtual robotic tools are normally only used for off-line programming of motion sequences. Complex motion commands such as *circular* motion commands can be difficult to translate. The motion sequences consists of "simple" motion commands in the virtual robotics system which are translated to a simple controller specific motion command, for example moveto(x,y,z) is translated to posh(xyz). Virtual robotic systems must be extended with methods for the realistic handling of signalling between devices if complete off-line programming of robot programs including sensory interaction and logic reasoning is to be achieved. In this research study a black-board

architecture has been implemented and used for successfully handling device interaction and physical properties.

Enhancement of virtual robotic systems with the addition of tools based upon formal verification methods, such as colour-Petri nets or state graphs, and methods for simulating sensors, such as the generic sensor model, would provide a good platform for off-line generation and debugging of fully functional event-driven robot programs.

7.6 References

[Axelsson 1995] Axelsson S., *Interview with Stefan Axelsson.* Volvo Research, Gothenburg October 1995.

[Chung et. al. 1993] Chung H.L.C., Narayan O. H., and Davies B.J., *An Event-Based, Robot-Served Flexible Manufacturing Cell Modeller*; International Journal of Advanced Manufacturing Technology , 1993, Vol 8, pp. 102-110.

[Jacak and Rozenblit 1992] Jacak W. and Rozenblit J. W., *Automatic simulation of a robot program for a sequential manufacturing process*. Robotica 1992, Vol. 10 pp 45-56.

[Lahti 1995] Lahti T., *PLCLINK WHITE PAPER*, Tehadsmalit AB 1995.

[Mahajan et al. 1993] Mahajan S.K., Brewera S.K and Bien C.D., *QUEST-QUING EVENT SIMULATION TOOL*. Proceedings of the 1993 Winter Simulation Conference, pp 269-275.

[McCarragher 1993] McCarragher B. J., *Task Level Adaption using discrete event controller for robotic assembly*. Proceedings of the 1993 IEEE International Conference on Intelligent Robots and Systems, ISBN 0-7083-0823-9.

[Meijer and Hertzberger 1988] Meijer G.R. and Hertzberger L.O., *Off-Line Programming of Exception handling Strategies*. IFAC Symposium on robot control 2, Karlsruhe 1988, pp. 431-436.

[Meijer et al. 1992] Meijer G.R., Caglioti V., Somalvico M. and Negretto U., *Execption handling*. Integration of Robotics into CIM, Chapman & Hall 1992, ISBN 0-442-31243-1.

[Okano et al. 1988] Okano A., Shimbada K. and Kawabe S., *A concurrent motion simulator and interactive debugger for multiple robots*. International Symposium and Exposition on robotics 19 , Sydney 1988, pp 998-1009.

[Petri 1962] Petri C.A., *Kommunikation mit Automaten*. PhD Thesis 1962, University of Bonn, Germany.

[Remboldt et al. 1993] Rembold U., Nnaji B.O., and Storr. A. *Analysis Tools for Manufacturing*. Computer Integrated Manufacturing and Engineering, pp. 127-135, ADDISON-WESLEY 1993, ISBN 0-201-56541-2.

[Yeung and Moore 1996] Yeung W.H.R. and Moore P.R., *An Integrated MMS and colour petri/net model for the distributed control of flexible assembly systems*. International conference on Computer Integrated Manufacturing, Singapore, 1995, vol. 1 pp. 831-838.

Chapter 8 'Pre-emptive Learning' and Adaptive Robotics

Robot programs developed with contemporary programming techniques are generally limited in their ability to take into account uncertainties in the environmental model and deal with environmental changes. The programmer and system designer must consider tolerances in object geometry, uncertainty in grip positions, take into account sensor readings and anticipate the degree of uncertainty (predictable variance) in the environment at the planning stage. Machine learning is beginning help to overcome some of these problems [Van de Velde 1993]. One of the technical goals of robot learning is to automate the construction, modification and organisation of a robot's internal representation. Such representations consist of properties and relationships which are external to the robots, such as knowledge about effects of actions etc. A robot is autonomous when it is self-governing, where such a robot organises its own internal structure in order to behave adequately with respect to its goals and the world [Van de Velde 1993]. Learning robots can handle environments that have some degree of unpredictability but in which programmers are readily at hand. An example of this is the Helpmate robot for hospital environments [Engelberger 1989].

Robots can learn to react to environmental changes by using sensors [Nehmzow 1992], [Van der Smagt1996]. However, here the thesis proposed is, that *real robots can use skills learnt by virtual robots trained in virtual robotic systems using simulated sensors.*

Two mobile robot platforms, the FRANK2 mobile robot illustrated in figure 10, from TAG Inc. and a Nomad 200 robot from Nomadic Technology Inc. illustrated in figure 11, have been used to test this thesis. Van de Velde states "A mobile robot is the archetypical example of an autonomous system: equipped with sensors and effectors, it moves around and interacts with its environment in order to achieve some goals" [Van de Velde 1993]. Two different virtual mobile robots with similar control architectures have been trained in a virtual world, to be able to decide actions based on information from sensor readings. The response from the controller is determined by an artificial neural network. The low level control of the robots is based on instincts. Instincts are sets of sensor conditions that must be satisfied. The artificial neural network in the virtual environment is trained to map sensor readings, of sim-

ulated sensors mounted on the virtual mobile robots, to the correct actions of the robots. The robots were taught two types of behaviour; (i) move forward without collisions and if a corner is encountered, choose a manoeuvre which gets the robots out of the corner; and (ii) traverse a corridor and maintain the robot in the middle of the corridor.

The robots used for the evaluation had sensors with different characteristics and different numbers of sensors. This resulted in different network architectures being used in the controllers. The artificial neural networks trained in the virtual worlds were then transferred to the real mobile robots which were then able to accomplish the defined tasks after some training time used for "refining" of knowledge. The on-line training is necessary to adjust the internal representation, kept by the artificial network controller, of the robot and its sensors. The internal representation has been constructed in the artificial neural network during training in the virtual world and must be adjusted to accommodate the differences between the virtual and the real environments. This continuous learning guarantees that the system will identify and learn future changes in the system and its environment. It is also possible to calibrate the virtual model by transposing the neural network that has been 'refined' by exposure to the real world. Using a completely untrained artificial network in the controller, causes the robot to take unpredictable actions at the beginning of the training period. The robot can for example run into a wall a hundred times before learning to avoid a collision and move forward. This can cause material damage to both the robot and its environment and can be dangerous. Learning from scratch is very time consuming in that the system must be physically reset after each trial. If the robot is powered with batteries they will be discharged rapidly during the training phase, which places demands on keeping a relatively large stock of charged batteries, otherwise the training will take excessive periods even for relatively easy tasks.

8.1 Control architectures for mobile robots

There are many control architectures and strategies for autonomous robots described in the literature. However, two main contrary approaches can be found:

(i) The reactive approach suggested by Brooks [Brooks 1986] where the control system is composed by task achieving behaviours, as illustrated in figure 34. Purely reactive control

systems have a set of pre-programmed condition-action pairs, these systems maintain no internal models, but simply look-up and command the appropriate action for each set of sensor vectors [Mataric´ 1994].

(ii) The layered control where the control system is divided into functional modules [Alami et al. 1992], [Crowley et al. 1991]. In layered control systems a planner uses a world model to assist the planning of actions, with respect to goal definitions and current sensor information as depicted in figure 35.

Between these extremes there are many hybrid architectures, which usually employ reactive strategies for low-level control and a planner for high level goal achieving, for example the architecture presented by Nehmzow et al. [Nehmzow et al. 1993]. The control architecture used in this study as illustrated in figure 36, is a hybrid architecture which has similarities with the architecture suggested by Nehmzow et al. [Nehmzow et al. 1993].

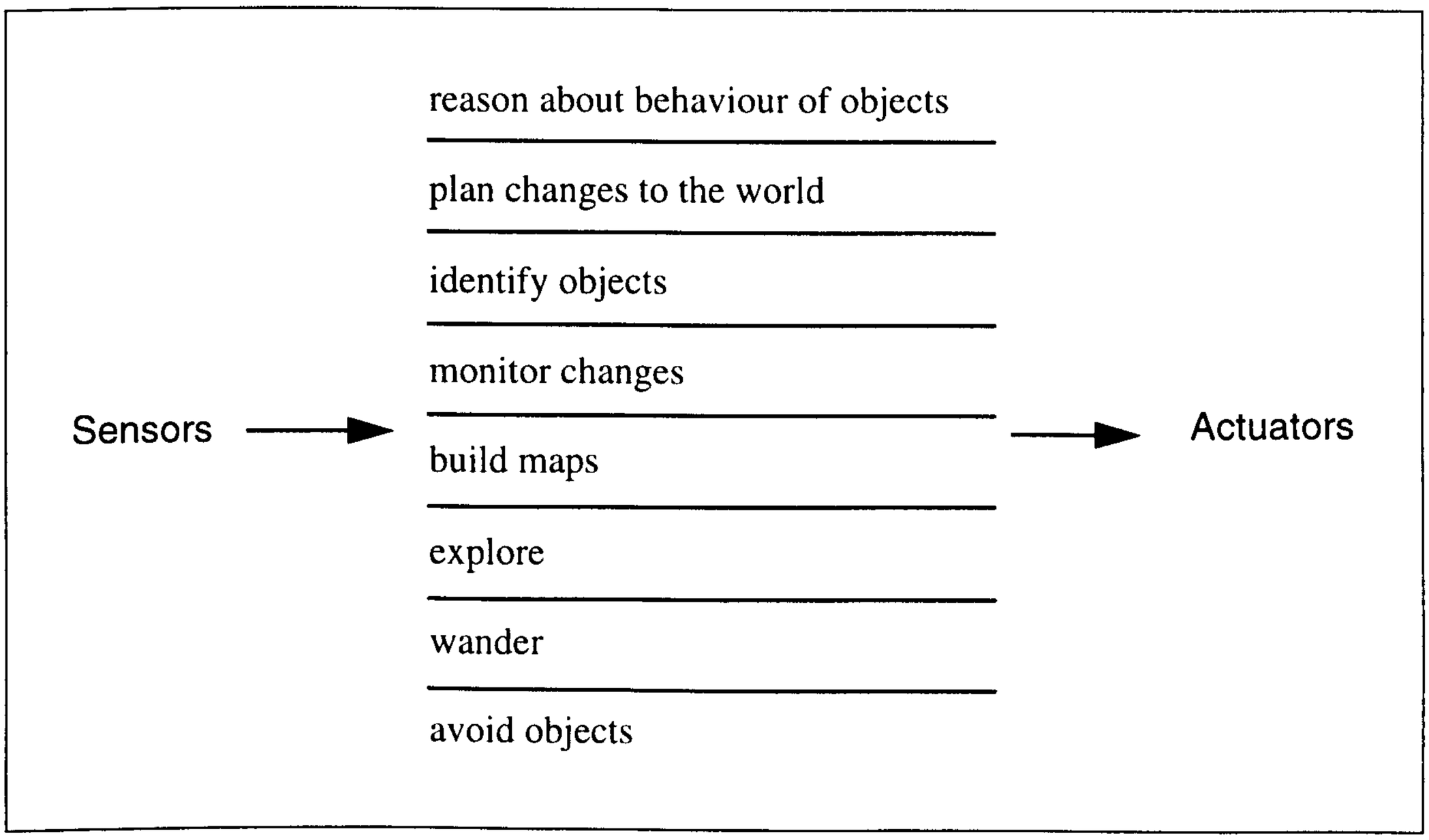


Figure 34: Behaviour based control, as suggested by Rodney A. Brooks

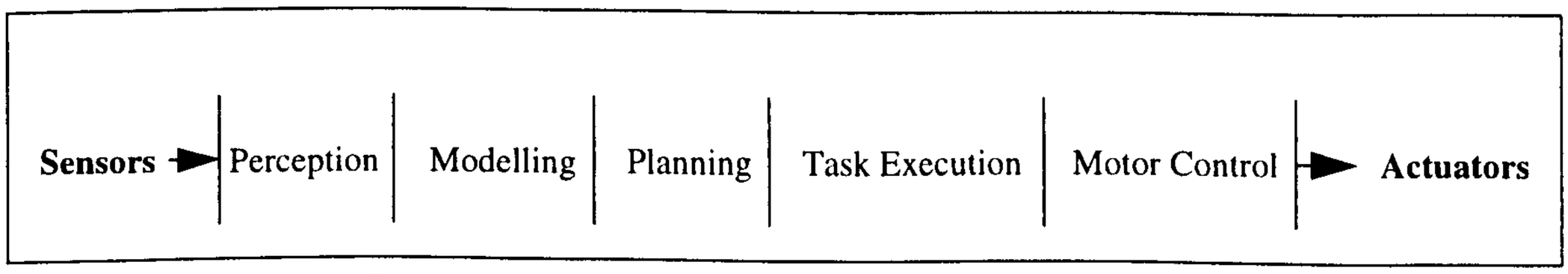


Figure 35: Layered control system for autonomous robots

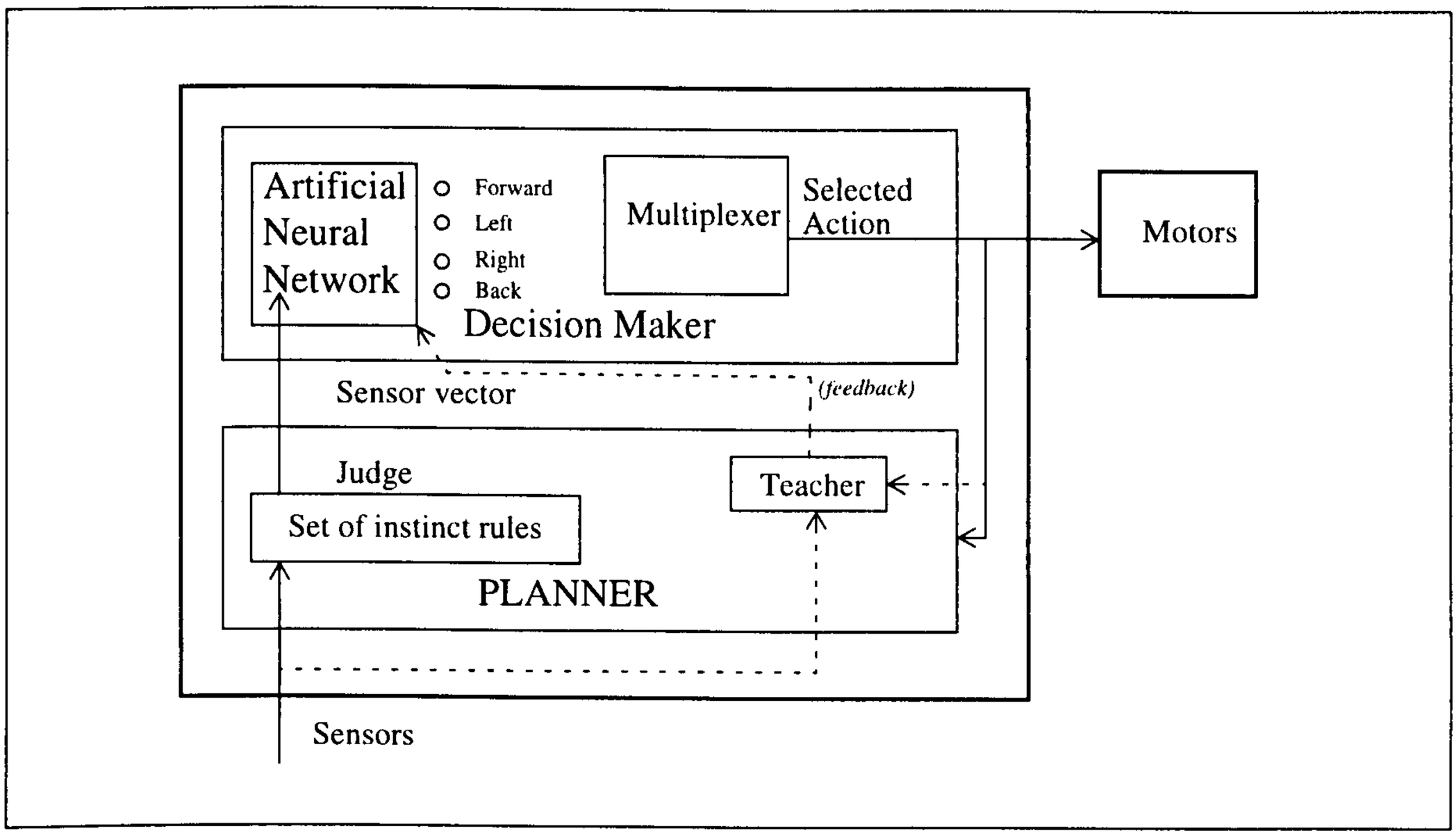


Figure 36: Control architecture for mobile robots used in the experiments

The low-level control is achieved through 'instincts' as introduced by Nehmzow [Nehmzow 1992]. Instincts are sets of sensor conditions that should be maintained, table 26 shows examples of instinct rules. Sensor conditions could be, for example, for avoid collision, "keep infrared sensor value to 0", which would mean nothing is detected. The robots overall goal is controlled by the "Planner"

Table 26: Examples of instinct rules

`Move Forward`
`Avoid collision`
`Follow wall`
`Follow corridor`

While trying to achieve the overall goal, the low-level controller reads the sensors to detect if there is any violation of the instincts. If any of the instinct rules are violated, the robot has to move to a state where the violation of the rule stops, before proceeding to accomplish the overall goal. The instincts are set in hierarchical order so that a violation to an instinct at a lower level must be stopped before trying to stop the violation of an higher order instinct. If for example the robot has a sub-goal to follow a corridor and an obstacle which would cause a collision is detected *(the instinct rule **avoid collision** is violated)* at the same time as the

robot fails to follow the corridor *(the instinct rule **follow corridor** is violated)*, the robot has to avoid the collision before trying to follow the corridor again.

An artificial neural network is used to solve the problem of mapping the relationship between a violated instinct and an action which stops that violation. If there is a violation against any instinct rule the 'Planner' feeds a sensor vector to the artificial neural network. This sensor vector contains the current sensor state. The output nodes of the ANN corresponds to different primitive motor actions. A multiplexer is used to choose the strongest output node produced by the ANN for a particular sensor vector.

The artificial neural network is trained using a supervisor unit called 'Teacher' in the control architecture. The robot starts by learning the most basic instinct, namely to 'Move Forward'.

When an instinct is learnt, the overall repertoire of the mobile robot increases. Some example behaviours are listed in table 27. The artificial neural network maintains the previously learnt knowledge whilst learning new responses as shown by Nehmzow et al. [Nehmzow et al. 1993]

Table 27: Extended repertoire of behaviours

Move Forward
Avoid Collisions
Turn in Corner

8.1.1 Control objectives to learn in a virtual world

The main objectives for the control of the robots were defined as:

Control objective I, the robot should learn to move forward. The robot should avoid collisions with obstacles. The robot should, when it encounters a corner, choose to turn in the direction which allows the robot to proceed by moving forward into 'free space'. These behaviours when learnt should allow the robot to move clockwise or counter-clockwise along the walls of a room.

Control objective II, the robot should learn to follow corridors. The robot should keep in the

middle of a corridor even if the width of the corridor changes. The robot should, if positioned at a random location, be able to move to the middle of the corridor and continue to follow the corridor.

8.2 The learning phase

Starting from an untrained network, the robot tries to learn the correct response to a specific sensor vector. When an instinct rule is violated the multiplexer starts by choosing the strongest output node and executes the action that corresponds to that node. The action is performed and the controller then checks whether the violation is still occurring. If the violation is still present the multiplexer chooses the next strongest output node and performs the corresponding action and so on. If all nodes have been tested the multiplexer starts with the strongest node again. When an action has removed the violation, the correct pattern on the output nodes are known. This pattern is used for calculating the errors on the output nodes. The correct node is set to 1 and the others to 0, the output of each node is then compared with this value and the error is calculated. This error is then used by the error back propagation algorithm to adjust the weights and train the ANN.

A method to determine if the network has generalised is to evaluate the neural network error. A goal of neural network training is to find a network weight vector that minimise the error [Hecht-Nielsen 1989]. There exist several measures for the neural network error. Calculating the difference between the actual output neuron and the target value (given for an input-output pair) and then sum the differences over all output neurons ($E=\sum(t-o)$) is an error measure defined by [Rumelhart and McClelland 1986].This method for error calculation is on a pair-by-pair basis. Another measure of the network error is the average sum-squared error value ($E=0.5*\sum(t-o)**2$) [Eberhart and Dobbins 1990]. The basis for this error calculation method is on the entire training set. Other error measures are maximum absolute error, mean absolute error and median squared error [Hecht-Nielsen 1989]. In this research study the error signals (used by the error-back propagation algorithm) are summed over the output nodes. The summed error signals are monitored to determine the generalisation.

