Robotica (2001) volume 19, pp. 217–231. Printed in the United Kingdom © 2001 Cambridge University Press

Behaviour-based robot program invariance

Giovanni C. Pettinaro

Mechatronic Group, Dept. of Systems Engineering, ABB Corporate Research, Gideonsbergsgatan 2, S-721 78 Västerås, (Sweden).
E-mail: giovanni.pettinaro@secrc.abb.se

(Received in Final Form: August 5, 2000)

SUMMARY
Despite the numerous technical improvements, robot programming is still today a very tedious activity requiring considerable efforts for fine tuning programs written off-line. Moreover, such programs are usually not particularly portable from one robot to another, sometimes even when the robots are from the same maker.

The issue addressed in the work reported here is how to overcome these shortcomings. In this respect, the argument is to develop programs within the behaviour-based assembly paradigm which views tasks in terms of goal-achieving entities (behaviours). The focus on the description of a task more than its implementation is the key to generality and portability. As shown with the aid of two similar, but not identical, assembly benchmarks, behavioural programs are independent of any specificity of each assembly kit and can, indeed, be constructed using the very same code.

KEYWORDS: Behaviour-based robotics; Robot programming; Assembly robots.

1. INTRODUCTION

As it is well known, industrial robots are still today nothing more than dummy playback machines which repetitively perform the same sequence of actions time after time without any real *understanding* of what they are actually doing or even simply any *awareness* of their surroundings.[1] Thus, it is clear that robust and reliable programming of them is very difficult, because it requires substantial experience on the operator side in both the process to be automated and the programming means to be employed.

In order to simplify such an activity and to ease the human-machine interaction (HMI), the trend, since the beginning of the Eighties, has been to introduce gradually cheap computers on the factory floor. Such a trend allowed over the years a plethora of computer-like programming languages to emerge,[2–6] however, none has so far been good enough to dominate and replace the others. Indeed, they have all shared the same limiting characteristic of being machine-oriented.

Having acknowledged the problem, the research community has devoted, since the end of the Seventies, great efforts to simplify and ease the activity of robot programming. The result was a progressive investigation of task-level planning systems.[7–9] As opposed to the more conventional off-line programming ones, task-level systems aim at taking as input simply a description of the job to be carried out in terms of macroscopic tasks to be accomplished rather than in terms of the single atomic motions to be performed. It is then up to the *task planner* to generate as output the corresponding robot program in terms of the required manipulator motions necessary to carry out each macroscopic task, and hence the assigned job.

The advantage which these systems could potentially offer the operator is the possibility of reasoning about a task in terms of the logics with which it should be carried out rather than the details necessary to perform it. Such a characteristic would considerably raise the level of abstraction (HMI) with which to view industrial tasks; however, its implementation unveils all the complexities of human reliability and robustness which are still core problems to be solved in order to build successful autonomous *intelligent* systems. Thus, despite considerable research efforts, the prototypes proposed and built so far have up to now failed to be reliable and robust enough to convince industry, and they are still today no more than expensive academic toys.[10,11]

The reasons for such a failure can be traced back to the fact that many of the characteristics of dexterity and flexibility, which we commonly ascribe to humans, are far too often superficially oversimplified by introducing sensing equipment, often of one single kind, without actually tackling the real problem which is understanding what is perceived. In this regard, it is true that reliability and robustness with respect to the unknown are for humans or other animals inherently bound to their capability of sensing their surroundings and react accordingly; however, the basic principles underlying perception and awareness of the external world are still not well understood. Thus, although external sensors are a necessary condition in order to have artificial autonomous systems, electronic sensors, which are mere transducers of physical phenomena into electric pulses, cannot always provide a comprehensive picture of the surroundings, and hence guarantee a reliable and truthful response of the agent. Because of this reason, all commercial robot systems' manufacturers have always been very cautious in introducing them at all. Indeed, they have rather preferred to re-engineer the robot surroundings so as to eliminate any source of possible unpredictability. The result of such a policy is still visible today on modern industrial manipulators where the limited, if not totally absent, availability of on-board sensing capabilities requires a large program in order to perform even a simple task

https://doi.org/10.1017/S0263574700003015 Published online by Cambridge University Press

reliably and robustly within a restricted working world, and, if this is not enough, the success of the whole task itself depends strictly on the kind of strategy employed. Such an observation is particularly noticeable in industrial assembly where the optimal solution for one assembly may well be not easily extendible to other cases, such as different members of the same family.*

Since the real world is well known to be neither perfect nor ideal, it is plain that the use of sensors provides more general solutions in a more unstructured world. However, it is also true that automatic planners face a complexity explosion in the attempt to synthesise sensor-based programs capable of taking into account all the details concerning tolerances and strategies to be employed.

