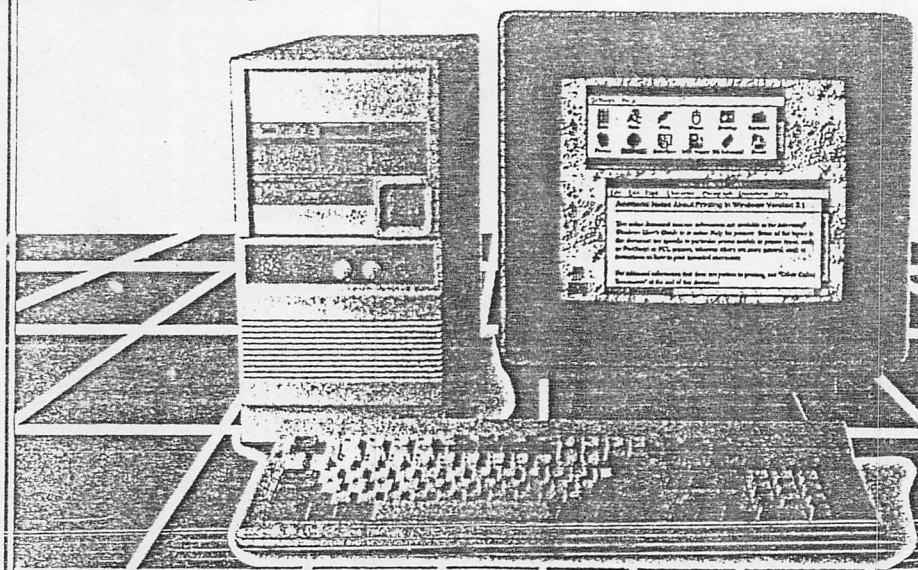
การประชุมวิชาการคอมพิวเตอร์

2537

National Computer Symposium 1994

Proceedings



โดยได้รับการสนับสนุนจาก

● ศูนย์เทคโนโลยีอิเล็กทรอนิกส์และคอมพิวเตอร์แห่งชาติ (NECTEC)

วิธีการโปรแกรมหุ่นยนต์แบบใหม่โดยใช้โมดูลพฤติกรรม

ประภาส จงสถิตย์วัฒนา
ภาควิชาวิศวกรรมคอมพิวเตอร์
จุฬาลงกรณ์มหาวิทยาลัย, กรุงเทพ 10300

บทคัดย่อ

รายงานฉบับนี้มุ่งปัญหาสองประการในการโปรแกรมหุ่นยนต์คือ หนึ่ง การโปรแกรมอุปกรณ์จริงนั้นยาก สอง ยิ่งยากมากขึ้นเมื่อต้องพบกับความไม่แน่นอนในโลกจริง โดยเสนอหลักการเพื่อแยกโครงสร้างระบบหุ่นยนต์ให้เป็นส่วนย่อย โดยที่โปรแกรมหุ่นยนต์สำหรับระบบนั้น ทำงานอย่างเชื่อถือได้ในภาวะที่มีความไม่แน่นอน ส่วนย่อยนี้เรียกว่า "โมดูลพฤติกรรม" ปัญหาของความไม่แน่นอนจะถูกกำจัดวงอยู่ภายในโมดูล มีตัวอย่างของโมดูลพฤติกรรมจากระบบหุ่นยนต์จริงหลายระบบเพื่อแสดงถึงวิธีโปรแกรมหุ่นยนต์แบบใหม่นี้

Behavioural Module : A new method to program robots

Prabhas Chongstitvatana
Department of Computer Engineering
Chulalongkorn University, Bangkok 10300, Thailand
email fengpjs@chulkn.ac.th

Abstract

This work addressed two problems in programming a robot. First, it is generally hard to program a real world device. Second, it is harder still to cope with the uncertainty in the real world. Some architectural principles are proposed which address the problem of decomposing robotic tasks into modular units such that a robot program works realiably in the presence of uncertainty. These modular units are called behavioural modules. The problem of uncertainty is dealt with by encapsulating sensing and variation-reducing strategies inside these modules. Examples are drawn from several existing robotic systems to illustrate the validity of this new programming method.