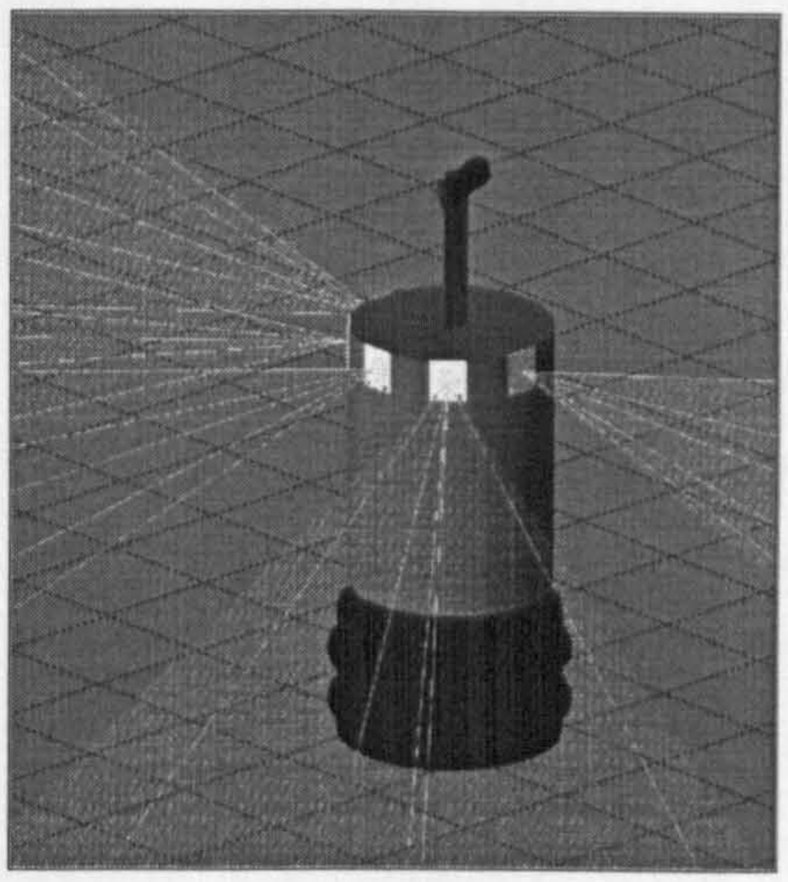
The minimum error level varies for different situations and can be determined for the particular 'application' using a validation test [Hecht-Nielsen 1989]. To determine if the minimum

error level has been reached, a set of test pairs not included in the training set can be used. These test pairs are presented to the network and if the network produces correct outputs while keeping the error level close to the level reached before the test session then the network can be considered fully trained (i.e it as generalised). In this research study no fixed training data sets are used during the training sessions, the input-output pairs are generated as situations occur during the training. The generalisation is tested by exposing the networks to a complete new situation and they are considered fully trained if it can accomplish the mapping while keeping the error level close to the level reached before the test.

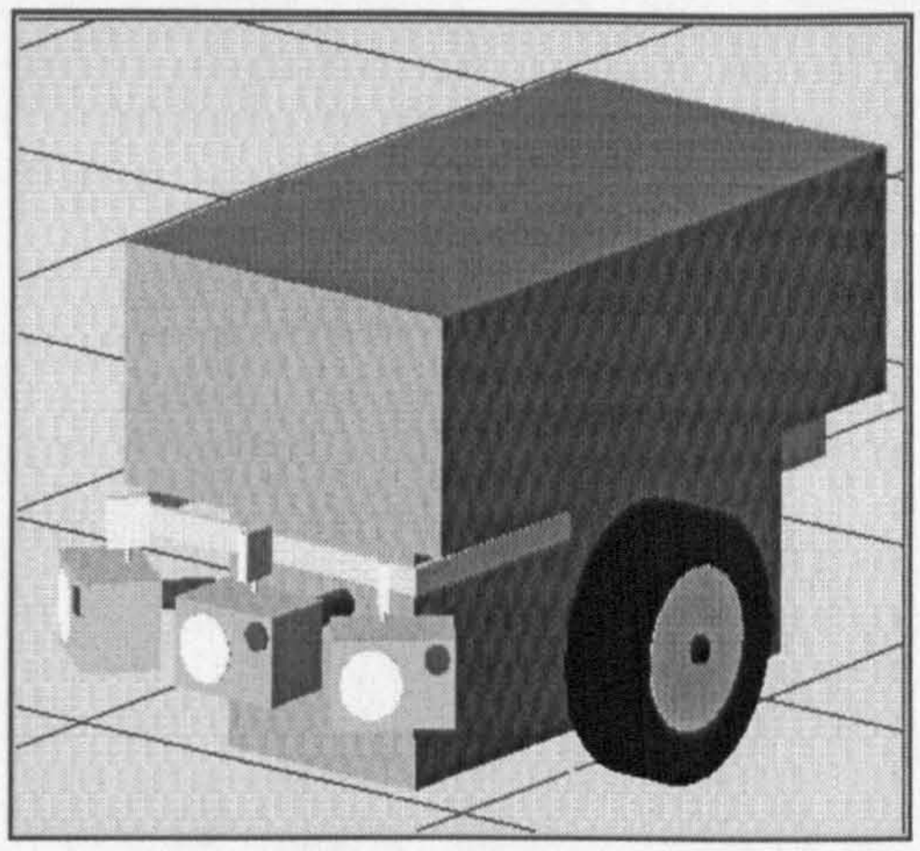
The 'target' error level used in these experiments was determined through experiments using the Frankie robot and the network illustrated in figure 48. The network was trained in batches of 100 training steps and exposed to a 'test environment'. The robot accomplished the task successfully when the neural network error was below 0.10. This value was determined to be the level to be reached before the training was stopped. The same level was used for all experiments in this study.

8.2.1 Virtual robots

Two virtual mobile robots were created in the CimStation virtual robotics environment as illustrated in figure 37. A virtual model of the FRANK2 robot was created, called Frankie (figure 37b). It was equipped with simulated versions of the sensor pods, the Polaroid ultrasonic sensor which is described in section 6.4. A virtual model of the Nomad 200 robot called Nomadie was also created. The Nomadie robot was equipped with arrays of the simulated Polaroid ultrasonic sensors as illustrated in figure 38, and bumper sensors. The Polaroid ultrasonic sensors on the Nomad 200 are driven by a Polaroid 6500 ranging board, giving them different characteristics to the sensors on the FRANK2 robot. The simulated sensors where implemented with the generic sensor model. Table 28 shows the main function of the simulated Nomadie ultrasonic sensors. The sensors where created using information about sensor characteristics provided by the robot manufacturer.



a) Nomadie virtual Nomad 200 robot



b) Frankie virtual FRANK2 robot

Figure 37: Virtual mobile robots

The virtual robots were implemented with the same motion control schemes as their corresponding real robots. The control architecture described in section 8.1 was implemented in the SIL language. The objective was to implement a control architecture with the same structure and syntax on both real and virtual robots. To enable execution of realistic motion commands for controlling the virtual robots, the homogenous transform describing the relationship between the mobile robot's base coordinate frame and the 'WORLD' coordinate frame must be calculated. This homogenous transform must be calculated and updated regularly during the motion of the virtual robots. A process was implemented whose purpose was to check that the robots did not pass through fixed objects such as walls, etc. This process is started by every motion command and monitors for collisions between the robot and the rest of the 'world'. If the virtual robots bounce into a wall the motion is stopped, and the robot stands in front of the wall. The commands for moving the robots and reading the sensors where made to have equivalent 'syntax' to the corresponding real robot.

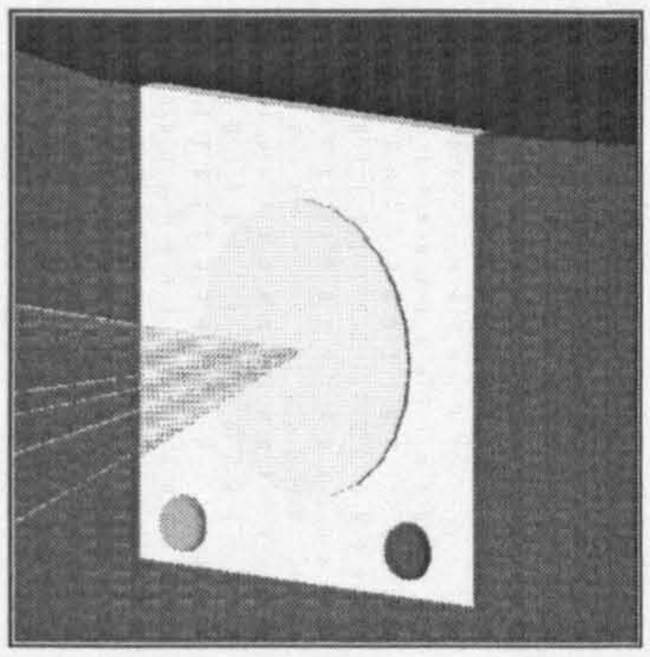


Figure 38: Virtual ultrasonic sensor on Nomadie

Table 28: Main procedure for simulated sensor on Nomadie

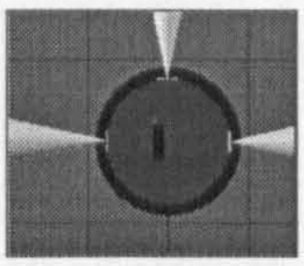
```
procedure SENSOR15_Sample();
Var distance: integer;
Begin
     Distance:=report_sonic_sensor15(647,'SENSOR15_RANGE');
     Distance:=roundoff(distance/2.54);
     SENSOR15_DISTANCE:=Distance;
End;

Sample_sonic_SENSOR15==mk_ticker(
     mk_application("SENSOR15_sample,
     emptylist(universal)),Sample_interval);
```

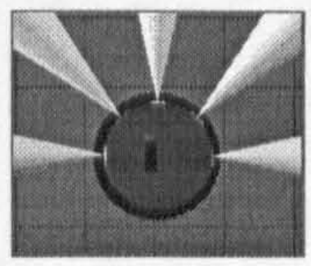
8.2.2 Training of Nomadie

The Virtual Nomad 200 robot, Nomadie, was taught the two control objectives; (i) move forward; avoid collisions and move out of corners; and (ii) follow corridors. Depending on the control objectives to be learnt, different ultrasonic sensors and different numbers of sensors were used in the ultrasonic sensor ring, as illustrated in figure 39.

This resulted in two different network architectures to train, as illustrated in figure 40 and figure 42.



a) Nomadie with 3 sensors used



b) Nomadie with 5 sensors used

Figure 39: Sensor configurations on Nomadie

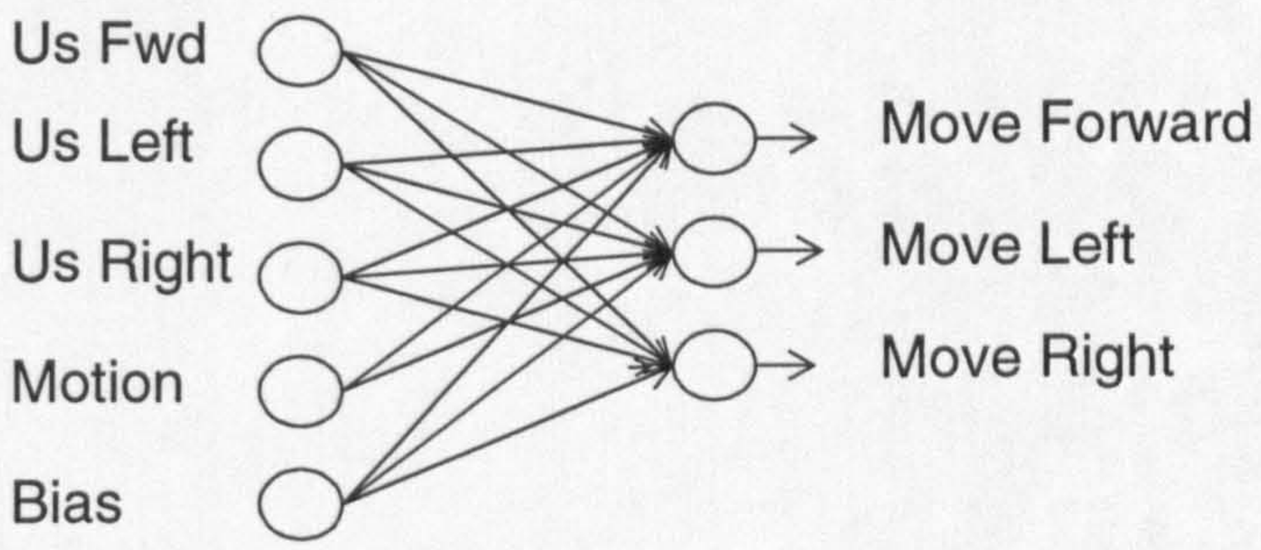


Figure 40: Artificial neural network for learning Nomadie control objective I

8.2.2.1 Train Nomadie for control objective I

The objective was to teach the robot to move forward if no other instinct was violated, The second instinct rule was to avoid collisions. Move forwards is the first instinct rule to be learnt.

By using just these simple instincts and appropriate sensor configurations, the robot can learn in which direction to turn to get out from corners without getting trapped. The behaviours possible to accomplish are illustrated in figure 41A. Figure 41C illustrates what is considered to be a bad action when encountering a corner as depicted in figure 40A and figure 41B illustrates a good action, which is accomplished after learning. The priority of the instinct rules was set, so that the avoid collision rule must be satisfied before trying to satisfy the move forward instinct rule. Three sensors, front, right and left (figure 39A) were used to accomplish these behaviours. The network trained is illustrated in figure 40. The inputs are the three ultrasonic sensors and a motion input.

The input node **motion** was fed with information about the robot's motion direction. The

current motion direction is indicated with a value of *1* for moving forward, *2* for moving left, 3 for moving right and 0 for no motion. The actions left and right, turn the robot 90 degrees in the respective direction. All input data were normalised to values between 0.0 and 1.0. Training was started with a randomly initialised weight matrix in the neural network. The robot was positioned at a random location in the room (with the forward direction aligned with a wall). The controller begins by reading the sensor values and the motion direction variable, these values are checked against the instinct rules. As the learning starts with the robot standing still, the rule move forward is broken. An input vector is produced from the sensor values and the motion variable, this vector is propagated through the network. The output nodes are selected by strength, the strongest first. When the move forward node is selected and executed the violation is stopped and an input-output pair to be used by the back-propagation algorithm is established. When the robot moves forward no instinct rule is broken until the robot encounters a wall (i.e. it detects an object with which it will collide if it continues moving forward). A new input vector is produced and propagated through the network. The objective is to identify the output node (action) which will stop the violation i.e. move the robot into a position as illustrated in figure 41 B.

At the beginning of the learning phase the robot (in most cases) first tries to move forward as this was the first instinct learnt. When the robot is at a position where a collision had been avoided, a new input vector is produced and the controller detects that the move forward instinct is violated and tries to avoid this violation. The behaviour of moving forward was learnt after approximately 5-10 iterations (learning steps). The mapping between the avoid collision instinct and appropriate action means that the relationship between the sensor values and the turning direction is determined. The initial mapping of this relationship was achieved after 10-30 learning steps. Initial mapping means that it learns to move in the correct direction with these particular starting conditions, such as clock-wise or counter clockwise starting direction and with the properties of this particular room (distances detected by the sensors). After 50 learning steps the move direction was changed 180 degrees. When the avoid collision instinct was violated the network could not map the input to a correct output pattern. The robot moved in the new direction for 50 learning steps and eventually manages to determine the correct mapping. The move direction was then again changed 180 degrees with the result that the controller had 'forgotten' some of the initial mapping for this direction.

When the controller had learnt the mapping sensor values-turning direction (this decision was

based on observation of the robots behaviour) the robot was moved to rooms of other sizes. As the ultrasonic sensors on the Nomad 200 robot have a detection range of 431 to 6477 mm it was necessary to alter the robot between virtual rooms of different sizes to allow faster training and better generalisation. This provides the artificial neural network with a broader set of training data and enables it to generalise better. Training the robot in one room, only provides the network with a limited range of input data from the sensors, thus not training the artificial neural network to generalise. The syntax of the instinct rules are illustrated in table 29. The training was performed for three different randomly initialised networks. The amount of training steps and training time each network needed to generalise and obtain an acceptable performance (network error level < 0.10), are shown in table 30. When the target value for the network error was reached ten consecutive times the robot was positioned in a room which had not been used in the training. The robots behaviour and the network error were monitored and if the network was able to solve the situation while keeping the error level then the network was considered to have generalised. This test was performed for each of the three networks

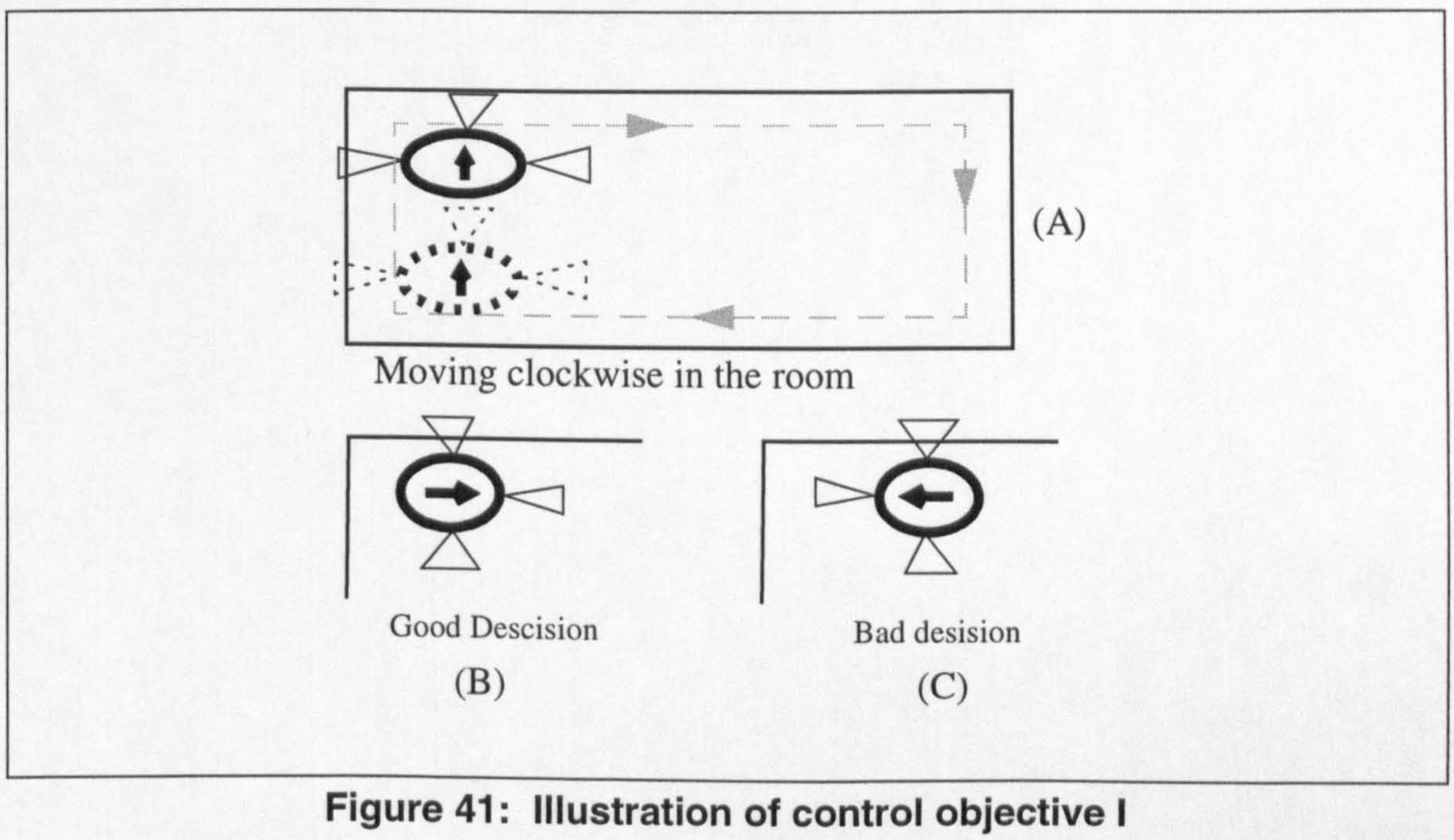


Figure 41: Illustration of control objective I

Table 29: Instinct rules for control objective I

```
Begin
     If (US1_Value<30) /* Forward sensor 30 inches*/
     Then Violation:=Collision;
     Else Violation:= None;
     If (Violation=None) Then
     Begin
          If motion<>Move_forward
          Then Violation:=Not_Forward;
          Else Violation:=None;
     End;
End;
```

