The Design and Implementation of Vision-Based Behavioural Modules for a Robotic Assembly System

Prabhas Chongstitvatana

Ph.D.
University of Edinburgh
1992

Abstract

The work described in this thesis is about how to program robots to work reliably in the presence of uncertainty. Some architectural principles are proposed which address the problem of decomposing robotic assembly tasks into modular units such that a robot program can be implemented efficiently, tested easily, and can be maintained or modified without undue complexity. This architecture also provides a framework to integrate sensors into a robotic assembly system.

These modular units are called behavioural modules. They perform their tasks reliably. The problem of uncertainty is dealt with by encapsulating sensing and variation reducing strategies inside these modules. Experiments are performed with a working robotic assembly system using vision based behavioural modules. Analysis of this system validates the principles presented in this thesis.

Acknowledgements

First and foremost, I would like to thank my supervisor, Chris Malcolm, for his untiring support and encouragement, and Herzmark Malcolm, for her hospitality. Thanks to Tim Smithers, who taught me the difference between a scientist and an engineer, and Bob Fisher, who showed me how to resolve a dilemma. Thanks to all my former teachers: Robin Popplestone, Jim Howe, Susak Thongthammachart, Vinai Varanyananda, and Supachai Tangvongsarn, who lead my life smoothly into this path and taught me to be the way I am. Also thanks to my examiners: Gillian Hayes and Nigel Hardy, whose comments have helped to improve the presentation of this thesis.

My father teaches me about life and gives me support in every way, my sister and brother always help me. Many friends have helped me to finish this work. Special thanks to Graham Deacon who laboured over my English, and Jaruloj Eamsiri who listened to all my ideas. Jeremy Wyatt, Edward Jones and Manuel Trucco helped me greatly, and Kittisak Yongsiri did many illustrations. Thanks to all my friends and colleagues: Alistair Conkie, Lykourgos Petropoulakis, Li-Dong Cai, Mitch Harris, Peter Balch, Brian Logan, Howard Huges, Myra Wilson, Jim Donnett, and many others, who made my life in Edinburgh a happy and stimulating one. Thanks to the staff of the Department of Artificial Intelligence, especially Janet Lee and David Wyse, and ACME/SERC who provided travelling funds. Thank you again, with all your helps I have learnt how to write a thesis not a second too late.

Finally, I would like to dedicate this thesis to the memory of my mother, I couldn't find enough words to thank her.

Declaration

I declare that this thesis has been composed by myself and that the work described in it is my own, except where stated otherwise.

P. Chongstitvatana

Table of Contents

List of Figures

Chapter 1

Introduction

What is a symbol, that intelligence may use it,
and intelligence, that it may use a symbol?

Warren McCulloch (1961)

Reliable assembly with a robot must meet the challenge of dealing with the problem of uncertainty in the real world. Today's assembly robot systems cannot cope with uncertainty nor handle sensing adequately and are in general hard to program. The work described in this thesis is about how to program robots to work reliably in the presence of uncertainty. It addresses the problem of decomposing robotic assembly tasks into modular units such that a robot program can be implemented efficiently, tested easily, and can be maintained or modified without undue complexity. It also provides a framework for integrating sensors into a robotic assembly system.

1.1 Programming an assembly robot

An assembly robot deals with physical objects and their spatial relationships. The relationships between objects are constrained by their geometry. The environment and the task determine the robot motions.

Robots have not been used successfully to perform assembly for many reasons. The assembly task needs dextrous manipulation. Actions like insertion and part mating can not be easily done by blind positioning without using sensors when a high precision is required. The current generation of industrial robots are primarily

position controlled devices and therefore sensing systems, like force sensing or visual sensing, are still difficult to incorporate. The industrial robot can be programmed but the programming is done at a low level; in terms of robot motions.

A popular method is to program by teaching in which a programmer leads the robot through the required action and stores all the relevant motions to be replayed in use. Because of this method, even a small alteration may requires a lot of re-teaching. People have to use the robot to do programming and debugging. Debugging the robot program is time consuming. In order to get good reliability, several hundred test runs are required. Inevitably, there are variations in the shape or position of the parts. The robot motion has to take into account these uncertainties. Finding a robot motion strategy to achieve a desired part motion is difficult.

One way to reduce the uncertainty is to use sensors. For example the dimension of a part can be measured by both contact or non-contact sensing devices such as tactile sensors and vision. Its position or orientation can be measured by many means such as by analysing range data from laser ranging or a stereo vision system. Interpreting the data from sensors is also difficult because of the noise in the measurements. Very frequently the data are inconsistent. To get good results the environment and sensors need to be set up carefully and precisely calibrated. It makes programming with sensors difficult and of very limited use.

A program is built upon the layers of abstraction in the languages. Robot programming languages can be characterised by the programmer's view of the world offered by the languages. The successive layers of abstraction are built from the set of primitives provided by the level below. Malcolm and Fothergill (1986) describe these layers:

- The joint-level is the lowest level. A robot is programmed in terms of positions of the end-effector specified by the joint-angles or joint-displacements.

- In the manipulator-level, the trajectory of motion is specified in terms of Cartesian co-ordinates in a reference frame. The robot follows trajectories with a specified velocity and acceleration.

- In the object-level, a programmer specifies geometric constraints on objects. The program statements specify the constraints on the relationships between objects, for example *face-1 against face-2, shaft-5 colinear with bearing-3.* These statements are translated into the robot motions that will achieve the desired object relationships.

- In the task-level, a programmer specifies only the goals for the relationships among objects, rather than the motions needed to achieve those goals, for example, *put peg-1 into hole-4.* This high level description is then translated into a program at the lower level.

1.2 Why a robot is difficult to program

At the joint-level and the manipulator-level, to program a robot involves thinking about a sequence of motions to manipulate objects into the desired configurations. Although a trajectory is parameterised and symbolised such that a programmer does not have to think in terms of numeric coordinates, the sequence of motions is often long and complex and prone to errors. A program is sensitive to change in the geometry of objects. A small change may result in a large change in the robot program because the change in the geometry of objects may affect all subsequent motions.

At the object-level, a program is stated in terms of the relationships of the objects. Knowledge of object geometry and robot kinematics is used to translate the program statements into sequences of robot motions. The robot must know the exact geometry and position of objects in the environment and the precise actions to be performed. This translation does not cope satisfactorily with the uncertainty in the real world. To cope with the uncertainty, sensors can be used to access the real world. Sensing depends on the geometry of the task. Programming with sensors results in numerous branchings in program flow because of the enumeration of the possible sensing situations and the possible actions. There is no effective way to foresee all the possibilities. The generation of robot motion sequences incorporating sensing automatically is still an active area of research.

There are many problems that must be solved in order to program a robot at the task-level: how to acquire the part; how to choose the location in the workcell to perform assembly operations; how to choose the fixtures to hold the parts to the required accuracy; how to synthesis the fine motions for parts mating; how to grasp each part; and how to generate collision-free paths for the manipulator and the parts it carries; etc. These problems make the translation of the task-level specification to the lower level specification very difficult. A simple task-level description leads to a complex lower level program because of the presence of uncertainty. Incorporating a sensing strategy into the automatic translation process is of limited success (Yin, 1987; Donald, 1987; Jennings and others, 1989; Hutchinson and Kak, 1990).

The difficulty in programming a robot stems from the fact that, to increase the flexibility of the robot, its sequence of motions must accommodate changes in the real world. A robot must be competent to work reliably and robustly. To do this, it must cope with the uncertainty in the real world, which requires the use of sensors. However, a robot with sensors is difficult to program. Programming at a low abstraction level is a burden and error prone and the program is also sensitive to minor task changes. If a higher abstraction level is to be achieved, an effective way to program the use of sensors must be found. These are problems which have limited the progress of assembly by robots.

1.3 Classical approach to robotic assembly

It should be much easier to program a robot at task level, for example, *put peg in hole*, *move top plate to mate with subassembly*. From this task level specification the system analyses and generates the sequence of robot motions automatically. Planning the motion generation is complicated by many factors. First of all the motions generated must be collision free. Secondly sensing is needed to correct for deviations that might occur. It is also desirable for the system to be able to recover from small errors without resort to human operators.

To achieve task-level programming, past research has focused on the study of supporting functions. These functions are, for example, grasp selection, collision free trajectory planning, motion planning, error detection and recovery (Lozano-Pérez

and Brooks, 1985). The emphasis is on the modelling of geometrical characteristics of objects and the planning system. The planner has only limited capability in dealing with uncertainty in the real world. Taking uncertainty into account in reasoning during plan time is very complicated (Brooks, 1982). There are many sources of uncertainties: from the part tolerance; whether two parts fit together given their tolerances (Fleming, 1987), from the programming of sensors; what sensors to use and how to use them, from the possibility of error that can occur in each step of assembly, etc. It is an enormous task to analyse them.

The classical approach to assembly robot programming systems can be charac terised as based on the geometry of the objects, relying extensively on the exact knowledge of this geometry to generate robot motions. Sensors are used to update a representation of a world model. Robot motions are planned based on this world model. Because robot motions are based on planned motions, the uncertainty in the real world is accommodated by sensing and updating the world model. The problem then lies in the quality of the sensing and how accurately the world model reflects the real world. The detailed geometric representation of the world, however, makes it difficult to use sensors effectively. The world model might include complete geometric descriptions of objects, the kinematic model of the robot, the physical characteristics of the robot: speed, positioning accuracy, workspace bounds, etc. In this kind of terms, the space of interpretation of sensor readings is very large. A more detailed model increases the sensing accuracy which improves the reliability of robot operations but it also increases the problem of interpreting the sensed data.

1.4 Behaviour-based approach to robotic assembly

The behaviour based approach argues that the classical decomposition of the prob lem is not correct. Instead of breaking it into functional modules which are controlled by a central system, with perceptual modules as inputs and action modules as out puts, it should be broken into many task achieving units, where the task is some useful accomplishment in the assembly world in question such as acquiring a part. Each unit individually connects sensing to action, each unit pursues its specific goal

but co-operates with other units to achieve the desired goal. Rather than rely on a world model, the individual unit concentrates on those aspects of the world that are directly relevant to it, i.e. a minimal distributed world model rather than a centralised world model. They may work in parallel and interact with each other.

Brooks has demonstrated this approach successfully in his series of wheeled mobile robots (Brooks, 1985, 1986, 1987) and six legged walking robots (Brooks, 1989). In robotic assembly, a system by Malcolm (1987) called SOMASS, can plan and perform assembly in the domain of the Soma-4 world.[1] Given the desired final shape of the assembly, the system plans the sequence of operations that will put the seven component parts together. The planning of motions is decomposed in terms of task-achieving units called behavioural modules. The uncertainty is dealt with within these units. The robot program is more robust and requires significantly less debugging as reported in (Malcolm and Smithers, 1988a).

1.5 To make a robot easier to program

Smithers and Malcolm (1989) propose that a program for robotic assembly should be in terms of task-achieving behavioural modules. Behavioural modules isolate the details of the real world and its uncertainty from a programmer. Sensors can be integrated such that these modules can be guaranteed to perform their intended assembly tasks reliably within a certain range of uncertainty.

This enables a robot program to be constructed without a programmer having to worry about the uncertainty of the real world. Given suitably competent behavioural modules a program can be described at the task-level. The programmer makes sure that the ranges of parameters of the task fall within the competence of the behavioural modules. It is also possible (as the SOMASS system demonstrates in its limited domain) to generate a reliable robot program from a task-level description automatically.

This thesis aims to study the problem of decomposing assembly tasks into behavioural modules. The central claim of this thesis is that this decomposition should be based on the principles that:

---

[1] A detailed description of this system is given in chapter 4.

- there is no reliance on a central model of the world for combining sensing and action;
- sensing and action should be tightly coupled within the module;
- prefer to pass control via perception of the world rather than by parameters.

These principles are verified by a series of experiments. A working robotic assembly system is demonstrated. Behavioural modules used in it are designed and implemented based on the above principles and tested on a number of assemblies successfully. Visual sensing is used in the experiments. The first experiment uses a single camera system and the second experiment extends that to a two camera system. The third experiment adds mobility to the two camera system which increases the range of assembly tasks that can be performed.

1.6 Organisation of thesis

The next chapter presents previous work on robotic assembly: the attempt to raise the level of description of an assembly task, the use of a geometrical world model in planning and reasoning about robot motions, and the problems of using a world model in robotic assembly. It goes on to describe this new emerging paradigm in robotics, the behaviour based approach. The use of visual sensing in robotics is discussed, in hand eye co ordination tasks and in dynamic situations. Relevant work in the computer science literature about design methodologies and the modularisation of programs is also examined.

Chapter 3 describes the central idea of this thesis: the concept of behavioural modules. It addresses the problem of robot programming and the use of sensors. The difficulty in introducing sensors into robotic assembly systems is discussed. Then the behaviour-based approach to robotic assembly is introduced. The characterisation of behavioural modules and the principles for decomposing assembly tasks into behavioural modules are discussed. An example of two different decompositions is given. The issues in design and implementation of behavioural modules are examined. The next three chapters (4,5 and 6) present a series of experiments to verify these ideas.

Chapter 4 describes an experiment of integrating visual sensing into a robotic assembly system. The experiment is based on an existing system: SOMASS, which originally did not use sensors, but whose architecture was designed to facilitate their incorporation. This system is extended to incorporate a camera. The task of acquiring a part is implemented using a new behavioural module. This new module uses visual sensing. The sensing strategy is described. The result shows that visual sensing can be integrated seamlessly into the existing system. The vision system also works without having to rely on any central world model.

Chapter 5 describes the second experiment. The limitation of the single camera system of the previous chapter is that the sensing gives only two-dimensional data, therefore in order to resolve depth ambiguity along the line of sight some *a priori* knowledge must be available. This experiment introduces the second camera and does not require that knowledge. This two camera system is uncalibrated. It does not use a central world model. It uses a similar sensing strategy to the single camera system. A simple task of stacking blocks is demonstrated.

Chapter 6 describes an experiment that adds mobility to the two camera system. This enables it to change its view. This increases the range of assembly tasks that can be performed using visual sensing. New behavioural modules are designed to move the camera head around the object of interest without calibration between the position of the camera head and the object. This experiment demonstrates the use of a mobile camera head to find good views that are suitable for assembly operations. It is also used to perform a part-mating operation which require close range visual sensing. The result shows that moving cameras to view different parts can be achieved without requiring calibration between the robot's and cameras' reference frames, thus preserving the 'no central world model' principle.

Chapter 7 analyses all robot programs used in the experiments. The programs are examined and behavioural modules are explained: how they work, why they work and what are their assumptions about the world. The analysis is done along three dimensions: connections between modules, coupling of sensing and action in modules, and implicit assumptions of modules. The application of the principles of task decomposition is discussed. Finally, the conclusion drawn from this analysis is presented.

Chapter 8 describes the visual routines that are used in the behavioural modules. The justification of the choice of algorithms is discussed and is supported by the experimental data. The vision process is based on segmentation of image using a global threshold. The objects are presented by their attributes such as: centroid, area, corners, lines, etc. The results of implementation and the limitations of algorithms are discussed.

Chapter 9 summarises the work presented in the previous chapters. It notes the problems and points out important directions for future research. The appendix contains a full catalogue of behavioural modules.

1.7 A note on support and prior publication

The SOMASS system and its development to include sensors was the subject of SERC/ACME research grant GR/E/68075 during my studentship, one of the principal investigators being my supervisor, Chris Malcolm. I would like especially to acknowledge the assistance of Alistair Conkie, my friend and colleague, at the time a research fellow employed by the grant, and with whom I have published a number of papers on aspects of this work.

My original ideas are the single camera system described in chapter 4 and the mobile camera system in chapter 6. Conkie invented the strategy to use two cameras to control a robot. I performed the implementation and all the experiments, Conkie contributing discussion of many of the issues. Most of the ideas about behavioural modules are the result of many years of discussion and experimentation in the robotic research group in the Department of Artificial Intelligence, University of Edinburgh. Thus many colleagues contributed to the concept which I have distilled to present my own version in chapter 3.

An abridged report of the experiments in chapter 4 is in the paper: 'Behaviour based assembly experiments using vision sensing', presented at the Vision Interface Conference 1991, Calgary, Canada (Chongstitvatana and Conkie, 1992a). Chapter 5 grew out from the paper: 'An uncalibrated stereo visual servo system', presented at the British Machine Vision Conference, 1990, Oxford, UK (Conkie and Chongstitvatana, 1990). Chapter 6 grew out from the paper: 'Active mobile stereo vision for robotic assembly', presented at the 23rd Symposium on Industrial Robots, Barcelona, Spain, 1992 (Chongstitvatana and Conkie, 1992b).

Chapter 2

Previous work

In order to understand the problems of robotic assembly, we will review the development of robotic assembly systems. We will start from a historical perspective and then cover related work in incorporating sensing into robotic assembly tasks, with emphasis on visual sensing. The behaviour-based approach is introduced, and finally a discussion of the problems of previous approaches to robot programming, and the technical material which is the foundation of this thesis are presented.

2.1 Review of classical robotic assembly systems

In the early days, the Freddy system (Ambler and others, 1975) demonstrated robotic assembly in a simplified world. This world consisted of several parts piled into a heap. The Freddy system separated the individual parts, recognised, and assembled them. It demonstrated the use of force and tactile sensors to handle and assemble the parts, and 2D vision combined with manipulation to recognise and locate parts. It showed the importance of assembly strategies, the use of fixtures and demonstrated robot programming.

Much work has gone into raising the level of robot programming languages, in order to make robots easier to program. An overview can be found in Lozano-Pérez (1982). A method of specifying an assembly in terms of the spatial relations between parts (i.e. object-level programming) has been developed into a robotic assembly programming language called RAPT (Popplestone and others, 1978, 1980). This

system relies on symbolic inferences for spatial reasoning. A user describes how the various objects relate to each other in various situations in terms of their surface features such as planes, cylinders, holes, etc. The assembly process is described by the series of distinct situations required. The spatial relationships specifiable between the features include such concepts as *against*, *coplanar*, *aligned*, *fits*, and *parallel*. The type of feature and relationship specified determines the geometric relationships between the parts, for example, *edge-1 against face-2*. From this description, the RAPT system infers facts about the actual position of objects in each situation. The robot motions are derived directly from these positions and knowledge of the relationship between the robot arm and the parts. The success of such a program depends on the robot motion strategies adopted to reduce the uncertainty in locations and dimensions of the parts, and this must be decided by the user. RAPT does not perform any planning or determination of intermediate positions.

The spatial reasoning engine employed in RAPT has also been used to plan motions in contact (Koutsou, 1986). The plan is formulated in terms of the interactions between the features of the objects involved. This suggests the use of tactile sensing for part mating operations.

The ultimate aim of robotic assembly research is to be able to interpret a high level task specification into robot motions automatically. One good example is TWAIN (Lozano-Pérez and Brooks, 1985). In their study, they decomposed the system into many parts: part feeding, layout, fixturing, fine motion, grasping, gross motion. They advocated the use of number of powerful planning modules, the use of constraint propagation to choose feasible values of parameters, the use of skeleton programs, and the use of configuration space to find a collision free path (Udupa, 1977; Lozano-Pérez and Wesley, 1979; Lozano-Pérez, 1983). Skeleton programs (Taylor, 1976) are parameterised robot programs for particular tasks which include motions, error tests, and computations, in which error estimates are used to instantiate parameters in the skeleton.

Brooks (1982) developed a formal method for using symbolic constraints to propagate error computations. It is used to check the plan in the presence of errors in the parts' placement and tolerances. The computation can be reversed and the desired resultant tolerance can be used to infer the required initial tolerances or the necessity for sensing. An analysis of the spatial uncertainties arising from tolerancing was

presented in (Fleming, 1987). The uncertainties come from the constraints of individual relationships and the difference between the nominal sizes of parts. Fleming used spatial reasoning to analyse the possibility of parts of given tolerances fitting together. This method can be used in a planner which works in an ideal world to cope with tolerancing.

In part mating operations, compliant motion is necessary but difficult for humans to specify. The study in (Lozano-Pérez and others, 1983) described a formal approach to the synthesis of compliant motion strategies from geometric descriptions of assemblies and explicit estimates of errors in sensing and control. Using configuration space, the pre images of a goal region are computed recursively back towards the current position (backward chaining) and the resulting strategy is a sequence of guarded motions[1].

Variation in the position of an object can also be taken into account in configuration space by creating a sphere around the point representing the manipulated object, where the radius of the sphere represents the positional uncertainty (Lozano Pérez and others, 1983). Erdmann (1984) developed a representation of friction in configuration space which is used in Erdmann and Mason (1988) to predict the motion of a sliding object.

Donald (1987, 1990) proposed a strategy to incorporate uncertainty in the geometry of the environment (model errors). His system takes into account sensing errors, control errors and model errors. It plans by using sensor based gross motions, compliant motions, and pushing operations. Some experimental work is reported in Jennings, Donald and Campbell (1989).

This kind of motion planning is highly analytical. In practice, motion planning is computationally expensive because the required information about the geometry of the parts, the tasks and the environment is extensive, and because a large amount of knowledge is required to support the reasoning systems. In coping with the real world, some simplifications are adopted to reduce the complexity of calculations. These systems also cannot deal with deviations that occur at execution time. In

[1] A guarded move is the strategy of moving under trajectory control until some sensed condition is met, and then halting motion.

order to cope with events that cannot be pre-computed, sensors are incorporated into the system.

2.2 Incorporating sensing into robotic assembly tasks

Sensor feedback was used in assembly as early as (Inoue, 1974). He used a robot with force feedback to achieve tight tolerance assembly. The assembly strategies adopted played a significant role. For a peg-in-hole task, the alignment was achieved by sliding on contact surfaces. Other strategies adopted were tilting, aligning, and pushing. These techniques are known as compliant motion. An overview of compliant motion can be found in (Mason, 1984). One robot programming system based on compliant motion is (Buckley, 1987). The planner plans compliant motion strategy by decomposing the task's configuration space into a finite state space, then searches and expands states to find a path from start state to goal state. States are linked to each other by arcs, which represent reliable compliant motions.

In his work on uncertainty reduction for robotic assembly from a manufacturing point of view, Gordon (1986) studied two methods: one is by engineering the world, and the other one entails measurement and correction. The latter method used a light stripe vision technique to measure the alignment errors for peg-in-hole assembly. To reduce the cycle time of an assembly, the sensing is performed while the robot is moving. A light strobe is used to freeze the image and the position of the part is determined by reading the robot position at the same time. This work showed that by careful measurement, tightly toleranced and high speed assembly can be achieved.

Yin (1984, 1987) extended RAPT (Popplestone and others, 1980) with its modelling system ROBMOD (Cameron, 1984), to include visual sensing to determine the transformation between the nominal part position and the actual part position. A user can specify a vision task to determine the deviation of an actual position from a planned position. The system reasons about spatial constraints at compile time and leaves run time variables to be instantiated by the vision system. However

the use of sensors to change the ordering of actions was not easily achieved without bringing more information into the planning system.

The Handey system (Lozano-Pérez and others, 1987) is an exercise in system integration. Handey can locate a part in a pile of objects, choose a grasp, plan a motion to pick it up, and move to place it at the destination. It used a laser range finder with a vision system to locate the parts. In order to integrate the system, some simplifications were adopted. Collision free path planning is in a reduced dimension: the last three links, the hand and any object in the hand are replaced by a box. Grasp planning was performed based on the potential field method (Khatib, 1986) instead of fine motion synthesis. They concluded that, although the heuristic planner left much to be desired, using constraint propagation and satisfaction techniques were extremely difficult and computationally expensive, so they were not used.

Spar (Hutchison and Kak, 1990) plans assembly tasks and takes into account spatial uncertainty. Spar uses knowledge about uncertainty in the world description to assess the possibility of run-time errors and adds sensing to reduce uncertainties, or, if the resultant uncertainty is still too large, the system adds post-verification sensing. Spar uses a hierarchical refinement method of planning: from high-level operations (i.e. pick up a block), to geometric configurations of actions (i.e. the position and orientation of the hand relative to the object that is to be grasped) in an ideal world (e.g. without uncertainty), then adds uncertainty-reduction to the ideal world plan. When the plan fails because the uncertainty is too large, sensing operations are added to the plan for verification, and when possible, precompiled error-recovery plans are also added.

Spar has limitations: 1) it concerns only the end points of actions, and does not plan the motions required to move the manipulator (neither can it plan fine motions or compliant motions), and 2) its geometrical reasoning is limited in that its constraint manipulation system cannot distinguish between the relevant and irrelevant features of geometric configurations, because that depends on the task.

The InFACT system (Hardy and others, 1992; Loughlin, 1992) uses separate sensing and action modules which are linked by a supervisory system. Uncertainties and variations are dealt with by measurement strategies which are only used in situations where problems or ambiguities may occur. Each recognised problem is

matched with a specific measurement strategy to deal with it, for example, a strategy of moving a part past a tactile sensor in a specific direction may resolve an ambiguity about the orientation of a part.

2.3 Visual sensing

Visual feedback has been used to guide robots in hand-eye coordination tasks since the early days of robotics research. Jones (1971) used a simple tracking method to control a camera's aim and derived useful information from the image in terms of pixel coordinates. He demonstrated the use of visual tracking methods to perform block stacking and loose insertion. The system was limited by the technology of that period: PDP 10 computer, framestore of 252×238 pixels, with acquisition time of 32 ms, 6 bits per pixel. Real time performance was accomplished by using a small window of 36×36 pixels to track simple features of objects, i.e. area, centroid, perimeter, etc.

A geometric modelling system was introduced to facilitate the method of specifying objects, and to allow programming robots in terms of objects' features. The geometric model is also used to reason about spatial occupancy, for collision avoidance, and to plan robot actions. The use of geometric modelling together with the control of a robot hand in a Cartesian coordinate system gives rise to the need to map objects into a common coordinate frame.

In a traditional robot vision system, to work out how to move a robot to manipulate an object, the location of that object must be found in terms of a common reference frame between the vision system, the workspace and the robot hand, (the so called 'world coordinate frame'). It requires a calibrated vision system, i.e. camera parameters must be known. The systems must incorporate explicit geometric models of the environment, the robot, and the camera. Success of operation relies heavily on the accuracy of these models and the calibration of the vision system with respect to the robot and the environment. Methods of calibration to get an accurate model of camera geometry are well established (Tsai, 1987; Chang and Liang, 1989).

Also, a traditional robot system works in its reference frame (the so called 'base coordinate frame'). A model of the kinematics of the manipulator is used to translate between the Cartesian frame and the measured joint space (Paul, 1981). The measurements come from the information from the encoders in the joints of the manipulator. While the ultimate accuracy is limited by the resolution of the joint encoders in a particular robot configuration, in practice the accuracy of going to a location depends mainly on the accuracy of the kinematic model.