Introduction

We look at two broad categories of robotic systems : autonomous (mobile) and assembly robots. We ask the question "*why it is difficult to program a robot?*". The reason is that of the uncertainties in the real world and that the world is always

changing therefore it is difficult to predict all the necessary facts from the robot internal theory and representation.

Classical approach to robot programming

The past research has focus on programming a robot at a high level of description so called *task-level* such as *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. 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-Perez 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).

The classical approach to robot programming can be characterised 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.

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.

New decomposition

In 1985, MIT mobile robot group has advocated a different architecture for autonomous systems (Brooks, 1986a; 1986b). Instead of decomposing the architecture into a series of *functional units* such as perception, modelling and planning (fig. 1a) the architecture is decomposed into task-achieving modules, called *behaviours*. (fig.1b).

Brooks' argument is that the functional decomposition tends to make the system fragile because the success of a task depends on the success of every units. Each unit requires the correct output from the previous unit. Also sensors and actuators are not closely connected therefore it is difficult to achieve a sufficiently fast reaction time to cope with the change in the real world. The architecture exhibits *sense-think-act* behaviour.

In behavioural decomposition, every units (or modules) run in parallel. Each unit can have sensors and actuators tightly connected. Each unit pursues its own goal. All units have been arranged to work together to achieve the task. The benefits of this decomposition are: the performance of the system is not dependent on the performance of the weakest link because there are many parallel units; the robustness is achieved by using more than one behaviour for a given situation (therefore creates redundancy via parallel systems).

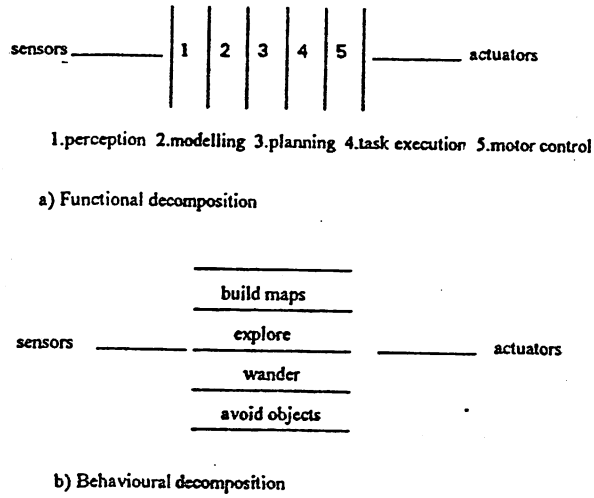


fig. 1 Two decompositions of a mobile robot

Chongstitvatana (1993) argues that these task-achieving units (called *behavioural modules*) facilitate the use of sensors to cope with uncertainties. By hiding the specifics of the use of sensors inside the modules behavioural modules abstract away some of the uncertainty management from the user of a robot. The decomposition of a task into behavioural modules should be based on the principles that:

1. there is no reliance on a central model of the world for combining 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.

By avoiding a centralised world model, the problem of sensing and updating the world model in the classical approach is avoided. By a tight coupling of sensing and action, a fast reactive system can be realised. By passing control via perception, the need for

an intermediate representation of the state of the world is lessen, in other words the world itself is used as its own representation. (the arguments about representation in this sense can be found in Brooks,1991.)

Characterisation of behavioural modules

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 an environment is devised by a human designer. There is no systematic method yet to design a behavioural module. The general guiding principles are proposed in the previous section. 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. For example, a module can make use of the knowledge of a motion that it has created to find the invariance in its perception. A block on the table can be distinguished from a block in the robot hand by moving the hand and observing the block that doesn't move (assuming only two blocks are in view and the camera is stationary).

Behavioural modules prefer communicating via the world.
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, whenit 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.

Examples of Behavioural Modules

In this section, three examples of the decomposition of an architecture into behavioural modules will be presented. These examples are extracted from the existing and working robotic systems. The first example is a mobile robot that can wander aimlessly around a room without hitting objects or people (Brooks, 1986a, fig 2a). The second example is a robot arm mounted on top of a mobile robot. This arm can reach for and grasp a soft drink can (Cornell, 1989, fig 2b). The third example is drawn from an automatic robotic assembly system that performs the assembly of Soma shapes (Malcolm, 1987, fig 2c).