Table 30: Training data on nomadie for objective I

Learning process	Learning Steps	Learning Time(hours)	Error level
1	967	16	0.08
2	1340	28	0.10
3	1103	23	0.09

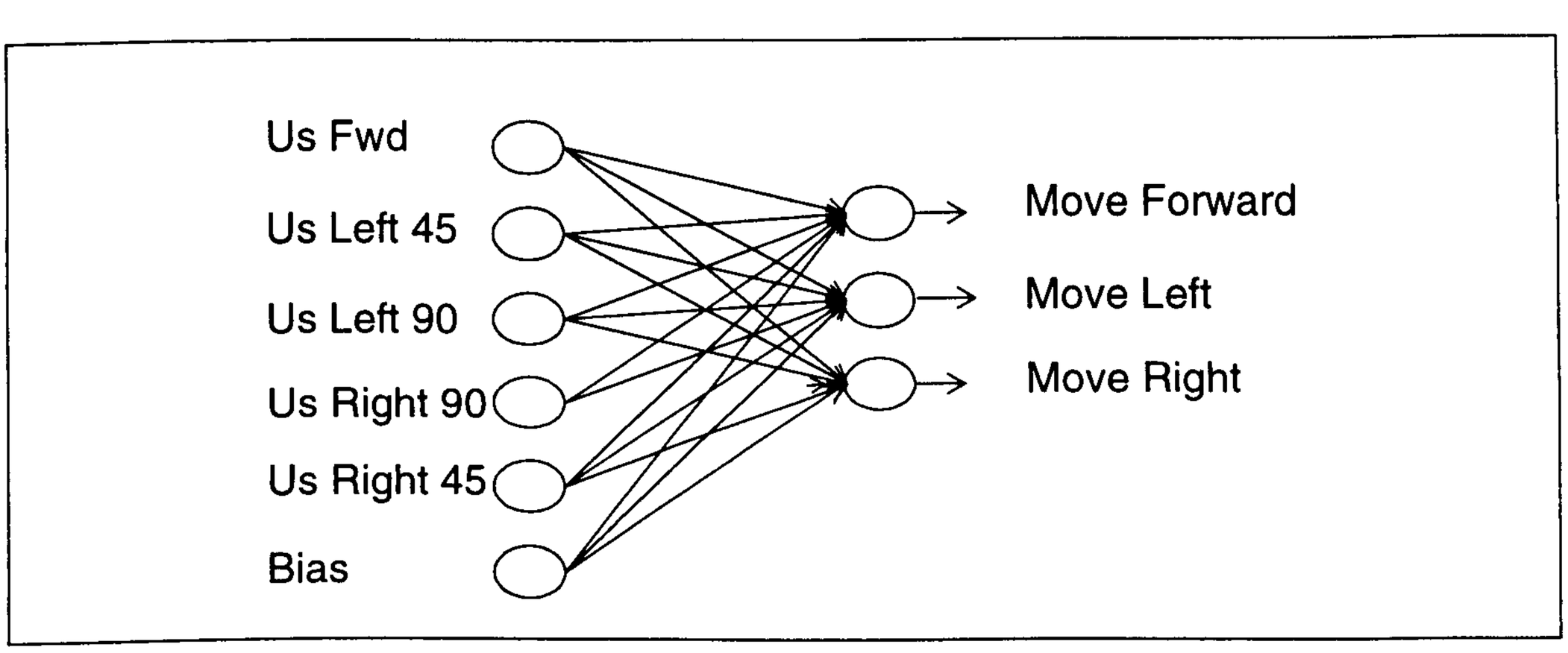


Figure 42: Artificial neural network for learning Nomadie control objective II

8.2.2.2 Train Nomadie for control objective II

The virtual Nomad 200 robot was taught control objective II using 5 ultrasonic sensors, arranged as illustrated in figure 39B. The instinct rule was designed to keep the difference between detected distance from sensor 5 (Us left 90) and 13 (Us Right 90) less than 380 mm.

There was no particular reason in choosing this value. The action set used were move forward, move left and move right. The actions left and right turn the robot 10 degrees in the respective direction. The network is illustrated in figure 42. The weights of the artificial neural network were randomly initialised. The learning rate used in the backpropagation algorithm was set to 0.3.

The training of the robot started by positioning it at a random location in a training corridor. The position was not in the middle of the corridor. The sensor values were read by the controller, which compared them with the instinct rule. As the controller detects a violation an input vector was produced from the sensor values and propagated through the neural network. The strongest output was selected and the corresponding action was performed. If the action stopped the violation against the corridor following instinct or produced a difference between the sensor readings that was less than the previous difference an input-output pair was established and the weights of the network were adjusted. If no adjustment took place the second strongest input was choosen and then the third. When an adjustment had been performed (one learning epoch) the sensors were read again to check for a violation, if no violation occurred the robot moved forward 300mm and the sensors where checked again. If the robot moved into a wall the robot was placed at a new starting position. When the robot performed adequately in the first training environment (this was determined through visual observations of the robot), the robot was subsequently moved to three other corridors to continue training. To provide a wider range of input-output pairs for the training of the neural network all of these corridors had different widths between the walls. When the networks had reached an error level of 0.1 or less for ten consecutive times, the robots were positioned in two test corridors, illustrated in figure 43.

These environments provided test sets of input-output pairs, for evaluation of the training of the networks. The evaluation was performed by observation of the robots behaviour, the criteria was that the training was successful if the robot managed to traverse the test environments. Three random initialised and untrained networks where used to establish the difference in the behaviour of trained and untrained networks. The three networks all managed to traverse these new corridors. The robot where not able to traverse the test corridors when the controller was loaded with any of the three random initialised networks. Picture sequences of successful and unsuccessful attempts to traverse one of the test corridors

(corridor B figure 43) are illustrated in figure 44 and figure 45 respectively. The syntax of the instinct rule is illustrated in table 31. The training time needed with different randomly initialised networks, for the artificial neural network to generalise and to perform acceptably are illustrated in table 32.

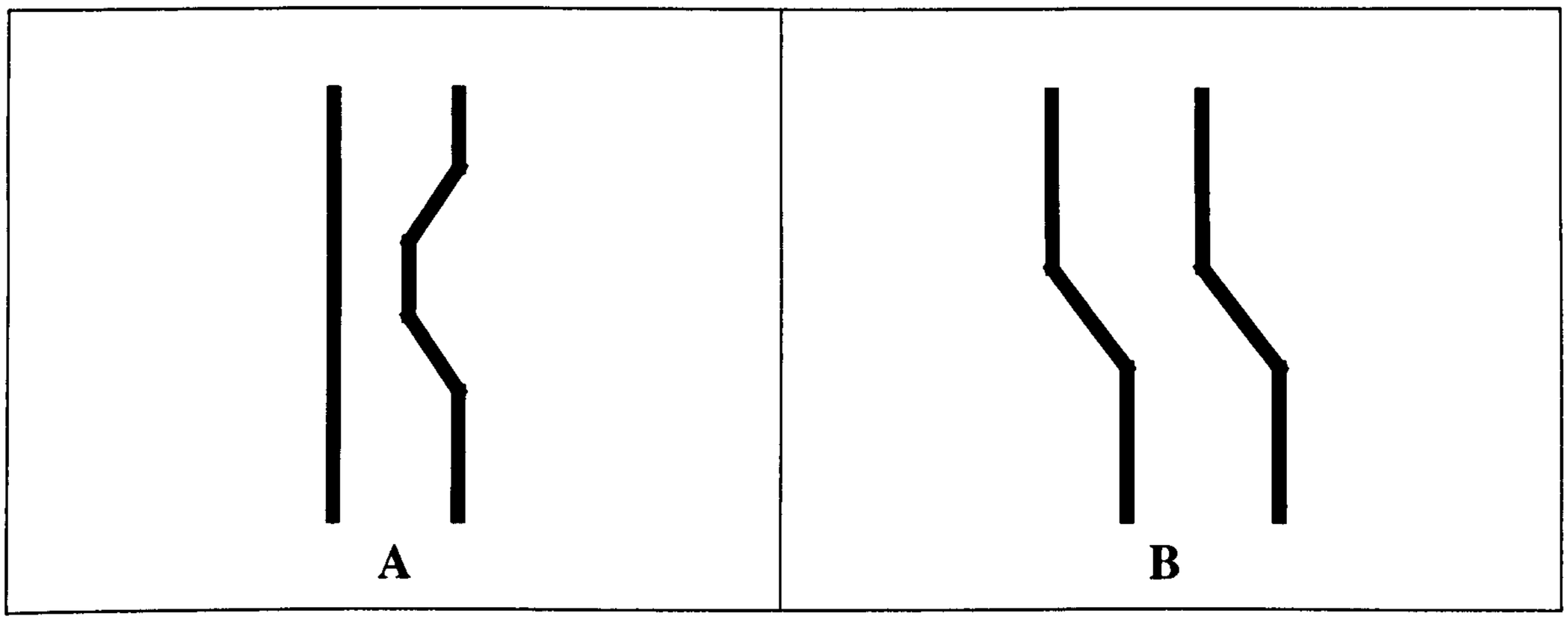


Figure 43: Test corridors used to verify the training of control objective II

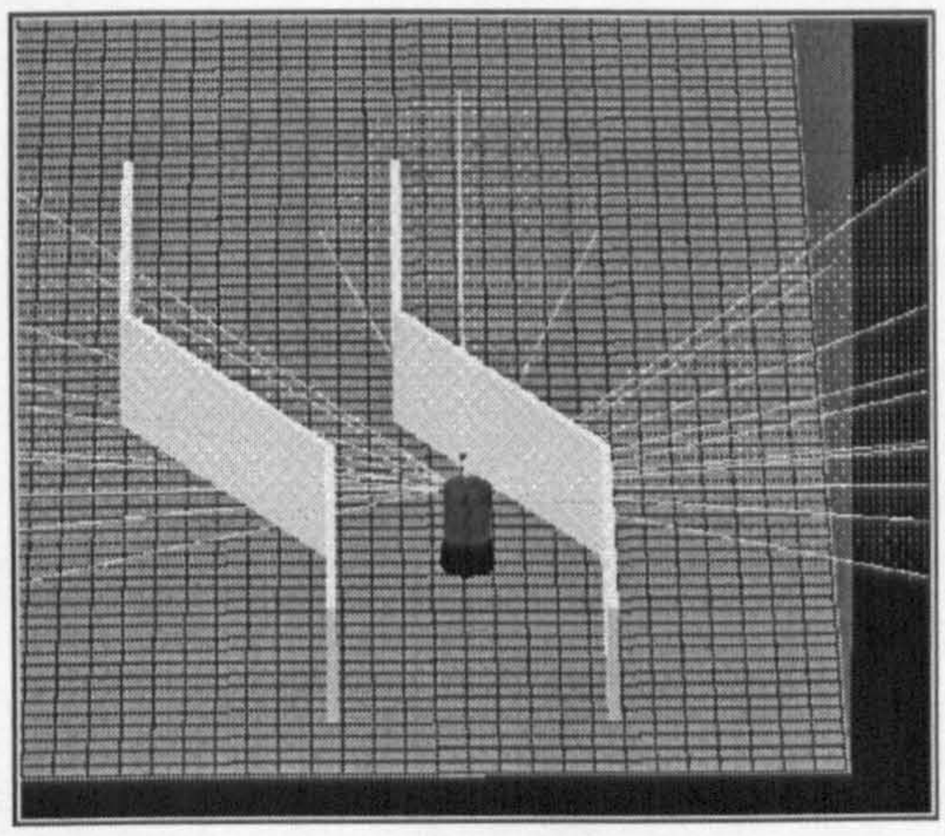
.

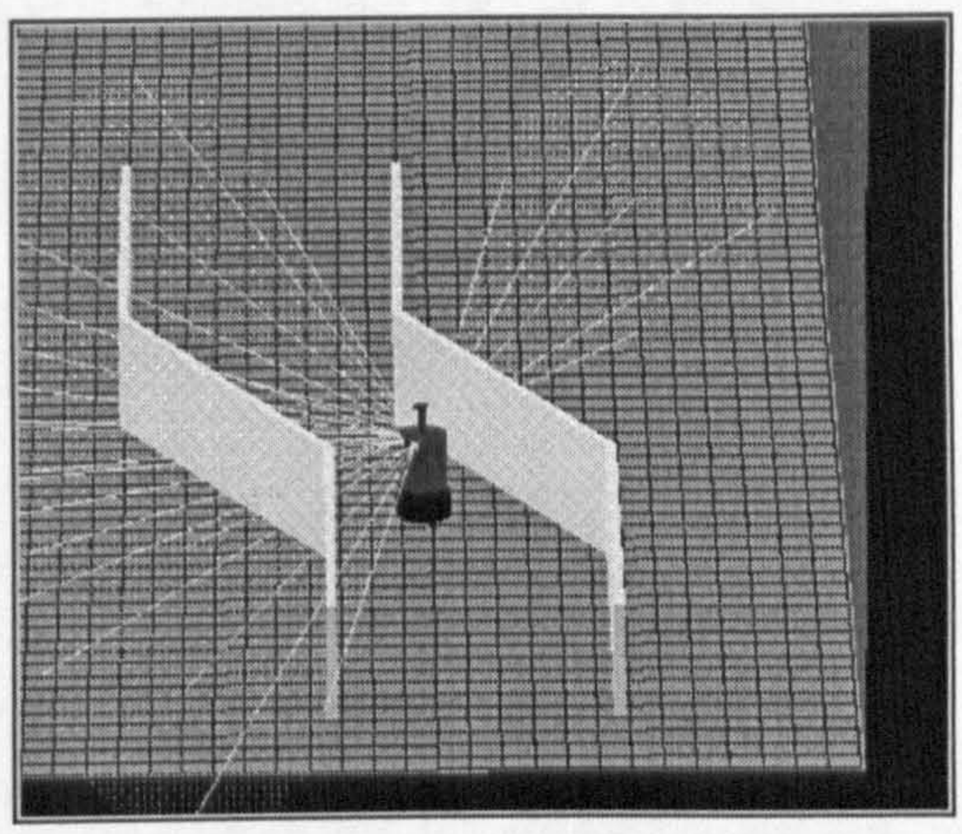
Table 31: Instinct rules for control objective II

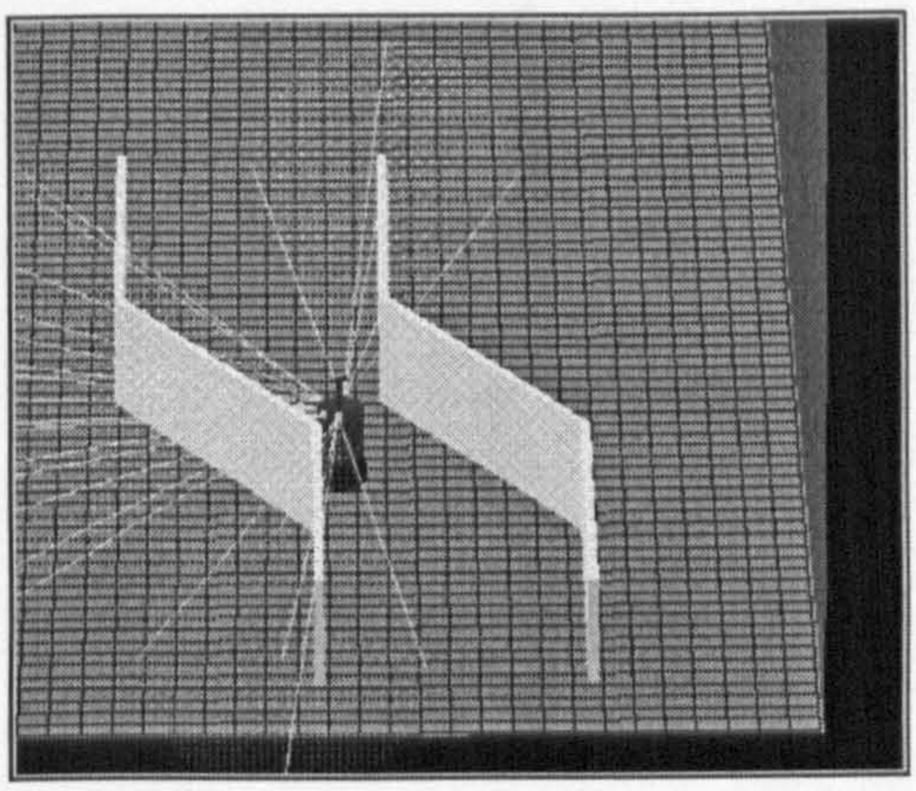
```
Begin
     Us_Diff:=ABS(US3_Value-US4_Value);
     If (US_Diff<15)
          Then Violation:=Not_Corridor;
          Else Violation:=None;
End;
```

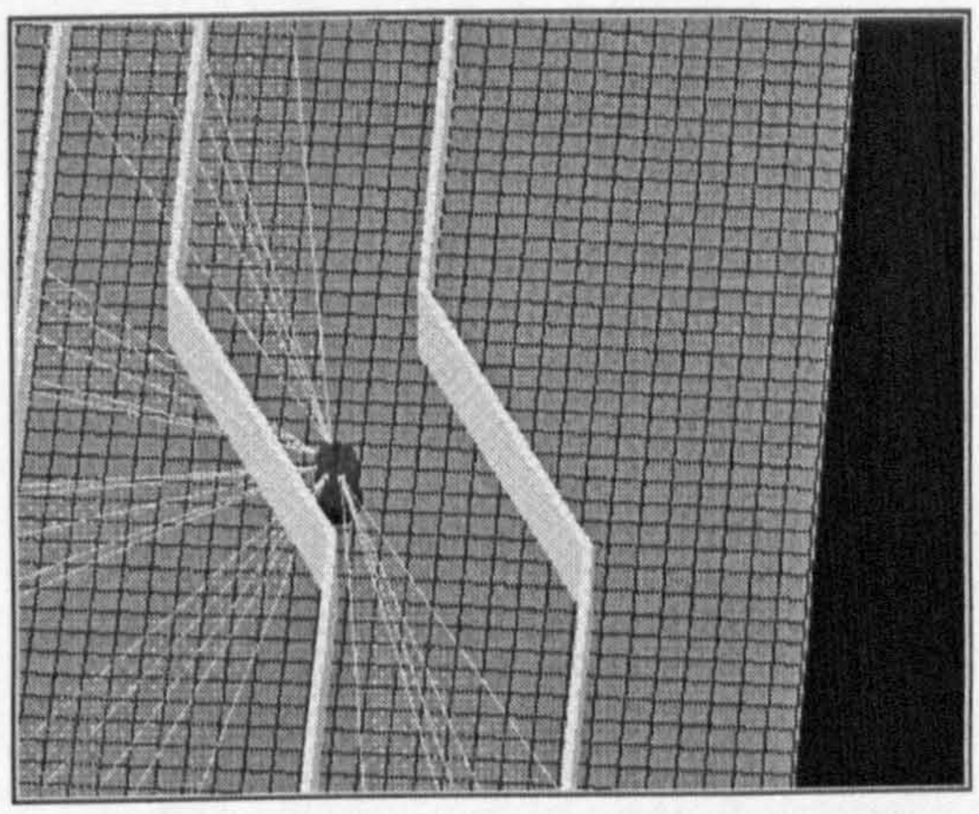
Table 32: Training data Nomadie objective II

Learning process	Learning Steps	Learning Time(hours)	Error level
1	1350	26	0.10
2	1680	32	0.09
3	1223	24	0.10