For a long time it was thought that the bottleneck was due to the algorithms used in the planners; thus a great deal of research efforts were devoted to the development of more efficient ones. However, the successful introduction of the *behaviour-based approach*[12] in the field of mobile robotics showed that, besides the classic *sense-plan-react* solution of the problem of robot navigation, there exists also an alternative one which completely rejects the use of any planning. Since it is not possible for assembly agents to get away from plans because of the inherently structured world they have to deal with, the ideas of the behavioural approach were translated into the *behaviour-based assembly architecture*[13] which may be seen as a hybrid. Basically, such a system requires still a planner to generate the robot program; however, this is defined in terms of task-achieving units called behaviours rather than in terms of manipulator motions.

The advantage gained with the use of such an approach is twofold: On one hand, it allows a plan to be totally insensitive to the uncertainty embedded in the real world, and, on the other, it provides considerable independence of the plan from the actual hardware onto which it will be executed as long as such a hardware (manipulator agent) is capable of performing all the behaviours required by the plan.

The paper herewith focuses on the latter of the two aforementioned advantages provided by the behaviour-based assembly approach. The discussion is developed as follows: Section 2 describes the related research works in this specific topic; Section 3 states the assumptions underlying the experimental research reported here; Sections 4 and 5 discuss the two assembly experiments carried out in order to show the invariance of the software written within the lines of the behaviour-based paradigm; Section 6 then discusses the results obtained from the experimental work and their implications; concluding remarks are given in Section 7.

2. RELATED RESEARCH

Since the end of the Seventies, when textual robot programming languages were first introduced,[14] all robot manipulator manufacturers, such as Unimation, Fanuc, Cincinnati Milacron, IBM, or ABB, have each introduced their own programming languages exploiting particular hardware features of their own models. By doing so, they not only fiercely battled against each other for increased market shares, but they also prevented any possible standardisation.[14,15]

Nowadays few of the initial pioneering companies actually survived; however, the trend did not change. Robot programs written in those languages are still today tailored to the specific task to be carried out by a particular manipulator. Such a limitation is especially evident in the field of industrial assembly where slightly different variations of the components can seriously affect the successful accomplishment of a program, even when the sequence of operations required remains unchanged (family of assemblies).

The idea proposed in this work aims at bridging the gap between standardization and generality within a specific task domain. In this regard, one of the first serious attempts tackling this problem was the research carried out by Malcolm and Smithers at the Department of Artificial Intelligence at the University of Edinburgh.[13] The system which they introduced and implemented (*behaviour-based assembly*) was greatly inspired by Brooks' *subsumption architecture*,[12] one of the first pioneering work in behaviour-based robotics. They maintained that Brooks' argument against planners did not actually suit industrial manipulators, which have to face and solve problems in a well defined and structured world. In fact, an industrial manipulator, as opposed to a mobile platform, requires to know in advance what it has to do and it can not afford a delay during the work cycle.

The paradigm introduced with the *behaviour-based assembly system* still assumed to have a planner for synthesising a robot program; however, such a plan was expressed in terms of reactive task achieving units (behaviours) instead of low-level robot motions. Such an assumption meant that a task could be thought of in classical terms and executed in behavioural ones. As a consequence, as long as different behavioural agents can understand and perform the aforementioned task-achieving units (behaviours), programs described in such terms could be totally portable from one agent to another regardless of the actual maker. As was later implicitly proposed by Pettinaro,[16] such a characteristic recalls very much the principle underlying Java™ applets running on different hardware.

The question which now spontaneously emerges from the implementation of a *behaviour-based assembly* architecture concerns the number of different behaviours required in order to have a running system. In this regard, an interesting early work by Nevins and Whitney[17] showed that the world of industrial assembly could be subdivided into a well defined and limited set of 12 different processes of which screw tightening and peg-in-hole insertion were the most common categories. They also showed that, in order to ease the design and installation of automation equipment, the most common assembly attitude was vertically down.

Nowadays assembly processes, as well as methods used, have both changed. However, despite the considerable increase of assembly automation in industry, the number of

* Two assemblies belong to the same family if they require the same sequence of operations in order to be realised.

https://doi.org/10.1017/S0263574700003015 Published online by Cambridge University Press

different operations has instead not changed significantly. Indeed, as shown by Balch,[18] their number is at most no bigger than 25, which, by considering the same classification used by Byrne[19] that covers 80% of all the assembly processes, it shrinks to a mere 10, as shown in a previous work of the author.[20]

The accomplishment of each of these 10 operations can be viewed by an external observer as the result (goal) achieved by a macroscopic *behaviour* performed by the executing robot agent. In this regard, I would like to point out that each behaviour may have different implementations (*behavioural modules*), though, as long as they all achieve the same goal, they would be considered as the same behaviour. Thus, programs expressed in terms of these modules would be not only portable but also robust with respect to future optimizations or developments of each module itself.