Stereo vision can be used to derive depth information. To do so, the location of the two cameras must be known in order to do triangulation calculations. For example, edge-based binocular stereo can provide accurate depth data, provided that the camera geometry is known (Pollard and others, 1989).

An example of the use of vision to determine the position of the part is (Horn and Ikeuchi, 1983). The vision system determines the position and attitude of a part in a pile of parts using photometric stereo. The attitude of an object is determined using a histogram of the orientations of visible surface patches, matched against prototype models. Because the photometric stereo does not provide absolute depth information, a robot executes guarded moves to pick up the part using the infrared light beam sensor at the finger.

Lougheed and Sampson (1988) described bin picking by a robot that can operate in real-time. The system uses a laser-ranging sensor and a special cellular array pipeline processor to construct 3D information of the parts. Using this information the best grasping point is chosen from the contour map of the scene.

In dealing with a dynamic world, a ping-pong player robot (Andersson, 1988) combines real-time visual feedback with the task that demands high speed response. The system uses special hardware and sophisticated calibration techniques to achieve the accuracy required.

Inoue and Inaba (1984) presented a system which used visual feedback to control a robot hand to handle a flexible object, a rope. It used measurements of specific information, the tip of the rope. In a more general approach, a full geometric model of robot hands, parts and workcell were used in Sakane and others (1987), Heikkila and others (1988). They focussed the visual processing on small sub-windows and

tracked edge-type features. They used two cameras to guide a hand and planned a viewpoint based on stability criteria to get an occlusion free viewpoint.

Weiss (Weiss, 1984; Weiss, Sanderson, and Neuman, 1987) proposed visual servoing of a manipulator system based on position or image data, and later an adaptive control scheme based on image feature references was developed, where the values of image features were used in the control loop. One method suggested was a system that measures features in image space. This could be applied to 'teach-by-showing' in the control of a camera in hand robot. The results of studies performed show the possibility of its use in 1, 2, and 3 DOF task. The features to be used in feedback are selected based on the degree of coupling between features and the desired control parameters and the minimisation of the sensitivity to change along the desired trajectory. In higher DOF tasks, the selection is done by trial and error.

Aloimonos and others (1987) presented some theoretical work on active vision. An active observer performs activities to control the geometric parameters of visual sensing. It is shown that the controlled alteration of viewing parameters yields stable and robust perception in a more computationally efficient way than a passive observer. It is worth noting that the technique of using image sequences, rather than static scene analysis, helps to simplify stereo calculations (Brown, 1988) and is useful for deriving depth information (Hayes, 1989).

An uncalibrated stereo vision system (i.e. no prior calibration, or self-calibrating on the run) has been used to navigate a mobile robot (Brooks, Flynn, and Marill, 1987). Sarachik (1989) also employed an uncalibrated vision system to determine the size and shape of a room by a mobile robot. The advantage of an uncalibrated vision system is that it does not rely on precise modelling or precisely calibrated equipment.

Wolfe and Richards (1990) demonstrated in a 2D hand-eye system that by using a vision system to track an end-effector in real-time in conjunction with a Cartesian controller, a significant improvement in robustness to the effects of vision system miscalibrations and errors in the parameters of a kinematic model can be achieved.

The type of vision sensing that is used in systems that employ a world model (Horn and Ikeuchi, 1983; Inoue and Inaba, 1984; Sakane and others, 1987; Heikkila and others, 1988; Lougheed and Sampson, 1988; Andersson, 1988) requires accurate

calibration to establish the mapping between the sensor's field of view and a common coordinate system. This type of vision system cannot easily cope with changes in camera parameters, such as changing the position of the camera or changing the view point, zoom, etc. Uncalibrated vision systems avoid problems normally associated with static cameras and also exploit the motion of the observer to derive some useful parameters for the purpose of accomplishing the task in hand (Brooks, Flynn, and Marill, 1987). Visual servo control, which uses image features (i.e. image areas, and centroids) as feedback control signals, eliminates a complex interpretation step, i.e. interpretation of image features to derive world coordinates (Weiss, 1984).

2.4 Behaviour-based systems

The behaviour-based approach to robotics began with mobile robots. In his now well known paper Brooks argued against the traditional AI approach of relying on the symbol manipulation system (Brooks, 1986). His approach to mobile robotics contrasted strongly with that of other mobile roboticists (Nilsson, 1984; Moravec, 1983; Giralt and others, 1984). He advocated the decomposition of system into *task achieving units*[2] instead of the traditional *functional units*.[3] For Brooks, the *program* of a robot is composed of levels of competences in which higher levels can subsume lower levels. At each level, behavioural modules tie their sensors to actions, are interconnected and execute concurrently. The robustness of the system comes from the fact that lower levels can perform their own tasks despite the absence of higher levels. Brooks' robots were built incrementally, i.e. higher levels can be added on without any alteration to lower levels. He called this the *subsumption architecture*.

He demonstrated the new approach in a series of mobile robots (Brooks, 1991b). His early robots have simple goals, for example roaming the room without hitting any objects. More sophisticated behaviours are demonstrated in a mobile robot with an arm on board that can find and pick up a can (Connell, 1989) and a six-legged

[2]We will call these, *behavioural modules*.

[3]*Function* here being interpreted as computational functions, i.e., a modularisation based on information processing requirements.

walking robot (Brooks, 1989). The use of a vision system is demonstrated in a mobile robot that can chase an interesting object (Horswill and Brooks, 1988).

How to create this *program* seems to be a difficult task. Presently, the programs to generate particular behaviours are individually hand-crafted. Kaelbling (1988) proposed a scheme to synthesise behaviours from a goal specification. The code generated by her scheme was decentralised and employed fixed priorities to resolve any behaviour selection conflict at compile time. In a sense, this attempts what (Lozano-Pérez, Mason, and Taylor, 1983) try to do in a well characterised situation in assembly tasks. Brooks (1990) produced a language, the *Behavior language*, which could be used to implement behaviours on mobile robots. This language compiles into subsumption architecture programs with a fixed priority arbitration scheme.

A formal model of distributed computation for sensory-based robot control has been studied in (Lyons, 1986). Behavioural modules can be seen here as perception action agents. The model of sensori motor action is based on schemas (Arbib and others, 1984) and can be used to describe behavioural modules. His intention was to use it for the formulation and verification of robot programs.

There has been very little work on behaviour based robotic assembly systems due to its youth. Emerging from previous experience with the classical robotic assembly system (RAPT), we can find arguments for the behaviour based approach in Smithers and Malcolm (1989). It describes the complexity of geometric and uncertainty reasoning in the construction of robot programming support systems that make use of sensors to reduce uncertainty. Smithers and Malcolm argue that robot programs should be constructed in terms of task achieving behavioural modules. Each behavioural module contains its own sensing. The distinction between sensing and action is hidden from higher levels. A robotic assembly system based on this concept is reported in Malcolm (1987). It is a complete and fully autonomous system. It plans and executes the assembly of Soma parts[4]. Although this system doesn't use sensors, it is designed to facilitate their inclusion.

The behaviour-based architecture avoids the problems of complexity inherent in classical systems by not trying to build a model of the world, therefore the computa-

[4]A detailed description of this system is given in chapter 4.

tional problems of uncertainty are reduced. In order to achieve a task, sensing-action feedback loops are used in a tight coupling with the world. Also this paradigm advocates building and testing robots in the real world so that from the bottom-up, design decisions have to be made to ensure that robots operate robustly in the real world.

2.5 Conclusion

From the early days (Ambler and others, 1975), it has been shown that robot programming is difficult. Attempts to raise the level of robot programming languages revealed many hard problems. For example the use of a geometric model of the objects leads to analysis which is complex and computationally expensive, such as configuration space, uncertainty analysis, etc. Planners that deal with uncertainty are very complicated. Besides various efforts on these problems; the integration of robotic assembly systems has been discouraging, these complicated techniques must be simplified in order to make the robot work in real-time.

The use of sensors is necessary in robotic assembly but the attempts to automate the generation of strategies for using sensors have also met with similar problems. The emergence of behaviour-based systems showed that these problems can be solved in a different way. Most of the work so far on behaviour-based systems is on mobile robots, but the architectural problems faced by assembly robots are similar. We conclude from this that it should be possible to develop robot systems with complex behaviour, and reliable achievement of their goals, with notably less computational complexity than suggested by the classical approach, by using the behaviour-based approach. The key concept is in the decomposition of tasks into *task-achieving units*.

The work in this thesis is built on the foundations of the behaviour-based robotic assembly system proposed in (Malcolm, 1987; Malcolm and Smithers, 1988b; Smithers and Malcolm, 1989). This thesis sets out to incorporate vision sensing into a robotic assembly system. The basis of the vision system is based on the method of visual servoing described by (Weiss, 1984), and is inspired by (Jones, 1974) in the use of simple processing to satisfy real-time constraints.

Chapter 3

Behavioural modules

What magical trick makes us intelligent?
The trick is that there is no trick.

Marvin Minsky (1987)

This chapter describes the main idea of this thesis: the concept of behavioural modules. We address the problem of how to program assembly robots to perform reliably in the presence of uncertainty. The classical approach to this problem is described and its difficulties in dealing with uncertainty are analysed. We introduce the behaviour based approach as a means to solve this problem. It employs an alternative to the classical approach to task decomposition. Criteria for task decomposition are proposed and discussed. These criteria directly address the problems of the classical approach. In this new approach, a task is decomposed into task achieving units called behavioural modules. An example is given to compare the two methods of task decomposition. We then discuss the characterisation of behavioural modules, their design and implementation.

3.1 Problems with the classical approach

The purpose of the process of assembly is to bring the parts involved into desired relationships as specified in the final state of assembly. These relationships and the ways in which they can be achieved are constrained by the geometry of the parts. The ordering of the assembly cannot be arbitrary. It is constrained too, by the shape

of the parts. In order to assemble two parts together, some part mating strategies are required. The assembly at this level is described in terms of part motions. But current assembly robots are programmed only in terms of their motions. If we know exactly the shape of the parts, and the relationship between the robot and the parts, then we can derive the robot motions from the desired part motions directly (Popplestone and others, 1980). But 1) in the real world the shape of the parts are not perfect; 2) the relationship between the robot and the parts are not known exactly because of the uncertainty in the location and dimensions of the parts; 3) and it is useful to use compliant and constrained motions where the motion of the part is only indirectly – if at all – related to the motion of the robot (Mason, 1985), for example: *dropping* a peg into a hole, or the use of remote centre compliance device (RCC) (Whitney and Nevins, 1979). Therefore it is difficult and sometimes impossible to derive a robot motion strategy to achieve the desired part motion from the knowledge of the required specified relationships.

In the classical approach, an assembly system plans for robot motions. It makes assumptions about the predictability of the physical world and only works when the assembly work cell is engineered to keep those assumptions true. It achieves reliability by 1) using a world model and 2) engineering the environment to be close to this model. If this engineering goes wrong, the assembly will not work. This is because the assembly is achieved as a side-effect of robot motions that interact with the environment to achieve the desired part motion. Because the goal is not known it has no control over this interaction, and therefore cannot correct the deviations of the actual situation at run-time from that predicted by the model.

The classical approach bases the interpretation of sensed data on a central world model. It relies on the calibration of sensors, which needs a world model and an accurate model of the sensors. For example, in visual sensing, calibration is done to establish the transformation from an image reference frame to the world coordinate frame. Lens parameters are also derived from the calibration based on the camera model. If the camera has moved, the new transformation is calculated from the previous one. Its accuracy depends on the accuracy of motion, which relies on the accuracy of the kinematic model of the device that moves the camera. For example, if a resolution of 0.5 mm/pixel is required and the camera is at a distance 50 cm, the resolution of the angular motion must be at least 1 milliradian: this is a severe

requirement of motion accuracy, especially when carrying what is often a relatively (for a robot) heavy camera.

The problems with the classical approach are that 1) it assumes the world to be perfect, then 2) the uncertainty in the geometry of the parts and their locations are added in and are analysed to determine whether sensing is necessary to reduce uncertainty to an acceptable level. This makes it necessary to do the uncertainty analysis after a plan or partial plan has been created. The results of this analysis are then used to correct or modify the initial plan (Brooks, 1982). Sensing can be introduced to reduce uncertainty at some point in the plan (Lozano-Pérez and Brooks, 1985; Hutchinson and Kak, 1990). This cycle of plan, analyse, and modify is repeated until the analysis uncovers no further problems. This makes the production of each robot program a large task since the problem of uncertainty plagues all aspects of a program.

The classical approach deals with uncertainty in an *ad hoc* way which is based upon computationally expensive analytical techniques which are only able to represent a subset of the types of uncertainty actually experienced at execution time. As a result the task of planning and programming a robot is made very complicated and the uncertainty analysis carried out is specific to each task. And yet the programs so produced are known not to work in practice without further on line testing.

3.2 Behaviour-based robotic assembly systems

The behaviour based approach suggests an alternative way of decomposing the problem of programming robotic systems. It separates the problem of task planning and programming from the problem of dealing with the inevitable uncertainty of the real world. An assembly system is decomposed into two parts: a planning system and an execution system. They are interfaced via a plan. The plan describes part motions that achieve the desired assembly. These part motions are translated into robot motions by the execution system.

The difference between this approach and the classical approach as typified by TWAIN (Lozano-Pérez and Brooks, 1985), Handey (Lozano-Pérez and others, 1987),

and Spar (Hutchinson and Kak, 1990), lies in the fact that in behaviour-based systems the planner relies on the execution system to cope with uncertainty. As a consequence it is able to plan in an ideal world, unlike the classical systems; TWAIN and Spar plan from task-level specifications and synthesise sensing and motion strategies down to the manipulator-level; Handey, although it uses sensing in its run-time system to locate objects and to make decisions about grasping, still relies largely on the planning system to plan for robot motions.

In the behaviour-based approach the execution system is composed of task-achieving units. These modular units are designed, implemented and tested in the real world to perform their intended tasks reliably, and they can be guaranteed to perform their tasks within a certain range of uncertainty. They are called behavioural modules. Behavioural modules facilitate the use of sensors to cope with uncertainties, therefore increasing the reliability of assembly tasks. It is an abstraction that makes an assembly robot more easily programmed. This is achieved by hiding the specifics of the use of sensors as much as possible from a planner and by avoiding *explicit* reasoning about uncertainty.[1] The planner, therefore, deals with an ideal world in which the robot carries out its operations reliably in the presence of uncertainty. This leaves the planner to deal with the problem of reasoning about the ordering of the assembly and other high level assembly strategies rather than concerning itself with the actual details of robot motions in the assembly cell.

A behavioural module is implemented as robust, goal achieving actions that are executed in the real world. Programming a robot is a process of composing abstract operations of robotic devices, i.e. robot motions, the use of sensors, object manipulations, etc. The behavioural module maps these abstract operations onto a physical instance of the task.

The interplay between robot motions, objects, and sensing is encapsulated in a behavioural module. It makes use of action coupled with sensing to gain information about the world to achieve the task in an economical fashion; this kind of

[1] *Implicit* reasoning about uncertainty, i.e., reasoning by the system designer which is incorporated into the system design, and so is not accessible to the system's own reasoning, is permitted. This distinction is discussed in (Malcolm and Smithers, 1989).

dynamic sensing can reduce the computational requirement of sensing a lot (Aloimonos and others, 1987; Brown, 1988). The encapsulation of sensing within a behavioural module is important. It allows the management of complexity in coping with uncertainty and facilitates the use of sensors. It enables the uncertainty to be encapsulated within a behavioural module and therefore limits the propagation of uncertainty. It trades some sensory/action redundancy for the benefit in computational simplification.

A behavioural module can be parameterised for a range of objects. Although it depends on the geometry of the objects, it can be parameterised for a class of objects with similar features, for example, similar shaped objects of different size. This makes behavioural modules usable for a class of tasks. Behavioural modules can be regarded as library routines for assembly tasks. They provide a set of capabilities of a robotic assembly system to a user. The user can then use these available behavioural modules without having to invent new behavioural modules for every new assembly in that class.

An important point of behavioural modules is that they abstract away some of the uncertainty management from the user of a robot such that the module will 'do the right thing' for its task. This abstraction is important because, by using behavioural modules, a user can compose a robot program to do assembly without having to be concerned with the details of the use of sensors to manage uncertainty and the program will work reliably. There are many levels of abstraction, from the details of physical devices, to the intermediate level of the strategy of assembly to the highest level such as a behavioural module that 'makes' the whole assembly. A behavioural module will not be useful if it is expressed at an inappropriate level. A robot program comprised of behavioural modules is easy to write and to test for its correctness because of the level of abstraction that behavioural modules provide. It can raise robot programming to the task-level.

The main difference between behaviour based systems and classical systems is that in behaviour-based systems the management of uncertainty is at the behavioural modules' design stage rather than at the planning stage. Most of the uncertainties are dealt with at a local level. A planner for behaviour based systems is therefore much more tractable than a planner for classical systems. Also behavioural modules are designed and tested in the real world, so they are likely to be more robust than

classical systems which have to take into account large amounts of detail about the real world in order to plan robot motions to achieve a given task.

3.3 Criteria for decomposing a task into behavioural modules

How should a task be decomposed into behavioural modules? We shall refer to some work in the computer science literature on system design and program design for providing a basis to develop our answer to this question (Parnas, 1971, 1972; Yourdon and Constantine, 1979). Parnas suggested decomposing a system into *design units*. Design units are related by interfaces which are any sort of interrelationship or interdependency. A structured description of a system shows the system divided into a set of modules, gives some characteristics of each module, and specifies some connections between modules.

The structure of a system is important because it has impact on: 1) the ability of the system to tolerate changes and 2) the ability to check for correctness. A system should be divided into a number of modules with well-defined interfaces; each one is small enough and simple enough to be thoroughly understood and well implemented.

The connections between modules are the assumptions which the modules make about each other. A good design should strive for a system that has weak connections because it will give the system the ability to tolerate changes. A change that occurs in a single module will not affect other modules only as long as the connections still 'fit'. A module should have a simple interface; that is, it should minimise the amount of information flow passed in the connections. This suggests that the connections should contain as little information as necessary.

The question about task decomposition will be addressed by using these concepts of modularisation.

The central problem of assembly systems is the presence of uncertainty. The classical approach addresses this problem by using a centralised representation of the world which is updated by sensors and needs the added complexity of being

decorated by uncertainty specifications and relations. The use of an explicit world model requires that all knowledge about the world should be collected and converted to a symbolic form. There are problems of choosing a common representation and the interpretation of sensed data. Also the central world model encourages the use of complex and computationally expensive algorithms.

The connections between modules in the classical approach are centred on this world model, therefore the modules are strongly connected. When uncertainty arises in one module, it propagates and affects all the rest of the system. It is difficult to keep this world model accurate. Its accuracy depends on various factors: the accuracy of the model of sensors, the accuracy of the kinematic model of the manipulator, the calibration procedure, the accuracy of a motion etc. When there is a change in the system: for example, objects have moved, the lighting condition has changed, it might invalidate assumptions about the world of other modules. Because all modules are strongly connected it is difficult to limit the effect of changes. This sensitivity to small changes of position, uncertainty, etc., happens because the world model is constructed in terms of position and uncertainty in order to be able to construct a robot-motion-based plan.

Another criterion to be considered is the degree and nature of coupling between sensing and action. A robot might derive the state of the world from its input sensors, then use its reasoning system to decide what commands to send to the output effectors. This cycle will then repeat. The amount of symbolic processing affects the cycle time. This cycle time must be fast enough to pre-empt significant change in the world. In the classical system, this process of the deriving the state of the world from sensed data and reasoning about cause and effect can be very expensive (therefore slow) because all knowledge about the world must be included; it is difficult to select what is important, the so called *frame problem* (McCarthy and Hayes, 1969).

Agre and Chapman (1987) argue that general reasoning is not often required in many situations, for example, rather than stating that: 'for all car x, if robot is moving toward x then slow down', in order to use the knowledge that is indexed by the sensors, we could use: 'if thing in the centre of visual field is getting closer, slow down', thus avoiding searching the world model for x.

In fact, one of Brooks' mobile robots (Brooks and others, 1987) uses this scheme to avoid hitting objects by measuring 'time to collision' without recognising objects. Brooks and his colleagues (Brooks, 1989, 1991b; Connell,1989) have had remarkable success in building complex systems that can co-ordinate the behaviour of different modules without using a representation of the world and without direct communication between modules. Instead the modules are initiated by sensing the effect of activity of other modules or the perception of the world. Using the perception of the world, rather than parameters, as a control-passing mechanism reduces the assumptions made by the system, thus increasing reliability.

Ideally, sensing and action should be tightly coupled at a low level to simplify processing and to limit the information flow to other modules. At one extreme of the spectrum, sensing and action can be built into mechanical systems, like the remote centre compliance device (RCC) (Whitney and Nevins, 1979). Other examples are:

- a mobile robot (Brooks and others, 1987), which can wander around a room while avoiding hitting obstacles, using a cylindrical lens to compress an image into one horizontal strip, which works well for detecting vertical edges in the environment;

- Sarachik (1989) used the sequences of the central strip of the image to construct a view of a room to detect the height of the ceiling;

- retina chips by Mead (1989), an analog VLSI device that can detect visual motion in real-time using very little computational power.

This exploitation of contextual information and the tight coupling of sensing and action reduce the sense-act cycle time and simplify the computation.

In conclusion three criteria are proposed to decompose robotic assembly tasks into behavioural modules:

1. There is no reliance on a central model of the world for sensing and action.

2. Sensing and action should be tightly coupled within the module.

3. Prefer to pass control via perception of the world rather than by parameters.

The next section discusses the characterisation of behavioural modules and shows that their characteristics are the direct consequence of using these decomposition criteria.

3.4 An example

In this section, we give an example of two alternative decompositions of a task to illustrate the differences between the classical and behaviour based approaches.

Suppose we want to pick up a part. A camera is used to locate the part. The classical approach will decompose this task into two steps:

1. locate the part (`locate-part`);
2. move the hand to pick it up (`pick-up`).

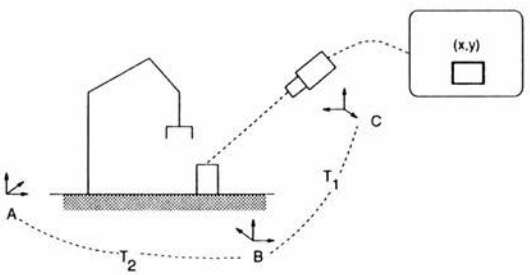

Figure 3–1: The classical approach

There are 3 reference frames: A the robot, B the world and C the camera. The camera sees the part at (x, y) in its image coordinates. The location of the camera must be known, i.e. the transformation T_1 from C to B. A calibration procedure is used to find the mapping of image coordinates to world coordinates. It must take into account the lens parameters (focus, distortion etc.). After this, the location of the part in the world coordinate can be calculated. This location then is transformed into the robot's coordinate, knowing the transformation T_2 of the reference frame B to A. Then the robot is commanded to pick the part up.

The success of this task depends on knowing T_1 and T_2 to a sufficient degree of accuracy. T_1 depends on the accuracy of the calibration of the camera.

Alternatively, using the behaviour-based approach, one way that we can decompose this task into 3 steps:

1. move the hand down to M (`move-down-half`);
2. move the hand from M to N, N is directly above the part (`move-above`);
3. move down to pick up the part (`move-down-pick`).

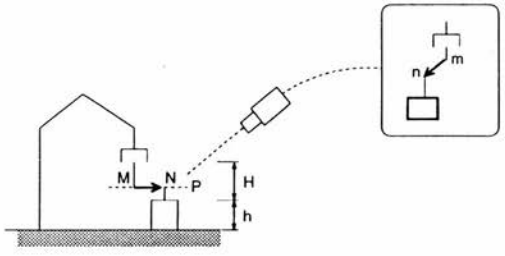

Figure 3–2: The behaviour-based approach

We assume that the hand is near the part so that the camera can see both the hand and the part at the same time.

Step 1: The hand moves down to M, half the height from the part (so the height of the hand above the part H must be known). It also relies on the fact that the hand moves down vertically.

Step 2: From M, which lies on a plane, the hand moves to N by visual servoing: the camera observes the relative position of the hand (m) and the point above the part in the image (n), and repeatedly moves the hand (only moving in the plane P)[2] to bring it to n. This is achieved by iteratively using a local mapping between the hand's commanded motion and its observed motion in image coordinates.

[2]This is because we don't know how far the part is from the camera. Using 2D information we can only control two of the degrees of freedom of a hand motion.

Step 3: The hand moves down and grasps the part.

This second method doesn't rely on knowing the camera position, nor the precise calibration of it. Instead, it employs a tight coupling between sensing and action to guide the hand to the part. The knowledge that is required here is much simpler than that needed in the first method; only H and h, instead of the transformation matrices T_1 and T_2. Because the second method doesn't use common coordinates to communicate, it doesn't matter if the camera has moved slightly. Because visual servoing is used, it will not matter if the robot doesn't have the required accuracy.[3] As long as the robot and the camera have enough resolution, the task will be performed successfully.

When we compare the connections between modules in both decompositions, we can see that the classical approach has much stronger connections than the behaviour-based approach. In the first method, `locate-part` and `pick-up` are connected by the world coordinates and the shared object model, thus the assumption about the accuracy of T_1 and T_2 is important. `locate-part` is also required that its knowledge about the camera calibration be accurate. In the second method, `move-down-half`, `move-above`, and `move-down-pick` are much more weakly connected. `move-above` doesn't need to know about the camera calibration, therefore the lens can be changed or adjusted and the camera can be moved. The behaviour based approach doesn't use T_2 either, so it can be robot independent.

[3]The *accuracy* of a robot refers to the precision with which the robot can go to a position described in terms of a general spatial co ordinate. Accuracy depends on the accuracy of the kinematic model of the robot, the construction of the robot and its calibration. *Resolution* refers to the smallest amount of movement that a robot is capable of detecting. Resolution depends on the robot construction and the resolution of the robot's joint encoders.

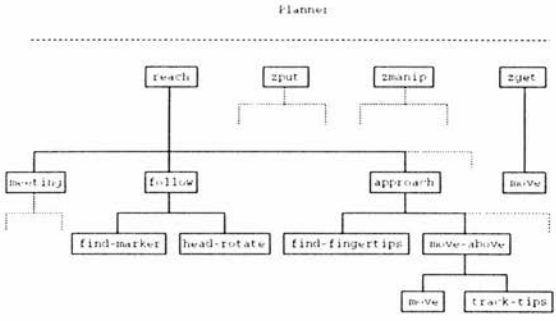

Figure 3–3: The hierarchical structure of behavioural modules

3.5 Characterisation of behavioural modules

In behaviour-based robotic assembly systems, the planning system relies on the execution system to perform the task reliably. The execution system is composed of behavioural modules which are combined together in a hierarchy with the top level interfaced to the planner (figure 3–3). We will discuss what are the characteristics of behavioural modules in the following paragraphs.