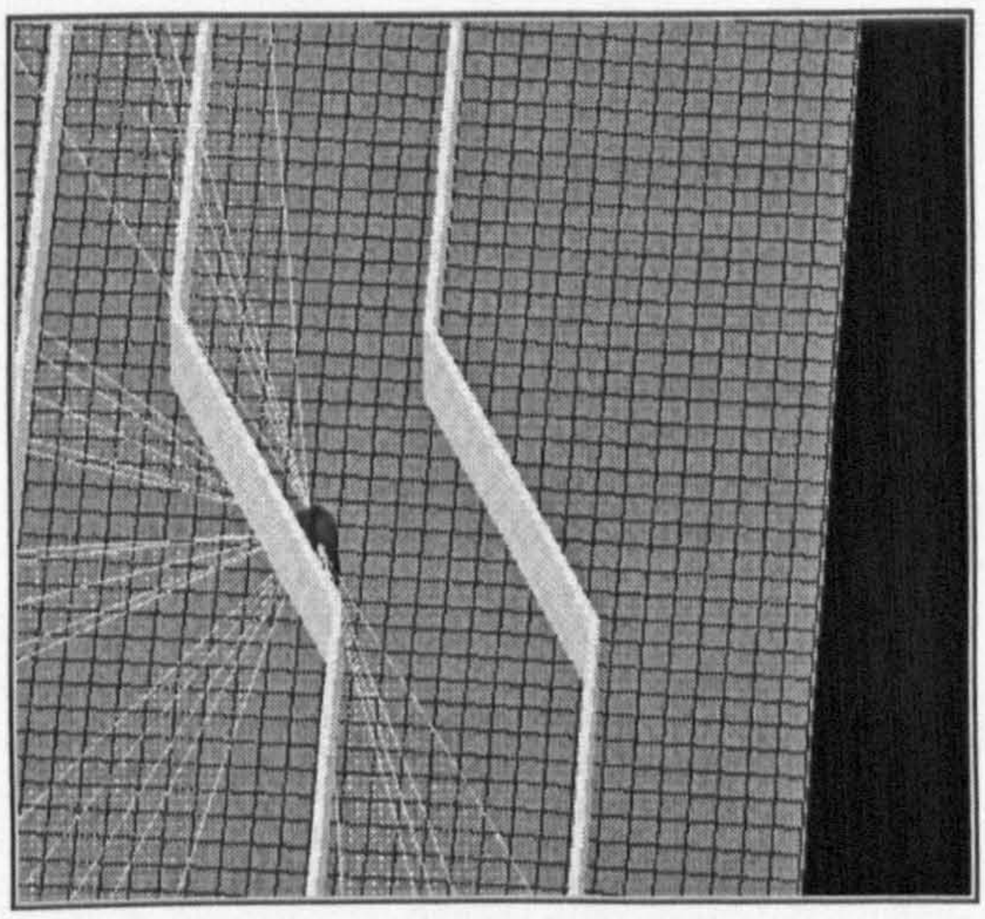
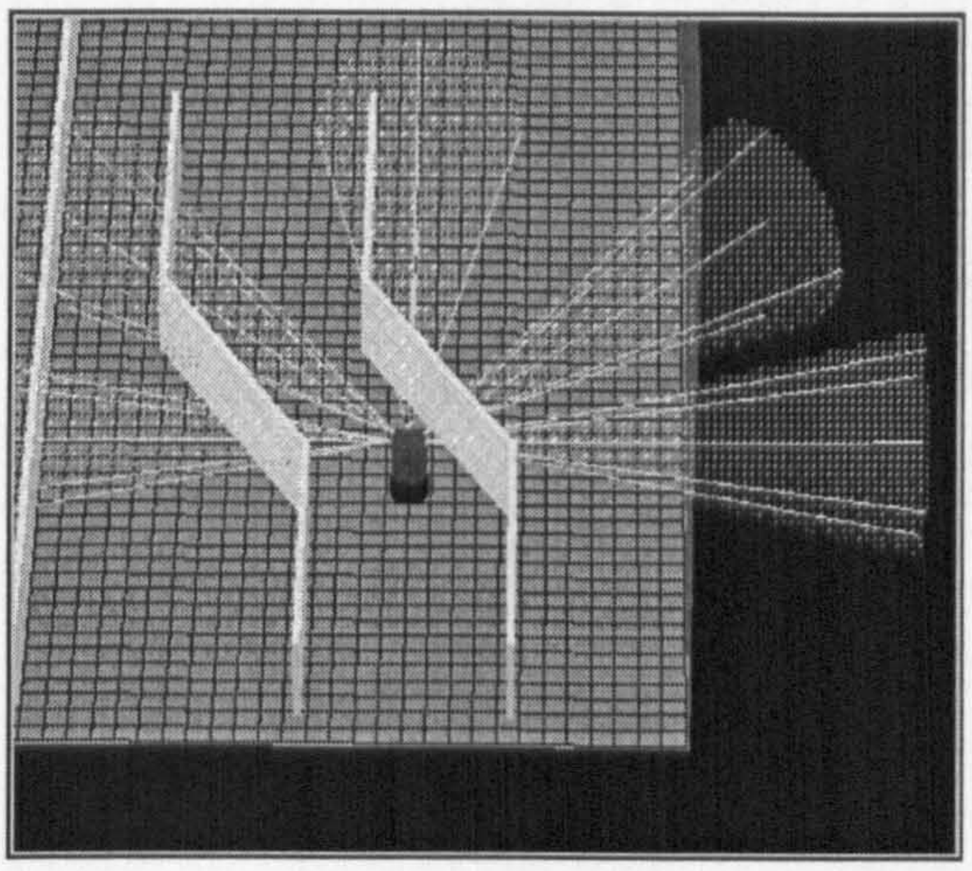
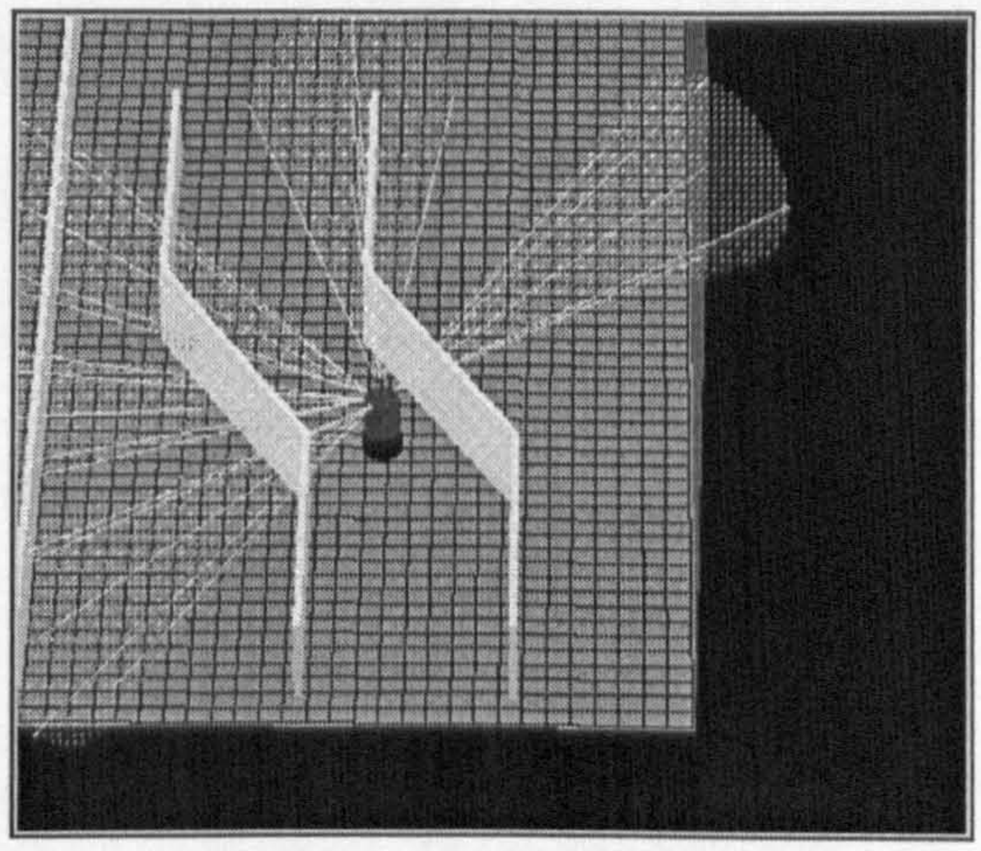


Figure 44: Untrained robot failing to follow an unknown corridor. (Motion sequence from A-E)

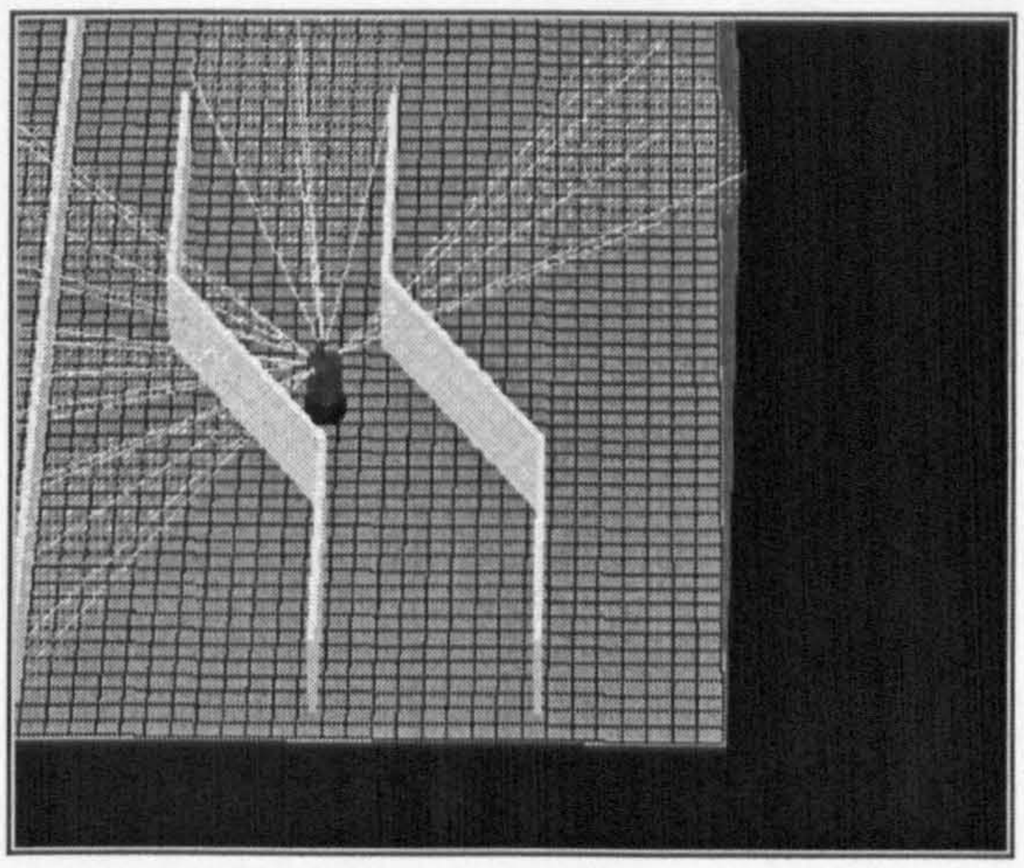
A



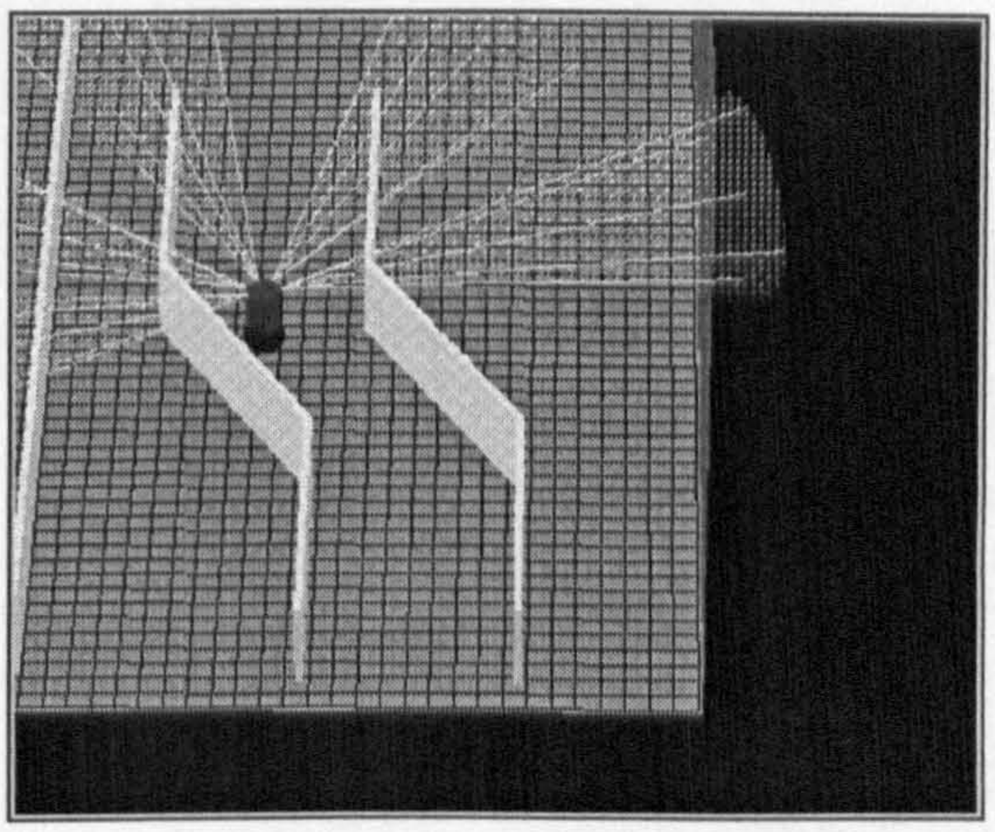
B



C



D



E

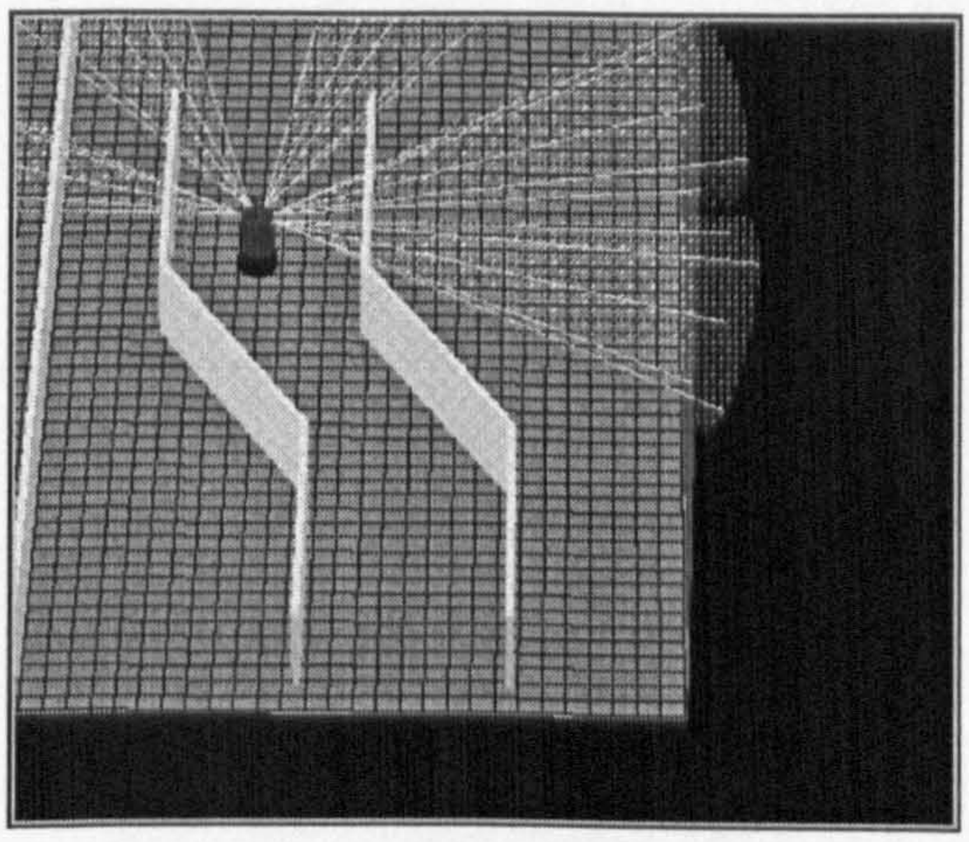
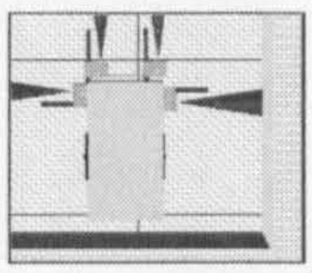


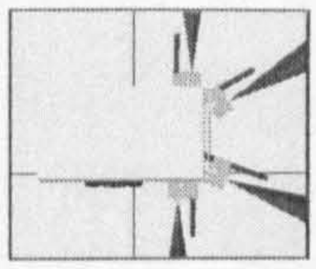
Figure 45: Trained robot successfully following an unknown corridor. (Motion sequence from A-E)

8.2.3 Training of Frankie

The Frankie robot was also taught the control objectives, (i) move forward; avoid collisions and move out of corners and (ii) follow corridors. Depending on the control objectives learnt, the sensor pods were arranged differently, as depicted in figure 46a and b. The two control objectives needed different numbers of input nodes. The architectures of the artificial neural networks used are illustrated in figure 47 and figure 48, which are fully connected perceptrons. A network where the last output state was copied to a memory layer (figure 49) was also used to train for control objective II. This architecture allows the network to learn sequential behaviours and multiple goals [Massone 1993], [Elman 1990].



a) Sensor configuration on Frankie for control objective I



b) Sensor configuration on Frankie for control objective II

Figure 46: Sensor configurations on Frankie

8.2.3.1 Train Frankie for control objective I

The objective, as with the Nomadie robot, was to learn the 'instinct' to move forward if no other instinct rule was violated, the second 'instinct' to learn was to avoid collisions. The instinct rules were set to the same priority as with Nomadie. The sensors where arranged as illustrated in figure 46a and b, with two sensors at each side of the robot and two sensors directed forward. The network architecture had 5 input nodes plus a bias node and three output nodes. Four input nodes for the sensors and one for detection of motion direction. The current motion direction is indicated with a value of 1 for moving forward, 2 for moving left, 3 for moving right and 0 for no motion. The actions left and right turn the robot 90 degrees in the respective direction. All input values were normalised to be values between 0.0 and 1.0. The robot was set to move 50 steps in each direction clockwise and counter-clockwise before changing to the other. The experiments where carried out in the same manner as with

the Nomadie robot (described in section 8.2.2.1). The sensors on the Frankie robot have a detection range of 100-2200 mm, which determined that rooms of different sizes (compared with the ones used with the Nomadie robot) where used for the training exercise. Three different randomly initialised networks where trained. The training was stopped when the network error reached a value equal or less than 0.10 ten time consecutively. The training results are shown in table 33. To evaluate the training the robot was positioned in a room which had not been used in the training. The robots behaviour and the network error were monitored and if the network was able to solve the situation while keeping the error level then the network was considered to have generalised. This test was performed for each of the three networks.