The architecture of the first two systems is composed of layers of control in which a higher level can subsume a lower level. Brook calls this architecture *subsumption*. The third system is hierarchical where a module can be composed of several submodules.

The first example, the robot has a ring of 12 ultrasonic sonars as sensors. Every second these sonars give 12 depth measurements. The control is built with layers of increasing levels of competence. Each layer composed of behavioural modules which are implemented as finite state machines. The two lowest layers will be examined: AVOID OBJECTS and WANDER (fig. 3). The first layer makes sure that the robot does not come into contact with other objects. The robot will move away if something approach it and if in course of moving itself it is about to collide with an object it will halt. The next layer, WANDER, enables the robot to wander around by changing its heading every 10 seconds. It also plans ahead using a simple heuristic to avoid potential collision.

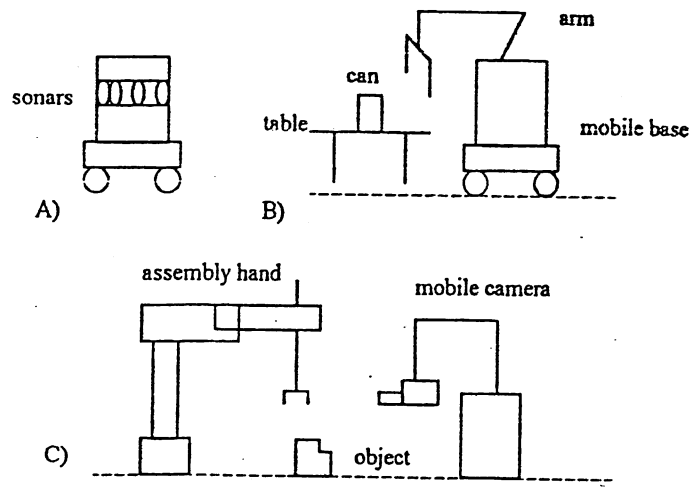


fig. 2 Three robotic systems in the examples

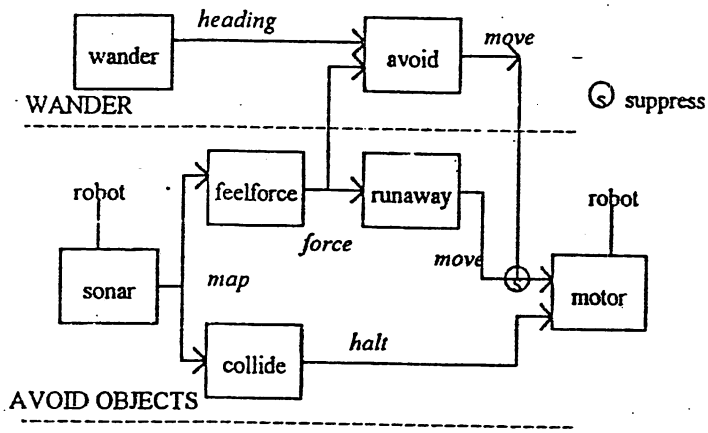


fig. 3 layer control of a mobile robot

In the AVOID OBJECTS layer, the *sonar* module takes sonar readings and produces a robot centred map of obstacles. The *collide* module monitors the sonar map and produces a halt signal if it detects objects ahead. The *feelforce* module generates a sum of the force vectors by considering each obstacles as a repulsive force. The output is passed through the *runaway* module which sends commands to the *motor* module to turn or to move if the force is significant.

In the WANDER layer, the *wander* module generates a random heading every 10 seconds. The *avoid* module treats that heading as an attractive force and sums it with the repulsive force from the *feelforce* module. The output suppresses the AVOID OBJECTS layer, forcing the robot to move according to the *wander* module.

Every modules run in parallel. These modules do not rely on a centralised world model therefore they can cope with a dynamic environment with ease. If a man walks up to the robot the *feelforce* and *collide* modules will move the robot. This is accomplished without requiring any prior representation of a man or any map

of the room. Although the sonar map is noisy the robot performs its task reliably because of the frequent sensing tends to average out the noise and the robot has the ability to cope with the change in the world.