As discussed by the author,[21] one set of behavioural modules implementing the aforementioned 10 operations can be viewed as a basic universal set of behaviours with which to represent by composition any other behaviour within a restricted assembly world made of rigid prisms and cylinders. As shown in reference 16 such a result is sufficient to infer and find, within the constraints of such a world, a behaviour-based robot language whose programs are portable and of more general use.

The modules as intended by Malcolm and Smithers recall very much the principles underlying object oriented programming. In this regard, it is worth mentioning the work of Zieliński[22] which describes the introduction of a software library containing building blocks for the construction of multi-robot system controllers tailored to meet specific demands of a task at hand.

On a similar line of research, the ORCCAD project carried out by Simon *et al.*[23] presented a programming environment for robotic systems capable of gathering, on the one hand, control laws in continuous time at low-levels and, on the other, logical aspects in discrete time at higher levels. The argument advocated in it was that, based upon a formal definition of robotic actions, complex applications could be designed, verified, and generated incrementally.

As far as robot behavioural programming robustness with respect to unknown environment, I finally would like to mention the research pursued by Liu and Qin[24] which addressed the issue of self-organization of behaviours for a mobile agent aimed at coping and adapting to its dynamic surrounding environment.

3. RESEARCH ASSUMPTIONS

In order to discuss the invariance of behaviour-based assembly programs, the first assumption I make concerns the kind of parts I want to deal with. Since the manipulation of flexible objects is not so common in assembly, I restrict my discussion just to rigid bodies. Moreover, since I assume to work with a rather common robot manipulator with a simple two-fingered jaw gripper, I limit my assembly world just to prisms with parallel faces and to cylinders.

In order to carry out the experiments on program invariance, I chose a family of two similar benchmarks: One made of metal, specifically aluminium (small benchmark), and the other made of wood (big benchmark).

As shown in Figure 1, each member of this family is composed of four pieces (a **jig**, an L-shape, a **plate** and a **peg**) and the process consists simply in mating these four parts together. In this regard I would like to point out that such a task could be accomplished in different ways depending on the particular strategy adopted for the actual parts mating.

The reason why I selected this particular kind of assembly task is that even in its simplicity it requires one of the most common and tricky manufacturing operations in industry: *peg-in-hole*.[19]

The assumption underlying the two kits belonging to the same family of assemblies is that they are similar but not equal, in the sense that they are not a mere scale of each other. This means that the behavioural program accomplishing the final assembly should be able to deal with it at a logical level by coping with the occasional differences among the various family members without requiring any specific tailoring to each of them. In other words, the only concern of the program should be with the similarities among the various assembly procedures of the same family

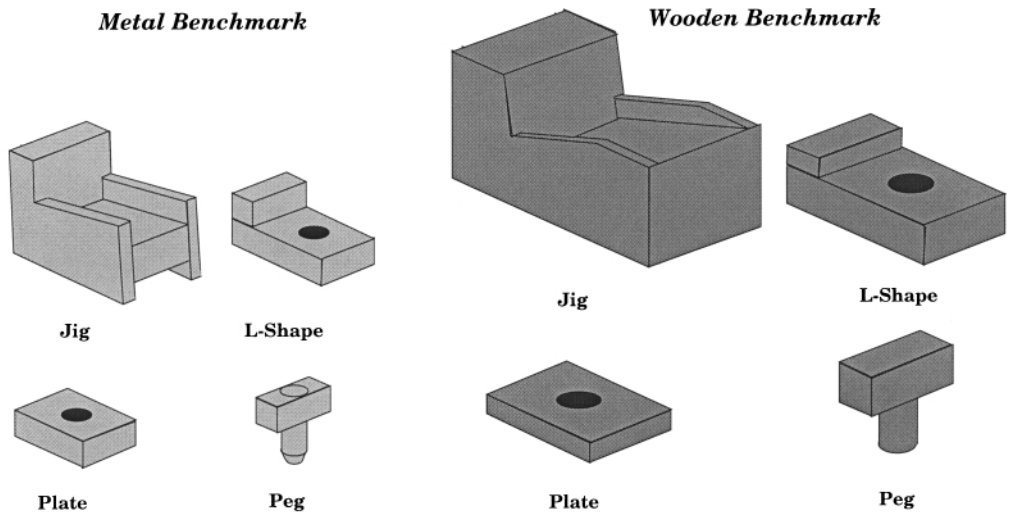


Fig. 1. Benchmark family parts.

https://doi.org/10.1017/S0263574700003015 Published online by Cambridge University Press

rather than with the actual assembly operations required for each particular set.