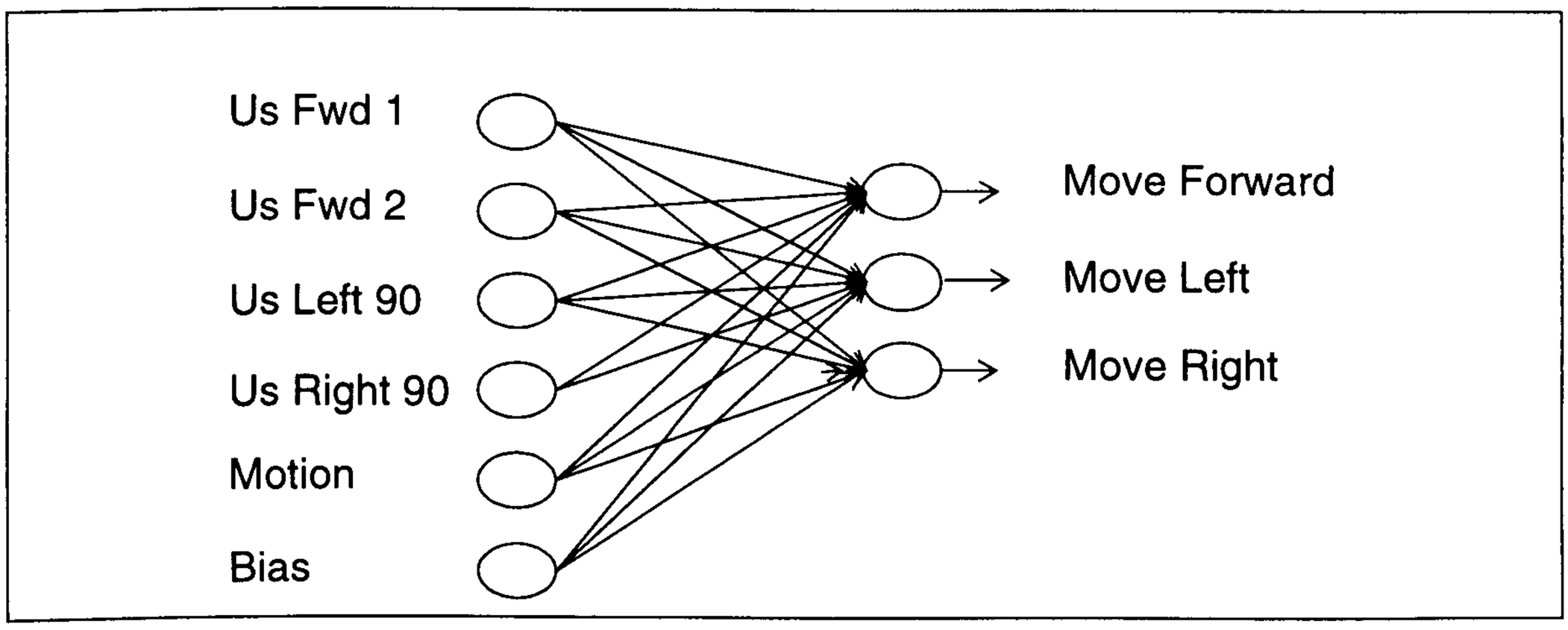


Figure 47: Neural network architecture in Frankie for control objective I

Table 33: Training data Frankie objective I

Learning process	Learning Steps	Learning Time(hours)	Error level
1	1350	26	0.10
2	1680	32	0.09
3	1223	24	0.10

8.2.3.2 Train Frankie for control objective II

The Frankie was taught control objective II (follow corridors) in the same environments as the Nomadie robot. The instinct rule was designed to keep the difference between sensor readings from the ultrasonic sensors in pod 1 and 4 within a certain tolerance. The tolerance was set to 500 bits, out of 4095 bits. The rule was violated if the difference became larger than 500. The coded instinct is illustrated in table 34. The sensor was mounted according to figure 46 b. The action set used where move forward, move left and move right. The actions left and right turn the robot 10 degrees in respective direction. The network architectures are illustrated in figure 48 and figure 49. The artificial neural network was randomly initialised at the beginning of the training phase. The learning rate used in the backpropagation algorithm was set to 0.2.The experiments where carried out in the same manner as when training the Nomadie robot control objective II (described in section 8.2.2.2). Three different random initialised networks where trained. Training times and the number of learning steps of the robot and its artificial neural network are listed in table 35. To evaluate the training the robot the same test environments was used as with the Nomadie (illustrated in figure 43). The evaluation was performed using the three trained and three untrained networks.Using any of the three trained networks the robot managed to traverse these test corridors. The robot where not able to traverse the test corridors using any of the three random initialised networks. Picture sequences of unsuccessful and successful attempts to traverse one of the test corridors (corridor A figure 43) are illustrated in figure 50 and figure 51 respectively.

The more complex network architecture was also trained using three different random initialised networks. The same set of training experiments where carried out. Using this network architecture resulted in longer training times (table 36) and for this control objective did not provide significantly better performance. The same test environments where used for this configuration. The architecture may well perform better for more complex learning strategies.

Table 34: Instinct rules in Frankie for control objective II

```
Begin
     If US_DIFF>500
     Then Violate_Corridore:=True;
     Else Violate_Corridore:=false;
End;
```

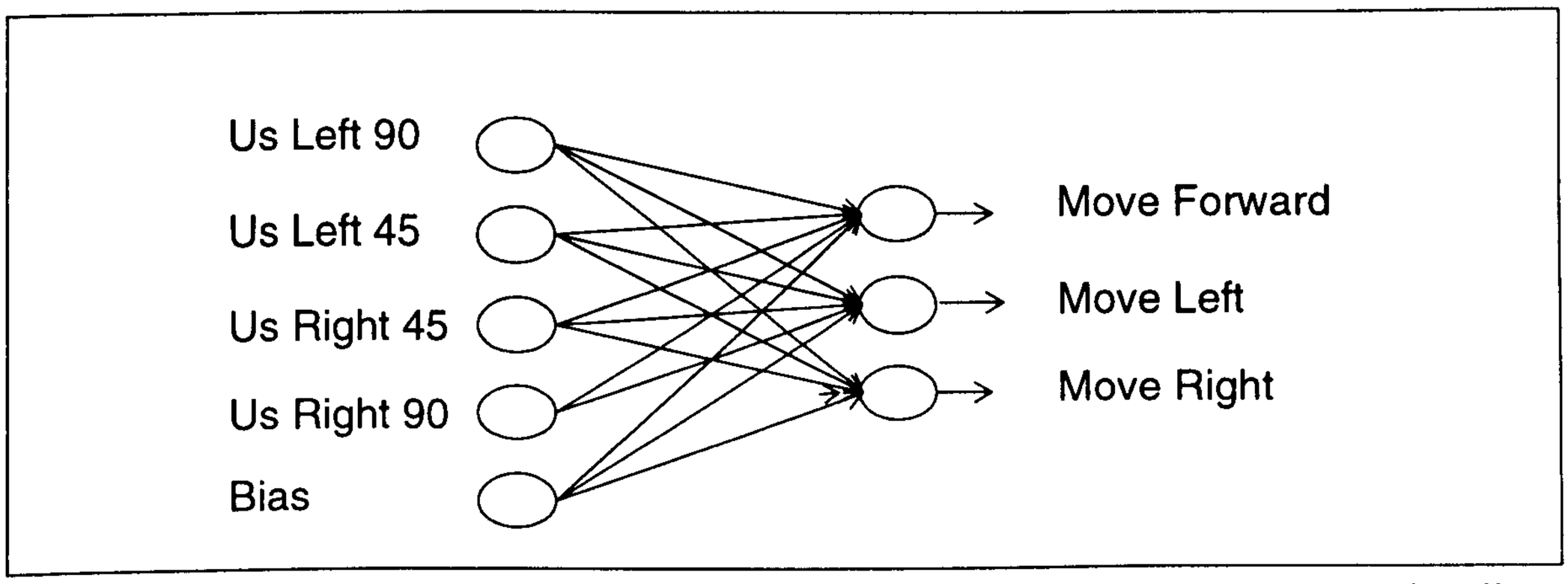


Figure 48: Neural network architecture in Frankie for control objective II

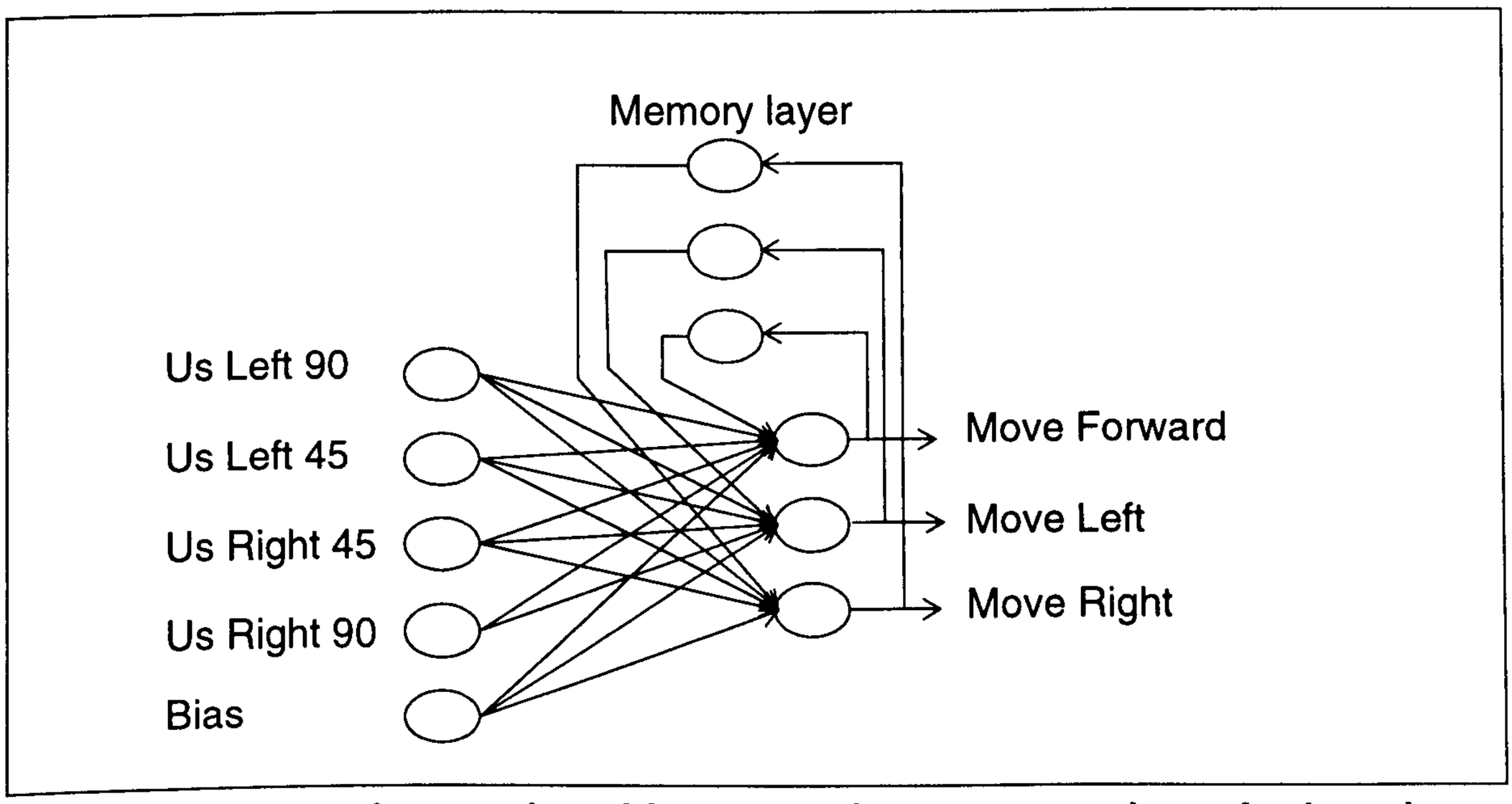


Figure 49: Neural network architecture using a memory layer for learning sequences, used in learning Frankie control objective II

Table 35: Training data Frankie objective II

Learning process	Learning Steps	Learning Time(hours)	Error level
1	1350	26	0.10
2	1680	32	0.09
3	1223	24	0.10

Table 36: Training data Frankie objective II using a memory layer

Learning process	Learning Steps	Learning Time(hours)	Error level
1	2130	36	0.08
2	2020	35	0.10
3	1660	31	0.10

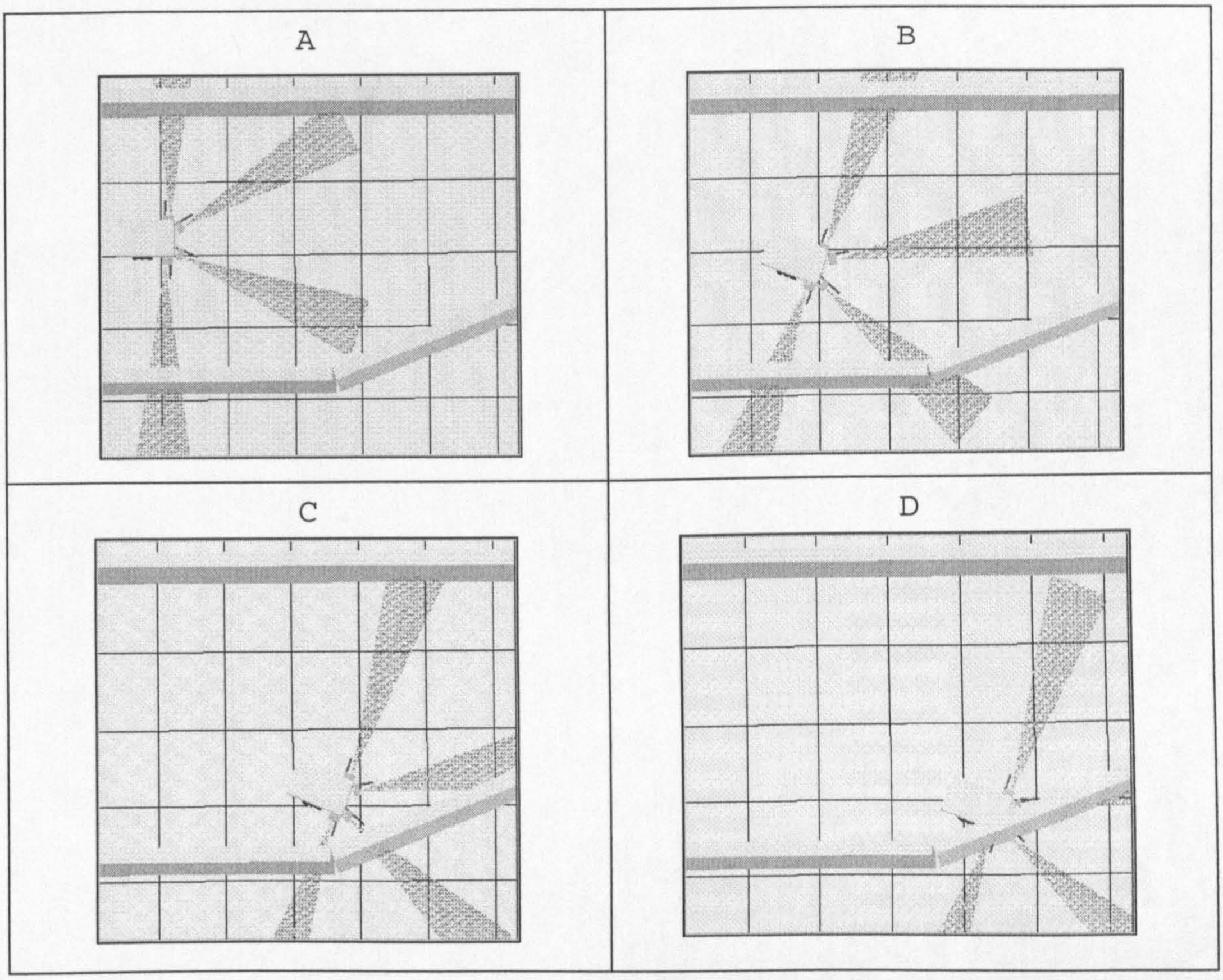


Figure 50: Frankie trying to negotiate an unknown corridor with two different randomly initialised and untrained networks in Frankie's controller. (Motion sequence a-d and e-j)

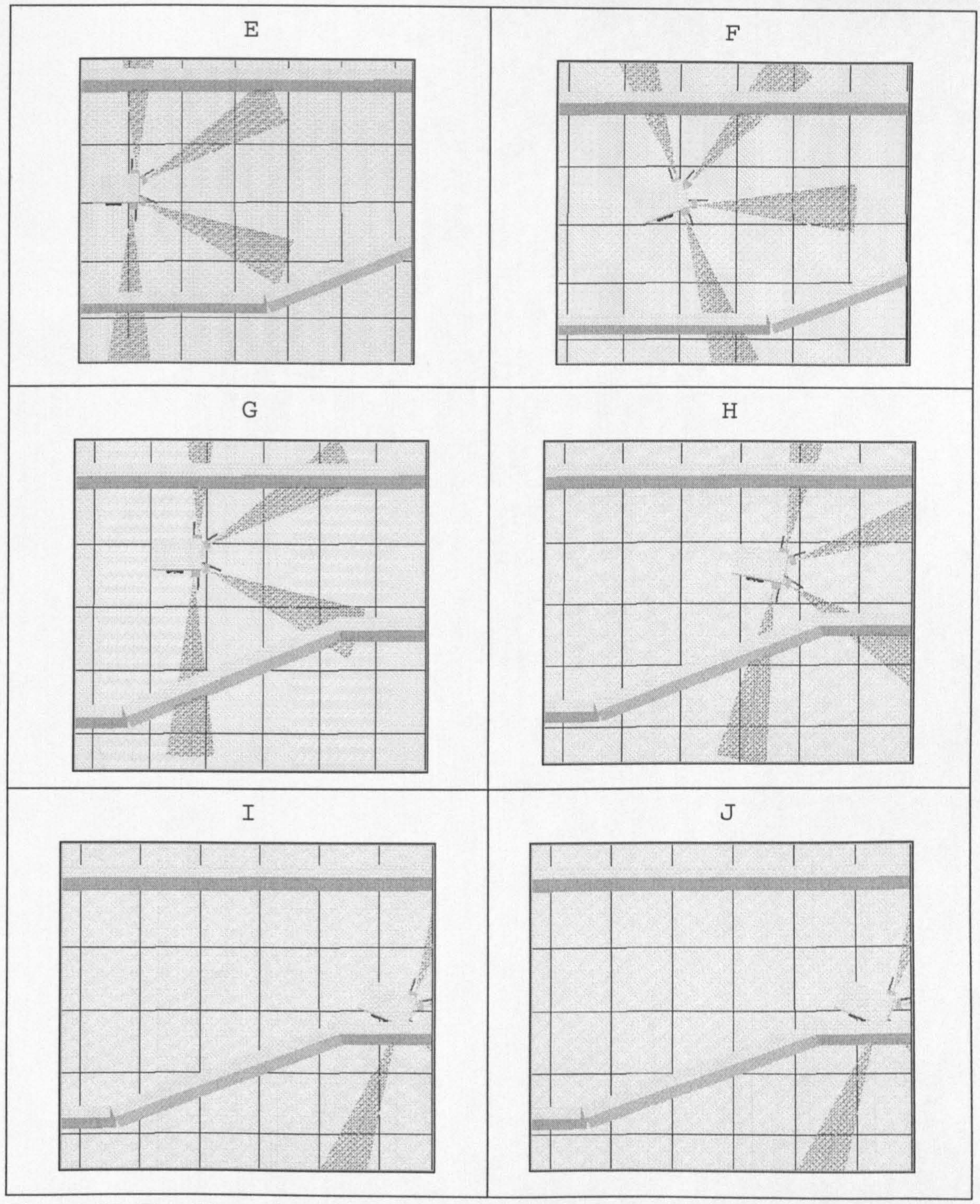


Figure 50: Frankie trying to negotiate an unknown corridor with two different randomly initialised and untrained networks in Frankie's controller. (Motion sequence a-d and e-j)

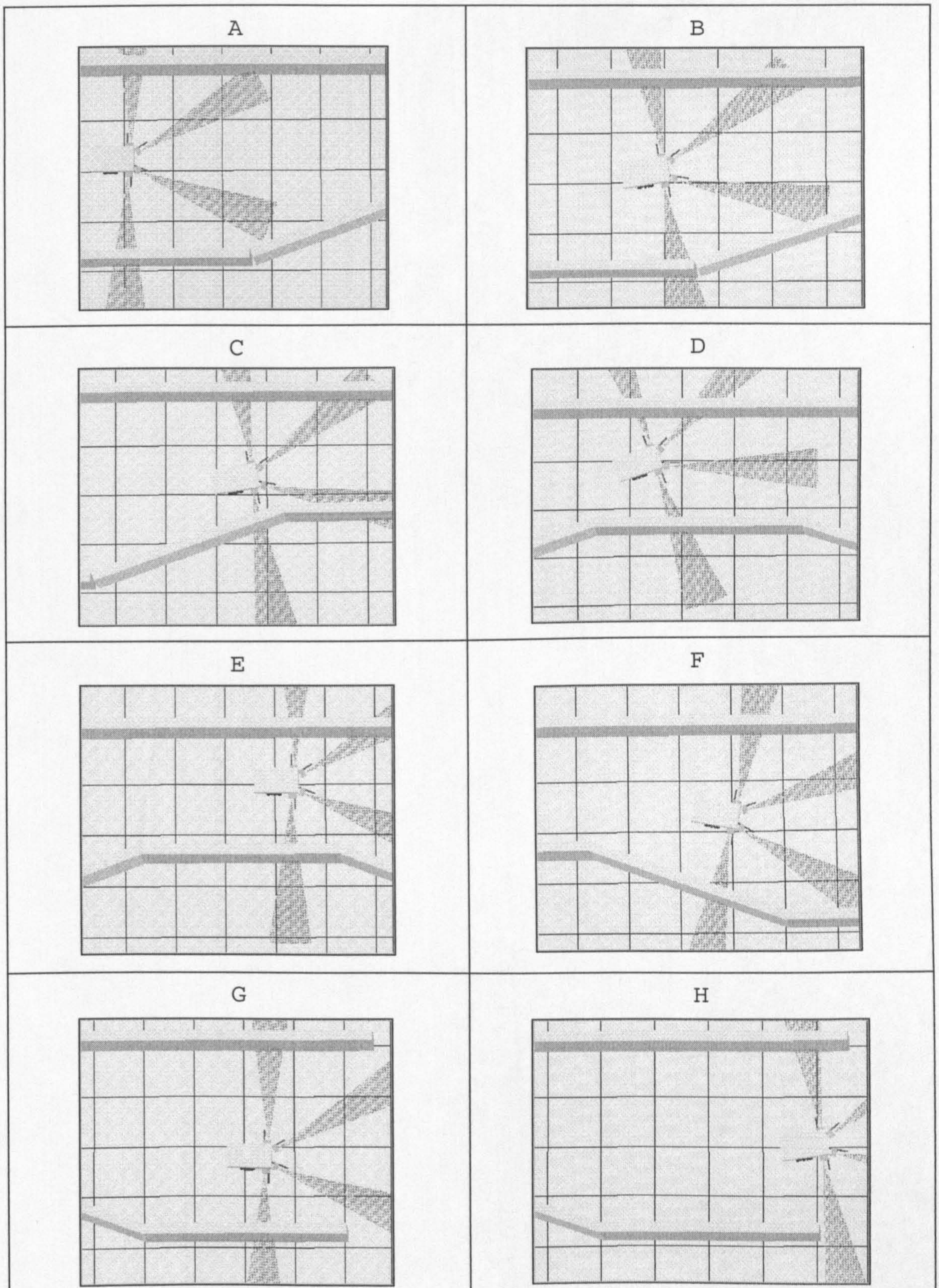


Figure 51: Trained Frankie negotiating an unknown corridor successfully. (Motion sequence, A-H)

8.2.4 Transferring trained ANNs

The artificial neural networks trained in the virtual environment are written to files. The files contain the adjusted weights of the network, which are real numbers. An example of a weight matrix of trained network is illustrated in table 37. The files were then transferred to the on-board computers of the robots. The trained network data is then read from the file during the initialisation phase for the robot controller.