The second example is the control of a robot arm that can grasp a can (Cornell, 1989, fig. 4). The robot composed of a mobile platform with a 2 degrees of freedom arm mounted on top. The mobile base has a laser ranger system that can detect a soft drink can. The base uses this sensor to move toward the target. The arm has several sensors mounted on the gripper. One sensor is the infrared near-range sensor. At the fingertips there are binary sensors that can detect the tips hitting a surface and a light beam between the fingers can detect if an object is presented.

The LOCAL layer is triggered by sensors detecting a can, the arm extends forward and down until the fingertips hit the surface of the table. The SKIM layer causes the hand to skim along the surface until it finds the can between the fingers then grasp it. The PARK layer retracts the arm to the home position.

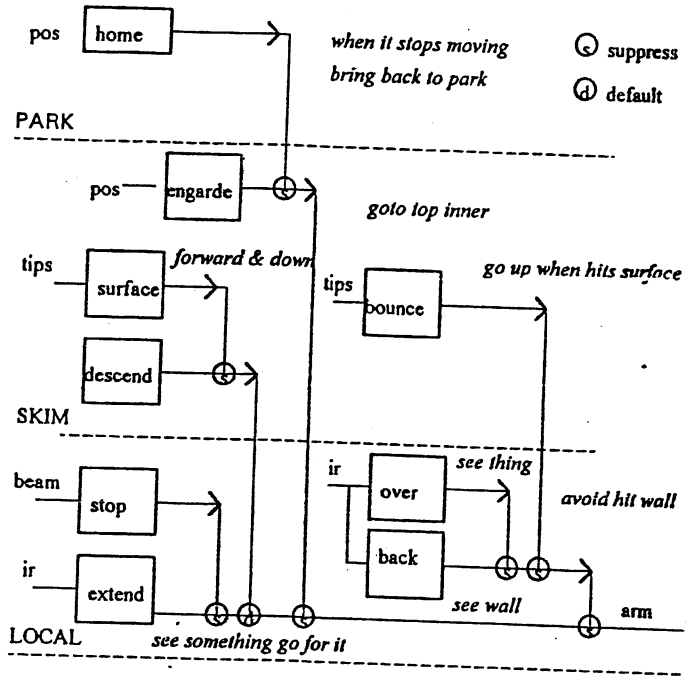


fig. 4 control of an arm

The lowest level is the LOCAL level which does interpreting the infrared near-range sensors. The *extend* module monitors the sensors and will extend the arm forward if it detects an object. *stop* causes the arm to stop when an object is between the fingertips. *back* prevents the arm hitting the wall by retracting the arm if IR sensors detected a big object. *over* lifts the arm up when it detects an object near the front of the hand. *extend* and *over* co-operate to move the hand toward the target and when lift the hand slightly when it is near the target so that the target will be between the fingers. The hand is stopped by *stop* and then the gripper will grasp the target.

The next level is the SKIM level. The LOCAL level extends the hand to reach a target. The SKIM level controls the height of the hand. It uses of the tips sensors to

detect the surface of a table where the target rested. *engarde* moves the arm from the home position, lifts the hand up and ready to extend the hand forward by the *extend* module in the LOCAL level. *descend* moves the arm straight down. *bounce* lifts the hand up slightly when the tips hit a surface. *surface*, *descend*, *bounce* and *extend* co-operate to "hop" the hand along a smooth surface.

The next level is the PARK level. The PARK level retracts the arm to the home position once the hand grasped the can. *home* monitors the movement of the arm when the arm stop moving it retracts the arm to the home position. It can be useful in other situation such as when the arm stuck in the some place *home* can move the arm back to a known position.