In order to achieve such a result, I undertake a stepwise approach by developing first the partial mating of **L-shape**, **plate** and **peg** (partial benchmark assembly), and then the full mating of the four parts (full benchmark assembly). The aim of such a *bottom up* policy is to show incrementally that the behavioural program for assembling one kit can be used for assembling a similar one belonging to the same family without any alteration to the code whatsoever.

Concerning the execution of the experiments themselves, I also make two further basic assumptions: First, the assembly jig, whose role in the case of the partial assembly is played by the **L-shape** part, is assumed to be fixed onto the working table, and second, the manipulator gripper employed as an end-effector is assumed to be equipped just with a simple differential touch sensor wrapped around each finger and capable of detecting only contact event signatures (Figure 2).

The first assumption is important for constraining the uncertainty due to eventual part slippages along the table, whereas the latter to show that the entire assembly task itself does not require very complex sensing capabilities in order to be accomplished.

The use of such a simple form of sensing enforces a further assumption concerning the initial home location of each assembly kit part. In fact, since my agent has no way to determine autonomously the locations on the working table where the assembly parts can be collected, I need to assume that such information is provided as input to the robot by means of taught world locations referred by appropriate labels.

As far as independence from the actual assembly family being handled is concerned, I assume my agent to be equipped with a general form of parallel jaw gripper (Figure 3) capable of grasping, as assumed at the beginning of this section, any prismatic object with parallel faces within its maximum gripping capability which is 103 mm.[25] Notice that such an end-effector (which I labelled Edinburgh gripper) can also grip cylinders and spheres, as long as the contact points are located at their diameter and, in the case of cylinders, also at the level of their centre of mass.

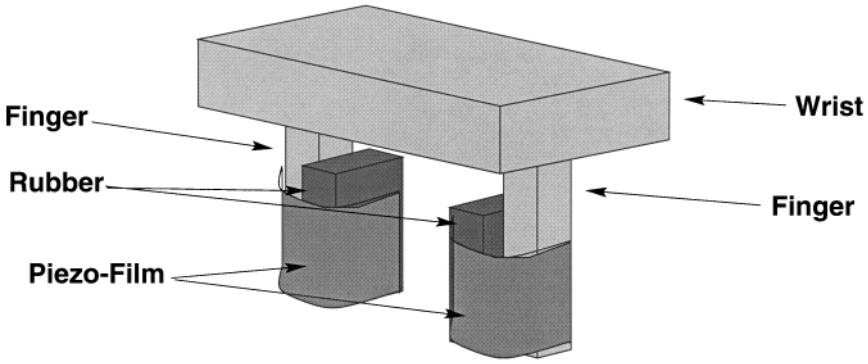


Fig. 2. Touch sensor.