Table 37: Weights of ANN for controlling Nomadie for objective I

3.0482	-2.1261	-1.5926
-0.0700	1.8066	-1.5813
-0.3647	-1.5101	2.2930
2.9331	-2.3152	-2.1373
-2.4647	0.4235	-0.1275

8.3 Evaluation on the real robots

The real robots Frank2 and Nomad 200 have their control architectures implemented in Pascal and C respectively. To verify the 'pre-emptive learning' the robots were exposed to new environments with characteristics similar to those of the virtual environments. The aim was to establish if the controllers had been able to generalise from the knowledge acquired in the virtual environment. The control strategies I and II (as previously described) were implemented in both robots and the performance of each was evaluated. Each robot was tested with both randomly initialised neural networks and with the artificial neural networks which had been 'pre-emptively learnt' in the virtual environment placed in the decision maker module of the controller.

To verify that control strategy I had been learnt, the robots were placed in a room. The robots should be able to move clockwise or counter-clockwise along the walls of the room avoiding collisions. The control architecture was formulated with instinct rules for control objective I. Firstly an untrained randomly initialised artificial neural network was loaded into the

controller. As to be expected the robots failed to accomplish their tasks and needed to be trained from scratch. Typical learning progress, on the Nomad robot, with randomly initialised weights is illustrated in table 38. The table illustrates the learning status after one "battery cycle" (that is starting with fully charged batteries and running the robot until the batteries are discharged). The robot needed some resetting, as it became "trapped" in the early stages of learning. The networks trained in the virtual robots were then loaded into the respective robots and each robot was able to accomplish their assigned tasks. The robots needed some training time to refine the weights of the ANN to match the specific properties of the real room. Both robots where tested with the three networks trained for respective robot.

Table 38: Typical learning progress on real Nomad robot, with randomly initialised weights.

Learning time (1 battery cycle)	**Learning steps**	**Error level**
2 h. 32 min.	98	0.34

To verify the learning of control strategy II the robots were placed in several corridors all which differed in properties, such as width and shape. The robot controllers were formulated with instinct rules for control objective II. The robots were placed at random locations in the corridors. With untrained randomly initialised artificial neural networks in their decision makers the robots could not accomplish the task, of following the corridors. The robots exhibited similar behaviour to the corresponding virtual robots in their initial stage of training, that is they constantly moved into walls. The files with the artificial neural networks trained in the virtual environment were loaded into their respective robot controllers.The robots were now able to respond with the correct actions to prevent violation against the defined instincts. Both robots where tested with the three networks trained for respective robot.

The networks trained with simulated sensors were shown to be able to map real sensor readings to correct actions to confirm the thesis proposed earlier in the chapter.

8.4 Summary

Graphical simulation can be used to provide initial knowledge for robotic systems with neural controllers before the final learning phase on the real robot is undertaken. In many instances it is impractical to train sufficiently on the real robot because of constraints such as limited time, availability of the robot, limited battery life, limited access to the working environment, safety considerations, etc. Furthermore, the use of realistic 3-D simulation of robots, sensors, tooling and the environment provides an opportunity for realistic evaluation of new theories and methods in the field of neural networks.

Experiments have shown it possible to train robots controlled by learning regimes (such as artificial neural networks) in virtual robotic environments using simulated sensors. Robots of two varieties equipped with different sensors and alternative combinations of sensors have been trained. The experiments have included training of five different network architectures, all of which have been able to generalise using information from simulated sensors.

The neural networks have been able to make decisions with acceptable performance as soon as they are transposed to the real robot and fed with data from real sensors. Further fine tuning of learning within the real environment can then proceed as necessary. It is anticipated that the inclusion of material properties within the objects in the virtual environment would further enhance the performance of the 'pre-emptive learning' for robotic systems. The generic sensor model would then need some additional functions for identification of the detected object's material properties and adjustment of the sensory data according to the material properties.

8.5 References

[Alami et al. 1992] Alami R., Chatila R. and Ghallab M., *Mission Planning and Control for Autonomous Mobile Robot*. Tolouse Cedex-France 1992.

[Bergolte 1994] Borgolte U., *The new German standard robot programming language: IRL*. Proceedings 25th International Symposium on Industrial Robots, 1994.

.

[Bernhardt 1994] Bernhardt R., *Realsitic Robot Simulation (RRS) Initiative, Technical Description*. IPK Berlin 1994.

[Brooks 1986] Brooks R.A., *A Robust Layered Control Systme For A Mobile Robot*. IEEE Journal of Robotics and Automation 1986, Vol.2, No.1 pp 14-23.

[CimStation 4] *Developers Guide*. Silma Inc, Cupertino CA, 1994.

[CimStation 5] *Command Reference*. Silma Inc, Cupertino CA, 1994.

[Crowley et al. 1991] Crowley J., O. Causse and P. Reignier. *Layers of Control in Autonoumouse Naviagation*. Conference on Robotics and Mechatronics, Aachen 1991.

[Eberhart and Dobbins 1990] Eberhart R. and Dobbins R., *Neural Network PC Tools*. Acedemic Press 1990, ISBN 0-12-228640-5.

[Elman 1990] Elman J.L., *Finding Structure in Time*.Cognitive Science 14 1990, pp 179-211.

[Engelberger 1989] Engelberger J.F., Robotics in sevice, Kogan Page,
ISBN 1-85091-358-7.

[Hecht-Nielsen 1989] Hecht-Nielsen R. *Neurocomputing*. Addison-Wesley 1989,
ISBN 0-201-09355-3.

[Massone 1993] Massone L.E., *A Biologically-Inspired Architecture for Reactive Motor Control.* Neural Networks in Robotics 1993, ISBN 0-7923-9268-X.

[Mataric´ 1994] Mataric´ M., *Interaction and Intelligent Behaviour.* PhD Thesis 1994 MIT, USA.

[Nehmzow et al. 1993] Nehmzow U., Smithers T. and McGonigle B., *Increasing Behavioural Repertoire in a Mobile Robot.* From Animals to Animates2, MIT Press 1993.

[Nehmzow 1992] Nehmzow U., *Experiments in Competence Acquisition for Autonomous Mobile Robots.* PhD Thesis Univeristy of Edinburgh 1992.

[Nnaji 1993] Nnaji B.O., Theory of Automatic Robot Assembly and Programming. 1993, ISBN 0-412-39310-7.

[Rumelhart and McClelland 1986] Rumelhart D. E. and McClelland J. L. *Parallel Distributed Processing: Exploratrions in the Microstructure of Cognition,* Vol. 1, MIT Press, Cambridge, MA, 1986.

[Van der Smagt 1996] Van der Smagt P., *A robot arm is neurally controlled using monocular feedback.* Self Learning Robots, IEE Colloqium Digest, Savoy Place London 1996.

[Van de Velde 1993] Van de Velde W., *Toward Learning Robots.*Toward learning Robots, The MIT Press, 1993, ISBN 0-262-72017-5.

Chapter 9 Conclusion

This chapter includes a discussion section a section with recommendations for further work and ends with a section with the contribution to knowledge and the major conclusions related to virtual robotics that can be drawn from the research study.

9.1 Discussion

Robots must be able to react and reason about information from other devices, such as machines and sensors, if more self-contained, robust and flexible robotic workcells are to be introduced. Contemporary robot installations are, in general fixed to highly repetitive tasks and/or advanced motion sequences, that are very limited in handling uncertainties and reasoning about new situations. The application of robotic systems to more diverse application areas such as robots for domestic use, robots in nuclear plants, flexible assembly cells etc., creates a demand for robots with better reasoning capabilities. Robots with better reasoning capability must use sensors to obtain information about their environment. For robots to be able to react to events and different states in an autonomous manner, the robot controller must be able to 'sense' the environment and then act according to these 'senses'.

Robots which are capable of handling complex situations, must have control programs which are able to 'reason' about events and future states, therefore these robot programs must contain a considerable amount of logic. Such control programs are difficult to develop and need to be thoroughly debugged and evaluated. Programming and debugging in the real workcell is not usually realistic as: (i) it is difficult to generate every possible state; (ii) on-line programming of logic is difficult; and (iii) resetting the workcell for each logic configuration is tremendously time consuming, thus too expensive to be a viable option. This dictates that such flexible robotic workcells should be programmed off-line using virtual robotic tools. To enable off-line programming of flexible robotic workcells, 'realistic' information about the robot and its interactions with the environment must be available. Flexible robotic workcells use sensors to detect environmental changes and states. The sensor interaction and reasoning about sensory information should be an integral feature in the off-line programming phase. This only becomes viable if virtual sensors are incorporated within the off-line

programming environment.

Most new buildings and environments are now designed using CAD and CAE systems. Future designs will almost certainly be developed using 3-D CAD, hence providing information that can be used for 'pre-emptive learning' by "*intelligent*" robots. By establishing links between 3-D design systems and virtual robotics tools, information about buildings, parts, etc. could be used by virtual intelligent robots for map-building and other information gathering tasks without the need to experience the real environment. The virtual environment would act like a '*school*' whereby basic knowledge and skills could be learnt and furthermore the knowledge could be tested and evaluated before exposing the real robot to the real environment and application.

When sensors are simulated in virtual robotics the interaction between the robot controller with its programs and the sensors must be representative of the interaction in the real environment, otherwise it can not provide information on which a reliable evaluation of the off-line generated program can be performed. This means that the program structure and statements must be comparable (if not identical) in both the virtual and real robot programs. If the program statements for sensory interaction used in the virtual world are simplified compared to the program statements used by the real robot these statements must be modified subsequently to add the necessary enhancements to the code after transferring it to the target system. This results in an significant amount of untested code thus dictating a need for a thorough on-line debugging phase, which severely limits the benefits of using virtual robotics. If sensory information has to be processed and transformed by the robot programs, for example to obtain a physical measure upon which a logic decision is made, the virtual sensors should provide information in the same format as the real sensors, which would enable the programming of the actual sensor interaction statements. An example of this is a range measurement sensor, which gives an output with a non-linear function between 0-10 volts. If the robot's action depends on the distance given by the sensor, then the distance must be calculated from the output of the device before an action is performed. Such calculations are often made in the robot program. Such sensor interaction can only be programmed, debugged and evaluated off-line if there is a realistic sensor simulation capability integrated into the virtual robotics tool.

A generic sensor model for simulation of sensors in virtual robotics has been proposed. The

generic sensor model is based on a geometric and a functional definition.

Geometric properties such as the sensor configuration and detection range are defined in the geometric definition. These properties are encapsulated within CAD models. Functional and operational characteristics such as output format, update rate etc., are defined using a generic sensor procedure and a template file. A sensor's detection volume is simulated with relatively simple objects, such as cylinders and cones. These estimated detection volumes are tested for intersection with other objects in the environment using collision detection. If there is a collision between the detection volume and an object in the environment, the relationship between the sensor's detection volume and the detected object is determined. Depending on the type of sensor simulated, different relationships such as distance to nearest surface of the detected object and the angle between the facing surface and the normal of the sensor are determined. These relationships are determined by calculating the interaction between the sensor and the detected objects. Trace-lines are used to provide a better estimation of the interaction between the sensor and detected objects, where the trace-lines are located within the detection volume. Each trace-line's interaction with the detected object is investigated. The more trace-lines that are used, the more accurate is the estimated relationships between the sensor and object in corresponding to the real relationship. The speed of execution of the sensor control programs is dependent upon the number of trace-lines used.

In this research study proximity devices, photoelectric devices and ultrasonic transducers have been simulated and the results verified by comparison with the corresponding real sensors. Virtual sensors created from the generic sensor model have been proven to simulate the corresponding real sensors with adequate results. With respect to the conventional use of sensors in robotic applications, such as detecting the presence of an object, distance to objects etc., the resemblance between the simulated detection volumes and the real sensor detection volumes are satisfactory. If better correspondence between virtual and real sensor characteristics is desirable, sensors characteristics must be simulated much more thoroughly.

Pomeroy et al. present work on the simulation of ultrasonic devices whereby each sound wave propagated is represented [S. Pomeroy et. al. 1993]. This is obviously very computer intensive, taking minutes to perform the simulation of each sound wave sent. Conversely, sensor simulations in 2-D simulators, as for example presented in [Erard et. al. 1995], are not relevant for most robot applications as robots operate in complex 3-D real environments.

Real sensors work within their own 'detection environments' and normally operate independently to other devices. Sensors operate in parallel and are only affected by events in their own 'sub-environment'. To duplicate the operation of sensors in real workcells, each virtual sensor created with the generic sensor model has its own control process, enabling the sensor to work independently from and concurrently with other processes in the virtual environment. The virtual sensors are activated and deactivated individually and are not controlled by the robot programs. The simulation speed will reduce with an increase in the number of sensors as more parallel activities must be catered for. If the virtual sensors are 'too detailed' this will also adversely affect the time for completing simulations. The simulation speed is directly related to the processing capability of the computer system, however this is not now seen as a major problem as faster computer systems are constantly being developed. The virtual sensors demonstrated in this research study have all been run in an interpretive mode. If each sensor's control program is compiled before execution the simulation speed would be increased.

Off-line programming of robot applications with a high degree of program logic, such as, event-driven robotic workcells requires off-line programming in the native language of the robot. Using one language for programming the virtual robots in the virtual robotics system and then having to use a post-processor for translating the program to the 'native' language of the real robot controller places severe restrictions on the process. Much of the logic in the program created in a general language is difficult to translate to a controller specific language. Many of the commands specific to the robot controller are also difficult to utilise. Conversely, implementing each controller specific language puts constraints on the flexibility when evaluating new installations etc., and it forces the programmers to use several languages. To gain benefits from the approach of using a general programming language it would be preferable if a standard language could be established in which both virtual and real robots are programmed. The German initiative IRL, which is based closely on the syntax of Pascal, provides a basis for such an approach.

Signalling and other 'physical' properties that occur in a real workcell must be catered for in the virtual environment. This can, for example, be solved by designing a blackboard architecture. A blackboard can store and display sensor states and other physical properties, ena-

bling global access to them. The demand on the sensors level of 'correctness' is not that high in most event-driven robot applications, it is more the actual interaction between the robot and the sensor that is of importance. This can allow less sophisticated models to be used for the simulation of sensors, thereby reducing the load on the processor running the simulation.

In response to demands for greater operational flexibility and responsiveness, 'event-based' control structures have been advocated. In this research study an event-driven robotic workcell was designed. Robot procedures and functions were created in the Pascal programming language and the same procedures and functions were made available in the virtual robotics tool, CimStation. The event-driven robotic workcell was programmed off-line using the virtual robotic tool, whereby sensory interactions, robot motions and program logic were programmed and debugged off-line. The programs were then down-loaded to the real workcell without any post-processing or additions to program statements. The virtual sensors were created using the generic sensor model. The virtual robotic system was expanded with features for realistic emulation of I/O, machine and sensory interaction. These features were 'hidden' to the programmer, thus minimising the differences between the language commands in the virtual and the real workcells. Utilising these emulation features, logic was extensively debugged and optimized. Using the same programming language for both the virtual and the real workcell enhances the potential for error free code to be down-loaded to the production facility. This study has demonstrated that complete sensory based event driven robot programs can be off-line programmed and tested before down-loading for execution. The programming was based on the Pascal programming language, thus realising the benefits of using a general purpose programming language for both virtual and real workcells.

There has been considerable research on self-learning robots and it is likely that, in the future, there will probably exist fully developed commercial robotic systems with reasoning capabilities. However, allowing real robots to move around in real environments (for example production areas) trying to learn about the world and tasks is not a realistic option as: (i) the production facilities must be complete and available before training can begin, thus extending the lead-time for the plant to be operational; (ii) an untrained robot could damage or destroy parts of the plant or itself in the early phases of training; (iii) in the real environment it is difficult and time consuming to create all possible states, thus leaving the robot unprepared

for some situations; and (iv) in some environments it is to dangerous to allow basic training. Intelligent self-learning robotic systems need equivalent of human schools.

This study has shown that it is possible to use 3-D CAD models for training robots using virtual sensors. Two representative autonomous robot platforms were used to conduct the investigation. A control architecture using artificial neural networks was designed and implemented in both robots. The artificial neural networks were trained to map sensory readings to correct robot actions. The robots learnt different tasks and used different network architectures dependent upon the platform and the task. Virtual robots were created having their sensors simulated using the generic sensor model. The robots used different sensory devices to correspond with their real counterparts. The same control architecture was implemented for the virtual robots. The virtual robots were trained for tasks in the virtual robotics system. After the completion of training the neural controllers where verified in the virtual environments by confronting the robots with new unknown situations. The trained artificial neural networks were transferred to the real robots and the real robots were able to accomplish desired tasks. The controllers needed some 'fine-tune' training after transfer from the virtual to the real robots. This training is necessary to adjust to the specific characteristics of each individual sensory device and to the exact characteristics of the new environments. The training in the virtual environment can be viewed as providing the robots with the 'basic' knowledge of how to perform a task and react in different sensor situations, however the last phase of training must be provided in the actual working environment to enable the robots to become 'skilled'. Human operators (after the initial training) need time in the real environment to become experienced and skilled.