Behavioural modules are not just software.
A behavioural module may consist of several parts: mechanical, electrical, computer software, etc. They act together in the real world to achieve a task. A behavioural module can contain other behavioural modules, thus forming a hierarchical structure. Behavioural modules are connected to the world via their sensors and actuators. This implies that a computer system cannot by itself be termed a behavioural module.

Behavioural modules have well-defined interfaces.
The interfaces between behavioural modules need to be clearly defined. They are documented in the form of module specifications. Part of the specification for such modules should contain a clearly stated objective: what the module tries to accomplish, whereas another part states what this module assumes other modules

guarantee. Other information contained in the specification is the details of the parameters that are passed in and out of the module.

Behavioural modules don't rely on a global world model.
Each behavioural module has its own representation of the part of the world that is relevant to it, so the information to pass to other modules is minimised. Unlike classical systems, for example, RAPT and Spar (Popplestone and others, 1978; Hutchinson and Kak, 1990), which try to use one representation of the world for the whole system and rely on powerful inference techniques to deduce (calculate) what to do, behavioural modules only need private and local representations that are tailored to their tasks.

Behavioural modules have tight coupling of sensing and action.
Sensing must be designed to match the task. Questions about how to use sensors, what to sense, what actions to take, etc., are determined by the task that is to be performed. A behavioural module will exploit contextual information. It can make use of the knowledge of a motion that it has created to find the invariance in its perception. For example, a block on the table can be distinguished from a block in the hand by moving the hand and observing the block that doesn't move (assuming only two blocks in view and the camera stationary).

Behavioural modules prefer communicating via the world.
To make the connections between modules weaker (to assume less about what other modules will guarantee), a behavioural module should prefer knowledge from direct sensing of the world to knowledge about the world that is deduced from information from other modules. It has the effect that a change in one module will not affect other modules. This also suggests that a behavioural module should minimise its reliance on *a priori* knowledge. The less knowledge it has to use, the less chance of it being affected by the change in that piece of knowledge.

Behavioural modules are context sensitive.
Behavioural modules are designed to perform in a chosen environment, therefore they are not context free. The context is dependent on the layers of abstraction of the robot program: from a high level that describes an assembly in terms of part motions and part relationships; to a low level that describes the individual behaviour of a physical device. A behavioural module can be composed of other behavioural

modules. In this hierarchical structure, there is a level at which some modules don't contain other modules, we called such modules *grounded*. At the point at which we can not identify a meaningful objective for a module with regards to the task, in other words, when it becomes free from the context that we are interested in, that module will cease to be designated *a behavioural module*. Behavioural modules are purposeful. It is connected to the world by purpose via sensors which grounds them and gives them context sensitivity.

In Malcolm, Smithers and Hallam (1989), an emerging paradigm in robotic architecture is discussed. Many of its characteristics agree with our discussion of behavioural modules. Especially, in Malcolm and Howe (1990) the following properties have been suggested as general properties of behavioural modules:

- they handle the variations and uncertainties typical of the task;
- they integrate sensing and action at a low level;
- they are computationally minimal;
- they know (i.e., represent symbolically) as little as possible.

All of these fit into our characterisation.

3.6 Design and implementation

Behavioural modules are hand-crafted by a human designer such that they can perform the task reliably. The overall strategy for achieving reliability in the presence of uncertainty and variations in the work cell is devised by the designer. The difficulty of part manipulation in an assembly task is dealt with by this person who has years of experience in object manipulation to call on when designing a behavioural module.

A behavioural module can be composed of other behavioural modules, thus a hierarchical structure is formed. A hierarchical structure has the benefits that: 1) individual modules are simplified because they can use other modules in turn; 2) some modules can be reused.

The decomposition criteria enable a clean and well-defined interface which makes behavioural modules easy to use and implement. These two properties: hierarchical structure, and clean decomposition, are desirable but independent.

The implementation of behavioural modules can be carried out in an independent manner from other parts of the system development once the specifications of the behavioural modules are defined. Each module can be implemented separately in a bottom-up fashion. Their specifications facilitate the testing of the modules and the system integration process.

3.6.1 Combining behavioural modules

Each behavioural module can be treated as a reliable component. Modules can be combined together to form a new module that has a different ability. The range of possible operations can be extended this way. For example, one module can pick up a cylindrical part, another module can pick up a square block. Given that the requirements of both modules are not in conflict, they can be combined to form a new module which can pick up either part.

Behavioural modules can also be combined to handle an exceptional case in which one module has failed, by activating another module to perform a recovery task. For example, assume there are two modules, one module can track the target (`track`), the other one can locate the target (`locate`). If the `track` module loses its target, the `locate` module can be activated to re-locate the target. In this sense, the new module that combines `track` and `locate` can handle this exception internally. Wilson (1992) constructed behavioural modules this way to increase the reliability of an assembly system.

3.6.2 The reuse of behavioural modules

The broad general competence of the behavioural modules in the SOMASS system (Malcolm, 1987) was shown by their incorporation in an automatic planning system in which they were tested over a wide range of assemblies (section 4.1.4 p. 45). Because of the hierarchical structure of behavioural modules it is often possible to reuse some of their component modules for a new task. Behavioural modules are not

context free, but the adaptation of existing behavioural modules to a new task is not difficult. Many modules, especially low level modules, contain little contextual information about the task (they gain context from sensing), hence they can be easily adapted. The strategy for achieving a reliable operation may be changed according to the task, but if the new modules for the task use the old modules, they can inherit the old modules' reliability. Most of the examples from our experiments (chapter 4,5 and 6) showed that this is the case. Many modules were reused in the construction of new higher level modules.[4]

In many cases existing behavioural modules have a wide enough general competence that they can be used in new tasks without change. Where change is required, this is usually no more than a change of parameters. For example, the module that tracks a moving target (section 4.5 p. 53) was reused in the module that moves the camera head while keeping a target in the middle of its view (section 6.4 p. 74), and the module that finds a robot hand by moving the robot fingers and taking the difference of two images (section 4.6 p. 54) was used to start the tracking-hand module and subsequently was reused to find the hand in the calibration of the stereo head module (section 5.4 p. 65). In these two examples, the existing modules were adapted without change. In some other cases, as in the tracking module, the parameters of the intended target (its size, and other attributes) need to be changed when reusing it in a new situation.

Another way to look at the reuse of behavioural modules is to think of their descriptions (as in appendix A) as a design specification similar to a description of an algorithm in computer science. The level of description given in this thesis is not totally machine independent (as it depends on some of the characteristics of the device used in the experiments) but one can usefully derive some analysis or implement a concrete system from this description. This is quite similar to the use of a hypothetical computer by Knuth (1968) in his famous text book on computer algorithms in which he describes and analyses algorithms, and presents concrete results based on the running of these algorithms on a hypothetical computer (MIX).

---

[4]The structure of the behavioural modules in the experiments are explained in section 7.7. Also see figure 7.1 for a classification of behavioural modules into low and high levels, and the appendix A.3 for the details of the hierarchical structure of behavioural modules.

The reuse then is a process of using the same description in another program (as much as one can implement Knuth's MIX assembly language programs using any programming language).

3.6.3 Implementation requirements

Behavioural modules can be executed concurrently, for example, a robot hand is moved by one module and at the same time in another module a camera tracks it Therefore the run time system must be capable of creating and synchronising multiple processes.

A robot program, which is composed of behavioural modules, can be implemented in many available computer languages. It does not require that the robot program has to be in any special form. Behavioural modules can be implemented in an ordinary computer language but to run them concurrently requires that the system has the ability to create and start concurrent processes. The robot program can use the data abstraction and control structures that are available in that computer language. The ability to pass data between concurrent processes depends on the underlying mechanisms of the support system.

In summary, the behavioural module concept is independent of the implementation language (provided it has sufficient resources) and can be implemented using existing computer languages.

3.7 Conclusion

The behaviour based approach to robotic assembly systems suggests a solution to the problem of achieving reliable operations in the presence of uncertainty by a different decomposition of the assembly task. The task is decomposed into modules which have weak connections as opposed to the classical approach which has very strong connections between modules. The strong connections between modules in the classical system come from the common use of the world model. The classical system uses a centralised world model to communicate between its modules which makes the problem of uncertainty permeate into every part of the system.

By avoiding this centralised world model, by employing tight coupling between sensing and action, and by communicating through the world, which also weakens connections, behaviour-based systems achieve reliable assembly operations in the presence of uncertainty.

A behaviour-based assembly system is comprised of two parts: the planning system and the execution system. The planning system plans in terms of operations in an ideal world and concerns only the high level strategy of the assembly task. The execution system, which is comprised of task-achieving units called behavioural modules, deals with the uncertainty in the world and achieves reliable operations.

Behavioural modules are connected with the real world, they have hierarchical structure, they prefer direct sensing in the world, and they exploit contextual information. The sensing in behavioural modules is simplified by its tight coupling with action. The uncertainties are reduced by the use of sensors and other manipulation strategies inside the module, therefore avoiding placing the burden on to the user.

Chapter 4

Extending SOMASS: visual sensing

The principles for the design of behavioural modules have been proposed and discussed in the previous chapter. In this chapter, and the next two chapters (5 and 6), they are applied in order to construct a working system. We start, in this chapter, with an extension of the SOMASS system that incorporates visual sensing using a stationary camera. We then describe a two camera system in chapter 5, before going on to the use of mobile cameras in chapter 6. These three chapters will describe experiments in the design and implementation of new behavioural modules and discuss the test results. The analysis of the system will be described in chapter 7. All the visual processing algorithms will be discussed in detail in chapter 8.

This chapter describes an experiment in extending SOMASS. The aim is to investigate the problems associated with integrating sensors into a robotic assembly system. SOMASS is chosen because it provides a reliable, working system that is already available. It is used as a base to carry out experiments with new behavioural modules so that they can be tested for a complete assembly task, in a realistic environment.

A detailed description of SOMASS is given in the next section in order to facilitate an understanding of its extension. The experiments in the design and implementation of new behavioural modules are then described. The limitations, the comparison of these modules with the old modules and the method of integrating new modules into the system are discussed.

4.1 The SOMASS assembly system

The SOMASS (Soma Assembly System) by Malcolm (1987) is a complete and integrated planning and execution system which performs the assembly of Soma shapes (figure 4–1 and 4–2). Given the desired final shape of an assembly, the system plans the sequence of operations that will put the component parts together.

SOMASS is divided into two parts: the symbolic planner system and the execution system. The symbolic planner uses an abstract representation of the task and isn't concerned with details of the real world. It generates plans which are then carried out by the execution system. The execution system deals with variations and uncertainty in the locations and dimensions of the parts.

4.1.1 The Soma assembly domain

The Soma cube was invented by the mathematician Piet Hin (Gardner, 1972). Soma shapes are built from individual cubes which are joined together to form shapes which have a surface concavity. The Soma-4 set is the set of Soma parts that contains 4 or less cubes in one of each shape (figure 4–1). There are many different shapes which can be constructed from these parts (figure 4–2). The *shape-dependent part-fitting* properties of Soma assemblies retain the essential features of the general assembly problem.

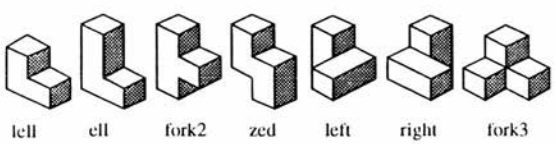

Figure 4–1: Soma-4 set

4.1.2 The planner

The planner is told the final shape of an assembly and it works out how the parts are to be disposed to form this assembly. There are rules of assembly embedded in the

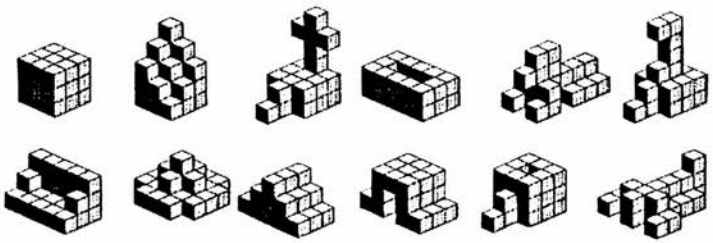

Figure 4–2: Several possible Soma-4 assemblies

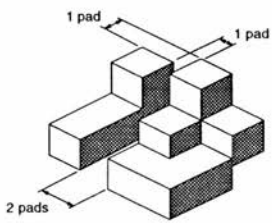

Figure 4–3: Padding of an assembly

planner. These rules aim to: 1) simplify the assembly operation, 2) deal with the fact that the robot that is used to perform assembly has only 5 degrees of freedom and 3) increase the reliability of assembly. The following paragraph lists some of these rules.

Soma parts are placed in a stable configuration; they have adequate space between them for the part acquiring operation; the parts are placed within known bounds; the parts are to be picked up by the top cube (the only exception is the *zed* shape which has two top cubes); the putting-down is done vertically; the topmost cubie (which is to be grasped) of each new part must not be lower than the topmost cubie of the rest of the assembly so far; etc.

The planner deals with the tolerance of the parts when they are placed into the final assembly by leaving gaps between them, called *padding* spaces (figure 4–3). When all parts are assembled, the final assembly can be pushed together; this operation is called *patting*. Salmon (1989) implemented a strategy that will pat most assembly shapes.

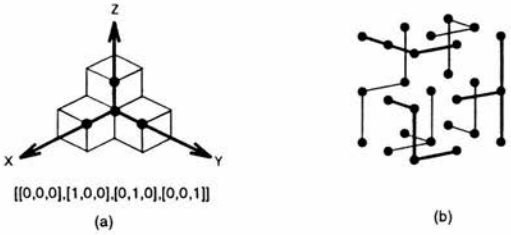

Figure 4–4: Representation of the shape a) a *fork3* part, b) a final assembly solution of a 3×3×3 cube

The planner was designed to have as little knowledge as possible about the details of the real world such as: the layout of the work cell, the size of the parts, etc. It uses an abstract representation of the shape of the part which is independent of the actual size of the part. A part is represented by a list of triples [x,y,z] of space that the cubes occupy (figure 4–4).

The planning process is based upon stepwise refinement: it forms a complete general solution at one level, and then adds constraints, solves the consequent problems, and adds details to the plan to form the next level. If any stage fails it backtracks. These stages in the processes are as follows:

1. A general solution is found in which the parts will fit into the final assembly.

2. A gravitationally stable ordering of the assembly is found. The part is to be inserted vertically and must come to rest in a gravitationally stable position.

3. Valid put-down grasps are determined. There must be clearance for the gripper fingers to open while releasing a part into the assembly without clashing with other parts.

4. Any necessary regrasps are planned. The intermediate put-down and regrasp strategy must be found for any part that has different pick-up and put-down grasps. The intermediate grasp must be a gravitationally stable orientation of the part. It takes into account the limited degrees of freedom of the robot.

After a final solution is found, the planner generates a plan incorporating all the parameters which are required by the execution system (figure 4–5).

```
; This is plan56 of the chair assembly using the soma4 part set.
;
    CALL sinit()
;
    CALL zjustz(b4.get, 2)
    ...
;
    CALL zpcalc(centre, b4.put, 1,-1,2,2*gap, 0*gap, drop)
    CALL zpcalc(centre, b5.put, -1,-1,2,0*gap, 0*gap, drop)
    CALL zpcalc(centre, b7.put, 1,1,2,2*gap, 2*gap, drop)
    CALL zpcalc(centre, b6.put, 0,1,3,1*gap, 1*gap, drop)
    CALL zpcalc(centre, b3.put, -1,1,3,0*gap, 1*gap, drop)
    CALL zpcalc(centre, b1.put, -1,1,5,0*gap, 0*gap, drop)
    CALL zpcalc(centre, b2.put, 1,1,5,2*gap, 0*gap, drop)
;
; ----------- The placing of fork3 -----------
    CALL zpatget(b4.get, RZ(90), -1.5,0.5, RZ(0), -1.5,0.5)
    CALL zget(b4.get:RZ(0))
; - No regrasping required.
    CALL zput(b4.put:RZ(-270))
;
; ----------- The placing of left -----------
    CALL zpatget(b5.get, RZ(90), -1.5,0.5, RZ(0), -1.5,0.5)
    CALL zget(b5.get:RZ(0))
; - Straight case.
    CALL zmanip(table, RZ(-180):RY(-90),1,0,2,RZ(0),0,0,2)
    CALL zput(b5.put:RZ(0))
;
; ----------- The placing of right -----------
    CALL zpatget(b7.get, RZ(90), -1.5,0.5, RZ(0), -0.5,1.5)
    CALL zget(b7.get:RZ(0))
; - Straight case.
    CALL zmanip(table, RZ(-90):RY(-90),0,-1,2,RZ(-90):RZ(0),0,0,2)
    CALL zput(b7.put:RZ(-90):RZ(0))
;
; ----------- The placing of zed -----------
    CALL zpatget(b6.get, RZ(90), -0.5,1.5, RZ(0), -0.5,0.5)
    CALL zpick(b6.get:RZ(0))
; - Reversed case.
    CALL zmanip(table, RZ(-90),0,0,2,RZ(-90):RZ(0):RY(-90),0,-1,2)
    CALL zput(b6.put:RZ(-90))
;
; ----------- The placing of fork2 -----------
    CALL zpatget(b3.get, RZ(90), -1.5,1.5, RZ(0), -0.5,0.5)
    CALL zget(b3.get:RZ(0))
; - Straight case.
    CALL zmanip(table, RZ(-270):RY(90),0,-1,2,RZ(0),0,0,3)
    CALL zput(b3.put.RZ(0))
;
; ----------- The placing of lell -----------
    CALL zpatget(b1.get, RZ(90), -0.5,1.5, RZ(0), -0.5,0.5)
    CALL zget(b1.get:RZ(-90))
; - No regrasping required.
    CALL zput(b1.put:RZ(-90):RZ(0))
;
; ----------- The placing of ell -----------
    CALL zpatget(b2.get, RZ(90), -0.5,2.5, RZ(0), -0.5,0.5)
    CALL zget(b2.get:RZ(0))
; - Straight case.
    CALL zmanip(table, RZ(-180):RY(-270),-1,0,3,RZ(0),0,0,3)
    CALL zput(b2.put:RZ(0))

.END
```

Figure 4–5: A plan generated by the planner

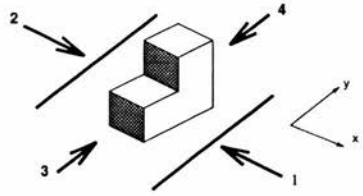

Figure 4–6: Sweeping motions to centre Soma part

4.1.3 The execution system

The execution system is composed of behavioural modules. Behavioural modules know nothing of the assembly order or the shape of the parts. The positions in the assembly are represented symbolically by the planner and instantiated at run-time. At run-time, the nominal initial locations of the parts, the nominal cube size, and several placing locations (such as a location which defines where the final assembly should be built) are instantiated.

Although the SOMASS system does not use sensors, its architecture was designed to accommodate them within the behavioural module. To cope with uncertainty without sensors, the SOMASS system uses constrained motion to reduce uncertainty (Mason, 1985; 1986). This technique is used in the modules that acquire parts. We will describe the behavioural modules in the system, they are: `zpatget`, `zget`, `zpick`, `zmanip`, and `zput`.

`zpatget` centres a Soma part by using a straight edge made of wood to sweep the part from four sides so that the part ends up in a known location. The first two sweeps constrain the x position and the last two constrain the y position (figure 4–6).

`zget` then is used to pick up the part from that location. `zget` does final adjustments to locate the part exactly in the middle of the grasp by making two snaps in orthogonal directions across the faces of the top cube (figure 4–7). For the *zed* part this is not possible because there are two cubes on the top. The module `zpick` is used instead for this part; `zpick` performs only one snap.

`zmanip` performs a regrasp operation. It transfers a part to a small regrasp table (due to the construction of the wrist of the robot that is used in this system, the gripper cannot reach the part from the side without hitting the table top, so a small

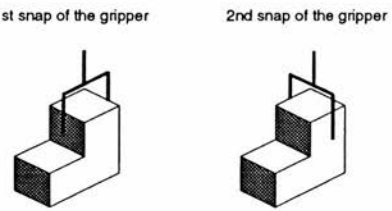

Figure 4–7: A double snap to centre a Soma part

table is needed to elevate the part) and then picks it up again in the new orientation. `zput` puts down the part into the assembly. The dimensional tolerance of the part is taken care of by the *padding* gap that is calculated by the planner.

4.1.4 Reliability

The results of some reliability tests for SOMASS were reported in Malcolm and Smithers (1988a). The planner generated 50 plans (39 for cube shapes and 11 for other shapes) and fast-running plans were selected for the repetitive test of a total 517 assemblies. This test represented the testing of about 4,000 lines of automatically generated robot code, in repetitive tests, for about 15 hours with 2.3 % failure (half of these were due to planner bugs).

4.2 Experiments in extending SOMASS: visual sensing

The SOMASS system uses no sensing. Instead it employs constrained motion to locate the parts. This section describes extensions to the SOMASS system which use vision sensing for acquiring the parts. This extension avoids the use of a global coordinate system for using the vision data and employs a self-calibration procedure to help achieve robustness. The position of the camera does not need to be known.

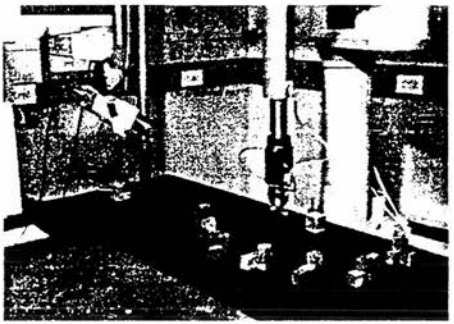

Figure 4–8: The assembly cell and Soma parts

As will be seen, the experiments also demonstrate that two robots, one holding a camera, and the other performing the assembly, can co-operate with each other in a very natural way without either having to know exactly where the other is

The first principle is to try and replace, as far as is possible, calculations by sensing. This equates to preferring perception over representation. Several techniques are used that contribute to this end. Frequent vision sensing, for example, allows motions in the world to be viewed as linear approximations, and only short term predictions of motion need be made for following an object. The tracking algorithm adopted is similar to that used by others (Jones, 1974; Brown, 1988).

The system is purposefully made as calibration free as possible. Relative quantities are used and self-calibration is done while carrying out tasks. This approach is possible because adequate sensing allows self-calibration. The need for calibration to establish the mapping of image data to world coordinates is absent from the system described here. The camera position need not be known.

The system does not rely on explicit models of the objects nor does each assembly agent need to know about the other(s). The system does not maintain representations of the Soma pieces, the camera, the robots nor of possible uncertainties due to inaccuracies of the model with respect to the real world.

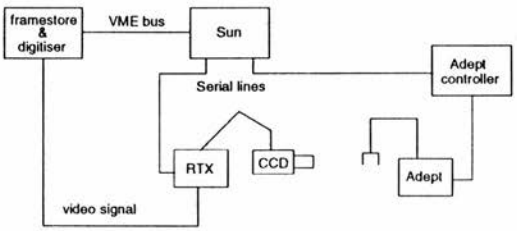

Figure 4–9: System architecture

4.3 Equipment and setup

The system adopted consists of two robots and a Sun workstation with some vision processing hardware (figure 4–9).

One robot is an AdeptOne industrial robot, with 5 degrees of freedom. This robot carries out the assembly task. It has a simple pneumatic close-open parallel gripper. The fingers are painted white and the rest of the hand painted black.

A second robot arm, an RTX made by UMI Ltd., is used to hold a small camera in the second experiment. The robot/camera combination is moved to track the movements of the Adept arm.

The Sun machine contains two MaxVideo boards: an image digitiser and a framestore. A subset of the data in the framestore is selected and analysed on the Sun. The hardware is driven in interrupt mode. Both robots are controlled via serial lines from the Sun. The software is written in C and Prolog.

The camera is a CCD video camera with automatic gain control, a Sony DXC-101P with a 16 mm lens. The work cell is lit by a white light lamp, and the surface of the table is covered by a black cloth, which gives adequate contrast.

The Soma set we used is made of hardwood and is not painted. It is in its natural texture and colour. The plan for the complete assembly is generated by the planner of SOMASS. Only the new vision-using part-acquiring module is substituted into the plan.

Two experiments were carried out.[1] In the first a behavioural module was devised (`hand-approach`) to pick up a Soma part from the work area. The camera is moved manually to aim properly at each part. Starting with the robot hand above the nominal (or predefined) position of the part, and the part in approximately its nominal position, a limited amount of image data combined with a simple strategy to move the hand can guide the hand to pick up the part. This was done without referring to robot, part, or camera coordinates.

The second experiment supplements the first. A behavioural module (`head-follow-hand-to`) capable of finding the robot hand is used. The camera is then able to follow the hand to a position directly above the nominal position of the part. This tracking behaviour thus aims the camera at the correct location for the `hand-approach` behavioural module to start, replacing the manual camera movements of the first experiment.

By using these two behavioural modules, the vision system does not have to recognise the individual part. Before a part is picked up the hand moves directly above the nominal position of the part and the camera follows the hand. Because the fingertips can be located their positions in the image are known. The area below the hand in the image is searched for the part (`find-part`). The camera is moved to aim at the correct part each time the new part is to be acquired.

4.4 First experiment : picking up a Soma part

In this experiment the camera is moved manually, for each part, so that both the robot hand and the part to be picked up are within the field of view. The part has a nominal position with the translation variance of the part being about twice its cube size and the rotation variance being plus or minus 30 degrees.

The camera looks down at an angle that can vary between approximately 30 and 60 degrees, and approximately plus or minus 15 degrees horizontally, from the

[1]The name of the relevant behavioural module is given in the form (`module-name`). They are also referred to in the analysis in chapter 7.

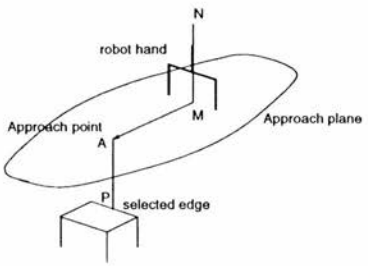

Figure 4–10: The diagram of the strategy

ideal 'head-on to the robot fingers' position. A wide range of positions are possible. The goal is to use the information derived from the image data to guide the hand to align the finger tips with one edge of the top cube of the part. The height of the part is assumed to be known, and the part lies flat on the working surface with the cube to be grasped pointing up. The technique used requires that both the fingers and the specified edge are in clear view.

4.4.1 Sensing-action strategy

The strategy is as follows (figure 4–10): a plane is defined, parallel to the work surface at a known height and above the tops of the parts, which is called the *approach plane*. There are two points of interest that are then projected onto the *approach plane*. The first (M) is the midpoint between the two fingers of the gripper. The second (P) is the midpoint of the top edge (in the image) of the part. The projection of this point is called the *approach point* (A). The robot is driven in the *approach plane* so as to minimise the distance between these two points (M and A).