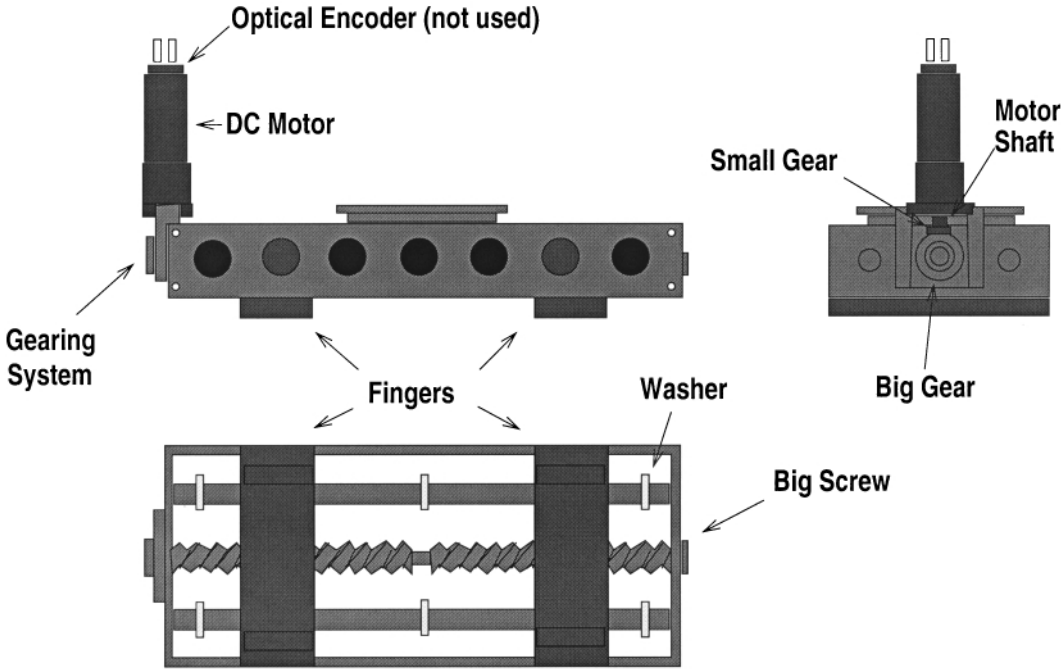


Fig. 3. Edinburgh gripper.

https://doi.org/10.1017/S0263574700003015 Published online by Cambridge University Press

Thanks to the above hardware assumption (general jaw gripper) and to its associated system of reading gaps between its fingers' inner faces, it is easy to make my robot agent realise autonomously which specific set a parts it holds actually belongs to. The only information which cannot be obtained by using solely such a gripper is the length of the peg shaft.

As discussed in one of my previous papers,[26] such information is crucial for establishing when a peg is fully inserted into a hole, or when, because of jamming or wedging, it is actually stuck. Notice that this information would not be required for the successful accomplishment of this kind of parts mating, if more powerful sensing capabilities such as vision would be employed. In such cases, in fact, the agent would be able to obtain it autonomously by simply processing its sensory vision data.

Nevertheless, even employing a very simple sensory system such as the one assumed, I can still make my agent measure the length of the peg shaft by itself. I can achieve this simply by first vertically griping the peg at its home location by its head with its shaft pointing downwards, and then by putting it down on the table: the length of the shaft would be computed as the difference between the vertical component of the location where the peg tip touches the table with respect to the robot frame system and the table height itself (Figure 4).

Notice finally that the technique assumes to find the position of the table surface by shifting the gripper +15 cm away from the position of the jig along the robot X-axis.

4. PARTIAL BENCHMARK ASSEMBLY

As pointed out in Section 3, the assembly experiments chosen as a testbed for showing the invariance of behavioural assembly programs are the benchmark assembly (Figure 1), which consists basically in placing first two hollow parts with coaxial holes (**L-shape** and **plate**) on top of a tilted jig and then in fixing them by inserting a peg through the common hole. However, in line with the choice of working with a *bottom up* approach, I temporarily disregard the jig and consider the assembly of just **L-shape, plate** and **peg** (partial benchmark assembly). Indeed, I consider two different sets of these parts: one made of metal (aluminium) and one made of wood (Figure 5).

By experimenting with this simplified version of the benchmark assembly, I aim at providing a gradual and clear understanding of the problems involved in this task as well as at identifying the behavioural components required to accomplish it. The final goal of this assembly experiment is to generalize opportunely these components so that they can be used to regardlessly assemble the metal benchmark kit as well as the wooden one.

As already described at the beginning of this section, the description of this assembly task is very trivial, since it simply consists in a sequence of one stacking operation (**plate** on top of **L-shape**) and one insertion (**peg** into a hole). Thus, taking into account that each part needs to be collected before being stacked, this assembly can be viewed as a typical pick-and-place problem. This means that what is needed is first of all one module for collecting parts (**pick-up**) and then one module for laying one part onto another (**stack**). A further module is then also required in order to perform the insertion operation (**peg-in-hole**), that is to insert the **peg** into the tandem hole passing through the other two parts (**plate** and **L-shape**).

As can be easily understood, the main result emerging from this task analysis consists in the three kinds of