This study has proven that robots which utilise artificial neural networks in their controllers can learn correct behaviours from training whereby simulated sensors are used in a virtual robotic system. Through the learning and generalisation capabilities of artificial neural networks, by exploring a virtual environment, robots could learn to navigate in factories and to negotiate changes in the environment, learn to adjust to material deficiencies when performing assembly operations etc. In [Biewald 1996], a robot which learns to navigate in factory-like environments by building maps consisting of feature recognition and state-graphs, is described. Features and places in the environment are recorded as sensor signatures. The relationship between the sensor signature and a place is established by training an artificial neural network. It should be possible to learn these sensor signatures in a virtual

robotic system using virtual sensory devices. Where map-building and learning in the real environment is not an option this approach could enable 'pre-emptive' off-line learning, for example, maintenance robots for nuclear plants can be 'taught' the plant navigation using a virtual plant. Intelligent assembly robots could be trained in the virtual environment and when new assemblies are introduced, the virtual robot can learn these processes using information from the product design before any real products are produced. This is a good example of concurrent engineering, as it enables information to be used at several instances simultaneously, thus reducing lead-time. Intelligent robots for the construction industry for applications such as wall plastering and rockwool spraying can be trained in the virtual system using the 3-D design environment. Having self-learning robots which have no initial knowledge, installed into a factory, is not a realistic option. It would be to costly, both in terms of increased lead-time and potential damage equipment, having these robots learning from scratch. Using virtual robotics could provide an opportunity for 'pre-emptive learning' of such tasks. Furthermore, the use of realistic 3-D simulation of robots, sensors, tooling and the environment provides a basis to allow more realistic evaluation of new theories and methods in the field of neural networks and other learning strategies. Learning regimes can easily be exposed to different situations and environments.

9.2 Recommendations for further work

To produce more realistic virtual environments for both off-line programming of event-driven robotics and 'pre-emptive learning' in virtual robotics, the virtual environments should provide information which might include material properties, variances in object geometry etc. Sensor characteristics are dependent on the material properties of the detected surface. For example, inductive devices will only detect metallic objects, ultrasonic transducers are influenced by the materials ability to absorb sound, etc. These properties could be accomplished by extending the CAD data structures used for modelling the environments with a data-field contain a 'material-tag'. When an intersection is detected the virtual sensor could interrogate this 'tag' and base any decision on this additional information. The first obvious decision would be to decide whether the object can be detected or not by a particular sensory device. Additional calculations based on information, such as, reflectivity, standard deviations in dimension, etc., could then be added to the generic sensor models *output format*

function. Goldenberger and McQuilian suggest that geometric uncertainties should be simulated while off-line programming assembly tasks [Goldenberger, McQuilian 1991]. They add a differential homogenous transformation to the nominal transform, which provides differences in the location and orientation of objects.

Creating methods for the realistic simulation of variation in object dimensions and locations would provide good training data to be used in 'pre-emptive learning' of adaptive robots. Variance in object locations can be provided by the addition of some random increments (noise) to the desired coordinates of an object. This is the basis of ongoing research at the University of Skövde.

Potential application examples are: (i) Surface treatment of parts with low tolerances, which normally would require expensive fixtures. Figure 50 illustrates a surface treatment application. The tolerances of the components and their locations is are variable, but it is important that the entire surface is coated. This is an example of an application where a robot with adaptive behaviours is suitable. The robot is guided through the use of ultrasonic sensors and could 'pre-emptively' learn how to follow the components whilst minimising the consumption of coating material;

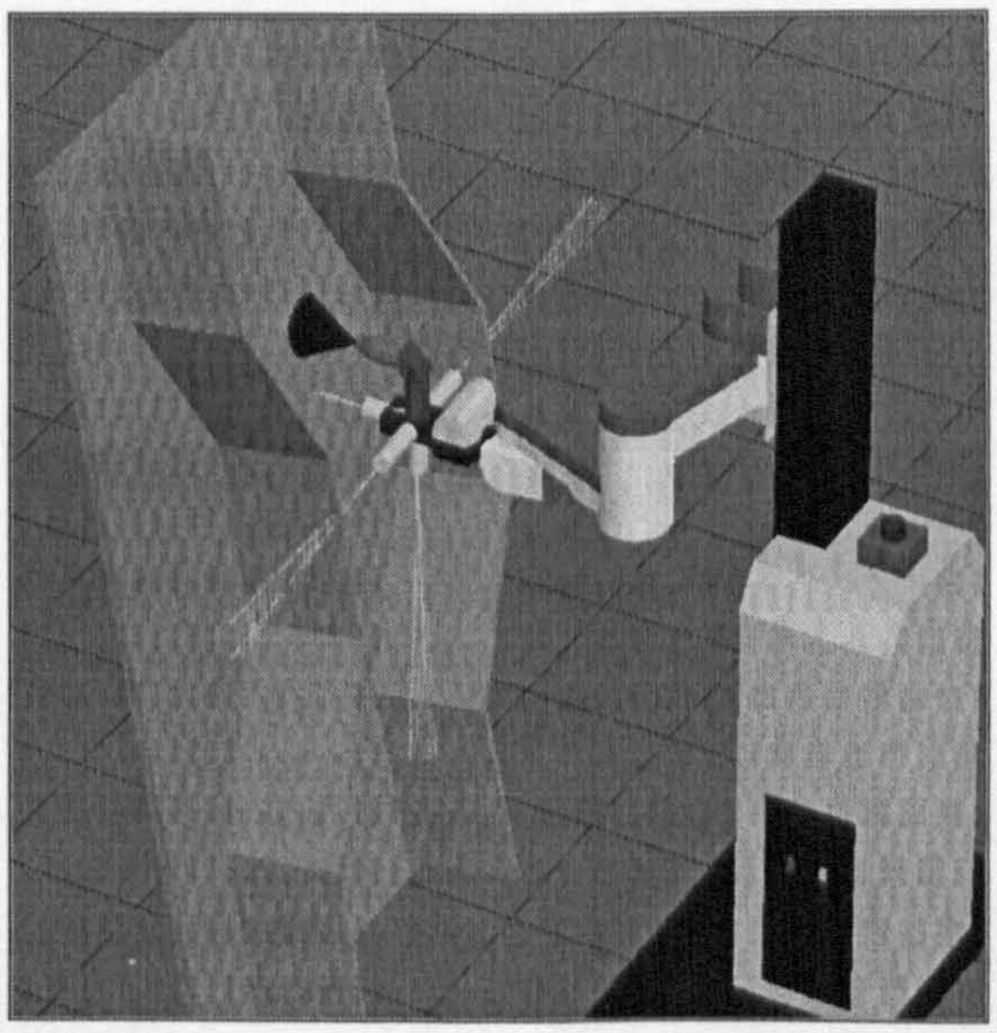


Figure 52: 'Pre-emptive learning' for a surface treatment robot capable of adaptive behaviours.

(ii) Training behaviours for semi-autonomous AGVs. For example obstacle avoidance in AGVs utilising reactive control could be developed and evaluated in virtual environments using simulated sensors; and (iii) Robots for domestic use capable of learning indoor environments, for example cleaning robots, hospital robots and maintenance robots for nuclear power plants;

Development of appropriate sensors and learning strategies together with 'pre-emptive learning' in virtual environments using simulated sensors should reduce significantly the need for on-line workcell calibration. The robots could be trained to detect and adjust to environmental changes. Such an approach should not only reduce the need for the initial adjustments to compensate for differences between virtual and real workcells, it would also enable the robots to detect changes over time.

9.3 Contributions to knowledge

Graphical programming systems / virtual robotic systems have been enhanced through the integration of simulated sensor components.

A generic sensor model can be used to create a variety of sensors within graphical simulation environments / virtual robotic systems.

Sensors simulated in virtual robotic systems and their interaction with the robot controller must be representative with the interaction that occurs in the real environment.

Realistic evaluation of event based robotic cells can now be undertaken through the use of an 'enhanced virtual robotics environment'.

It has been shown that complete off-line programming of sensor dependent robotic tasks can be undertaken in 'enhanced virtual robotic environments'.

It has been demonstrated that 'enhanced virtual robotic systems' can be used as a development mechanism for creating variations of neural controllers.

It has been proven that three dimensional graphical off-line programming environments can be used to train the behaviours of real robots which operate in three dimensional environments.

Graphical simulation can be used to provide the initial (or basic) knowledge to robotic systems with neural controllers before the final learning phase on the real robot is undertaken. This overcomes perhaps the biggest obstacle to the use of self-learning robots in real applications associated with problems of training (learning time, costs, availability and danger).

Sensor simulation can be used for both (i) off-line programming of conventional robot systems; and (ii) the training of adaptive robot systems.

9.4 Conclusions to be drawn

Contemporary graphical off-line programming systems can not be used for off-line programming of sensory interaction and extensive logic in robot programs.

Sensor simulation and characterisation is an area of significant research interest in flexible robotic systems.

Sensors should be autonomous components if true event-based systems are to be realised.

Using the same programming language for virtual and real robot devices and the associated 'sensors', enables off-line programming and debugging for sensor event-based robot applications.

Virtual robotics enhanced with simulated sensors can be used as a research tool for investigating new robot learning regimes.

Integration of graphical simulation, sensor simulation and artificial intelligence can be used to create robotic systems which include event-based operations and adaptive behaviours.

References

[Alami et al. 1992] Alami R., Chatila R. and Ghallab M., *Mission Planning and Control for Autonomous Mobile Robot*. Tolouse cedex-France 1992.

[Albus 1991] Albus J.S. *Outline for a theory of intelligence*. IEEE Transactions on Systems, Man and Cybernetics, 1991, pp 473-509.

[Ambler and Popplestone 1983] Ambler A. and Popplestone R., *RAPT: A language for specifying robot manipulations*. Robotic Technology 1983 pp125-141.

[Angermüller et al. 1989] Angermüller G., Niedermayr E. and Roth N., *Off-line Programming and Simulation of Flexible aasembly*. Assembly Automation 1989, Vol. 9
pp 97-102.

[Axelsson 1995] Axelsson S., *Inteview with Stefan Axelsson.* Volvo Research, Gothenburg October 1995.

[Aylor et al. 1992] Aylor S., Rabelo L. and Alkptekin S., *Artificial Neural Networks for Robotics Coordinate Transformation*. Computers in Engineering Vol. 22, No 4,
pp. 481-493, 1992.

[Bartels et al. 1987] Bartels R.H, Beatty J.C. and Barsky B.A., *An introduction, Splines for use in computer graphics & geometric modeling*. Morgan Kaufman publishing 1987,
ISBN 0-934613-27-3.

[Bergolte 1994] Borgolte U., *The new german standard robot programming language: IRL.* Proceedings 25th International Symposium on Industrial Robots. 1994.

[Bernhardt 1994] Bernhardt R., IPK Berlin. *Realistic Robot Simulation (RRS) Initiative.* Technical Description 1994.

[Bernhardt et al. 1991] Bernhardt R., et al., *Execution of Off-line generated and simulated application programs*. Mechatronics & Robotics, 1 1991 IOS press.

[Bezier 1986] Bezier P., *The mathematical basis of the UNISURF CAD system.* Butterworths & Co. Ltd. 1986, ISBN 0-408-22175-5.

[Bekey and Goldberg 1993] Bekey G. and Goldberg K., *Neural Networks in Robotics*, 1993 ISBN 0-7923-9268-X.

[Bien 1992] Bien C., *Simulation a necessity in safety engineering*. Robotics World, December 1992, pp 22-26.

[Biewald 1996] BiewaldR., *A neural network controller for the navigation and obstacle avoidance of a mobile robot.* Neural Networks for Robotic Control edited by A.M.S Zalzala and A.S Morris, Prentice Hall 1996, ISBN 0-13-19892-0.

[Blazevic et al. 1991] Blazevic P., Delaplace S., Fontaine J.G. and Rabit J., *Mobile robot using ultrasonic sensors: study of a degraded mode*. Robotica 1992, Vol. 9, pp 365-370.

[Bolmsjö 1989] Bolmsjö G., *Industriell robotteknik*. Studentliteratur 1989, ISBN 91-44-28512-4.

[Brooks 1986] Brooks R.A., *A Robust Layered Control Systme For A Mobile Robot*. IEEE Journal of Robotics and Automation 1986, Vol.2, No.1 pp 14-23.

[Brooks and Mataric 1993] Brooks R.A. and Mataric M. J., *Real Robots, Real Learning Problems*. Robot learning, Kluwer Academic Publisher 1993.

[Bruyninckx et al. 1992] Bruyninckx H. , et al., *A CAD-based contact force simulator as a learning tool for compliant motions*. Intelligent control IEEE conference 1992, pp287-292.

[Bullinger et al. 1989] Bullinger H.J., Menges R. and Warschat J., *GROSS- Graphic Robot Simulation System.* CAD/CAM, Robotics and Factories of the future, Vol. III, Springer Verlag 1989 ISBN 0-387-51134-0.

[Chen et al. 1991] Chen C., Trivedi M.M.,Bidlack C. and Lassiter N., *An Environment for Simulation and Animation of Sensor-based Robots.*Application of Artificial Intelligence 1X (1991) vol. 1468 pp 354-366.

[Chen et al. 1994] Chen C.,Trivedi M.M. and Bidlack C., *Simulation and Animation of Sensor-Driven Robots.* IEEE transaction on robotics and automation, Vol. 10, No. 5, 1994.

[Chung et. al. 1993] Chung H.L.C., Narayan O. H., and Davies B.J., *An Event-Based, Robot-Served Flexible Manufacturing Cell Modeller*; International Journal on Advanced Manufacturing Technology , 1993, Vol 8, pp. 102-110.

[CimStation 1] *CimStation Users Manual.* Silma Inc., Cupertino, CA.

[CimStation 2] *Meta-Kinematics Users Manual.* Silma Inc., Cupertino, CA.

[CimStation 3] *Dynamics User's Manual.* Silma Inc. 1995, Cupertino, CA.

[CimStation 4] *Developers Guide.* Silma Inc, Cupertino CA, 1994.

[CimStation 5] *Command Reference.* Silma Inc, Cupertino CA, 1994.

[Connell and Mahadevan 1993] Connell J.H. and Mahadevan S., *Introduction to robot learning.* Robot Learning, Kluwer Academic Publisher 1993.

[Craig 1988] Craig J.J., *Issues in the Design of Off-line Programming Systems.*Robotics research, MIT Press 1988, ISBN 0-262-02272-9.

[Craig 1989] Craig J.J., *Introduction to Robotics, Mechanics and Control.* Addison Wesley, 1989, ISBN 0-201-09528-9.

[Craig 1992] Craig J.J., *Robot Calibration Facilitates Off-line Programming*. Robotics World, March 1992,Vol 10, part 1, pp 24-25.

[Crane 1992] Crane III, C.D., *A computer graphics based approach to range sensor simulation*. Application of Artificial Intelligence X, machine vision and Robotics, 1992, pp 306-312.

[Crowley et al. 1991] Crowley J., Causse O., and Reignier P., *Layers of Control in Autonoumouse Naviagation*. Conference on Robotics and Mechatronics, Aachen1991.

[Dai 1989] Dai F., *Modelling of robot workcells for a programming and simulation system*. System modelling and simulation, Elsvier 1989.

[Dillman and Huck 1986] Dillman R. and Huck M., *A Software System for the simulation of Robot Based Manufacturing Processes*. Robotics (2) 1986, pp 3-18.

[Duffau and Kehoe 1991] Duffau B. and Kehoe A. 1991. *Overcoming Robot Workcell Uncertainties*. An Overview of ESPRIT Project CIM-SEARCH 5272.

[Eberhart and Dobbins 1990] Eberhart R. and Dobbins R., *Neural Network PC Tools*. Acedemic Press 1990, ISBN 0-12-228640-5.

[Edkins and Smith 1985] Edkins M. and Smith C.R.T. *The practical problems involved in off-line programming a robot from C.A.D. system.* Robots and automated manufacture 1995, ISBN 0-86341 053-7.

[Elmaraghy and Rondeau 1991] Elmaraghy A. and Rondeau J.M. *Automated planning and Programming environments for robots*. Robotics 1992 ,Vol. 10 pp 75-82.

[Elman 1990] Elman J.L., *Finding Structure in Time*.Cognitive Science 14 1990, pp 179-211.

[Engelberger 1989]. Engelberger J.F., *Robotics in service*. Kogan Page 1989, ISBN 1-85091-358-7.

[Erard et. al. 1995] Erard P-J., Fuhrer C. and Iff L., *A Synthetic Mobile Robot.* Proceedings of Computer Animation'95, IEEE Computer Society Press, 1995.

[Eshed 1] Scorbot-ER III User's Manual, Eshed Robotec 1992,, Tel Aviv, Israel.

[Flaig et al. 1994] Flaig T. , Neugebaur J-G. and Wapler M., *Virtual Reality for Robot Simulation and Off-line Programming*. Fraunhofer-Institute for Manufacturing Engineering and Automation (IPA) Stuttgart, Germany 1994.

[Freund et al. 1994] Freund E. , Rossman J., Uthoff J. van der Valk U., *Towards Realistic Simulation of Robotic Workcells*. 1994 International Conf. on Intelligent Robots and Systems. ISBN 0-7803-1934-6.

[Gigante 1993] Gigante M. A., *Virtual Reality: Definitions History and Applications*. Virtual Reality Systems, pp 3-14, Academic Press 1993, ISBN 0-12-227748-1.

[Goldenberger and McQuilian 1991] Goldenberger A.A. and McQuilian F.J., *Geomtric Uncertainty in Off-Line Programming of Robot Manipulators for Assembly Tasks*. Journal of Dynamic Systems, Measurement and Control 1991, Vol 113. pp 329-334.

[Goldman 1985] Goldman R., Design of Interactive Manipulator Programming Environment. UMI Researsch Press 1985, Ann Arbour, Michigan .

[Hamura and Kataoka 1995] Hamura M. and Kataoka M., T*he Arc Welding Robot System with AI function.* Proceedings 26th Int. Symposium on Industrial Robots 1995, ISBN 1-86058-000-9.

[Hecht-Nielsen 1989] Hecht-Nielsen R. *Neurocomputing*. Addison-Wesley 1989, ISBN 0-201-09355-3.

[Hill and Tang 1988] Hill J. L. andTang S., *Kinematic Simulation of Robotic Systems*. CAD/CAM Robotics and factories of the future 1988, Vol. 3 ISBN 0-387-51134.

[Hirzinger et al. 1994] Hirzinger G., Gombert B. , Dietrich J. and Shi J., *Transferring space robot technologies into terrestrial applications*. Proceedings 25th International Symposium on Industrial Robots 1994, ISBN 0-85298-939-3.

[Honeywell 1] Honeywell Europe. *Data sheet ultrasonic sensor 942-B3A-2D-1C1*. Belgium 1993.

[Jacak and Rozenblit 1992] Jacak W. and Rozenblit J. W., *Automatic simulation of a robot program for a sequential manufacturing process*. Robotica 1992, Vol. 10 pp 45-56.

[Kehoe et al. 1991] Kehoe A., et al., *Calibration Concepts for Off-line programming of robotic based manufacturing systems*. Mechatronics & Robotics 1, 1991 IOS press.

[Kugelmann et al. 1994] Kugelmann D. , Reinhart G. and Milberg J., *Autonomous Robotics handling Applying Sensor Systems and 3D simulation*. Proceedings Int. Conf. on Robotics and Automation 1994, pp196-201.

[Lahti 1995] Lahti T., *PLCLINK WHITE PAPER*, Tehadsmalit AB 1995.

[Leonard and Durrant-Whyte 1992] Leonard J.J. and Durrant-Whyte H.F., *Directed sonar Sensing for Mobile Robot Navigation*. Kluwer 1992, ISBN 0-7923-9242-6.

[Li and Zeng 1993] LI Y. and Zeng N., *A Neural Network Based Inverse Kinematics Solution in Robotics*. Neural Networks in Robotics, 1993 ISBN 0-7923-9268-X.

[Lieberman and Wesley 1977] Lieberman L. and Wesley M., *AUTOPASS: an automatic programming system for computer controlled mechanical assembly*. IBM Journal of Research and Development. 21 1977.

[Mahajan et al. 1993] Mahajan S.K., Brewera S.K and Bien C.D., *QUEST-QUING EVENT SIMULATION TOOL*. Proceedings of the 1993 Winter Simulation Conference, pp 269-275.

[Massone 1993] Massone L.E., *A Biologicaly-Inspired Architecture for Reactive Motor Control.* Neural Networks in Robotics 1993, ISBN 0-7923-9268-X

[Mataric´ 1994] Mataric´ M., *Interaction and Intelligent Behaviour.* Phd Thesis 1994 MIT, USA.

[Mayr and Held 1989] Mayr H. and Held M., *SMART: A Universal System for the Simulation of Machining and Robot Tasks.* Computer Application in Production and Engineering, Elsvier 1989.

[McCarragher 1993] McCarragher B. J., *Task Level Adaption using discrete event controller for robotic assembly.* Proceedingds 1993 IEEE International Conference on Intelligent Robots and Systems, ISBN 0-7083-0823-9.