There is a continuous 1-to-1 mapping from the image plane to the approach plane, assuming the camera optical axis is not lying in the *approach plane*. This means that in practice we can work with the image data only. The robot hand is driven vertically a known distance to lie in the *approach plane* (`hand-move-down-half`). The resulting pixel displacement of the hand allows us to deduce the *approach point* in the image. Successive estimates are made of how the robot should move which are refined based on the actual movements (again in the image plane), until the error

is reduced to an acceptable level (`hand-move-above`). Finally the hand is rotated so that it is parallel to the edge (`hand-rotate-to-parallel`) and then it can move down to the final position (`hand-move-down-close`).

The analysis of this strategy is discussed in the next two sections. The first part treats the method of visual servoing the hand and the second part analyses the perspective distortion involved in projecting the image into the camera.

4.4.2 Visual servoing

Consider the mapping of the motion in a plane in the robot coordinate frame to the image plane.

$$\mathbf{v} = \mathbf{T}\,\mathbf{m} \tag{4.1}$$

where $\mathbf{m}$ is a vector PQ joining the point P to the point Q in the robot coordinate frame which lies on the approach plane. $\mathbf{v}$ is the corresponding vector in the image plane formed by projecting P, Q into p, q. $\mathbf{T}$ is the transformation of the points in the approach plane into the image plane.

For the purpose of this vision-guided grasping the robot hand has two translational and one rotational degrees of freedom. The translational components are considered first. The transformation $\mathbf{T}$ is estimated iteratively. The first estimate of $\mathbf{T}$, $\mathbf{T}_1$, comes from observing the vector between the fingertips, with *a priori* knowledge about their position in the robot coordinate frame. In general, once an estimate of $\mathbf{T}_i$ and the image vector $\mathbf{v}_i$ are known, we can estimate $\mathbf{m}_i$ (the required motion) by:

$$\mathbf{m}_i = \mathbf{T}_i^{-1}\,\mathbf{v}_i \tag{4.2}$$

The robot is then moved by $\mathbf{m}_i$. The observed movement is $\mathbf{v}'_i$

$$\mathbf{v}'_i = \mathbf{T}\,\mathbf{m}_i \tag{4.3}$$

and

$$\mathbf{v}_i - \mathbf{v}'_i = (\mathbf{T} - \mathbf{T}_i)\mathbf{m}_i \tag{4.4}$$

From this a new estimate of $\mathbf{T}$ can be calculated. The next move $\mathbf{v}_{i+1}$ is given by

$$\mathbf{v}_{i+1} = \mathbf{v}_i - \mathbf{v}'_i \tag{4.5}$$

The iteration is stopped when the hand is close to the goal, i.e., when the term $\mathbf{v}_i - \mathbf{v}_i'$ is smaller than a preset value.

For the rotational degree of freedom, the robot hand is moved by small steps θ in the direction that will reduce the difference between the orientation of the gripper and the top edge of the part (these two lines are observed in the image). This is repeated until the gripper is parallel to the top edge. Although this parallelism is true only in an affine transformation (parallel lines in the world project to parallel image lines) and the perspective projection distorts this in the image, it is a sufficient approximation for matching the rotation of the gripper and the part. The result is within the tolerance required to successfully grasp a Soma part (see section 4.7 p. 55 for the test result).

4.4.3 Analysis of the perspective distortion

The visual servoing method described in the previous section assumes that the length $PA = MN$ (the approach point is found by moving the hand downward by half of the vertical distance from the top of the object thus making the line PA parallel to MN and of equal length). This is true only if it is orthographic projection. But because of the perspective distortion, there will be a discrepancy QA, (figure 4–11) which will create an error AA' in determining the approach point. We will derive QA in order to ensure that for a particular setup, this error can be made small enough so that the robot hand can reach the part within its range of grasping.

QA is determined as follows: d_1, d_2 are measured along the optical axis of the camera and f is the focal length of the camera.

From the triangle OMN,

$$\frac{ON}{\sin\beta_1} = \frac{MN}{\sin\gamma} \tag{4.6}$$

$$\frac{OC}{\sin\alpha_1} = \frac{BC}{\sin\gamma} \tag{4.7}$$

$$BC = h\,\frac{OC}{ON}\,\frac{\sin\beta_1}{\sin\alpha_1} \tag{4.8}$$

and similarly the triangle OPQ, where $BC = DE$

$$DE = PQ\,\frac{OE}{OQ}\,\frac{\sin\beta_2}{\sin\alpha_2} \tag{4.9}$$

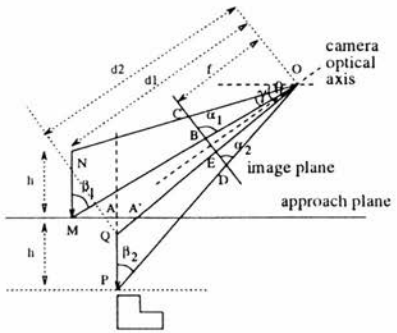

Figure 4–11: Determination of the perspective distortion

from $\frac{OC}{ON} = \frac{f}{d_1}$; $\frac{OE}{OQ} = \frac{f}{d_2}$

$$PQ = h \frac{d_2}{d_1} \frac{\sin\beta_1}{\sin\beta_2} \frac{\sin\alpha_2}{\sin\alpha_1} \tag{4.10}$$

$$QA = h - PQ \tag{4.11}$$

$$QA = h \left(1 - \frac{d_2}{d_1} \frac{\sin\beta_1}{\sin\beta_2} \frac{\sin\alpha_2}{\sin\alpha_1}\right) \tag{4.12}$$

if we align the optical axis along OQ and θ is the elevation angle

$$AA' = \frac{QA}{\tan\theta} \tag{4.13}$$

The camera position can be arranged such that QA is small. The above equation suggested that to do so h should be small. We found that, with the following ranges of parameters; the distance between the camera and the part 50-150 cm, the angle of the camera 25-50 degrees from horizontal, and setting $h = 40$ mm, AA' is within the bound 1.6 mm. A sample of the plot of errors as function of the angle and the distance is shown in figure 4–12. This bound is satisfactory because the gripper of the AdeptOne robot can tolerate the error of 5 mm in grasping a Soma part. To reduce h further will affect the accuracy of determining the vector in the image. This does not mean that the exact position of the camera must be known. The permissible range is large enough such that the camera can be positioned without any measurement tools.

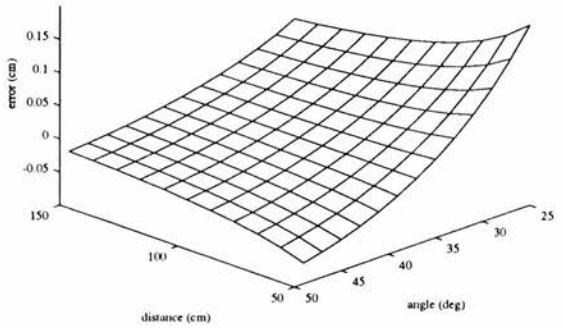

Figure 4 12: Errors as function of the camera angle and the distance. The part is at 3 cm from its nominal position.

4.5 Second experiment: tracking the robot hand

The manual positioning of the camera in the first experiment is replaced by automatic positioning using tracking. To establish the correct preconditions for the `hand-approach` behavioural module a second robot (the RTX) is used to hold the camera and to aim it correctly for each part to be picked up. This is done by tracking the hand. The method of tracking does not require any knowledge about the camera position. It starts with seeing the hand move, and follows it until it goes to the ready position to pick up a part. When the hand is there, the camera is already aiming at about the right place. The camera motion stops, the pick up behaviour carries on until it is finished, the part is placed in the assembly then the camera follows the hand to the next acquisition.

The tracking system is composed of two behaviours, one for following the moving hand (`head-follow`), and the second for locating the hand (`head-look-down`).

The `head-follow` behavioural module does not assume any knowledge of the shape of the hand. It is based on finding the difference between two successive images. The moving camera is instructed to centre its view on the part of the image that has changed. This in itself does not give accurately the location of the hand in

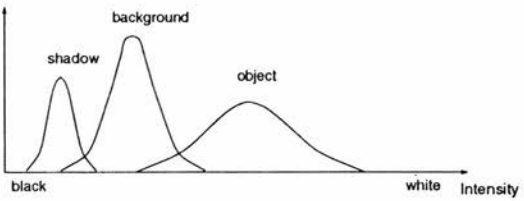

Figure 4–13: Components of the histogram

motion and cannot be used when the hand does not move. In conjunction with this behaviour, a second behaviour was used to locate the hand.

The behavioural module to locate the hand looks for a special mark on the robot hand (`find-hand-marker`). This mark is brightly coloured. A simple spectrum classification scheme based on intensity values of {Red,Green,Blue} is used to separate the mark from the background and calculate its centroid. Finally, the camera is view-centred on this point (`head-look-down`). The two behavioural modules are simple and complement each other well, and are sufficient to establish the proper preconditions for the `hand-approach` behavioural module.

4.6 Vision processing

The information needed from the image is the boundary of the Soma part, and the pixel coordinates of the fingertips.

In order to find in the first instance where the fingertips are, the difference of two pictures is calculated with only the fingers in different positions. Thus the initial finger positions can be located and two small windows enclosing the tips selected. Each fingertip is tracked throughout the hand motion by moving the windows to keep the finger centroids in the centres of the windows. The fingertips are painted white, so they are the brightest objects in the scene. This makes their identification more robust against the background.

The background is the easiest and most invariant thing to identify in the image. Use is made of knowledge about the image histogram (figure 4–13), in particular that

it is composed of three distributions; from the part, which has low peak and large variance, from the background which has high peak and small variance and from the shadow which is darker than the background. The histogram is first smoothed. To separate the background the maximum peak, which is the mean of the background distribution, is found. The variance of this distribution is analysed and the threshold selected. This algorithm works well even if shadows are present.

The system doesn't recognise the part in the usual sense. It follows the hand, using it as a pointer. A binary representation of the area of the image below the hand is made using thresholding as described above. The outline of the part is traced and a polygonal approximation of that shape derived (Malcolm, 1983). Then the most horizontal top edge is selected. It is the least obscured against the background and other objects, and the most reliable feature to look for. Since the assembly rules present the parts with a topmost 'handle' cube for gripping (figure 4-10), the actual grasping position is a simple offset from this.

4.7 Test results

We tested the complete assembly system for reliability by running the Soma assembly 2 times for each of 6 plans, a total of 12 runs (this amounted to about 1000 lines of the top-level plan). The parts were placed by disturbing them from their nominal positions randomly within a translational variation of 60 mm (twice the cube size of the Soma parts we used), and an orientation variation of ± 15 degrees. All runs were successful without a single failure. Although this number of runs is not large, it is sufficient to test the integration of the new behavioural modules into SOMASS. The SOMASS system itself has already been extensively tested (section 1.1.4 p. 45).

We performed a separate test to determine the accuracy of the part acquisition module (`reach`), which includes `hand-approach` and `head-follow-hand-to`. The setup parameters are as follows; the distance of the camera to the part is 540 to 600 mm, the height of camera above the part is 330 mm, the pitch angle of the camera is 25 to 35 degrees, the yaw angle is -10 to 20 degrees, the hand travelled downward 40 mm, the part varied in position ± 30 mm from its nominal position. The parts are placed by the robot at the positions calculated from their nominal

actual positions			reaching positions			errors		
x mm	y mm	r deg	x mm	y mm	r deg	dx mm	dy mm	dr deg
723.46	9.97	6.60	725.67	9.97	4.86	-2.21	0.00	1.74
698.16	88.28	-9.87	699.36	88.68	-9.03	-1.20	-0.40	-0.84
727.67	206.59	8.49	727.03	205.45	5.99	0.64	1.14	2.50
693.64	322.40	5.18	692.71	324.27	2.01	0.93	-1.87	3.17
712.88	-120.68	-13.23	711.50	-117.52	-12.91	1.38	-3.16	-0.32
610.35	418.27	-1.00	612.11	414.98	0.00	-1.76	3.29	-1.00
620.87	-219.62	17.25	622.57	-219.16	11.99	-1.70	0.46	5.26
688.54	-13.50	-1.64	689.65	-11.34	0.00	-1.11	-2.16	-1.64
706.69	99.58	14.79	708.83	99.02	16.03	-2.14	0.56	-1.24
731.14	193.09	14.06	731.83	193.06	9.98	-0.69	0.03	4.08
686.26	289.65	2.62	686.74	289.94	1.95	-0.48	-0.29	0.67
699.23	-138.43	18.36	696.32	-137.17	15.00	2.91	-1.26	3.36
632.28	407.38	1.93	634.12	409.01	0.00	-1.84	-1.63	1.93
613.74	-235.42	-4.37	613.92	-233.34	-5.03	-0.18	-2.08	0.66

Figure 4–14: Data from the accuracy test

positions plus random disturbances. We performed the part acquisition 14 times (7 different parts on 14 different places) and recorded the actual positions of the parts and the positions that the hand arrived at before picking up the parts. These positions are in robot coordinates, reading directly from the robot controller (figure 4–14). The errors are calculated to be $\sqrt{dx^2 + dy^2}$ and dr. The mean error and the standard deviation in reaching the position and orientation of the part are 2.10 mm, 0.91 mm, 2.02 degrees and 1.41 degrees respectively. Under these conditions, it took 3 to 5 moves on average for the hand to reach an *approach point* with the required accuracy of 2 mm.

The system will fail, as would be expected, if certain conditions are not met. If the contrast is sufficiently low the thresholding will fail. Tracking the fingers will fail if they appear too small in the image, for example if a wide angle lens is used. The angle of the camera with respect to the plane of the table is not critical, but near the horizontal the strategy employed gives less accurate results and near the vertical (top view) the fingers of the hand are more likely to be obscured. It was possible to arrange the environment so that all these failure modes were avoided without difficulty.

4.8 Discussion

Integrating the new behavioural module into the SOMASS system is simple. The new behavioural module (`reach`) replaces the old module (`zpatget`). The lines in the plan that call `zpatget` are replaced by a call to `reach`, for example:

```
CALL zpatget(b4.get, RZ(90), -1.5,0.5, RZ(0), -1.5,0.5)
```

is replaced by

```
CALL reach(b4.get)
```

No other change is necessary. This in fact simplifies the planning process because the number of parameters in `reach` is less than those in `zpatget`.

Visual sensing is integrated into SOMASS seamlessly. The new module does not alter any assumptions of any of the other modules of the system. Two additional pieces of knowledge are introduced into the system: the height of the hand above the part, and the meeting point of the hand and the camera. The height of the hand above the part is determined by h which is the amount that the hand travels downward to the *approach plane* (figure 4–11). The meeting point is the place where the camera can detect the hand initially before the tracking behaviour is started, this point is a predetermined position and is represented in the two robots (one is the AdeptOne robot, and the other is the RTX robot that holds the camera) in their respective reference frames.

We will compare the operations of `zpatget` with `reach`. `zpatget` requires more spacing between the parts in the layout (because of the sweeping motions) than `reach`, and also `zpatget` needs to move the hand to pick up the brush. Therefore the total distance of the hand movement in `zpatget` is more than in `reach`. It seems that `reach` should be faster than `zpatget`. But in an actual run the time taken by both methods is comparable because `reach` moves the hand slower than `zpatget` (the speed that `zpatget` moves the hand is 50 upto 100 mm/s and that of `reach` is 10 mm/s). This speed limitation occurs because of the processing time limits of the vision system and the RTX that holds the camera. Improving the hardware could

speed this up a lot, whereas the speed limits on `zpatget` are necessary to preserve the assumption of quasi-static movements, e.g. no part is bouncing etc.

The range of variation allowed in the orientation of parts in `zpatget` is ±40 degrees, but in `reach` it is limited to ±15 degrees. The reason for this is the ambiguity in the vision system in choosing the edge of the part. Consider the following scenario: when the camera faces the part squarely there is an ambiguity arising when the part is turned more than 45 degrees from the nominal position (figure 4–15), that is, one cannot tell whether the edge to the left or to the right of the top corner is corresponded to the top edge. In a normal case, where the part rotates much less than 45 degrees from its nominal position, this decision will be simply to choose the edge that is oriented more horizontally. Also in tracking the hand, the camera might be panned (in the worst case in our experimental set up) ±20 degrees, and this angle must be taken into account when considering the worst case. Therefore to avoid this ambiguity a part must not be turned more than ±25 degrees. We limit this to ±15 degrees in the experiments to be safe. This ±15 degrees limit could be substantially improved if knowledge about the angle that the camera has panned were incorporated into the module that makes this decision (`find-part`).

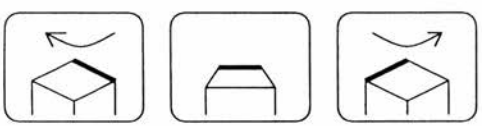

Figure 4–15: The ambiguity in choosing the edge

The `reach` behavioural module is designed using the principles discussed in the previous chapter. It doesn't use any central world model, nor common coordinates in communicating between modules. The tight coupling of sensing and action in following the hand to the part (`head-follow-hand-to`) simplifies the perceptual process (no recognition is required). The strategy for reducing variations in the part-acquiring task is encapsulated inside the module. This allows `reach` to be integrated into SOMASS almost without change to other parts of the system.

Chapter 5

Stereo visual sensing

Good judgement comes from experience,
and experience comes from bad judgement.

Fred Brooks[1]

The result of the experiments in the previous chapter is encouraging. The new part-acquiring behavioural module (`reach`) performs reliably and is integrated well into a complete robotic assembly system. In this chapter we explore a method to improve that module.

The method of visual sensing in `reach` has a limitation in that it can only control a robot hand in a 2 dimensional plane. This is because it uses only one camera and the information obtained is essentially 2D. By using two cameras we can have enough information to control a robot hand to move in 3 dimensions.

We describe experiments that use this idea to guide a robot hand to a visible target. The camera positions are known only approximately. The system does not use the details of the kinematics of the manipulator. There is no common frame of reference linking the vision system, the workspace and the robot hand. The stereo vision system gives information only in terms of image coordinates. This information is used to control a three degree of freedom robot hand in terms of lines and points visible in the images. We discuss the implications that arise from the results of these experiments.

[1] In Bentley (1988).

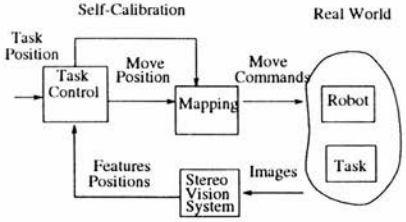

Figure 5–1: Control scheme

5.1 Introduction

A traditional robot system works in its reference frame (the so called 'base coordinate frame'). A model of the kinematics of the manipulator is used to translate between the Cartesian frame and the measured joint space (Paul, 1981). The measurements come from the information from the encoders in the joints of the manipulator. Some experimental systems do use external measurement to determine the position of the robot hand, for example the system by Inoue and Inaba (1985). (Sanderson and Weiss, 1983; Weiss, 1984; Weiss and others, 1987) proposed the visual servoing of a manipulator system based on position of image data. Wolfe and Richards (1990) also used image information directly to control 2 degrees of freedom of a robot hand in a pick-and-place task.

Our system measures positional displacements in the picture coordinates (2D pixel coordinates) of the object relative to the target. We use two cameras to derive the information to control 3 degrees of freedom of a robot. Because we measure the relative displacements and map them directly into the joint motion commands, we do not need to know the camera positions or optical parameters. The measurement is done in terms of the difference between the hand and the target in the images. Very little prior knowledge is required. The accuracy of the system does not depend on knowing the kinematics of the robot.

5.2 Stereo geometry

The cameras are set up so that the field of view of both cameras covers the area of interest. A point in the field of view of both cameras, **P**, can be represented as (x_L, y_L) and (x_R, y_R) in the left and right images respectively. Both y axes are assumed to point upwards, and the two optical axes are assumed not to be collinear. The elevation angles of the two cameras are arbitrary, but approximately equal. In an arbitrary but fixed global Cartesian space, **P** can be represented uniquely as $\mathbf{x} = (x, y, z)$. Each point **P** can also be represented uniquely by a vector in the space of points $\mathbf{p} = (x_L, x_R, y_L)$. The transformation between Cartesian coordinates and image coordinates, $\mathbf{J}_v$ is a function of the geometry of the cameras and lenses. Approximate values can be calculated by considering pinhole cameras in a specific configuration (Pollard and others, 1989). Consequently,

$$\mathbf{p} = \mathbf{J}_v\mathbf{x} \tag{5.1}$$

and

$$\mathbf{x} = \mathbf{J}_v^{-1}\mathbf{p} \tag{5.2}$$

We consider only 3 translation degrees of freedom for the robot. There is a transformation from the robot joint space represented by the set of points $\mathbf{m} = (m_1, m_2, m_3)$ and Cartesian space, as follows:

$$\mathbf{x} = \mathbf{J}_r\mathbf{m} \tag{5.3}$$

This matrix can be shown to be invertible for a 3 degree of freedom robot when it is restricted to cover a connected region with no singularities, and a function of $\mathbf{x}$ and $\mathbf{m}$. So

$$\mathbf{p} = \mathbf{J}_v\mathbf{J}_r\mathbf{m} = \mathbf{J}\mathbf{m} \tag{5.4}$$

and since $\mathbf{J}$ is invertible if we restrict the domain of interest to a connected region of space where both $\mathbf{J}_v$ and $\mathbf{J}_r$ are invertible (distant from any singularities of the robot):

$$\mathbf{m} = \mathbf{J}^{-1}\mathbf{p} \tag{5.5}$$

That is, given the uniqueness of the mapping we can attempt to control the robot using image data directly for the feedback and reference signals by observing the vector from the robot hand to the target in the images.

For control purposes, we found that **J** could be approximated sufficiently by measuring the result of a sequence of independent small motions of each of the motors m_1, m_2, m_3 and noting the resultant changes in **p**.

$$\mathbf{J} \doteq \begin{bmatrix} \frac{\Delta x_L}{\Delta m_1} & \frac{\Delta x_L}{\Delta m_2} & \frac{\Delta x_L}{\Delta m_3} \\ \frac{\Delta x_R}{\Delta m_1} & \frac{\Delta x_R}{\Delta m_2} & \frac{\Delta x_R}{\Delta m_3} \\ \frac{\Delta y_L}{\Delta m_1} & \frac{\Delta y_L}{\Delta m_2} & \frac{\Delta y_L}{\Delta m_3} \end{bmatrix} \tag{5.6}$$

This worked sufficiently well to allow us to use only an initial sequence of three movements to serve in calculating a value of **J** used in subsequent control. Clearly on a global scale this approximation is unjustified, but it worked over the control area of interest. With the equation 5.6[2] and with the target point and the robot end effector in the field of view, the measured parameters are the vectors from the robot hand to the target. This equation was subsequently derived by Domingo (1991) using tensor network theory as a part of his project. His work which used a similar technique to acquire a part will be discussed in section 5.7 p. 68.

5.3 The experiments

We used an RTX robot arm from Universal Machine Intelligence Ltd. The first camera was a Sony DXC-101P with a 16 mm lens; the second was a Panasonic F10 with a 10.5mm-84mm zoom lens (both have CCD sensors). We adjusted the focal length so that the scale of the images from the two cameras did not differ by more than about a third, which could be done easily by hand.

5.3.1 Controlling the robot hand

Conkie (Conkie and Chongstitvatana, 1990) demonstrated the validity of the application of the equation 5.6 by a clear and simple experiment. His setup will be

[2]It was formulated by Conkie (Conkie and Chongstitvatana, 1990).

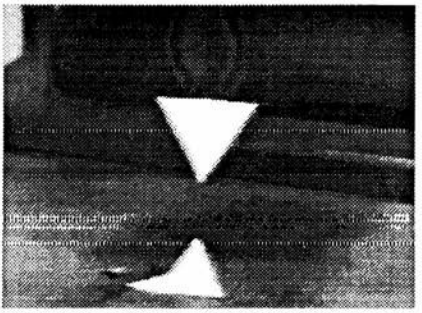

Figure 5–2: Aligning two triangles

explained as follows: the targets for the stereo vision system to track were simple, two white triangles. One was fixed to the robot hand the tip pointing downward, the other lay on the surface table in the working region with the tip pointing upward (figure 5–2).

The tips of two triangles were detected in both cameras. The goal was to align them in 3D space. The matrix of equation (5.6) was constructed as described above and then the motion strategy adopted in this case was to move the hand so as to align the two triangles vertically and then move downward half the calculated goal distance. The motion was decoupled into two separate error reducing processes. The goal position was achieved by repeating these processes several times.

The variation of the target position is in the area 10×10 cm and the height of the hand is 20 cm. It was found that the target was reached within 1 2 mm in every one of a dozen runs, with approximately eight iterations.

5.3.2 Stacking blocks

We applied the scheme above to a task of picking up and stacking three blocks (figure 5 3). For this task, beside the translational motion in 3 dimensional space the hand must rotate about the vertical axis in order to align the fingers with the grasping surface of a cube. The visual features that we used for control were the fingertips of the manipulator, which are painted white, and the centroid of the blocks. The

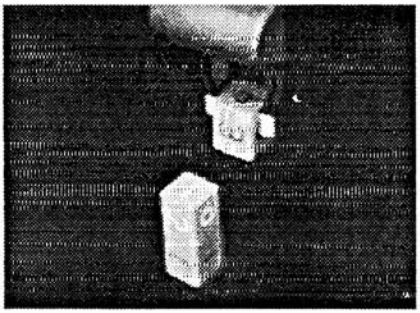

Figure 5–3: Stacking three blocks

hand position was taken as the point midway between the two fingertips with the hand empty (not holding a block) and the centroid of the silhouette of the block when there was the block in the hand.

Three blocks were laid in a line without touching each other. Their individual orientations were arbitrary and they were allowed to be anywhere inside the region visible to both cameras.

The strategy for picking up a block is similar to that of guiding the hand to a target in section 5.3.1 p. 62. The matrix (equation 5.6) is constructed by moving the robot hand by three small motions in independent directions and observing the resultant changes in the images (`calibrate`). The motion strategy is to align the hand vertically with the block and then approach using a downward motion (`hand-move-above-stereo` and `hand-move-down-half-stereo`). The gripper is then oriented with the block's faces (`hand-rotate-to-parallel`). This is done using the edge of the block and uses the view from only one camera; the task has only one degree of freedom therefore one camera is sufficient (this behavioural module is similar to the one used in the experiment in chapter 4). The last move is downward, to pick up the block without using visual feedback (`hand-move-down`). We didn't use visual feedback because when the hand is close to the block there will be an occlusion between the fingertips and the block and the vision system will lose track of the fingertips. Our present vision processing routines cannot cope with this occlusion as these routines are simple binary vision algorithms. Also, the last move can be reliably accomplished by 'blind' motion because the hand is quite close to

the block at this stage and the vertical distance between the hand and the block can be sufficiently approximated by using the 'calibration matrix' (equation 5.6) that was constructed earlier.