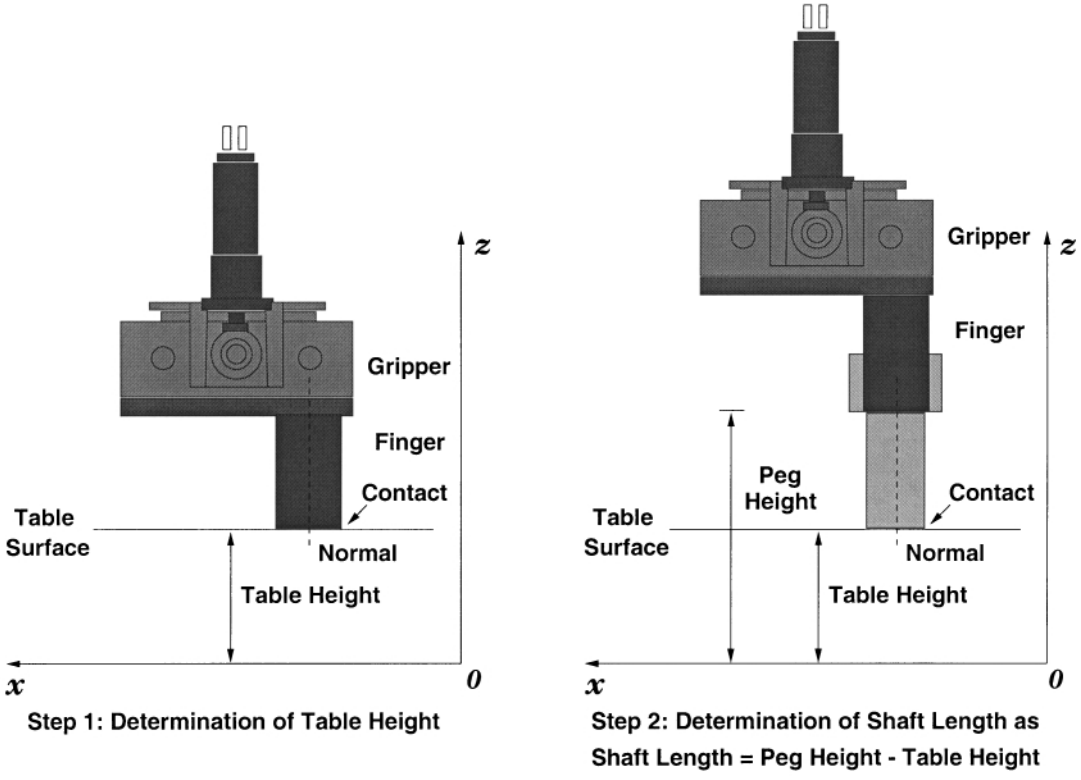


Fig. 4. Peg shaft length determination technique.

https://doi.org/10.1017/S0263574700003015 Published online by Cambridge University Press

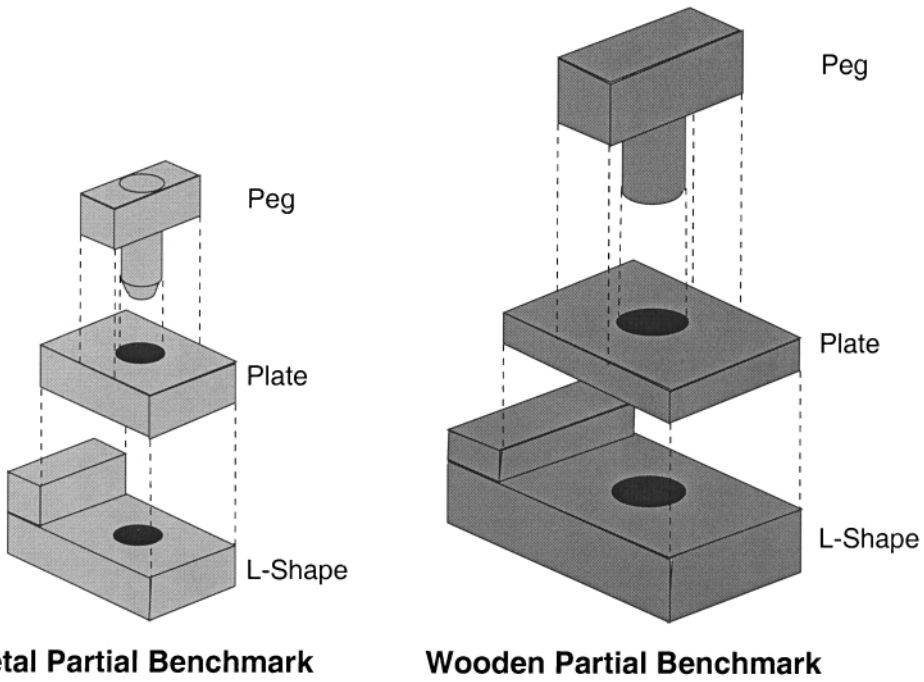


Fig. 5. Metal and wooden partial benchmarks.

components required to achieve it: **pick-up, stack,** and **peg-in-hole**. In this regard I would like to recall that a thorough analysis and discussion of **peg-in-hole** within the framework of the behaviour-based assembly paradigm can be found in a previous work of mine,[26] thus I dedicate Subsections 4.1 and 4.2 to the other two components and Subsection 4.3 and Subsection 4.4 to summarise the partial benchmark implementation and experimental work, respectively.