[McKerrow 1991] McKerrow P.J., *Introduction to Robotics.* Addison-Wesley 1991 ISBN 0-201-18240-8.

[McMahon and Brown 1993] McMahon C. and Brown J., *CAD CAM From principles to practice.* Addison-Wesley 1993, ISBN 0-201-56502-1.

[Meijer and Hertzberger 1988] Meijer G.R. and Hertzberger L.O., *Off-Line Programming of Exception handling Strategies.* IFAC Symposium on robot control 2, Karlsruhe 1988, pp. 431-436.

[Meijer et al. 1992] Meijer G.R., Caglioti V., Somalvico M. and Negretto U., *Execption handling.* Integration of Robotis into CIM, Chapman & Hall 1992, ISBN 0-442-31243-1.

[Milgram et al. 1995] Milgram P. , Rastogi A. and Grodski J. J., *Telerobotics Control Using Augmented Reality.* Proceedings 4th IEEE RO-MAN'95, 0-7703-2904-x/95 1995 IEEE.

[Nehmzow 1992] Nehmzow U., *Experiments in Competence Acquisition for Autonomous Mobile Robots.* Ph D Thesis Univeristy of Edinburgh 1992.

[Nehmzow et al. 1993] Nehmzow U., Smithers T. and McGonigle B., *Increasing Behavioural Repertoire in a Mobile Robot.* From Animals to Animates2, MIT Press 1993.

[Nehmzow and McGonicle 1994] Nehmzow U. and McGonicle B., *Achieving Rapid Adaptions in Robots by Means of External Tuition.* From animals to Animates 3, MIT Press 1994.

[Niklasson 1996] Niklasson L., PhD artificial neural networks, Interview 1996, Connectionist research group, University of Skövde, Sweden.

[Nitzan 1990] Nitzan D., *Encyclopedia of Artificial Intelligence, chapter robotics.* pp 923-944, New York, NY, John Wiley & Sons, 1990.

[Nnaji 1993] Nnaji B.O., *Theory of Automatic Robot Assembly and Programming.* 1993; ISBN 0-412-39310-7

[Nomadic 1] *Nomad 200 Users Manual*, Nomadic Technologies 1994, USA.

[Okano et al. 1988] Okano A., Shimbada K. and Kawabe S., *A concurrent motion simulator and interactive debugger for multiple robots.* International Symposium and Exposition on robotics 19 , Sydney 1988, pp 998-1009.

[Omron 1] *Type E3JK Photoelectric switch, Instruction manual.* Omron Corporation 1990.

[Omron 2] *Model TL-X-__-__E Proximity Switch, Instruction sheet.* Omron Corporation 1990.

[Owens 1994] Owens J., *RoboTrak: Calibration on a shoestring.* Industrial Robot. Vol.21 No.&, 1994 pp10-13.

[Petri 1962] Petri C.A., *Kommunikation mit Automaten.* PhD Thesis 1962, University of Bonn, Germany.

[Piguet et al. 1995] Piguet L., Fong T., Hine B., Hontalas P. and Nygren E., *VEVI: A Virtual Reality Tool For Robot Planetary Explorations*. NASA Ames Research Center 1995, MS 269-3, Moffett Field CA 94301,USA.

[Pomeroy et al. 1986] Pomeroy S.C., Dixon H.J., Wybrow M.D. and Knight J.A.G., *Ultrasonic Distance Measuring and Imaging Systems for Industrial Robots*. Robot Sensors, pp261-270, Springer Verlag 1986, ISBN 0-948507-02-0.

[Pomeroy et. al. 1993] Pomeroy S, Zhang G. and Wykes C. Variable Coefficient Absorbing Boundary Condition for TLM. Electronics Letters, Vol. 29, No. 13, 1993, pp 1198-1200.

[Pook and Ballard 1995] Pook P. K. and Ballard D. H., *Remote Teleassistance*. pp 944-949. Proceedings IEEE Int. Conference on Robotics and Automation 1995. ISBN 0-7803-165-6 .

[Ravani 1988] Ravani B., *World modeling for CAD based robot programming and simulation.* CAD Based Programming for Sensory Robots, Springer-Verlag 1988.

[Renfors et al. 1993] Renfors J., Aalto H. and Jokela M., *Accurate off-line programming for a 12 dof robot cell at Bronto skylift Ltd*. Proceedings Robotikdagar 2-3 Juni 1993, Linköping ISBN 91-7871-136-3.

[Rooney and Steadman 1993] Rooney J. and Steadman P., *Computer aided design*. Pittman publishing 1993, ISBN 0-273-02672-0.

[Rumelhart and McClelland 1986] Rumelhart D. E. and McClelland J. L. *Parallel Distributed Processing: Exploratrions in the Microstructure of Cognition,* Vol. 1, MIT Press, Cambridge, MA, 1986.

[Schlimmer and Langley 1992] Schlimmer J. C. and Langley P., Encyclopedia of Artificial Intelligence. 1992, 0-471-50307-X pp. 785-805.

[Sekiguchi et al. 1992] Sekiguchi M , Nagata S. and Asakawa K., Behaviour control for a mobile robot by structured neural network. Advanced Robotics, vol 6 No. 2, pp 215-230 1992.

[Shavlik et al. 1991] Shavlik J.W., Mooney R.J. and Towell G.G., Symbolic and Neural Learning Algorithms: An Experimental Comparision. Machine Learning, 1991, vol. 6, pp 111-143.

[Shröer and Bernhardt 1992] Shröer K. and Bernhardt R., Program execution. Integration of robots into CIM 1992. ISBN 0-412-37140-5.

[Smith 1992] Smith M.G., An Environment for More Easily Programming a Robot. Proceed ings of the 1992 IEEE International Conference on Robotics and Automation, Nice, France May 1992 pp 10-16.

[Smith and Gini 1992] Smith R. and Gini M., Error management for robot programming. Journal of Intelligent Manufacturing 1992, volume 3, pp 59-73.

[Stobart and Dailly 1985] Stobart R.K. and Dailly C., The use of simulation in the off-line pro gramming of robots. Robots and automated manufacture, Peter Peregrinus 1985, ISBN 0-86341 053 7.

[TAG 1] Frank2 Mobile Robot Research Platform ,User Guide and Reference, TAG 1994, Alnwick, Northumberland, Uk.

[Takahasi and Sakai 1991] Takahasi T. and Sakai T., Reality. Proceedings Int. Workshop on Intelligent Robots and Systems 1991, pp 1583-1588, IEEE Cat. No. 91TH0375-6.

[Tarnoff et al. 1992] Tarnoff N. , et al., Graphical Simulation for Sensor Based Robot pro-gramming. Journal of Intelligent and Robotics System 5, 49-62 1992.

[Theveneau and Pasquier1988] Theveneau P. and Pasquier M., A Geometric Modeller for an Automatic Robot Programming system. CAD Based programming of Sensory Robots, Springer Verlag 1988.

[Van der Smagt 1996] Van der Smagt P., A robot arm is neurally controlled using monocular feedback. Self Learning Robots, IEE Colloqium Digest, Savoy Place London 1996.

[Van de Velde 1993] Van de Velde W., Toward Learning Robots.Toward learning Robots, The MIT Press, 1993, ISBN 0-262-72017-5.

[Vietze 1992] Vietze L., Ultrasonic Modelling. Advances in Control Systems and Signal Processing, Contributions to Autonomous Mobile Systems., Vieweg 1992, ISBN 3-528-06383-1.

[Wapler and Neugenbauer 1994] Wapler M. and Neugebaur J-G., Improving the integration of Off-line programming and Robotic Simulation Systems with STEP. Proceedings 25th International Symposium on Industrial Robots, 1994.

[Wahrburg and Papperitz 1995] Wahrburg and Papperitz. Application of Enhanced Sensor-based Robots to Improve the Automation of Grinding and Deburring Tasks. Proceedings 26th International Symposium on Industrial Robots 1995; ISBN 1-86058-000-9.

[Warwick 1996]Wawick K., An overviewof neural networks in control applications. Neural Networks for Robotic Control, edited by A.M.S Zalzala and A.S Morris, Prentice Hall 1996, ISBN 0-13-19892-0.

[Wloka 1989] Wloka D. W., Efficient Calculation of Generalised Forces in a Robot Simula tion Environment. European simulation congress 3, Edingburg 1989, pp 595-601.

[WorkSpace 1] WorkSpace Users Manual. Robot Simulations, Newcastle Upon Tyne U.K.

[Wu et al. 1993] Wu C.M., Jiang B.C. and Wu C.H., Using Neural Networks For Robot Posioning Control. Robotics and Computer Integrated Manufacturing, Vol. 10 No. 3 pp 153-168, 1993.

[Wybrow and Wykes 1992] Wybrow and Wykes. An Ultrasonic system for object recognion in a manufacturing environment. Journal of Intelligent Manufacturing (1992) 3, pp 163-172.ots, The MIT Press, 1993, ISBN 0-262-72017-5.

[Yeung and Moore 1996] Yeung W.H.R. and Moore P.R., *An Integrated MMS and colour petri/net model for the distributed control of flexible assembly systems*. International conference on Computer Integrated Manufacturing, Singapore, 1995, vol. 1 pp. 831-838.

[Zalzala 1996] Zalzala A.M.S., Model-based adaptive neural structures for robotic control. Neural Networks for Robotic Control, edited by A.M.S Zalzala and A.S Morris, Prentice Hall 1996, ISBN 0-13-19892-0.

[Zomaya 1992] Zomaya A. Y., *Modelling and Simulation of Robot Manipulators*. World Scientific Series in Robotics and automated systems, Vol. 8, ISBN 981-02-1043-4.

Bibliography

Arbib M.A. and Buhman J. *Neural Networks* . Encyclopedia of Artificial Intelligence, 1992, 0-471-50307-X pp. 1016-1060.

Bekey G and Goldberg K. *Neural Networks in Robotics.* 1993 ISBN 0-7923-9268-X.

Bharath R and Drosen J. *Neural Network Computing*. McGraw-Hill 1994, ISBN 0-8306-4523-3.

Eberhart R and Dobbins R. Neural Network PC Tools. Acedemic Press 1990, ISBN 0-12-228640-5.

Picton P. *Introduction To Neural Networks*. Macmillan 1994, ISBN 0-333-61832-7.

Appendix A

Some virtual robotic systems reported in the literature

Software	Developer
AUTOMATOS	Institutet för Verkstadsteknisk Forskning IVF Göteborgs Sweden
CIMSTATION	Silma Inc. Cupertino, California USA
COSIMIR	IRF University of Dortmund Germany
GRASP	B.Y.G. Systems Nottingham, Nottinghamshire UK
IGRIP	Deneb Robotics Inc. Auburn Hills, Michigan USA
KISMET	Tele Robot Engineering Meersburg Germany
MOSES	AUTOCAM Dortmund Germany
ROBCAD	Technomatix Technologies Herzliya Israel
ROSI,	University of Karlsruhe Germany
SMART	AIS GES.M.B.H. Linz Austria
WORKSPACE	Robot Simulations Ltd Newcastle Upon Tyne UK

Appendix B

List of Publications

Eriksson P. and Moore P. *Improved Computer Aided Robotics with Simulation of Sensors*, Proceedings Mechatronics and Machine Vision in Practice, 1994, IEEE Computer Society Press ISBN 0-8186-6300-6

Eriksson P. and Moore P. *Simulation of Sensors to Enhance the Capability of Computer Aided Robotics*. Proceedings Robotikdagarna 95 Linköping, 1995, Sweden
ISBN 91-7871-566-0

Eriksson P. and Moore P. *An environment for Off-line Programming and Simulation of Sensors for Event Driven Robot Programs*. Proceedings Recent Advances in Mechatronics" ICRAM´95, Istanbul 1995, ISBN 975-518-063-X.

Eriksson P. and Moore P. *A Role for 'Sensor Simulation' and 'Pre-emtive Learning' in Computer Aided Robotics*. Proceedings 26th International Symposium on Industrial Robots ISI, Singapore, 1995, ISBN 1-86058-000-9

Eriksson P. and Moore P. *A Role for Computer Aided Robotics in Training Adaptive Robot Behaviours*. Proceedings Mechatronics ´96, Portugal, 1996, ISBN 972-8063-08-3

Appendix C

Implementation of a sensor, using the values from section 6.2

```
{**********************************************************************}
{* check if a collision was detected *}

function is_hit():boolean;
begin
    if null(who_hit_who) then
        is_hit:=false
    else
        is_hit:=true
end;

{**********************************************************************}
{*Calculates the angle between a traceline and a detected surface.
The function returns a valueof type real, the code can be extended with
code to handle distortion which may occur near the treshold value *}

Function REFLECT(Detect_shp: SHAPE;
        Trace_Line : LINE;
        Intersect : LINETRACE;
        Reflect_Angle : REAL ) : Real;

VAR
    Norm : POINT;
    Intersect_Pnt : POINT;
    Trace_Frame : FRAME;
    Trace_Pnt : POINT;
    ALFA : REAL;
    Direction : POINT;

BEGIN
    Norm:=Normal_At(Detect_shp,Intersect);
    Intersect_Pnt:=Intersect.pnt;
    Trace_Frame:=Pose_of(Trace_Line);
    Trace_Pnt.XC:=Trace_Frame.XC;
    Trace_Pnt.YC:=Trace_Frame.YC;
    Trace_Pnt.ZC:=Trace_Frame.ZC;
    Direction:=DIFFERENCE(Trace_Pnt,Intersect_Pnt);
    Direction:=Normalize(Direction);
    ALFA:=ACOS((NORM*Direction)/1) as_type Real;
    IF ((ALFA>= (1.5708)) AND (ALFA<=3.1416))
    THEN ALFA:=3.1416-ALFA;
    IF (ALFA<=Reflect_Angle)
    THEN REFLECT:=1.0
    ELSE REFLECT:=0.0;
```

```
END;

{**********************************************************************}
{Variables created for the Polaroid_1 sensor}
VAR

    Polaroid_1_Value :Real;
    Polaroid_1_Range :Shape;
    Polaroid_1_Trace-Line1: Line;
    Polaroid_1_Trace-Line2: Line;;


Polaroid_1_Update_rate == 0.250;
Polaroid_1_Default_Value == 10.0

Polaroid_1_Range :=wlkup('POLAROID_1/POL1_RANGE');
Polaroid_1_Trace-Line1:= wlkup('POLAROID_1/POL1_TRACE1');
Polaroid_1_Trace-Line2:= wlkup('POLAROID_1/POL1_TRACE2');

{**********************************************************************}
{* This function calulates the output value as given by the polaroid ultrasonic sensor
on the Frank robot *}

Function Polaroid_1_OutPutValue(distance: real ) : Real;
Begin
   Polaroid_1_OutPutValue:=((2.718**((1-(distance/200.0))**2))-1);
End;

{**********************************************************************}

FUNCTION Report_Polaroid_1(Range_shape : Shape;
          default_dist:real; Shape_name:string) : Real;
VAR
   collision, was_range_hit,detect: boolean;
   Shape1,Shape2,int_col_shpA,int_col_shpB : shape;
   dist,distA,distB,dist_Null :real;
   distAMin,distBMin :real;
   Angle :Real;
   who_hit_counter:integer;
   list1, list2 : list of linetrace;
   point1:point;
   intersection1: linetrace;
   intersection_pnt,calc_point:point;
   who_hit_who2,who_hit_who3 : list of ob; {Used for copies of the who_hit_who list}
   pose1,Calc_pose : Frame;
```

```
Begin
    who_hit_counter:=1;
    dist:=default_dist; {set the distance variable to not detect}
    Was_range_hit:=False;
    collision:=is_hit(); {is_hit is function in hitting.sil}
    distA:= default_dist;
    distB:= default_dist;
    distAMin:= default_dist;
    distBMin:= default_dist;
    {Checks if one of the colliding objects is the RANGE}
     if (collision=true) then
     begin
        for sh_pr in who_hit_who do
           If ( ((name(car(sh_pr) as_type shape))= Shape_name)
           or
           ((name(cdr(sh_pr) as_type shape))= Shape_name) )
           Then  Was_range_hit:=true;
 end;

if Was_range_hit=True then
begin
    v_init_coldet(); {From obj_obj_measure}
    Detect:=False;
    who_hit_who2:=who_hit_who;
    {This part extracts which object LINE1 is colliding with}
    For inter_pair in who_hit_who2 do
    begin
    Shape1:=(car(inter_pair) as_type shape);
    Shape2:=(cdr(inter_pair) as_type shape);
    If ((name(Shape1)='POLAROID_1/POL1_TRACE1') or
    (name(Shape2)= 'POLAROID_1/POL1_TRACE1')) then
    Begin
        detect:=true; {to get the first colliding object}
        if name(Shape1)='POLAROID_1/POL1_TRACE1'
        then int_col_shpA:=Shape2
        else int_col_shpA:=Shape1;
        { This part measure gives the distance to the colliding shape as an }
        { vector in the frame for line1}
        list1:=ln_shp_intersect(Polaroid_1_Trace-Line1,int_col_shpA);
        { Creates a list of linetraces scanns through all objects that colides}
        { with the line}
        if not null(list1) then
        begin
            intersection1:=car(list1);
            {Gets the nearest object }
            intersection_pnt:=intersection1.pnt;
            {get the intersectio0ns point in world frame}
            point1:=intersection_pnt in_frame pose_of(Polaroid_1_Trace-Line1);
            {convert intersection point to the lines frame}
```

```
            distA:=sqrt((square(point1.xc)+square(point1.yc)+square(point1.zc)));
            {calculate the vector length in the lines frame}
            Angle:=Reflect(int_col_shpA,Polaroid_1_Trace-Line1,intersection1,0.785);
            If Angle =0.0 Then DistA:= default_dist;
        end;
        if null(list1) then distA:= default_dist;
        If DistA<DistAMin then DistAMin:=DistA;
    end;
end;
Detect:=False;
who_hit_who3:=who_hit_who;
{This part extracts which object LINE2 is colliding with}
For inter_pair in who_hit_who3 do
begin
Shape1:=(car(inter_pair) as_type shape);
Shape2:=(cdr(inter_pair) as_type shape);
If ((name(Shape1)='POLAROID_1/POL1_TRACE2') or
(name(Shape2)= 'POLAROID_1/POL1_TRACE2')) then
Begin
    detect:=true;
    if name(Shape1)='POLAROID_1/POL1_TRACE2'
    then int_col_shpB:=Shape2
    else int_col_shpB:=Shape1;
    { This part measure gives the distance to the colliding shape as an }
    { vector in the frame for line1}
    list2:=ln_shp_intersect(Polaroid_1_Trace-Line2,int_col_shpB);
    { Creates a list of linetraces scanns through all objects that colides}
    { with the line}
    if not null(list2) then
    begin
        intersection1:=car(list2);
        {Gets the nearest object }
        intersection_pnt:=intersection1.pnt;
        {get the intersections point in world frame}
        point1:=intersection_pnt in_frame pose_of(Polaroid_1_Trace-Line2);
        {convert intersection point to the lines frame}
        distB:=sqrt((square(point1.xc)+square(point1.yc)+square(point1.zc)));
        {calculate the vector length in the lines frame}
        Angle:=Reflect(int_col_shpB,Polaroid_1_Trace_Line2,
        intersection1,0.785,0.959);
        If Angle =0.0 Then DistB:= default_dist;
    end;
        if null(list2) then distB:= default_dist;
        If DistB<DistBMin then DistBMin:=DistB;
    end;
end;
Detect:=False;
dist:=distAMin;
if dist>distBMin then dist:=distBMin;
```

```
    v_end_coldet();
end;
report_Polaroid_1:=dist;
end; {* procedure report_sonic() *}

{**********************************************************************}

procedure POLAROID_1_sample();
Var distance: integer;

Begin
    Distance:=report_POLAROID_1(Polaroid_1_Range,
    Polaroid_1_Default_Value,'POL1_RANGE');
    Polaroid_1_Value:=Polaroid_1_OutPutValue(distance);
End;

{**********************************************************************}
*  Sample_interval determines frequency of check in simulating time
*  NOTE: This global cannot be changed dynamically since the
*        application does not have a pointer to it.*}


POLAROID_1_Control==mk_ticker(
        mk_application(" POLAROID_1_sample,
        emptylist(universal)),Polaroid_1_Update_rate);

{**********************************************************************}
{**********************************************************************}
```