To stack blocks, we place them in a known place with the same orientation (`hand-move-to` and `put-on`). This is done using predefined heights to offset the stacking positions. We tried not to use predefined height in the earlier experiments, by using visual servoing instead. In order to do that, we tried to find a good set of image features to align the block in the hand with the block on the table. The features that we considered are: centroid, edges, corners. For example, in orienting the block in the hand, we used the bottom edge of the block in the hand and the top edge of the block on the table. For vertical alignment, we used visible vertical edges of both blocks. For mating blocks, the gap between them must be observed somehow, but when the blocks are getting close to each other this gap cannot be seen. This is because the camera view is looking down about 20 degrees (from horizontal), which is suitable for the pick-up task but renders the gap invisible when the blocks are close. We tried to use the distance between centroids but the centroid of the block in the hand is not sufficiently accurate because the block is partly obscured by the hand (figure 5.3). We also tried to approximate the vertical distance by using the 'calibration matrix' and moved 'blind'. The accuracy is not sufficient for this part mating task (overshoot is destructive in this situation). In the end, we had to resort to using a predefined height. This lack of accuracy, however, is not a failure in principle of the general strategy; it could be remedied by more sophisticated vision algorithms, and moving the camera for better views (as we will show in chapter 6).

5.4 Vision processing

The basic vision processing is similar to that described in chapter 4. Similar methods are used to locate the fingertips and to get the boundary and corner points of the blocks.[3] The difference is that here we need 2D information from both cameras,

[3]These visual routines are fully explained in chapter 8.

therefore the vision processing load is double. Several additional task specific constraints are used to locate the features of the blocks, as in finding vertical edges and finding the bottom edge of the block in hand. From the corner points, the edge lists are constructed and the edges are examined. The vertical edges are defined as the edges that lie within ± 15 degrees from the vertical axis (in image coordinates). Any small 'noise' edges can be rejected by examining their length. For vertical alignment of two blocks, the selected vertical edges are the two on the extreme left and right of the blocks. Also, we know that the block in the hand is higher (in image coordinates) than the block on the table, therefore the vision system can distinguish between the two blocks. Spatial information is also used in matching the blocks in the left and right cameras. By using the left-of, right-of relations amongst the blocks, we can match which blocks in the left image correspond to which blocks in the right image. We are not trying to solve a general vision problem, therefore these constraints, despite being specific to this task, are appropriate to our purposes in this experiment.

To guide the hand to the target, an error vector is measured. The error vector is defined as the vector in the image from a reference point in the robot hand to a point on the target. We used the middle point between two fingertips as the reference point of the hand and the centroid of the block on the table as the target point. To measure the error vector, both the hand and the target must be clearly seen. We get this information by tracking the selected features of the robot hand and the blocks. The visual routines run continuously at the rate of 3 Hz (using both cameras) while the hand is moving. In the situation where only one camera is used, the processing rate goes up to 6 Hz.[4]

[4]We performed a better 'low level' speed tuning for visual routines in the next experiment described in the next chapter. See the result in section 6.9 p. 79 and the tuning methods in section 8.7 p. 110.

5.5 Test results

The setup for the block stacking task is as follows: the blocks are toy wooden blocks of size 4×4×4 cm, the working area (representing the uncertainty in the position of blocks) is 25×18 cm, the two cameras have a base line of 60 cm, are 70 cm from the middle of the working area, look down 20 degrees (measured from horizontal), and are verged to the middle of working area. We performed 10 runs of the task to confirm its reliability and measured the accuracy of the hand in reaching the block which was 1 2 mm in every run. The orientation accuracy was within 5 degrees. This is measured from the 'ideal' position for grasping a block. The number of iterations of the hand movement to reach a specified goal was 3-4 moves.

We did not report the measurement in terms of world coordinates because the RTX that we used was controlled in terms of joint angles, and the best kinematics available for the RTX contains errors larger than the 1 2 mm we are concerned with here; in addition, backlash and other mechanical sources of error are larger than this, making more accurate kinematics pointless. We performed the measurement by hand, measuring the distance between the objects and the robot hand in the work cell, so they are only approximate, unlike the result reported in the previous chapter in which we used the Adept robot, and reading the world coordinates of the position of hand directly from the robot controller (the Adept that we used is very precisely calibrated).

5.6 Discussion

The principles proposed in chapter 3 are used in the design and implementation of behavioural modules for the experiments described in this chapter. The module to pick up a block (`pick`) arises from an attempt to extend the technique of the module `hand-approach` in the previous chapter to deal with 3D. The use of stereo visual servoing was captured inside the modules: `calibrate`, `hand-move-above-stereo` and `hand-move-down-half-stereo`. The sensing-action strategy for a hand to reach a target is similar to the idea used in the previous chapter. The module

`hand-rotate-to-parallel` is exactly the same as the one used in the previous chapter (section 4.4.1 p. 50). In fact, the module `stack-blocks`, which is the top-level module for this task, can be regarded as 'a plan' that is composed of the sequence: `pick`, `hand-move-to` and `put-on` (compare it with the plan in figure 4-5). It is also interesting to compare `pick` and `hand-approach`. `pick` uses less a priori knowledge than `hand-approach` in that it doesn't need to know the height of the hand above the part, and the height of the part. Mating the parts is still a problem without this piece of knowledge. We will describe a solution to this in the next chapter. The detailed analysis of these behavioural modules is given in chapter 7.

The result shows that the *accuracy* achieved by our technique in guiding the RTX robot to reach a target is better than that of using the inverse kinematics of the RTX. The RTX has position *repeatability* 0.5 mm at best and from our measurements typically it has *accuracy* with in 2 mm with small rotations as in these experiments; with large rotations the *inaccuracy* can be as much as 10 mm. The *accuracy* depends on initial arm measurements, calculations, and calibration errors on an individual robot. In real use, it is worse than that because the RTX has low stiffness. For instance, a force of 100 gm will move the hand by 1 mm in the horizontal direction or 0.5 mm vertically. We generally expect *accuracy* no better than within 3-4 mm from the RTX that we use.

Alistair Conkie verified this method's independence of robot kinematics by re-implementing `pick` on the AdeptOne robot. Similar results were achieved, despite the different kinematics between the RTX and the AdeptOne robots.

5.7 Multiple matrices

One interesting piece of work has come out of the technique used in this chapter. During a discussion with Brendan McGonigle,[5] we were informed about the possibility of the 'memory' of this 'calibration matrix' (equation 5.6) in animals. The work

[5]Department of Psychology, University of Edinburgh.

of Flook and McGonigle (1977) shows that monkeys and humans can learn and re tain the adaptation of hand eye co ordination to sensory distortion conditions. Their experiments[6] demonstrate the ability of their subjects to retain *multiple* adaptations.

It is interesting that these multiple adaptations can both be remembered and applied in the immediate situation. The subject learned to adapt to one kind of visual displacement distortion, and then learned a second adaptation with a different displacing prism. These adaptations did not interfere with each other. Humans have been shown to be able to recall the adaptation immediately once having recognised the apparatus. This can be due to some clue about the prisms: the converging lines near the rim of the glass, the reflection on glass edges (rainbow), etc.

Based on this observation, Domingo (1991) employed the technique of the stereo visual servoing presented in this chapter and extended it to work over a wider area. He calibrated and stored a number of calibration matrices over the working area and used grid co ordinates to index them. The interpolation of these matrices then is used to guide a hand to any location in that area. To determine how many matrices are required for a given task, some details of this system will be discussed. In order to have calibration matrices vary smoothly over a working area, a constraint was imposed, such that any element of the matrix must not change more than 10% from one point to an adjacent one. Using an RTX robot and a working volume of $25\times50\times35$ cm this constraint can be satisfied by setting the grid size to 5×5 cm. Therefore there are $6\times11\times8 = 528$ matrices.

These multiple matrices are rather like an implementation of the mapping be tween sensor state space and motor state space which Churchland (1986) called a *state space sandwich* system. In Domingo's system, the robot hand is moved, by approximating its inverse kinematics, to a grid point near the target. The calibra tion matrices in that neighbourhood are used to interpolate the required motion to reach the target. The visual servoing is a function that maps the sensed target point into motor space. This is a proposed solution for the problem of sensorimotor coordination in the neurosciences (Churchland, 1986).

---

[6]The subjects look through the window, in which laterally displacing prisms are in serted, locate a target and reach for it through the slit aperture located below the window.

Chapter 6

Active mobile vision

In the last chapter, we demonstrated a visual servoing technique that is capable of controlling a robot hand in three dimensions. But because the cameras are stationary, their assembly task applications are limited. In the previous experiment, we could not find a set of features to be used to perform part-mating operations reliably. This is because the cameras' viewing angle is not suitable.

This chapter describes a mobile stereo vision system that is actively controlled to guide a robotic assembly task. The system is uncalibrated, and we apply principles similar to those used in the vision system of the previous chapter; the camera positions are known only approximately, the vision system does not rely on precise modelling of the parts or of the cameras, and there is no need to use a common frame of reference linking the vision system, the workspace and the robot hand.

A block stacking task is demonstrated. A robot hand moves to pick up randomly placed blocks and stacks them. A stereo camera head is moved to find good views and is used in feedback control of the robot hand. The visual feedback control uses signals which are measured in terms of the feature space in both cameras. The task was performed by a 4 degree of freedom (DOF) robot hand and a 5 DOF mobile stereo camera head.

6.1 Introduction

The method of visual servoing assumes that the cameras have clear views of the relevant area all the time while the task is in progress. The ability to move the camera head provides the flexibility to use visual servoing for various tasks. It enables the choice of views that are best suited to the task for different situations. For example, a close view is good for fine motions of the hand, as occurs in mating the parts, a more distant view is better for covering a large area, to facilitate locating the objects, a view from above is needed for observing the orientation of objects about their vertical axis, and a horizontal view is more appropriate for observing the vertical alignment of two objects.

This chapter is a logical development of our previous work on uncalibrated vision. In chapter 4, we presented a single camera system that performs a part acquiring task. Chapter 5 presents a work that generalises one camera to two cameras but with a fixed viewpoint. This chapter presents a mobile two camera system with multiple possible viewpoints that are not predefined. A mobile stereo camera is used to find good views such that suitable features for visual servoing can be detected and tracked reliably. The experiment emphasises the strategy for using vision for manipulations that require a close up view, for example fine motions of a robot hand and part mating tasks. This is different from other work (Heikkila and others, 1988; Sakane and others, 1987) which use preselected alternative viewpoints. In this work, the movement of the camera pair is determined by continuously tracking the change of parameters of the selected features. In Zheng and others (1991) a similar idea is presented but the movement of a camera was from a front view to a side view only. In our experiments the movement of a camera pair is used to accomplish many tasks; occlusion free viewing, alignment and part mating.

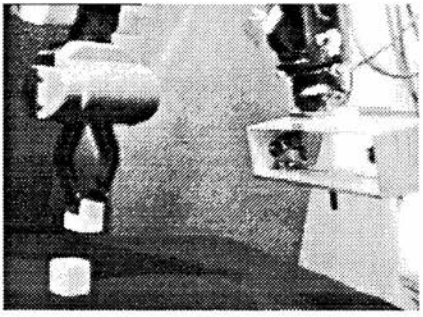

Figure 6–1: A view of the assembly system, the RTX robot (left) manipulates blocks. The Adept robot (right) holds the camera head.

6.2 The equipment setup

The system consists of two robots and a Sun workstation. The vision equipment consists of two cameras and Maxvideo boards: an image digitiser and a framestore. The hardware is driven in interrupt mode. Both robots are controlled via serial lines from the Sun.

An Adept-One industrial robot, with 5 DOF, is used to carry a mobile stereo head. The camera head consists of two small colour CCD cameras, Panasonic WVCD1E, with 1.6 f and focal length 7.5 mm lenses, mounted with a fixed base line of 18 cm and verged to a distance 38 cm ahead. Another robot, an RTX robot, is used to manipulate the parts. The control of the RTX is limited to 4 DOF; x, y, z translations and a rotation around the z-axis, rz. These robots are position controlled via their standard controllers in a look and move fashion. The image processing is all done in the workstation (figure 6–1).

6.3 The task

The task of picking up randomly placed blocks and stacking them is demonstrated. Two blocks lie on the table. We assume that the blocks are at least a few centimeters apart, such that it is possible for the robot hand to grasp one block without colliding with the other, and that their orientations are random.

First, the vision system looks from a distance to get an overall view of the work cell and to locate the blocks (figure 6-2). The camera head is moved to find a view of a block that is occlusion-free (`view-separate`) (this method will be described later). The behavioural module `pick`, described in the last chapter, is used to pick up one block. The choice of the block is arbitrary, for example, the block on the left. The hand moves into the scene, and performs self-calibration (`calibrate`). It picks up the chosen block. The camera head adjusts its view by centering and zooming to get both blocks in its field of view with a reasonable size (`view-closeup`). It does this by moving closer to the block in the hand while monitoring its area, then moves up and looks down at the block at an angle of about 20-30 degrees (`head-vertical`). The hand performs self-calibration for this view and moves to place one block above the other (`hand-move-above-approximate`), then rotates this block until the two blocks are in the same orientation (`hand-with-block-rotate-to-parallel`). The camera head moves down to get a side view that can see the edge view of the mating surfaces (`view-gap`), i.e. the bottom edge of the block in the hand and top edge of the block on the table are almost horizontal. The hand self-calibrates again for this view and moves to align the two blocks vertically, then moves half way down (`hand-move-down`), aligns them again to improve the accuracy (`hand-move-above-precise`), then slowly puts down the block until the two blocks mate using visual feedback (`mate-blocks`).

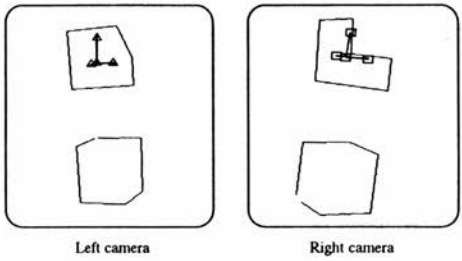

Figure 6–2: The image processing in the system extracts the outlines of the objects. The figure shows typical shapes seen from the left and right camera. The upper shapes are the block in the hand. Also shown are the self-calibration vectors from both cameras.

6.4 The camera motion

The camera head can move in various ways. The camera head motion can be tangential to the line of sight while maintaining the field of view on the objects in the scene. Another motion is to zoom in or out, along the line of sight. In all there are 6 movements : left, right, up, down, zoom in, zoom out (figure 6–3). The motion of the camera head need only be approximate, because we are not interested in knowing the distance from the objects.[1] A camera keeps its fixation point on a target in the image and rotates the head to centre its view on that point when the head is moved, thus describing a some what elliptical path when it moves around the objects.

[1]It is possible to calculate the distance, using the knowledge of the camera head motion and the angular change of the line of sight.

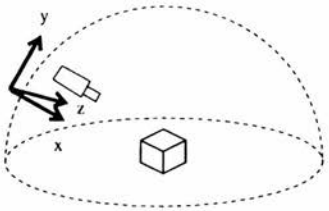

Figure 6–3: The camera motion

6.5 Finding an occlusion-free view

If both cameras register two objects in their fields of view, then there is no occlusion. When an occlusion occurs, the images of two objects merge into one. We use the axis of eccentricity, as defined in Ballard and Brown (1982, p. 255) to guide the move to find an occlusion-free view. The *eccentricity* is a measure of 'elongation' of a shape, it can be calculated from the ratio of principal axes of inertia. Assume we select a camera (left, or right) to guide the head movement. It tries to move left or right so that the axis of eccentricity becomes more horizontal, and detects when the objects separate, e.g. when there are two objects in the image (figure 6–4). The choice of camera depends on which one has the better view initially. Motion is then in the direction of the camera that registers no occlusion (occlusion might not occur in both cameras simultaneously), or registers a more horizontal axis of eccentricity.

6.6 Visual servoing of the hand

To find the displacement of an object from its goal position, we use the shape attractor algorithm (Harris, 1992), in which, given the outlines of two shapes, the algorithm will find the translation, scaling, and rotation vectors that will make the two shapes 'fit' together. This method does not require any heuristic to choose the edges to be matched (figure 6–5). We use the block on the table as the goal to attract the block in the hand. The translation vectors from both cameras are used to control the movement of the hand to put one block above the other. The height

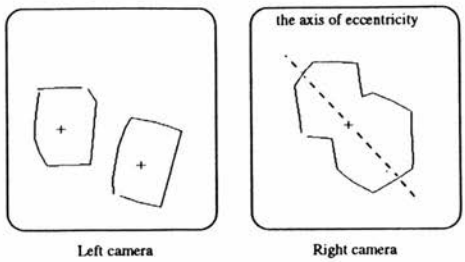

Figure 6–4: A view with occlusion. The left camera registers no occlusion, the right camera registers an occlusion. The axis of eccentricity is used to determine the direction to move the camera head.

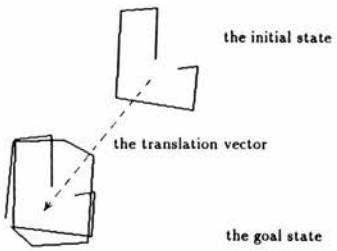

Figure 6–5: The shape attractor algorithm provides the translation vector that will bring the two shapes together.

(vertical movement) is controlled separately. This is because of the fact that the translation vectors calculated are actually for transferring one block into the other rather than above it.

In aligning two blocks, only some edges are used for matching in order to calculate the translation and rotation vectors. To move one block precisely above the other before mating them, vertical edges are chosen. The camera view is from the side such that the vertical edges can be seen clearly. The most prominent edges are selected. In the left camera, the right vertical edges are used, and vice versa for the right camera.

The two bottom edges are used in rotating a block in the hand to align it with

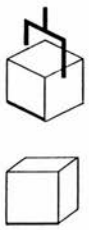

Figure 6–6: Two bottom edges

the block on the table (figure 6–6). Only one camera is needed because the task has only 1 degree of freedom. We choose the camera that can see the view of the bottom edges most clearly. The choice is based on the length of the edge and its slope.

When the camera head changes views, especially when it zooms in and looks down, the image projection becomes subject to perspective distortion (in general, we assume an orthographic projection). The vertical lines will verge to a vanishing point. Therefore, the vertical alignment must be compensated by a factor which is extracted from the self-calibration matrix (z-axis calibration). This factor is used to take account of the tilt of the vertical axis. It is used in the calculation of the translation vector in placing one block above the other block.

6.7 Mating blocks

The goal is to reduce the gap between two blocks. This gap is determined from the side view of both blocks. A small window is assigned to track the moving bottom edge. The small size of the window enables a fast visual feedback loop because it reduces the amount of data to be processed. The hand is stopped when the gap is closed (figure 6–7). The terminating condition is that the gap is less than 5 pixels. With a reasonably close view, a final gap less than 1 mm is achievable.

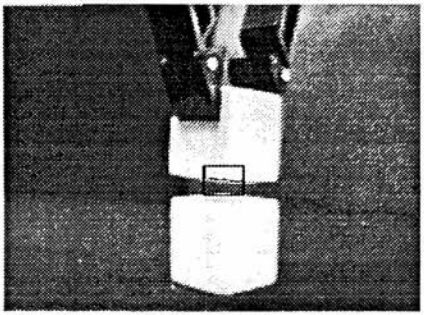

Figure 6–7: Mating blocks by tracking the edges until the gap between two blocks disappears.

6.8 Choosing the features to track

To track an object while a motion occurs (either the camera head motion or the motion of the hand), we use the centroid of the object or the aggregation of them if more than one object is involved. In some circumstances, when we want to focus on a particular object, and if there is more than one object in the window, we choose to track the object that is most similar (in shape) to the previous target. The success relies on the assumption that the object will not change its appearance suddenly.[2] There is one case when this is not true. This is when the motion of the camera head causes the image of two objects to merge into one shape (two outlines merge into one) or when one shape becomes separated into two. We treat this as a special case by choosing to focus on the new shape (if they merge) or vice versa when it separates.

[2]It depends on the ability to track the object fast enough, compared to the change.

6.9 Test results

The average linear speed of the hand and the head is 16 mm/s. The slow speed of the hand in mating blocks is 3 mm/s. The average sampling rate of the visual feedback ranges from 8 Hz, with a window 200×318 pixels, to 12.5 Hz, with a window 40×40 pixels whilst mating blocks (this is measured for one camera). The number of iterations of the hand movement to reach a specified goal is 3-4 moves. The accuracy in stacking the blocks is 1-2 mm. When mating the blocks, the final gap is less than 1 mm before the hand releases the block from its grip. The tolerance is large because of the nature of the mechanics of the RTX robot, but this is better than the accuracy that can be achieved by driving it using inverse kinematic calculations alone.

6.10 Discussion

It has been shown that active mobile vision works well for the task demonstrated. The ability to move the cameras based on the configuration in the workcell can simplify vision tasks because a view that is suitable for a particular situation can be selected, rather than a static and therefore possibly unsuitable view. For example, a view from above is used to align the orientation of two blocks, a side view is used in mating them. Once a suitable view is achieved, the tracking of selected features simplifies the detection of goal states. An example is of tracking appropriate edges to mate the blocks. Further work is necessary to increase the generality of the feature selection.

The integration of visual sensors using an uncalibrated vision system is useful in practice and is achieved by using position control via standard interfaces to commercial industrial robots. The system would certainly benefit both in terms of responsiveness and speed if more sophisticated servoing techniques were to be adopted, but the basic results would remain unchanged. Many parts of the system, given their generality, can be carried over to different assembly tasks, and towards the ultimate aim of being able to specify tasks simply and have them achieved robustly.

The criteria proposed in chapter 3 have been applied in the design of the new behavioural modules used in the experiment described in this chapter. By not relying on a central world model; by employing a tight coupling between sensing and action; and by communicating via the world, we have built a system that performs a fairly complicated task reliably. The experiments show that we can add, extend, improve, and reuse a set of behavioural modules to perform new tasks without difficulty.

Chapter 7

Analysis

Meno: And how will you inquire, Socrates, into that which you know not?
What will you put forth as the subject of inquiry? And if you find
what you want, how will you ever know that this is what you did not know?

Plato

In this chapter we look at the behavioural modules that are used in the experiments. We explain what they do and how they do it, then go on to analyse their structures and their relationships in order to understand why they work. The analysis is along three dimensions: 1) the connections between modules: their assumptions about the world and other modules; 2) the coupling between sensing and action in modules; and 3) the communication between modules.

The analysis follows the order of the experiments: the single camera system of chapter 4, the stereo camera system of chapter 5, and the mobile camera system of chapter 6. Finally, observations about the structure and the size of the programs are given.

7.1 Naming convention

The names of behavioural modules are given in the form `Noun-Verb-Object` and will be unique for each module. `Noun` denotes the agent like `hand`, `head`, `camera`. When there is no ambiguity it is omitted. A long name usually indicates that it is similar to another module but with extra qualifications, for example, `hand-move-above`

and `hand-move-above-stereo`. The appendix lists the complete pseudo code of all the programs used in the experiments and also the hierarchical structure of the behavioural modules. They should be consulted if more details about the programs are needed.

7.2 SOMASS

We start with behavioural modules in the original SOMASS. To perform a pick-and-place of one part, the sequence of calls to behavioural modules might be:

```
zpatget(b1.get, sweeping parameters...)
zget(b1.get)
zmanip(table, putgrasp..., getgrasp...)
zput(b1.put)
```

The parameters that are instantiated at run-time are the predefined locations: b1.get, table, b1.put, etc. Only `zpatget` and `zget` communicate by a parameter b1.get, the other modules don't communicate. `zget` assumes that the part will be in a predefined location. It can cope with small uncertainty by using a double snap strategy. It relies on `zpatget`, which can cope with a wider range of variations in locations and orientations of a part, to put the part in a predefined location. `zmanip` assumes that the part is in the hand with the right orientation. It also relies on the fact that, during a regrasp the part is stable and can be picked up properly. `zput` assumes that the part has the right orientation and the place where it is to be put down is surrounded by sufficiently large gaps to allow for the tolerance in dimensions of the part.

These modules are loosely coupled and the reliability of the whole assembly comes from the ability to arrange for the assumptions of all modules to be true. For example one of these assumptions, that the part is stable during a regrasp in `zmanip`, is the cause of half of the failures that involved instabilities as reported in Malcolm and Smithers (1988). It is because the L-shape part used in their test is

not stable when it is in an upside-down position so it falls over or wobbles out of position during a regrasp.[1]

In single camera experiments, `zpatget` is replaced by `reach`. Because `reach` doesn't break any assumptions of `zget` and because it doesn't have any communication with other modules, its substitution doesn't affect other modules.

7.3 Single camera experiments

Now we will go on to analyse the behavioural module `reach`. `reach` moves the hand to the nominal position of the part, and uses the camera to locate the part, then moves the hand to the part. It updates the actual position of the part so that `zget` can pick it up later. The code for `reach` is shown below:

```
reach( p )
        meeting
        head-follow-hand-to( p )
        edge = find-part
        hand-approach( edge )
```

`meeting` brings the hand and the camera to their initial states. Two predefined locations are needed; they refer to the meeting place: 1) in hand co-ordinates, 2) in camera co-ordinates. `meeting` also confirms the presence of the hand by using `find-hand-marker`.

```
meeting
        hand-move-to( x )
        head-move-to( y )
        find-hand-marker  ;; check to see that camera can see hand

head-follow-hand-to( p )
        loc = find-hand-marker
        while hand-move-to( pos )
                head-follow
        head-look-down
```

[1]There are two simple remedies: 1) tell the system this is unstable; and 2) add restorative patting to `zmanip`.

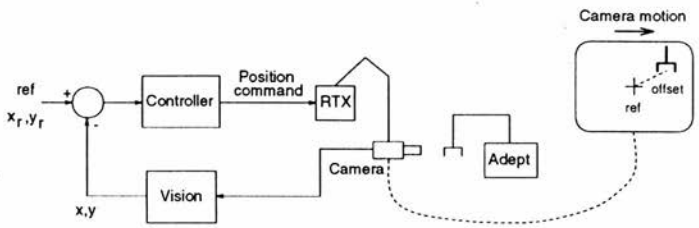

Figure 7–1: The control loop of `head-follow`

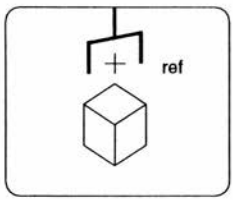

Figure 7–2: Shift the reference point

`head-follow-hand-to` uses `head-follow` to track the hand. The visual routine `find-moving` locates the moving hand and then `head-rotate` pans and tilts the camera to centre the view on the target. The control loop is a simple look-and-move with a proportional controller. The feedback error is the offset of the position of the hand from a reference point (figure 7–1). The control gains are found empirically. The performance is acceptable as long as the sampling rate is adequate.

```
head-follow
        while hand-is-moving
                loc = find-moving
                m = the move to centre view on loc
                head-rotate( m )
```

Once the hand stops, the camera tilts down to aim at the part. `head-look-down` uses `find-hand-marker` to locate the hand and centres the view on the part underneath the hand. `head-look-down` uses a similar feedback loop to `head-follow` but it shifts the reference point (figure 7–2).

```
head-look-down
        loc = offset
        repeat
                x = find-hand-marker
                m = the move to shift view from x to loc
                head-rotate( m )
        until  hand-marker is at loc
```

Before we go further, we will discuss these modules. The precondition of `head-follow` is that the camera has a proper initial view (the hand is in the view), and the postcondition is that the hand is near the part (at the nominal position of the part) with the camera aiming at the part. `head-follow` has an assumption that the head doesn't move too fast for it to track. This depends on the vision process `find-moving` and the motion of `head-rotate`. `find-moving` uses the difference between successive images to detect a moving object. It assumes that there is no other object moving except the hand.