4.1 Pick-up module

As outlined above, this module is responsible for collecting an object lying at a certain location. Stable grasping is the key to a successful execution of the task, and much research has been carried out for both simple parallel jaw-grippers and multifingered end-effectors.[27] The former type is less versatile than the latter, because it can only manage just about 40% of the total number of possible manipulations which a human 5-fingered hand can do.[28] Incidentally, I would like to point out that a three-fingered gripper would reach at least 90% of them.

Earlier in Section 3, I assumed to employ a parallel jaw-gripper as my agent's end-effector. The implication of such an assumption is that what my agent can safely and stably grip consists basically in prismatic objects with parallel faces. However, I ought to observe that such a prehensile tool can allow my agent to grasp cylinders and spheres too, assuming of course to grip them at their diameters.*

Several studies have analyzed and modeled object contacts with hard fingers. In this regard, an interesting exception, considering my assumption of using a differential touch sensor wrapped around rubber padded fingers, is represented by the work of Akella and Cutkosky,[29] in which they propose an analytical modeling of stability conditions during contact between an object and soft fingertips filled with powders or plastic fluids.

As far as stability is concerned, grasps can be sorted out in two classes: spatial and contact grasps.[30] The former may be regarded as the tendency of the grasped object to return to an equilibrium location in space, whereas the latter as the tendency of the points of contact to return to an equilibrium position on the object's surface. Concerning contact itself, the problem of resolving its location and of determining forces and moments involved with it is addressed by Bicchi *et al.*[31] in which they propose a method for gathering contact information by measuring robot internal forces and torques.

There are specific positions which guarantee optimal gripping stability. In this respect, an algorithm for calculating non-graspable regions of a part is proposed by van Bruggen *et al.*[32] The method suggested in such an algorithm reduces the 3D grasp planning problem to a number of $2\frac{1}{2}$ D subproblems based on the mechanical properties of industrial grippers. A different approach is proposed by Caselli *et al.*[33] who managed to develop a tool for selecting feasible grasps of a robotic hand under various situations by using a rule-based expert system with a neural net based classifier. Another solution for planning robust grasping operations is suggested by Joukhandar *et al.*[34] in which they propose the use of a technique employing physical models, labelled as physically-based, which allows one to analyze the dynamic object/hand interactions.

Interesting because it involves grasp planning for a two-fingered gripper with parallel faces, is the work carried out by Taylor *et al.*[35] in which they resolve the problem of grasping unmodeled 3D objects by using visual information. In this regard, it is also worth mentioning the method proposed by Rodrigues *et al.*[36] in which they search for

* Cylinders can actually be grasped also by their parallel bases, if their height is within the range of the parallel jaw-gripper.

https://doi.org/10.1017/S0263574700003015 Published online by Cambridge University Press

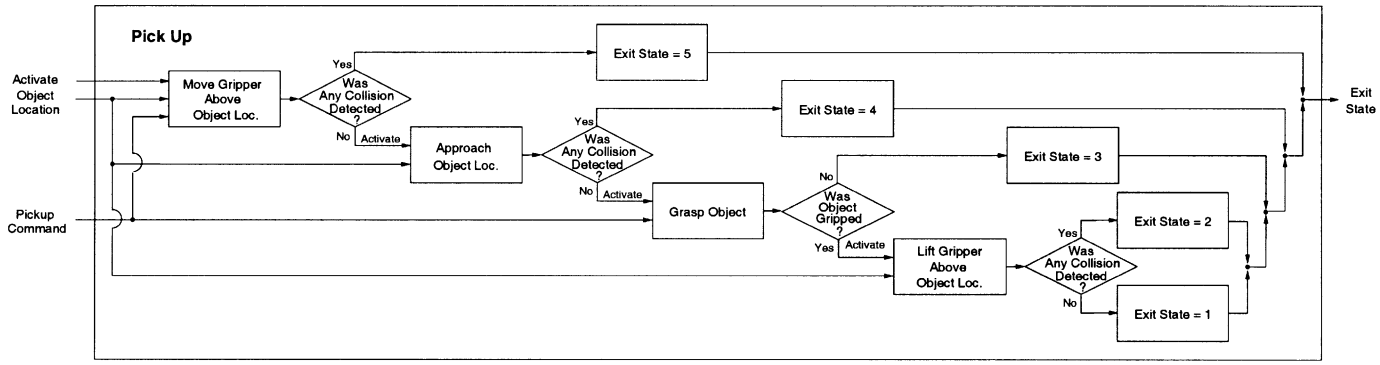


Fig. 6. Pick up diagram.

suitable gripping points on the outline of generic shapes gathered by a vision system.