The last example is from a robotic 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. 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 (fig. 5 is a part of a plan). The execution system deals with variations and uncertainty in the locations and dimensions of the parts. To cope with uncertainty, the SOMASS system uses a self-calibrating vision to acquire parts (Chongstitvatana and Conkie, 1992).

```
; This is plan56 of the chair assembly using the soma4 part set
....
; ------- The placing of fork2 -------
   CALL reach(b3.get)
   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 reach(b1.get)
   CALL zget(b1.get:RZ(-90))
; - No regrasping required.
   CALL zput(b1.put:RZ(-90):RZ(0))
.END
```

fig. 5 A plan generated from an automatic planner

The execution of highest level behavioural modules is sequencial most of the time i.e. doing the task according to the plan step by step. But inside a module there are several submodules that run two or more robots simultaneously. The main behavioural modules in the system are: *reach*, *zget*, *zmanip*, and *zput*. *reach* moves the robot hand to acquire a part using a self-calibrating vision. *zget* then is used to pick up the part from that location. *zmanip* performs a regrasp operation to change the orientation of the part (when necessary) and *zput* puts down the part into the assembly.

In the SOMASS system, the behavioural modules form an hierarchical structure inwhich a module can be composed of several submodules (fig. 6).

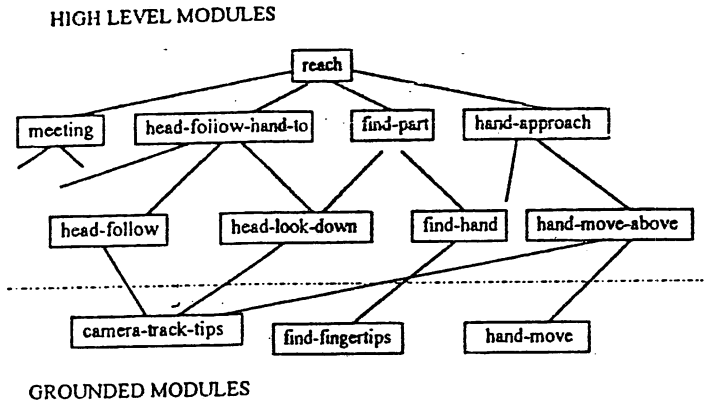


fig. 6 Hierarchical structure of behavioural modules

The *reach* module will be examined in more details. The *reach* module is composed of many submodules. *meeting* moves the hand and the mobile camera to a meeting place where the visual tracking can start. *head-follow-hand-to* moves the hand while keeping the mobile camera tracking it until the hand reaches a location near a part. *findpart* looks underneath the hand to find the part. *hand-approach* moves the hand until it is directly above the part using visual-servo method. At the termination of *reach*, the hand is in the position to pick up the part using the *zget* module.

The communication between these modules is through the world. For example, *head-follow-hand-to* uses the hand to lead the mobile camera without passing any location parameters to the vision system and the camera stops at the right place where the vision system can detect the part. This enables the vision system to work without having to know the location of the part as *a priori*. Most of the time the part itself is used to communicate between modules without using any common co-ordinate system. For example, *zget* module picks up the part where *reach* module left the hand at.

From these three examples it can be seen that the proposed principles to decompose a task into modular units work well in practice. By not using a centralised world model, the problem of keeping the model correct under the frequent changes of the real world is avoided. This is possible because a behavioural module keeps only a partial representation of the world that is suitable for its task.
For examples, the *collide* module uses a sonar map which is robot centred; the *extend*, *over* and *back* in the LOCAL level use different interpretation of the infrared sensors. The use of tight coupling between sensing and action can be seen clearly in all three s systems especially the method of visual servo and visual tracking in the SOMASS system. The communication through the world can be seen in the second example in the SKIM level. The *surface*, *descend* and *bounce* modules use the surface of the table to glide the hand along the surface. And in the SOMASS system, there is a use of the part itself to communicate between the *reach*, *zget*, *zmanip* and *zput* modules.

Conclusion

This work 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. Many examples from several existing systems 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 co-ordinate 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. The uncertainties are reduced by the use of sensors and other manipulation without putting the burden on to the user.

References

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

Brooks, R.A. (1986a). A robust layered control system for a mobile robot. *IEEE Jour. of Robotics and Automation*, RA-2(1):14-23.

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

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

Chongstitvatana, P., and A. Conkie. (1992). 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.

Chongstitvatana, P. (1993) Vision-based behavioural modules for robotics assemble systems, *Proc. 3rd. Int.Conf.on Advanced Science and Technology transfer to Thailand*, Bangkok, Thailand.

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

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.

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

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

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