In `find-part`, the camera is already aiming at the part (as a consequence of executing `head-look-down`). `find-part` looks for a part underneath the hand. It finds the hand by using the motion of the fingers: the difference of two images is calculated with only the two fingers in different positions (open and closed). `find-part` returns an edge that is to be used by `hand-approach`. An appropriate view for `find-part` depends on the layout of the workcell and the location of the camera. When the camera tracks the hand and arrives at a part, the camera should have a clear view of that part; the part should be near the hand and not overlap with other parts. The next step is that the hand moves to reach for the part by `hand-approach`.

```
hand-approach( edge )
        z = down half height
        hand-move-track( z )
        a = mid point
        hand-move-above( a )
        hand-rotate-to-parallel( edge )
        z = down close to near point
        hand-move-track( z )
        b = near point
        hand-move-above( b )
```

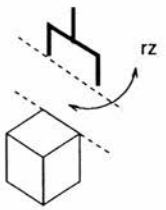

Figure 7–3: The hand rotates to parallel to the edge

The location of the hand is represented by the middle point between the fingertips. `hand-move-track` moves the hand down, which creates an image vector to be used to calculate the *approach point*. `hand-move-above` servos the hand to this point. `hand-rotate-to-parallel` aligns the hand with the edge of the part (figure 7–3) by using `hand-rotate` in small steps until the gripper is parallel to the edge. Next, `hand-move-track` moves the hand down again nearer to the part. Then `hand-move-above` is used again to move the hand to the final place. From the location of the hand, the actual location of the part can be found.

The use of tight coupling between sensing and action appears in these modules. The success of the strategy in `hand-approach` is based on knowing the height of the hand above the part. The motion of the hand is part of the sensing strategy. `hand-move-above` works because it uses the observed motions of the hand and employs a feedback loop to reduce the deviation of the hand from the target. `head-follow` and `head-look-down` use tracking. These modules use their own reference frames, for example the camera uses image coordinates and the command to move the hand is in hand coordinates. They need no common coordinates to operate. The communication between these modules is via the world. There is no need to keep any centralised world representation.

Because the vision system is uncalibrated, it doesn't need to keep precise knowledge about the camera locations, or the lens parameters. It is therefore tolerant to changes of these values.

The behavioural modules in `reach` prefer sensing the world to performing deduction from information given by other modules or by *a priori* knowledge. For example: `head-follow-hand-to` uses `find-hand-marker` to locate the hand; `find-part` lo-

cates the hand by itself using `find-hand`; `hand-move-track` locates the fingertips while tracking them continuously.

7.4 Stereo vision

We will begin with the top level module: `stack-blocks`. At this level, the sequence of operations is simply: locate the parts, pick them up one by one, and stack them at a predefined location.

```
stack-blocks
        ;; place, table, block1, block2 are predefined
        get-three-blocks( b1, b2, b3 )
        pick( b1 )
        hand-move-to( place )
        put-on( table )
        pick( b2 )
        hand-move-to( place )
        put-on( block1 )
        pick( b3 )
        hand-move-to( place )
        put-on( block2 )
```

`get-three-blocks` is a visual routine that finds the locations of the blocks in both cameras (more details about the visual routines in the next chapter). `pick` picks up a block. The sequence of `hand-move-to`, `put-on` is a sequence of 'blind' operations that stacks the blocks. We will now analyse `pick`:

```
pick( p )
        find-hand-stereo
        calibrate
        hand-move-above-stereo( p )
        hand-move-down-half-stereo( p )
        hand-move-above-stereo( p )
        edge = choose one camera and get a block edge
        hand-rotate-to-parallel( edge )
        z = vertical distance between hand and block
        open-fingers
        hand-move-down( z )
        close-fingers
        hand-move-up
```

`find-hand-stereo` is similar to `find-hand` in the previous experiment but it uses both cameras. `calibrate` moves the hand in 3 orthogonal directions and calculates the 'calibration matrix'. The key strategy of `pick` is the combination of `hand-move-above-stereo` and `hand-move-down-half-stereo`. The hand is controlled by the method described in chapter 5 using the observed vector from the hand to the target. `hand-rotate-to-parallel` is the same as in the previous experiment, in which the camera to be used is chosen arbitrarily. The sequence of `open-fingers`, `move-down`, `close-fingers`, and `move-up` is a sequence of 'blind' operations, but it is adequate because the hand is sufficiently close to the block such that there is no need to use visual feedback to servo the hand.

```
hand-move-above-stereo( p )
        repeat
                x = vector from hand to block p
                v = Jinv * x
                hand-move( v.x )  ;; no vertical motion
        until no movement

hand-move-down-half-stereo( p )
        v = Jinv * z              ;; z is height from hand to p
        hand-move-down( v.z / 2 )
```

This strategy of 'move above then move down half' is based on decoupling of the movement of the hand in the vertical and horizontal planes. It assumes that the vertical alignment of the robot z axis is within about ± 10 degrees (empirically determined), which is not difficult to arrange.

It can be clearly seen that `pick` contains the sensing-action strategy to achieve the part-acquiring operation. `pick` makes no assumption about the kinematic model of the robot (with some restriction on the vertical axis, but this is only for this experiment; the method described in chapter 5 is general enough to move a hand in 3 dimensions, the vertical axis restriction merely simplifies the strategy for the hand to reach a target). The program at the top-level (e.g. `stack-blocks`) does not have to concern itself about the sensing strategy at all.

7.5 Active mobile vision

At the top-level:

```
stack-two-blocks
        view-separate
        view-facing
        b = choose a block      ;; use two cameras
        pick( b )
        view-closeup
        head-vertical( y )      ;; to increase tilt angle
        view-gap
        hand-put-block-down
```

It starts with the camera head trying to find an occlusion free view of two blocks using `view-separate` and then `view-facing`. `pick` (from the previous experiment) picks up one block. The camera head then moves close in by `view-closeup` to prepare to stack the block. `head-vertical` moves the camera head up and tilts it down, and `align-blocks` moves one block to be above the other. `view-gap` changes the view to a side view and `hand-put-down` mates the blocks.

The method of finding an occlusion-free view will now be analysed.

```
view-separate
        get-blocks
        while see-one-object
                s = calculate the axis of eccentricity of the object
                d = motion in the direction according to the slope s
                head-horizontal( d )
                get-blocks

head-horizontal( x )
        head-centre
        repeat
                head-move( dx )  ;; dx is small step in x direction
                head-centre
        until head had moved x
```

The camera head is moved until it registers two objects. In moving the camera head, small steps are used (dx). The assumption about moving the camera head is

that the step size is small enough to allow the object to remain in the view all the time. Once two objects are seen, `view-facing` moves the camera head to face them so that two objects are seen well separated.

The camera head motions are achieved by a tight coupling of sensing and action inside the `head-centre` module. `head-centre` pans and tilts the camera head to centre the view on the target. The modules that want to move the camera head do not have to concern themselves with keeping the object in view and only need to be concerned about their task, for example, `view-facing` moves the camera head and just keeps monitoring the position of the two blocks without having to deal with the details of the camera head motion, `view-closeup` is similar but monitoring the area of the block.

```
view-facing
        ;; move head to face two blocks squarely
        ;; measure the slope of the line between centroids
        while | slope | > s            ;; s is a small constant
                head-horizontal( d )
```

When a good view is achieved, `pick` is used to pick up the part. `pick` from the previous experiment is used without any change. The next step is to align the blocks. `align-blocks` uses a similar technique of 'hand moves above then rotates':

```
align-blocks
        find-hand-stereo
        calibrate-with-block
        hand-move-above-approximate
        hand-with-block-rotate-to-parallel
```

`calibrate-with-block` is similar to `calibrate` but uses the block in the hand instead of just the hand (it is also simpler to find the block than to find the hand). `hand-move-above-approximate` differs from `hand-move-above-stereo` only in the use of the shape attractor method. The shape attractor method has the benefit that it uses all the edges of the image without having to select which edges to be used (as opposed to the experiment with stereo visual sensing in which the vertical edges are used). `hand-with-block-rotate-to-parallel` is similar to `hand-rotate-to-parallel` but uses the bottom edge of the block in the hand to align with the edge of the block on the table.

`view-gap` moves the camera head by `head-vertical` and monitors the tilt angle. It assumes that when the camera head is horizontal, the view of the gap between the two blocks will be an edge-on view. This assumption is quite contrived but it works for this experiment. A more general strategy, for example, monitoring the edges of the gap until they are parallel, can be used instead without affecting other modules. As long as `view-gap` achieves its post condition of having the view of the gap edge-on, other modules will continue to work. The final step is to mate the blocks:

```
hand-put-block-down
        calibrate-with-block
        hand-move-above-approximate
        z = calculate gap
        hand-move-down( z/2 )
        hand-move-above-precise
        window at gap use right camera
        mate-blocks
        open-fingers
        hand-move-up
```

`hand-move-block-down` uses the same strategy of 'move above then move down half' to place the block nearer to the target. `hand-move-above-precise` uses the vertical edges to align the block because we found that the shape attractor method doesn't give enough accuracy to mate the blocks. This inaccuracy arises from the fact that the shape of the block in the hand is partly obscured by the fingers so that it didn't match the shape of the block on the table well. `mate-block` monitors the gap until it disappears and stops the hand. `mate-block` needs a fast feedback loop to prevent overshoot. This is achieved by using a small sized window.

```
mate-blocks
        while hand-move-down-slowly
                repeat
                        get-two-blocks
                        calculate gap
                until gap < e              ;; e is a small constant
                hand-stop
```

7.6 Discussion

The sensing-action strategy is encapsulated in each module. The strategies used are suitable for the task of each individual module. In selecting the view, aligning the blocks, putting down the block, etc., each module has its own appropriate strategy. Some modules from one experiment are used in another experiment without change: `calibrate`, `pick`, etc. Some are modified: `hand-move-down-stereo`, `hand-move-above-approximate`, etc.

The ability to combine modules can be used to create a more robust module. For example, one may recover from the failure in tracking the fingertips in `hand-move-track` by using `find-hand` to re-establish the location of the hand and then continue the previous hand motion.

```
hand-move-track( m )
        p = find-fingertips
        while hand-move( m )
                camera-track-tips( p )

hand-move-track-improve( m )
        p = find-fingertips
        while hand-move( m )
                camera-track-tips( p )
                if lost-track then
                        hand-stop
                        find-hand
                        continue
```

The three principles proposed in chapter 3 are applied thoroughly in the design and construction of the behavioural modules used in the experiments described. The demolition of the centralised world model is very important and has a profound effect on the structure of the whole system. The sub-systems do not rely on a fixed set of knowledge so they are not sensitive to any change in that knowledge. Things like camera calibrations, camera locations, etc., do not play important roles in the system. Their absences simplify the system and make the system more robust. The assumptions in each module are contained within the module so the higher level modules become easier to use. The reuse of modules has been shown to be simple.

7.7 Structure

By looking at the code in the system, we can construct the hierarchy of modules (Appendix A). The modules can be classified into *ground* (not containing other modules) and *non ground*. Almost all *ground* modules belong to one of two groups: visual routines and robot motion routines. The list of all the module names is given in figure 7 4. Thus *ground* modules are like the lowest level *instruction set* that other modules can use. From this level, other modules are created by combining the lower level modules into new functionality at the intermediate level (figure 7 5), and these intermediate level modules are combined into the modules at the top level. Most of the *ground* modules reflect the capability of the physical devices.

7.8 Size of the system

To give an idea about the complexity of the system, we shall present some statistics about the size of the implementation. All programs that are used in the experiments were written in a combination of PROLOG, C, and VAL-II languages. The planner, and most of the top level and some near top level modules, are implemented in PROLOG, which then calls other modules implemented in C and VAL-II. All visual routines are implemented in C and some robot control programs are implemented in VAL II for the Adept One robot. Other robot control programs are implemented in C. The approximate size of various parts of the system are shown in the following table:[2]

[2]The planner isn't implemented by the author, the rest of the code is. The planner is the work of Malcolm (1987).

```
                    GROUND MODULES

          Visual routines
find-hand-marker
get-blocks
find-fingertips
camera-track-tips
take-picture
substract-pictures
get-three-blocks
get-two-blocks
get-vertical-edges

          Hand and head motions
head-move-to
head-move
head-rotate
close-fingers
open-fingers
hand-move-to
hand-move
hand-rotate
hand-move-up
hand-move-down
hand-move-down-slowly
hand-stop

                    NON-GROUND MODULES
reach
meeting
head-follow-hand-to
find-part
hand-approach
head-follow
head-look-down
find-hand
hand-move-above
hand-move-track
hand-rotate-to-parallel

stack-blocks
find-hand-stereo
calibrate
pick
put-on
hand-move-above-stereo
hand-move-down-half-stereo

stack-two-blocks
view-separate
view-facing
view-closeup
view-gap
align-blocks
calibrate-with-block
hand-move-above-approximate
hand-move-above-precise
hand-with-block-rotate-to-parallel
hand-put-block-down
head-vertical
head-horizontal
head-centre
head-zoom
mate-blocks
```

Figure 7–4: The list of behavioural modules

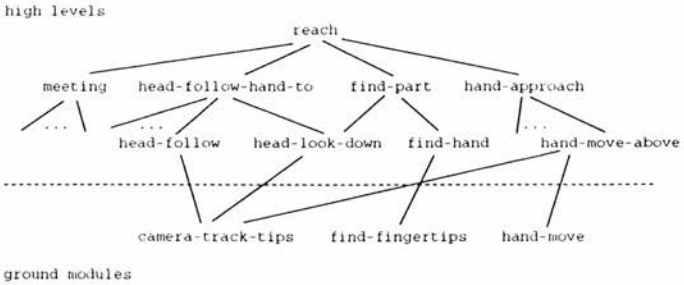

Figure 7–5: The levels in the hierarchy of behavioural modules

Program	lines
planner (PROLOG)	6,000
behavioural modules (PROLOG)	1,300
behavioural modules (C)	3,400
visual routines (C)	9,700
robot control (C)	2,000
robot control (VAL-II)	200
total	22,600

These 22,600 lines of code represents the system described in this thesis. (This doesn't count the interface to the user which is a PostScript like script language running under the X windows system. The interpreter of this script language is implemented in about 2,000 lines of C). The assemblies described in the experiment of SOMASS with vision were based on 6 plans, with a total amount of 502 lines in the top-level plan.

Chapter 8

Visual routines

This chapter discusses the vision algorithms used in the experiments. The justification of the choice of algorithms is discussed and supported by the experimental data. The data structures representing the objects are explained.

How to choose amongst so many algorithms available in the literature of machine vision? The answer is to look at the aim of our experiments, what we try to achieve. This thesis is about programming a robotic assembly system and how to integrate sensing into it. We didn't try to invent new vision algorithms, instead we tried to use at best the existing techniques. The Soma assembly domain turned out to simplify vision tasks a lot.

The methods we used are as follows. An image is acquired and segmented by global thresholding. Then it is scanned to find objects and the boundaries of the objects are traced. From the boundary lists the attributes of the objects are computed; the choice of attributes depends on the particular task. The corners of objects are found and linked together by straight lines to form structures of objects. The corners of an object are found by the corner filter. The sections that follow discuss all algorithms in detail.

8.1 Real time constraint

Our experiments use vision in closed loop control and also in tracking a target. The time limit is a real constraint in the system. We will present a 'back of an envelope' calculation to illustrate this point. Assume that a robot hand moves at the speed 20 mm/s and a camera set up has a resolution of 0.5 mm/pixel, the camera then will observe the hand moves at the rate 40 pixel/s. Assume the sampling rate of 4 Hz is required for the tracking task. The target will travel 10 pixels within one sampling period. Assume the target (hand) is 40-50 mm, e.g. about 100 pixels. It then follows that the region that needs to be processed must be 120×120 pixels in size (allow the object to move 10 pixels in any directions) and the processing must be done within 250 ms.

We checked the equipment we used to find out some basic facts. An image can be acquired every 20 ms and then the data can be transferred. The transfer times are:

size in pixel	ms
512×512	800
256×256	200
128×128	50

It can be clearly seen that there is not much time to process one image at the rate of 4 Hz. The 512×512 is impossible, the 256×256 leaves us 30 ms, and the 128×128 leaves us 180 ms. Therefore the algorithm that can be used must be a fast one.

8.2 Image segmentation

Segmentation is a process of partitioning an image into units that are homogeneous with respect to one or more characteristics. A popular technique is thresholding, which is computationally simpler than other existing algorithms, such as boundary detection or region-dependent techniques. A survey of thresholding techniques for image segmentation can be found in Sahoo and others (1988). Lee, Chung and Park (1990) also gave the performance evaluation of five global thresholding algorithms: simple image statistic method (Kittler and Illingworth, 1985); between class variance method (Otsu, 1979); entropy method (Kapur, Sahoo and Wong, 1985); moment preserving method (Tsai, 1985); and quadtree method (Wu, Hong and Rosenfeld, 1982).

8.2.1 Thresholding

An image contains objects and background. We can pick a threshold that divides the image pixels into either objects or background:

$$T = T[x, y, p(x, y), f(x, y)] \tag{8.1}$$

when $f(x, y)$ is the intensity of point (x, y) and $p(x, y)$ denotes some local property measured in the neighbourhood of this point. We then can divide the image by using the following rule:

(x, y) is part of object if $f(x, y) > T$
otherwise it is part of background

Assume that the intensity of the object is greater than the intensity of background. When T depends only on $f(x, y)$ the threshold is called *global*. If T depends on both $f(x, y)$ and $p(x, y)$ then it is called *local* and if T depends on the spatial coordinates x and y it is called a *dynamic threshold*. A global threshold is one that divides the entire image with a single threshold value, whereas a dynamic threshold partitions a given image into subimages and determines a threshold for each of these images.

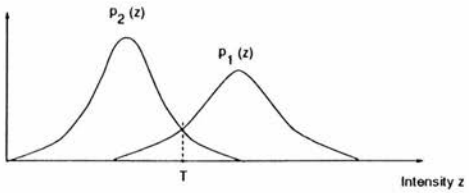

Figure 8–1: The sum of two probability density function

Global thresholds have applications in situations where there is a clear separation between objects and background and where illumination is relatively uniform.

8.2.2 Optimal threshold selection

If we assume bimodal images, the histogram is the sum of two probability density functions. The function approximating the histogram (figure 8–1) is given by:

$$p(z) = P_1 p_1(z) + P_2 p_2(z) \tag{8.2}$$

when z is a random variable denoting intensity, $p_1(z)$ and $p_2(z)$ are the probability density functions, and P_1 and P_2 are called *a priori* probabilities. It is required that

$$P_1 + P_2 = 1 \tag{8.3}$$

Let us form two functions of z, as follows:

$$d_1(z) = P_1 p_1(z) \tag{8.4}$$

$$d_2(z) = P_2 p_2(z) \tag{8.5}$$

We classify the pixel as an object if $d_1(z) > d_2(z)$ or as background if $d_2(z) > d_1(z)$. The average error of misclassifying an object as background or vice versa is minimised by choosing z such that

$$d_1(z) = d_2(z) \tag{8.6}$$

if we set $z = T$. We have that the optimal threshold satisfies the equation:

$$P_1 p_1(T) = P_2 p_2(T) \tag{8.7}$$

Thus if the function forms of $p_1(z)$ and $p_2(z)$ are known, we can solve for the optimal threshold that separates objects from background.

We can model a multimodal histogram as the sum of n probability density function so that

$$p(z) = P_1 p_1(z) + \cdots + P_n p_n(z) \tag{8.8}$$

Then the optimal thresholding problem may be viewed as classifying a given pixel as belonging to one of n possible categories. The minimum error decision is based on n functions of the form:

$$d_i(z) = P_i p_i(z) \qquad i = 1, 2, \ldots, n \tag{8.9}$$

A given pixel with intensity z is assigned to the kth category if $d_k(z) > d_j(z) \quad j = 1, 2, \ldots, n; j \neq k$. The optimum threshold between category k and category j, denoted by T_{kj}, is obtained by solving the equation:

$$P_k p_k(T_{kj}) = P_j p_j(T_{kj}) \tag{8.10}$$

8.2.3 The method in the experiments

To decide what method to use we took many images of the typical scene in the workcell of the Soma assembly domain (figure 8–2 to 8–5). Figure 8–2 shows a white plastic part with a good contrast. Figure 8–3 shows a wooden part; the part has a wider spread of its intensity value. Figure 8–4 shows a wooden part with shadow; the histogram is trimodal. Figure 8–5 shows a wooden part, with a smaller window; the area of the part has a larger proportion than all previous pictures, and shadows are present. We collected histograms and found that they were strongly bimodal, and only became trimodal when there were shadows. From this knowledge, we decided to use the simplest method of global thresholding. Relying on the most invariant feature of the histogram, that is the peak of the background, our method searches for the valley to the right of the background peak to find a threshold that divides the image into objects and background. As seen from the typical images, the histograms are quite noisy, and must be smoothed before doing the search to prevent choosing the wrong valley (local minima).

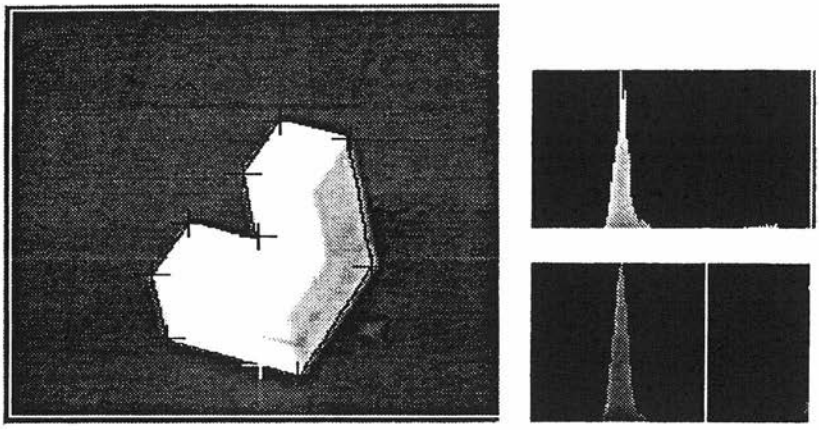

Figure 8–2: A typical scene 1

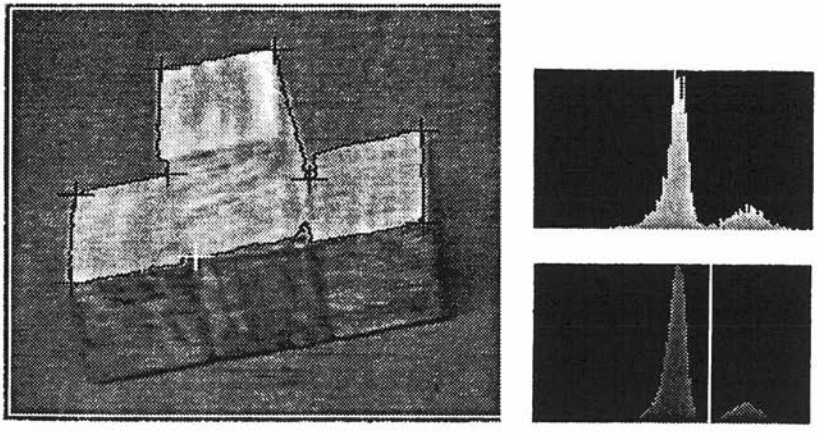

Figure 8–3: A typical scene 2

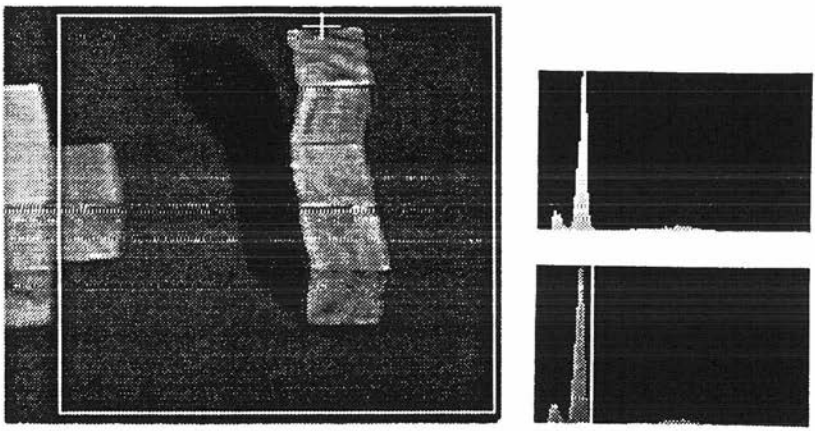

Figure 8–4: A typical scene 3

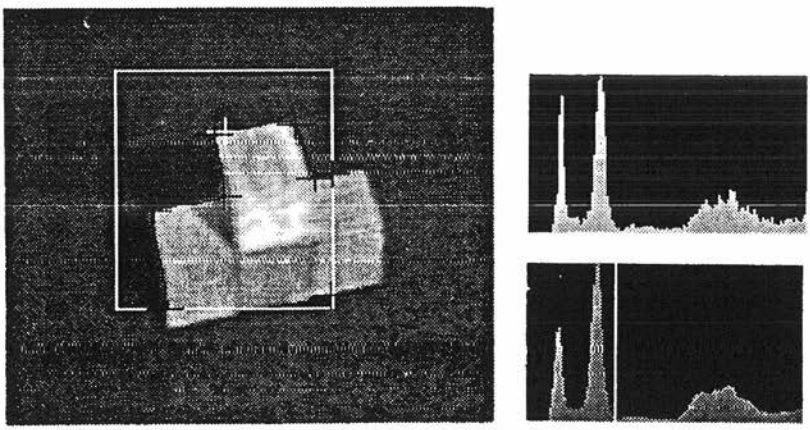

Figure 8–5: A typical scene 4

We use Gaussian smoothing by repeated averaging using a 1×3 mask. There are many possible masks. To speed up the smoothing, one can design a Gaussian smoothing mask with a smaller central coefficient.[1] For example, start with a Gaussian mask $1/16 \times [1, 14, 1]$, move to a spline smoothing mask $1/6 \times [1, 4, 1]$,[2] and further to a Gaussian mask $1/5 \times [1, 3, 1]$, even a Gaussian mask $1/4 \times [1, 2, 1]$, where each of the 1×3 masks must be subject to a formula for 3 coefficients: $[\beta,\ 1-2\beta,\ \beta]$. We use the mask $1/4 \times [1, 2, 1]$. It is derived from a 3×3 Gaussian mask which is the optimal integer value mask in overall accuracy approximation as shown in Cai (1990, theorem 3.8, p.42).

This mask is the best one in the sense that it has the smallest central weighting coefficient, except the simple averaging mask $1/3 \times [1, 1, 1]$, thus it provides the maximal Gaussian smoothing rate. Also, all the weighting coefficients within this mask are powers of 2. Hence, only operations of shifting and addition are involved in the smoothing, thus working much faster than any other Gaussian masks where multiplications are required.

8.3 Tracing the boundary

The algorithm to trace a boundary is as follows (figure 8–6): imagine an observer sits between the pixels at the boundary of an object, this observer traces the boundary by looking at 2 pixels ahead (L and R) and keeps the object on its right-hand side, it uses the following rules (X = background, O = object):

L	R	Action
X	X	turn right
X	O	report this point and go ahead
O	O	report this point, turn left (move diagonally)
O	X	same as above (this is to prevent going 'into' the object)

[1] Li-Dong Cai pointed this out to the author.

[2] B-spline mask is equivalent to a specific Gaussian mask, see (Cai, 1990, chapter 3) for more details.

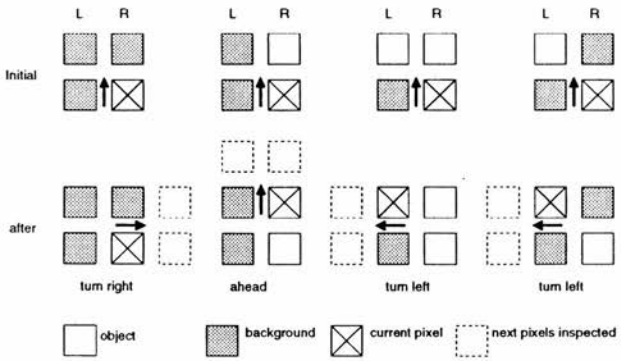

Figure 8–6: Tracing the boundary

Terminate upon returning to the starting point with the same orientation. This algorithm reports 8-connected boundary lists. It uses only 4 directional headings. Of 8-connected vectors in a list, four right angle components are produced by the move ahead and the other four 45 degrees components are produces by the left turn. The algorithm inspects as few pixels as possible. It turns right to inspect two pixels on the other side of the current pixel and does not inspect the same pixel twice in a move.

To find a starting point, an image is scanned until it hits the object pixel. To prevent a false trigger by small noise, the scanner checks the width of the object by counting the number of consecutive object pixels, which must be greater than k, a constant which denotes the minimum width of the object of interest. Many algorithms to follow contours of irregular 8-connected figures are given in Rosenfeld and Kak (1982).

8.4 Attributes of objects

From the boundary of objects, simple attributes can be calculated. We used the following, when $x_i y_i$ are coordinates on the boundary list:

1) centroid

$$x_c = \frac{\sum_{i=0}^{n-1} x_i}{n} \tag{8.11}$$

$$y_c = \frac{\sum_{i=0}^{n-1} y_i}{n} \tag{8.12}$$

2) area; the area of a closed curve is given by

$$\frac{1}{2} \mid \sum_{i=0}^{n-1} (x_i y_{i+1} - x_{i+1} y_i) \mid \tag{8.13}$$

3) perimeter; it is the length of boundary list n 4) compactness

$$\frac{perimeter^2}{area} \tag{8.14}$$

5) eccentricity (from Ballard and Brown, 1982, p.255);
start with the mean vector

$$\mathbf{x}_0 = \frac{1}{n} \sum_{\mathbf{x} \; in \; R} \mathbf{x} \tag{8.15}$$

when R is the boundary list, then compute the ijth moments M_{ij} defined by

$$M_{ij} = \sum_{\mathbf{x} \; in \; R} (x_0 - x)^i (y_0 - y)^j \tag{8.16}$$

The orientation, θ, is given by

$$\theta = \frac{1}{2} tan^{-1} \left(\frac{2M_{11}}{M_{20} - M_{02}} \right) + n\left(\frac{\pi}{2}\right) \tag{8.17}$$

and the approximate eccentricity e is

$$e = \frac{(M_{20} - M_{02})^2 + 4M_{11}}{area} \tag{8.18}$$

8.5 Corner filter

Malcolm (1983) reported an algorithm to find the corner points from a boundary list. The goal of the corner filter was to find a good polygonal approximation to an object outline as fast as possible. The corner points are the basis for discovering features of the object relevant to assembly. We used an updated version based on that algorithm.[3] The algorithm is easy to visualise by imagining a worm crawling around the boundary. As it moves, the gradient of the line joining its head and tail will change. The determination of the corner points is based on this change.

We considered $2w$ pixels on the boundary. $\mathbf{P} = (x, y)$ is a point on the boundary, n is an index into the boundary point, $\mathbf{P}_n$ is at the head of the worm, its mid-point is at $\mathbf{P}_{n-w}$ and its tail is at $\mathbf{P}_{n-2w}$. The mid-point is used to report a corner.

We define the estimated gradient by forward differencing:

$$\nabla \hat{\mathbf{P}}_i \doteq \mathbf{P}_{i+w} - \mathbf{P}_i \tag{8.19}$$

An estimate of the curvature of P's trajectory is the second difference. We use its absolute value because we are not interested in its sign:

$$D_n = \mid \nabla \hat{\mathbf{P}}_{n-w} - \nabla \hat{\mathbf{P}}_{n-2w} \mid \tag{8.20}$$

and therefore

$$D_n = \mid \mathbf{P}_n - 2\mathbf{P}_{n-w} + \mathbf{P}_{n-2w} \mid \tag{8.21}$$

By setting a threshold at a, the local maximum between the points at which $D_j > a$ and $D_k \leq a$ is reported as the corner point (figure 8–7).

For a slow changing curve, the point cannot be reported by $D_n > a$. The accumulated change is used instead:

$$T_n = \sum_{i=p+1}^{n} D_n \tag{8.22}$$

---

[3]Our algorithm uses $s = w$, when s is the number of pixels over which the smoothing total is maintained and w is the size of the worm in that report.

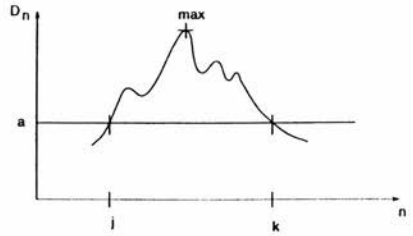

Figure 8–7: Local maxima of a sharp corner

when p = the last reported corner point. The mid-point is reported when $T_n > t$. There is a 'left over' accumulation (T_n) when the worm goes out from a sharp corner, therefore we set $T_n = 0$ until $n > p + 2w$.

The last kind of corner point reported is when the boundary deviates from the straight line between the head of the worm and the last point reported.

Let **O** be the middle point on the boundary between $\mathbf{P}_n$ and $\mathbf{P}_p$, and **M** the middle point of the line joining $\mathbf{P}_n$ and $\mathbf{P}_p$. The deviation is the distance between **O** and **M** (figure 8–8).

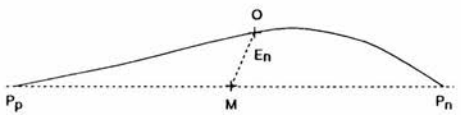

Figure 8–8: Deviation measure

$$E_n = \|\mathbf{O} - \mathbf{M}\| \tag{8.23}$$

when $\|\mathbf{H}\|$ is the magnitude of **H**. The mid-point is reported when $E_n > e$. In short, the corner finder is a function:

$corner(w, a, t, e)$

when w = smoothing interval
a = threshold for sharp corners
t = threshold for curve corners
e = threshold for deviation corners

(Malcolm, 1983) contains the details of a fast implementation of this algorithm. Figure 8 9 shows the result of applying this algorithm to a typical image. In the experiments, we set the parameters $w = 10$ and $a = 7$, and only sharp corners are considered (t and e are set arbitrarily large). This is because the features of interest are characterised by sharp corners (i.e. top edges of a cube).

8.6 Shape attractor

Harris (1992) reported an algorithm to align two sets of edges together despite the presence of significant distortion and missing or extraneous line segments. This algorithm works by iteratively translating, rotating and dilating one set of line segments onto its target. The following sections are summarised from that paper.

8.6.1 Closeness function

Consider the m moving line segments and n target segments. The algorithm computes a weight for all $m \times n$ possible mapping of mobile segments onto target segments. The weight w_{ij} is bigger the closer the ith (mobile) and jth (target) segments are to each other. *Closeness* is defined to be a function of orientation, length and distance apart of line segments. It depends on the dot product of and the divergence (δ) between the line segments. If two lines are represented as follows:

$$\begin{aligned} \vec{\mathbf{p}}(t) &= \vec{\mathbf{p}}_1 + t\vec{\mathbf{p}}_2 \\ \vec{\mathbf{q}}(t) &= \vec{\mathbf{q}}_1 + t\vec{\mathbf{q}}_2 \end{aligned} \tag{8.24}$$

divergence is defined as:

$$\delta^2 = \int_0^t \|\vec{\mathbf{p}}(t) - \vec{\mathbf{q}}(t)\|^2 \, dt \tag{8.25}$$

This works out to be

$$\delta^2 = \|\vec{\mathbf{r}}\|^2 + \frac{1}{3}\|\vec{\mathbf{s}}\|^2 + \vec{\mathbf{r}}.\vec{\mathbf{s}} \tag{8.26}$$

where

$$\begin{aligned} \vec{\mathbf{r}} &= \vec{\mathbf{p}}_1 - \vec{\mathbf{q}}_1 \\ \vec{\mathbf{s}} &= \vec{\mathbf{p}}_2 - \vec{\mathbf{q}}_2 \end{aligned}$$

The *closeness* value is given by:

$$w = (\|\mathbf{p}_2\| + \|\mathbf{q}_2\|)e^{r(C\cos\theta - K\delta^2)} \tag{8.27}$$

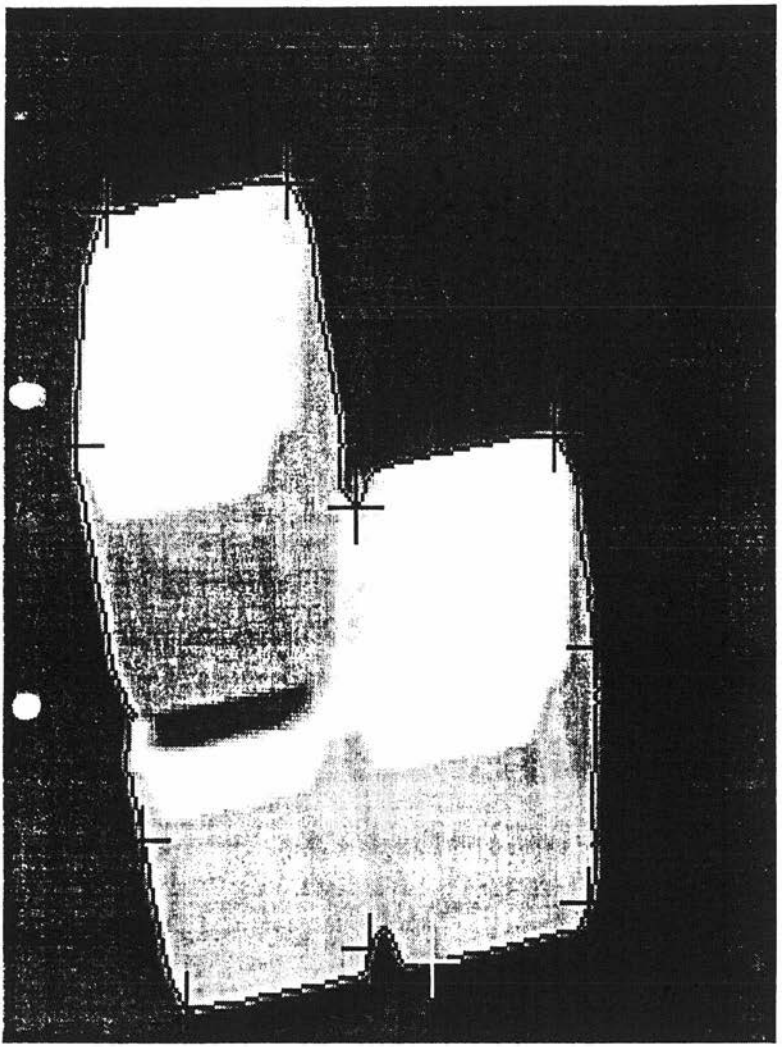

Figure 8–9: The result of applying the corner filter

C and K are positive constants and $0 \leq r \leq 1$. This function is Gaussian with respect to the divergence of the two line segments. When r is 1, K determines the spread of the Gaussian and thus the range of distance over which differences in separation have a strong effect on w. K depends on the amount of noise expected in the data. We used $C = 5$ and $K = 0.001$ where coordinates are pixel values.

8.6.2 Computing the translation

The overall translation applied to the mobile segments, $\mathbf{v}'$, is the weighted sum of the individual $\mathbf{v}_{ij}$ between the centres of segments i (mobile) and j (target):

$$\mathbf{v}' = \frac{\sum_{ij} w_{ij} \mathbf{v}_{ij}}{\sum_{ij} w_{ij}} \tag{8.28}$$

Similarly, for rotation the individual rotations are those that would make segment i parallel with segment j. The centroid of the mobile segments is used as the centre for the resulting rotation vector:

$$r' \begin{pmatrix} \cos\theta' \\ \sin\theta' \end{pmatrix} = \frac{1}{\sum_{ij} w_{ij}} \sum_{ij} w_{ij} \begin{pmatrix} \cos\theta_{ij} \\ \sin\theta_{ij} \end{pmatrix} \tag{8.29}$$

For the first iteration r can be set close to zero. Each weight then becomes the sum of the lengths of the two segments and the resulting translation aligns the centroids of the two figures, with no overall rotation. For the next iteration, the radius r' is used.

8.7 Tune up for speed

To measure what is the processing rate we can achieve, we tested a simple process of acquiring an image, thresholding it and tracing the boundary, then finding all corners. The scene (figure 8–2) has only one Soma part in it and the background is clear (only single object detected). A straightforward implementation is timed:

Process	time in ms for 256×256	128×128
acquire image	20	20
transfer data	200	50
compute histogram	120	60
smooth and threshold	200	200
trace	40	40
corner	40	40
total	620	410

From this profile, the data transfer becomes the bottle-neck when an image is large, also the smoothing which is an iterative process becomes the next bottle neck. We reduced the time to transfer data by using subsampling to compute a histogram, and by accessing only the required pixel in scanning to find objects and in tracing the boundaries. We avoided computing a new threshold whenever possible, especially when it is in a tight loop and a fast processing speed is required. For example, in tracking the fingertips, the threshold is found once in the beginning and is used throughout the subsequent processing to find the fingertips. With a careful implementation, we achieved the rate of 8 Hz with a window of size 200×318 pixels and 12.5 Hz with a small window of size 40×40 pixels while tracking the gap in the part-mating task.

In many circumstances speed of processing is not essential. For example, in finding the hand by locating its marker or in finding a part to be picked up, we were not worried if the visual routines took one second to complete the task.

8.8 Discussion

The choice of algorithms is constrained very much by the real time limit. Our visual routines are task specific but they performed well in the experiments. The Soma domain simplifies the task of vision processing because we can rely on using just the outside edges and other attributes that can be computed using the boundary list. The limitation arises when there is a need to inspect the 'inside' of an object. The algorithms we chose consider only the 'silhouette' (or outline) of objects. For

the purpose of using the 'internal' edges a standard edge detector like Canny can be used (Canny, 1983). In tracking features, 'snakes' as originally proposed by (Kass, Witkin, and Terzopoulos, 1987), can be used for locating features of interest in images and tracking their image motion as long as the feature does not move too fast. A 'snake' is an active contour model which minimises an expression of energy to locate image features. (Curwen, Blake, and Cipolla, 1991) also reported an implementation of this mechanism using B-spline model of the feature (to reduce the state space of the problem). For the corner finder, a new algorithm is reported in Smith (1992) that can work with gray levels without image segmentation. Because of the real-time constraint, special hardware might be necessary in order to compute more elaborate algorithms.

Chapter 9

Conclusion

All things transitory
But as symbols are sent:
Earth's insufficiency
Here grows to Event:
Here it is done:
The Woman-Soul leadeth us
Upward and on!

from Faust
Johann Wolfgang von Goethe

How to program robots to work reliably in the presence of uncertainty?

The classical approach to this problem is to analyse the uncertainty of the parts' geometry and their locations to determine the sensing and actions that are necessary to reduce the uncertainty to an acceptable level. The problem with the classical approach is that the analysis of uncertainty and the synthesis of sensing and action to reduce the uncertainty are computationally expensive and generally intractable. Presently, this analysis of uncertainty also is only able to represent a subset of the types of uncertainty actually experienced at execution time.

The behaviour-based approach suggests a different solution: use a competent execution system to deal with the inevitable uncertainty of the real world. This has a profound effect on the architecture of robotic assembly systems. The sensing operations are embedded in the run-time system, which allow the programming (or task planning) to be carried out in an ideal world in which robots perform their tasks reliably.

This thesis suggested an architecture for the execution system and proposed the criteria for decomposing a task into modular units which are called behavioural modules. Behavioural modules are task-achieving units (the task is in the world, e.g., computational tasks don't count). Programming a robot in terms of behavioural modules leads naturally to task-level programs. Experiments described in this thesis show that behavioural modules can encapsulate the essential information about the world and communicate without relying on a centralised model of the world and without a global coordinate system. An individual module only receives the information that is required to perform its task. The tight coupling of the sensing and action inside individual modules is an important idea in coping with the uncertainty of the real world.

Behavioural modules in this work have the following general properties :

1. In terms of computational structure and modularity; behavioural modules can be easily combined to create new modules.

2. In terms of the assembly operations they achieve; the behavioural module interface easily matches to the appropriate atomic plan terms.

3. Generality over variations of the forms and positions of the parts; the planner does not deal with uncertainty, leading to simplification of the planning and assures reliability.

4. Independent of technology; the independence of the programmer from knowing the sensing strategies and type of sensors.

The rest of this chapter consists of three sections. The first briefly lists the contributions of the work described in this thesis. The second discusses some directions for extending this research. The third briefly summarises the work in terms of the expectations originally set at in chapter 1.

9.1 Contributions

The work described in this thesis has made a number of contributions to solving the problem of programming assembly robots to perform reliably in the presence of uncertainty. They are summarised below.

Integration of sensing into assembly systems

The decomposition of tasks into task achieving units gives rise to a new modularisation of assembly systems in which the modules may contain sensing operations. The work in this thesis demonstrates that this modularisation allows sensing operations to be seamlessly integrated into a robotic assembly system (section 1.8 p. 57). Moreover, the introduction of sensing into the system doesn't require that the programmer or the task planner has to know the details of the sensors or the strategy to use them. This contrasts to the previous work in this area which required the intimate connection of sensing operations and task planning.

Uncalibrated vision

The uncalibrated vision presented in this work shows that visual sensing in a robotic assembly system doesn't have to rely either on precise calibration or on using a centralised model of the world. This visual sensing can be used in a feedback control loop, and it can be self calibrating so that the locations of cameras or their lens parameters are not required as *a priori* knowledge (chapters 4 and 5). This enables the use of visual sensing in a more flexible way because the camera can be moved (chapter 6). The scheme of visual servoing presented is computationally cheap and can be applied to commercially available position controlled manipulators. It also is demonstrated in one case that the accuracy achieved by this visual servoing method is better than the accuracy of using the robot's inverse kinematics calculation (section 5.6 p. 67).

The design and implementation of behavioural modules

The design of behavioural modules is dictated by the criteria of the task decomposition (section 3.3 p. 26). From the analysis of the system presented in this work, it is shown that behavioural modules have hierarchical structures and their interfaces can be clearly defined. The assumptions about the world in which they operate and the

assumptions about the operations of other modules can be made explicit (chapter 7). The implementation of behavioural modules can be carried out in an independent manner from other parts of the system development once the specifications of behavioural modules are defined. Each module can be implemented separately in a bottom-up fashion. The specification of behavioural modules facilitates the testing of the modules and the system integration process (section 3.6 p. 34).

9.2 Future work

There is a long list of interesting variations and extensions that could be made to the work described in this thesis. Some of them which promise direct applications are described here.

Extension of the stereo vision system

The uncalibrated stereo vision system presented in this work could be extended so that the calibration matrix can be updated on the run. This would be a generalisation of the method for storing and indexing the calibration matrices in Domingo (1991). The limitation of the storing and indexing method is that the matrices are built only for a fixed camera location. The ability to update the calibration matrix on the run will enable cameras to be moved about without recalibration for different views.

Sensor fusion

The architecture proposed in this work allows the use of multiple sensors, even with sensing of a different modality. Although this work demonstrates vision sensing, other work had demonstrated tactile sensing (Wilson, 1992) and force sensing (Balch, 1992) in a similar framework. Although none of these systems demonstrates the use of more than one mode of sensing, fusion of multiple modes is a clear logical development. Sensor fusion can be made based on a task, e.g., several behavioural modules which have different sensing modalities, each of which achieves its task, can collaborate to achieve a desired task.

Multiple robots co-operation

This work demonstrates two robots co-operating in sensing and manipulation but

only one robot manipulates the objects. It is possible to envisage more than one robot that can co operate in the task of object manipulation. Steels (1989) has given examples of how distributed agents can co operate and Brooks (1991a) suggested an approach to studying the control of multiple agents. Behavioural modules could be designed to take into account more than one agent performing a task. Behavioural modules can be used at task level so the programming of a system of this type should not be overly complicated. It will be interesting to understand how resources can be shared amongst the agents. For example, a camera can observe both robot hands at the same time, therefore it can participate in the control of both robots. The interaction between agents inside a module and between modules will be of great interest.

9.3 Epilogue

Programming robots and sensors to do assembly tasks is very much simplified by using behavioural modules. Simply by inspecting the code for robots to do the assemblies presented in this work, one can see that, at one extreme, the code at the level of individual motions is lengthy, and this makes it hard to maintain or to modify. At the other end of the spectrum, a plan in terms of behavioural module calls, is much shorter, easier to understand and can be adapted to a new task without much difficulty. Visualising a robot motion in three dimensional space to achieve an assembly task reliably is a skill that can be acquired only after a lot of practice. There are numerous surprises when one deals with real objects, since not everything is exactly as one first expects. Using sensors can also be intriguing. When I was trying to find a way to use information from a camera to guide a robot hand, I was puzzled about how to use 2 dimensional data to control 3 dimensional motions. Only after I tried to control the robot manually while looking through the robot's eye (the image from the camera displayed on a monitor), did I find the solution presented in chapter 4. The behavioural module concept captures this sort of skill quite nicely and also hides the complication of using sensors from a user. This concept of purposeful modules, that contain sensing and action, is my answer to the question of programming robots which I posed in the beginning of this chapter. The next important question is how well it scales to a complex domain of assembly? Maybe the only way to find out is to try.

Appendix A

Catalogue of behavioural modules

The appendix contains all robot programs used in this thesis. There are two parts: the first part contains the listing of pseudo codes of behavioural modules, and the second part lists the hierarchical structure of behavioural modules.

A.1 Naming convention

The names of behavioural modules are given in the form `Noun-Verb-Object` and will be unique for each module. `Noun` denotes the agent like `hand`, `head`, `camera`, when there is no ambiguity it is omitted. A long name usually indicates that it is similar to another module but with extra qualifications, for example, `hand-move-above` and `hand-move-above-stereo`.

A.2 Pseudo code

SINGLE CAMERA

```
reach( p )
        ;; hand moves to nominal position of a part p and then moves to it
        meeting
        head-follow-hand-to( p )
        edge = find-part
        hand-approach( edge )

meeting
        ;; move camera and hand to a meeting point
```

```
        ;; x,y  are meeting points in hand coordinate and head coordinate
        hand-move-to( x )
        head-move-to( y )
        find-hand-marker        ;; check to see that camera can see hand

head-follow-hand-to( p )
        ;; move hand to p, camera follows and then looks at the part
        loc = find-hand-marker
        while hand-move-to( p )
                head-follow
        head-look-down

find-part
        ;; look for a part underneath the hand
        p = find-hand
        make window under hand to look for part at p
        repeat 3 times or until get good edge
                get-blocks
                choose the one nearest to the hand
                get two edges at top corner
                edge = select edge that most parallel to fingertips
                check good edge by size and slope
        return( edge )

hand-approach( edge )
        ;; move hand close to a part
        z = down half height
        hand-move-track( z )
        a = mid point
        hand-move-above( a )
        hand-rotate-to-parallel( edge )
        z = down close to near point
        hand-move-track( z )
        b = near point
        hand-move-above( b )

hand-move-track( m )
        ;; move hand and keep tracking fingertips
        p = find-fingertips
        while hand-move( m )
                camera-track-tips( p )

find-hand-marker
        ;; locate hand marker in the image
        use camera red channel
        find blobs with properties of hand marker
        loc = centroid of blob
        turn camera back to black and white channel
        return( loc )

find-hand
        ;; locate hand by difference two images which fingers move
        close-fingers
        take-picture
        open-fingers
        take-picture
        w = subtract-pictures
        p = find-fingertips( w )
        return( p )

find-fingertips( w )
        ;; locate left and right fingertips
        make two windows correspond to two fingertips in w
        find blobs with properties of white fingertips
        p = centroid of two lowest blobs
        return( p )

head-follow
        ;; pan-tilt head to follow the moving hand
        while hand-is-moving
                loc = find-moving
                m = the move to centre view on loc
```

```
                head-rotate( m )

head-look-down
        ;; pan-tilt head until hand-marker is at offset
        loc = offset
        repeat
                x = find-hand-marker
                m = the move to shift view from x to loc
                head-rotate( m )
        until  hand-marker is at loc

hand-move-above( dest )
        ;; move head, currently at p, to dest while keep camera tracking
        ;; p in image coordinate (x,y)  T is transformation
        repeat
                v = p - dest
                m = map( v, T )
                hand-move-track( m )
                v1 = observed move of v
                T = remap( v1 , m )
        until p near dest

hand-rotate-to-parallel( edge )
        ;; rotate hand until it is parallel to the edge. hand is empty
        repeat
                dr = small rotation to make hand parallel to edge
                while hand-rotate( dr )
                        camera-track-tips( p )          ;; p is current position
        until hand paralle to edge

camera-track-tips( p )
        ;; tracking the location of p while hand move
        while hand-is-moving
                make small windows centred at each fingertips in p
                find blobs with properties of white fingertips
                choose the most similar to the previous blobs
                p = centroid of blobs


                                STEREO  VISION

stack-blocks
        ;;  pick up and stack three blocks at a place
        ;;  place, table, block1, block2 are predefined
        get-three-blocks( b1, b2, b3 )
        pick( b1 )
        hand-move-to( place )
        put-on( table )
        pick( b2 )
        hand-move-to( place )
        put-on( block1 )
        pick( b3 )
        hand-move-to( place )
        put-on( block2 )

pick( p )
        ;; pick up a block at p
        find-hand-stereo
        calibrate
        hand-move-above-stereo( p )
        hand-move-down-half-stereo( p )
        hand-move-above-stereo( p )
        edge = choose one camera and get a block edge
        hand-rotate-to-parallel( edge )
        z = vertical distance between hand and block
        open-fingers
        hand-move-down( z )
        close-fingers
        hand-move-up

find-hand-stereo
```

```
        ;; same as find-hand but do both cameras

calibrate
        ;; find the approximate mapping between hand motion and image
        ;; by move in 3 orthogonal directions and observe 3 vectors
        ;; calculate Jinv which map these 3 vectors to motions

hand-move-above-stereo( p )
        ;; move hand to point above p, no vertical motion
        repeat
                x = vector from hand to block p
                v = Jinv * x
                hand-move( v.x )
        until no movement

hand-move-down-half-stereo( p )
        ;; z is height from hand to p, v is z converted to robot motion
        v = Jinv * z
        hand-move-down( v.z/2 )

put-on( h )
        ;; put block down, by dead reckoning
        hand-move-down( h )
        open-fingers
        hand-move-up


                                ACTIVE MOBILE VISION

stack-two-blocks
        view-separate
        view-facing
        b = choose a block          ;; use two cameras
        pick( b )
        view-closeup
        head-vertical( y )          ;; to increase tilt angle
        align-blocks
        view-gap
        hand-put-block-down

view-separate
        ;; move head to have a view which two objects are not overlapped
        get-blocks
        while see-one-object
                s = calculate the axis of eccentricity of the object
                d = motion in the direction according to the slope s
                head-horizontal( d )
                get-blocks

view-facing
        ;; move head to face two blocks squarely
        ;; by measuring the slope of line between centroids of two objects
        while | slope | > s                ;; s is a small constant
                head-horizontal( d )

view-closeup
        ;; move head closer until area of the block in hand > a
        b = choose the block in hand
        head-centre
        while b.area < a                    ;; a is a constant
                head-zoom( dr )             ;; dr is a small movement

view-gap
        ;; move head while keeping object in the view until head is
        ;; almost horizontal, looking parallel to mating surfaces

align-blocks
        find-hand-stereo
        calibrate-with-block
        hand-move-above-approximate
        hand-with-block-rotate-to-parallel
```

```
head-horizontal( x )
        ;; move head left or right while keeping object in the view
        head-centre
        repeat
                head-move( dx )             ;; dx is small step in x direction
                head-centre
        until head had moved x

head-vertical( y )
        ;; similar to head-horizontal, but move up or down

head-zoom( z )
        ;; similar to head-vertical, but move in or out

head-centre
        ;; pan-tilt head until target is in the centre of the view

head-move( d )
        ;; move head relate to the current position
        ;; along x or y or z  reference to head coordinate

hand-move-above-approximate
        ;; move to above a block using shape-attractor to find translation
        repeat
                get-two-blocks
                x = translation vector by shape attractor
                v = Jinv * x
                hand-move( v.x )
        until no-movement

hand-move-above-precise
        ;; same as hand-move-above-approximate but
        ;; use translation-use-vertical-edge
        repeat
                get-two-blocks
                get-vertical-edges
                x = translation vector by shape attractor
                v = Jinv * x
                hand-move( v.x )
        until no-movement

hand-put-block-down
        ;; put block down while monitoring the gap until blocks are mated
        calibrate-with-block
        hand-move-above-approximate
        z = calculate gap
        hand-move-down( z/2 )
        hand-move-above-precise
        window at gap use right camera
        mate-blocks
        open-fingers
        hand-move-up

mate-blocks
        ;; move block in hand down until no gap
        while hand-move-down-slowly
                repeat
                        get-two-blocks
                        calculate gap
                until gap < e           ;; e is a small constant
                hand-stop

hand-with-block-rotate-to-parallel
        ;; rotate hand until block in hand parallel to block on table
        get-two-blocks
        get slopes s1, s2               ;; s1,s2  slopes of bottom edges of blocks
        while | s1 - s2 | > e           ;; e is a small constant
                dr = small rotation to reduce difference of slopes
                hand-rotate( dr )
                get-two-blocks
                get slopes s1, s2
```

```
calibrate-with-block
        ;; same as calibrate but use block in hand instead of the hand
```

A.3 Hierarchical structure

SINGLE CAMERA

```
reach
        meeting
        head-follow-hand-to
        find-part
        hand-approach

meeting
        hand-move-to
        head-move-to
        find-hand-marker

head-follow-hand-to
        find-hand-marker
        hand-move-to
        head-follow
        head-look-down

find-part
        find-hand
        get-blocks

hand-approach
        hand-move-track
        hand-move-above
        hand-rotate-to-parallel

hand-move-track
        find-fingertips
        hand-move
        camera-track-tips
```

```
find-hand
        close-fingers
        take-picture
        open-fingers
        subtract-pictures
        find-fingertips

head-follow
        find-moving
        head-rotate

head-look-down
        find-hand-marker
        head-rotate

hand-move-above
        hand-move-track

hand-rotate-to-parallel
        hand-rotate
        camera-track-tips
```

STEREO VISION

```
stack-blocks
        get-three-blocks
        pick
        hand-move-to
        put-on

pick
        find-hand-stereo
        calibrate
        hand-move-above-stereo
        hand-move-down-half-stereo
        hand-rotate-to-parallel
        open-fingers
        close-fingers
        hand-move-down
        hand-move-up
```

```
hand-move-above-stereo
        hand-move

hand-move-down-half-stereo
        hand-move-down

put-on
        hand-move-down
        open-fingers
        hand-move-up
```

ACTIVE MOBILE VISION

```
stack-two-blocks
        view-separate
        view-facing
        view-closeup
        pick
        head-vertical
        align-blocks
        view-gap
        hand-put-block-down

view-separate
        get-blocks
        head-horizontal

view-facing
        head-horizontal

view-closeup
        head-centre
        head-zoom

view-gap
        head-vertical

align-blocks
        find-hand-stereo
        calibrate-with-block
        hand-move-above-approximate
        hand-with-block-rotate-to-parallel

head-vertical
        head-centre
        head-move

head-horizontal
        head-centre
        head-move

head-zoom
        head-centre
        head-move

head-centre
        head-rotate
```

```
hand-move-above-approximate
        get-two-blocks
        hand-move

hand-move-above-precise
        get-two-blocks
        get-vertical-edges
        hand-move

hand-put-block-down
        calibrate-with-block
        hand-move-above-approximate
        hand-move-down
        hand-move-above-precise
        mate-blocks
        open-fingers
        hand-move-up

mate-blocks
        hand-move-down-slowly
        get-two-blocks
        hand-stop

hand-with-block-rotate-to-parallel
        get-two-blocks
        hand-rotate
```

Bibliography

Agre, P.E., and D. Chapman. (1987). Pengi: an implementation of a theory of activity. *Proc. of 6th National Conf. on Artificial Intelligence*, pp. 268-272.

Aloimonos J., I. Weiss and A. Bandyopadhyay. (1987). Active vision. *Proc. of 1st Int. Conf. on Computer Vision*, London, IEEE computer society press, pp. 35 54.

Ambler, A.P., H.G. Barrow, C.M. Brown, R.M. Burstall, and R.J. Popplestone. (1975). A versatile system for computer controlled assembly. *Artificial Intelligence*, 6:129-156.

Andersson, R.L. (1988). *A robot ping-pong player*. MIT Press.

Arbib, M.A., K.J. Overton, and D.T. Lawton. (1984). Perceptual systems for robots. *Interdisciplinary Science Reviews*, 9:31-46.

Balch, P. (1992). Force sensing in an industrial assembly. unpublished manuscript.

Ballard, D.H. and C.M. Brown. (1982). *Computer vision*. Prentice Hall, NJ, p. 255.

Bentley, J. (1988). *More programming pearls*. Addison-Wesley, p. 63.

Brooks, R.A. (1982). Symbolic error analysis and robot planning. *Int. Jour. of Robotic Research*, 1(4):29-68.

Brooks, R.A. (1985). A robust layered control system for a mobile robot. MIT AI Lab., AI memo 864.

Brooks, R.A. (1986). Achieving artificial intelligence through building robots. MIT AI Lab., AI memo 899.

Brooks, R.A. (1987). A hardware retargetable distributed layered architecture for mobile robot control. *Proc. IEEE Int. Conf. on Robotics and Automation*, Vol. 1, pp. 106-110.

Brooks, R.A. (1989). A robot that walk: emergent behaviours from a carefully evolved network. MIT AI Lab., AI memo 1091.

Brooks, R.A., (1990). The behavior language: user's guide. MIT AI Lab., AI memo 1227.

Brooks, R.A. (1991a). Challenges for complete creature architectures. In J. Meyer, and S.W. Wilson, (Eds.), *Proc. of the first Int. Conf. on Simulation of Adaptive Behaviour*, pp. 434-443.

Brooks, R.A. (1991b). Intelligence without representation. *Artificial Intelligence*, 47:139-159.

Brooks, R.A., J. Connell, and A.M. Flynn. (1986). A mobile robot with onboard parallel processor and large workspace arm. *Proc. of 5th National Conf. on Artificial Intelligence*, Philadelphia , Vol. 2, pp. 1096-1100.

Brooks, R.A., A.M. Flynn, and T. Marill. (1987). Self calibration of motion and stereo vision for mobile robot navigation. MIT AI Lab., AI memo 984.

Brown, C.M. (Ed.) (1988). Rochester robot. University of Rochester, Computer Science, technical report 257.

Buckley, S.J. (1987). Planning and teaching compliant motion strategies. Ph.D. thesis, MIT, MIT AI Lab., AI memo 936.

Cai, L. (1990). Scale-based surface understanding using diffusion smoothing. Ph.D. thesis, University of Edinburgh.

Cameron, S.A. (1984). Modelling solids in motions. Ph.D. thesis, Department of Artificial Intelligence, University of Edinburgh.

Canny, J.F. (1983). Finding edges and lines in images, MIT AI TR-720.

Chang, Y.L., and P. Liang. (1989). On recursive calibration of cameras for robot hand-eye systems. *Proc. of IEEE Int. Conf. on Robotics and Automation*, Vol.2, pp. 838-843.

Chongstitvatana, P., and A. Conkie. (1992a). Behaviour-based assembly experiments using vision sensing. In A. Colin, and E. Emil, (Eds.), *Advances in Machine Vision*. World Scientific Press, Singapore, pp. 329-342, also DAI RP 466.

Chongstitvatana, P., and A. Conkie. (1992b). Active mobile stereo vision for robotic assembly. *Proc. of 23rd Int. Sym. on Industrial Robots*, Barcelona, Spain, pp. 393-397, also DAI RP 590.

Churchland, P.M. (1986). Some reductive strategies in cognitive neurobiology. *Mind*, 95:279-309.

Conkie, A., and P. Chongstitvatana. (1990). An uncalibrated stereo visual servo system. *Proc. of the British Machine Vision Conference*, Oxford, pp. 277-280, also DAI RP 475.

Connell, H.J. (1989). A colony architecture for an artificial creature. Ph.D. thesis, MIT.

Curwen, R.M., A. Blake, and R. Cipolla. (1991). Parallel implementation of Lagrangian dynamics for real-time snakes. In P. Mawforth (Ed.), *Proc. of British Machine Vision Conf.*, Springer Verlag, pp. 29-35.

Dennett, D.C. (1988). Cognitive wheels: the frame problem of AI. In *The robot's dilemma*, Z.W. Pylyshyn, (Ed.), Ablex Publishing.

Domingo, J. (1991). Stereo part mating, M.Sc. thesis, Department of Artificial Intelligence, University of Edinburgh.

Donald, B.R. (1987). Error detection and recovery for robot motion planning with uncertainty, Ph.D. thesis, MIT, MIT AI Lab., AI-TR-982.

Donald, B.R. (1990). Planning multi step error detection and recovery strategies. *Int. Jour. of Robotics Research*, 9(1):3-60.

Donald, D.E. (1968). *The art of computer programming Vol.1: Fundamental algorithms*, Addison-Wesley.

Erdmann, M.A. (1984). On motion planning with uncertainty. MIT AI Lab., AI-TR 810.

Erdmann, M.A., and M.T. Mason. (1988). An exploration of sensorless manipulation. *IEEE Jour. of Robotics and Automation*, 4(4):369-379.

Fleming, A. (1987). Analysis of uncertainties and geometric tolerances in assem blies of parts, Ph.D. thesis, University of Edinburgh.

Flook, J.P., and B.O. McGonigle. (1977). Serial adaptation to conflicting prismatic rearrangement effects in monkey and man. *Perception*, 6:15-29.

Gardner, M. (1972). Pleasurable problems with polycubes. *Scientific American*, Sept., p. 176.

Giralt, G., R. Chatila, and M. Vaisset. (1984). An integrated navigation and motion control system for autonomous multisensor mobile robot. In M. Brady, and R. Paul (Eds.), *Robotics research* 1. MIT Press, Cambridge, MA, pp. 191-214.

Goethe, Johann Wolfgang von. (1890). *Faust: a tragedy*, translated in the original meters, with copious notes by Bayard Taylor, edited by G.T. Bettany, 3rd ed. London : Ward, Lock.

Gordon, S.J. (1986). Automated assembly using feature localization, Ph.D. thesis, MIT, MIT AI Lab. AI-TR-932.

Hardy, N.W., H.R. Nicholls, and J.J. Rowland. (1992). The design of sensing commands in the InFACT assembly machine. *Proc. of 23rd Int. Sym. on Industrial Robots*, Barcelona, Spain, pp. 47-52.

Harris, M. (1992). Vision guided part alignment with degraded data. *Proc. of IFAC Sym. on Intelligent Components and Instruments for Control Applications*, Malaga, Spain.

Hayes, G.M. (1989). A real time kinetic depth system. M.Sc. thesis, Department of Artificial Intelligence, University of Edinburgh.

Heikkila, T., T. Matsushita, and T. Sato. (1988). Planning of visual feedback with robot-sensor co-operation. *Proc. 1988 IEEE Inter. Workshop on Intelligent Robots and Systems*, Tokyo.

Horn, B.K., and K. Ikeuchi. (1983). Picking parts out of a bin. MIT AI Lab., AI memo 746.

Horswill, I.D., and R.A. Brooks. (1988). Situated vision in a dynamic world: chasing objects. *Proc. of 7th National Conf. on Artificial Intelligence*, Minneapolis.

Hutchinson, S.A., and A.C. Kak. (1990). Spar: A planner that satisfies operational and geometric goals in uncertain environments. *AI magazine*, 11(1):31-36.

Inoue, H. (1974). Force feedback in precise assembly tasks. MIT AI Lab., AI memo 308.

Inoue, H., and M. Inaba. (1984). Hand-eye coordination in rope handling. In Brady M., and R. Paul (Eds.), *Robotics Research 1*. MIT Press, pp. 163-174.

Inoue, H., and M. Inaba. (1985). Monitoring 3D pose of robot hand by real time vision. *2nd Int. Conf. of Advanced Robotics*, Tokyo.

Jennings, J., B.R. Donald, and D. Campbell. (1989). Towards experimental verification of an automatic compliant motion planner based on a geometric theory of error detection and recovery. *Proc. of IEEE Inter. Conf. on Robotics and Automation*, pp. 632-637.

Jones, V. (1974). Tracking: an approach to dynamic vision and hand-eye coordination. Ph.D. thesis, University of Illinois, Urbana Champaign.

Kaelbling, L.P. (1988). Goals as parallel program specifications. *Proc. of 7th National Conf. on Artificial Intelligence*, Minneapolis, pp. 60-65.

Kapur, J.N., P.K. Sahoo, and A.K.C. Wong. (1985). A new method for gray-level picture thresholding using the entropy of the histogram. *Computer Vision, Graphics and Image Processing*, 29:273-285.

Kass, M., A. Witkin, and D. Terzopoulos. (1987). Snakes: active contour models. *First Inter. Conf. on Computer Vision*, London, IEEE Computer Society Press, pp. 259-268.

Khatib, O. (1986). Real-time obstacle avoidance for manipulators and mobile robots. *Int. Jour. of Robotics Research*, 5(1):90-98.

Kittler, J., and J. Illingworth. (1985). Threshold selection based on a simple image statistic. *Computer Vision, Graphics and Image Processing*, 30:125-147.

Koutsou, A. (1986). Planning motion in contact to achieve parts mating. Ph.D. thesis, University of Edinburgh.

Lee, S.U., and S.Y. Chung. (1990). A comparative performance study of several global thresholding techniques for segmentation. *Computer Vision, Graphics and Image Processing*, 52:171-190.

Lougheed, R.M., and R.E. Sampson. (1988). 3-D imaging systems and high-speed processor for robot control. *Machine Vision and Applications*, 1:41-57.

Loughlin, C., (Ed.). (1992). *InFACT: project, concept, machine.* MCB University Press Limited, Bradford.

Lozano-Pérez, T. (1970). The design of a mechanical assembly system. MIT AI Lab., AI-TR 397.

Lozano-Pérez, T. (1981). Automatic planning of manipulator transfer movements. *IEEE Trans. on Systems, Man, and Cybernetics*, SMC-11(10):681-698.

Lozano-Pérez, T.(1982). Robot programming. MIT AI Lab., AI memo 698.

Lozano-Pérez, T. (1983). Spatial planning: a configuration space approach. *IEEE Trans. on Computers*, C-32(2):108-120.

Lozano-Pérez, T., and M.A Wesley. (1979). An algorithm for planning collision-free paths among polyhedral obstacles. *Communications of the ACM*, 22(10):560-570.

Lozano-Pérez, T., M.T. Mason, and R.H. Taylor. (1983). Automatic synthesis of fine-motion strategies for robots. MIT AI Lab., AI memo 759.

Lozano-Pérez, T., and R.A. Brooks. (1985). An approach to automatic robot programming. MIT AI Lab., AI memo 842.

Lozano-Pérez, T., J.L. Jones, E. Mazer, P.A. O'Donnell, W.E.L Grimson, P. Tournassoud, and A. Lanusse. (1987). Handey: a robot system that recognises, plans, and manipulates. *Proc.of IEEE Int. Conf. on Robotics and Automation*, Vol. 2, pp. 843-849.

Lyons, D. (1986) RS: a formal model of distributed computation for sensory based robot control. Ph.D. thesis, U. of Massachusetts at Amherst, COINS tech. report 86-43.

Malcolm, C.A. (1983). The outline corner filter. *Proc. of the 3rd Inter. Conf. on Robot Vision and Sensory Controls*, pp. 61 68, also DAI RP 212.

Malcolm, C.A. (1987). Planning and performing the robot assembly of SOMA cube constructions. M.Sc. thesis, Department of Artificial Intelligence, University of Edinburgh.

Malcolm, C.A., and A.P. Fothergill. (1986). Some architectural implications of the use of sensors, DAI RP 294.

Malcolm, C.A., and J. Howe. (1990). Behavioural modules: a new approach to robotic assembly. ACME Grant GRIE 68075, Extend Report, Department of Artificial Intelligence, University of Edinburgh.

Malcolm, C.A., and T. Smithers. (1988a). Programming assembly robots in terms of task achieving behavioural modules: first experimental results. *Proc. of the*

Inter. Advanced Robotics Program: Second Workshop on Manipulators, Sensors and Steps Towards Mobility, also DAI RP 410.

Malcolm, C.A., and T. Smithers.(1988b). Symbol grounding via a hybrid architecture in an autonomous assembly system. *Workshop on Knowledge Representation and Learning in an Autonomous Agent*, also DAI RP 420.

Malcolm, C.A., T. Smithers, and J. Hallam. (1989). An emerging paradigm in robot architecture. Department of Artificial Intelligence, University of Edinburgh, research paper 447.

Mason, M.T. (1984). Compliant motion. In M. Brady, J.M. Hollerbach, T.L. Johnson, T. Lozano-Pérez, and T.M. Mason, (Eds.), *Robot motion: planning and control.* MIT Press, Cambridge, MA, pp. 305-322.

Mason, M.T. (1985). The mechanics of manipulation. *Proc. IEEE Int. Conf. on Robotics and Automation*, St Louis, pp. 544-548.

Mason, M.T. (1986). Mechanics and planning of manipulator pushing operations. *Int. Jour. of Robotics Research*, 5(3):53-71.

McCarthy, J., and P.J. Hayes. (1969). Some philosophical problems from the standpoint of artificial intelligence. In B. Meltzer, and D. Michie, (Eds.), *Machine Intelligence* 4. Edinburgh University Press.

McCulloch, W.S. (1961). What is a number, that a man may know it, and a man, that he may know a number. *General Semantics Bulletin*, Nos. 26 and 27, pp. 7-18.

Mead, C. (1989). *Analog VLSI and neural systems.* Reading, MA, Addison Wesley.

Mead, C., and M. Mahowald. (1988). A silicon model of early visual processing, *Neural Networks* 1:91-97.

Minsky, M. (1987). *The society of mind.* London Heinemann.

Moravec, H.P. (1983). The Stanford cart and the CMU rover. *Proc. of IEEE* 71, July, pp. 872-884.

Nilsson, N.J. (Ed.) (1984). Shakey the robot. SRI AI center tech. note 323.

Otsu, N. (1979). A threshold selection method from gray-level histogram. *IEEE Trans. Sys. Man Cybernetics*, SMC-9:62-66.

Parnas, D.L. (1971). Information distribution aspects of design methodology. Department of Computer Science, Carnegie-Mellon University.

Parnas, D.L. (1972). On the criteria to be use in decomposing systems into modules. *Communication of the ACM*, 15(12):1053-1058.

Paul, R.P. (1981). *Robot manipulators: mathematics, programming, and control.* MIT Press, Cambridge, MA.

Pollard, S.B., T.P. Pridmore, J.E.W. Mayhew, and J.P. Frisby. (1989). Geometrical modelling from multiple stereo views. *Int. J. of Robotics Research*, 8(1):3 32.

Popplestone, R.J., A.P. Ambler, and I. Bellos. (1978). RAPT: a language for describing assemblies. *Industrial Robot*, 5(3):131-137.

Popplestone, R.J., A.P. Ambler, and I. Bellos. (1980). An interpreter for a language for describing assemblies. *Artificial Intelligence*, 14(1):79-107.

Rosenfeld, A., and A.C. Kak. (1982). *Digital picture processing, 2nd ed.*, London Academic Press.

Sahoo, P.K., S. Soltani, and A.K.C. Wong. (1988). A survey of thresholding techniques. *Computer Vision, Graphics and Image Processing*, 41:233-260.

Sakane, S., T. Sato, and M. Kakikura. (1987). Model-based planning of visual sensors using a hand eye action simulation system: HEAVEN. *Proc. of Int. Conf. Advance Robotics*, pp. 163-174.

Salmon, J.C. (1989). Implementation of a generalised patting behaviour for the SOMASS system. M.Sc. thesis, Department of Artificial Intelligence, University of Edinburgh.

Sanderson, A.C., and L.E. Weiss. (1983). Adaptive visual servo control of robots. In Pugh, A. (Ed.). *Robot Vision*. Springer-Verlag, pp. 107-116.

Sarachik, K.B. (1989). Characterising an indoor environment with a mobile robot and uncalibrated stereo. *Proc. of IEEE Int. Conf. on Robotics and Automation*, pp. 984-989.

Smith, S. (1992). A new class of corner finder. *Proc. of the British Machine Vision Conf.*, Leeds, Springer-Verlag, pp. 139-148.

Smithers, T., and C.A. Malcolm. (1989). Programming robotic assembly in terms of task-achieving behavioural modules. *Journal of Structured Learning*, 10:137-156, also DAI RP 417.

Steels, L. (1989). Cooperation between distributed agents through self-organisation. AI memo 89-5, VUB AI Lab., Brussels.

Taylor, R.H. (1976). The synthesis of manipulator control programs from task level specifications. AI Lab. Stanford, AIM-282.

Tsai, W.H. (1985). Moment preserving thresholding: a new approach. *Computer Vision, Graphics and Image Processing*, 29:377-393.

Tsai, R.Y. (1987). A versatile camera calibration technique for high accuracy 3D machine vision metrology using off-the-shelf TV cameras and lenses. *IEEE Jour. on Robotics and Automation*, RA-3:323-344.

Udupa, S.M. (1977). Collision detection and avoidance in computer controlled manipulators. Ph.D. thesis, California Institute of Technology.

Weiss, L.E. (1984). Dynamic visual servo control of robots: an adaptive, image based approach. Ph.D. thesis, Carnegie Mellon University.

Weiss, L.E., A. Sanderson, and C. Neuman. (1987). Dynamic sensor-based control of robots with visual feedback. *IEEE Jour. of Robotics and Automation*, RA-3(5):404-417.

Whitney, D.E., and J.L. Nevins. (1979). What is the remote centre compliance (RCC) and what can it do? *Proc. of 9th Int. Symp. on Industrial Robots*, Washington DC, pp. 135-152.

Wilson, M.S. (1992). Achieving reliability using behavioural modules in a robotic assembly system. Ph.D. thesis, University of Edinburgh.

Wolfe, D.F.H., and R.J. Richards. (1990). Eye to hand coordination for vision-guided robotic pick-and-place operations. *Advanced Manufacturing Engineering*, Vol. 2, July, pp. 123-132.

Wu, A.Y., T.H. Hong, and A. Rosenfeld. (1982). Threshold selection using quadtrees. *IEEE Trans. Pattern Anal. Mach. Intell.*, PAMI-4:90-94.

Yin, B. (1984). Combining vision verification with a high level robot programming language. Ph.D. thesis, University of Edinburgh.

Yin, B. (1987). Using vision data in an object-level robot language RAPT. *Int. Jour. of Robotics Research*, 6(1):43-58.

Yourdon, E., and L.L. Constantine. (1979). *Structured design: fundamentals of a discipline of computer program and systems design*, Prentice-Hall, NJ.

Zheng, J., Q. Chen, and S. Tsuji. (1991). Active camera guided manipulation. *Proc. of IEEE Int. Conf. on Robotics and Automation*, California, pp. 632-638.