As shown above, the use of powerful sensing capabilities such as vision and force-torque sensors allows an agent manipulator to plan how to grip an object without requiring particular knowledge about it.[37] In this respect, I have to point out that I initially assumed to employ a very simple form of a differential touch sensor which can only detect on/off event signatures (cf. Section 3). Because of such an assumption, the behavioural module performing **pick-up** requires to have a work-cell location to which to drive the agent manipulator as an input parameter. Incidentally, I would like to observe that some parts such as hollow cylinders may be actually grasped by closing the gripper jaw-fingers as well as by opening them. Thus, in order to specify which kind of gripping has to be performed, I need to feed a second input parameter which encodes both the gap between fingers to be used for the grasping and the kind of gripping to be performed which can be by opening or closing fingers. The speed to be used to approach the vicinity of the part to be collected (gross motion) can also be encoded in this second input parameter.

Summarizing, **pick-up** requires the following two parameters as input:

- a work-cell location, and
- a pickup command.

At this point, having discussed the module input/output requirements, I can now analyse its internal structure in more detail. In this regard, I ought to point out that in order to drive the agent manipulator above the target location where the object to be collected is located, I first of all require a gross guarded motion. Then, if no obstacles have obstructed the way there, I need a further guarded motion to approach the target location. If again no obstacles have obstructed this second move, then I can activate a gripping module which would close or open the fingers according to the gripping command given initially as input to the picking module. If the clamping is successful, then I finally need to lift the gripped part.

As a remark to the task description given above, I would like to observe that I basically require two guarded motions for positioning the gripper above the object and approaching it, a grasping module for the actual collection, and finally another guarded motion for departing from the current location.

The part collection analysis as outlined here is synthesised in graphic terms in the diagram reported in Figure 6.

Considering **pick-up** as structured above, several outcomes are possible (cf. its diagram given in Figure 6). First, let me point out that if all the operations involved are carried out successfully, then the result of the whole module may be viewed as a successful part collection. All the other module outcomes come from possible failure along the sequence of operations involved in the collection. A close examination reveals four more outcomes besides the one corresponding to a successful pick-up.

In conclusion, I could say that the possible exit states of the part collecting module (**pick-up**) are:

(i) Part successfully collected
(ii) Obstacle obstructs part lifting
(iii) Part to collect not found
(iv) Obstacle obstructs part during downward approach
(v) Obstacle encountered on the way to the location above part.

In order to test the implementation of **pick-up** as outlined above, I considered 3 prismatic parts (a **plate**, an **L-shape** part, and a **peg** with a prismatic head), and 2 cylindric ones* (a large ring, and hollow tube).[38, 39] Then, I set a series of 10 collections for each part and recorded every time a successful collection. As a remark, I would like to point out that my **pick-up** module does not perform any grasp planning and it assumes† that the gripping position given as input corresponds to a stable grasping. As regards the aforementioned cylinders, I have to stress that I tested both inner and outer part collection and recorded success each time. However, I ought to stress again the point that outer grasping of cylinders is possible only if performed at their diameter and at the level of their centre of mass.

4.2 Stack module

A stack module may be regarded as a particular form of fine motion, which implies a part to be laid down at a specified work-cell location. In this regard, since I am not employing

* These parts are taken from a kit of common electric torches whose assembly was analysed and reported in two previous works of mine.[38,39]

† This is in line with the assumption of having a hybrid planner which provides this information to the module.

https://doi.org/10.1017/S0263574700003015 Published online by Cambridge University Press

sophisticated sensors, I need to assume, as previously done with **pick-up**, to feed such knowledge as an input parameter. Incidentally, I would like to remark that a part may be released on top of another one either by opening the gripper fingers or by closing them. Thus, **stack** and **pick-up** likewise require a second input parameter (stacking command) encoding the speed to be used to reach the vicinity of the stacking site and the way in which a part has to be released,* and hence stacked, on top of another one once the site has been reached.

As previously done for **pick-up**, after having discussed the input requirements of my **stack** module, I can analyse its internal structure.

To start with, let us observe that my agent needs first to perform a gross guarded motion in order to locate the gripper above the stacking location. Then, if no event contact has been detected, it needs to perform a fine guarded motion in order to approach the targeted stack. At this stage, if no hindrance has obstructed the approaching, it needs to release the part held by the gripper according to the stacking command given as input. If the release is successful, the result of the entire module is a successful part stacking. Other possible outcomes which may derive from part stacking are all caused by possible failures of the operations involved in stacking. In this regard, looking back at the task description discussed above, I can summarise the exit states of **stack** as follows:

(i) Part successfully stacked
(ii) No part to be stacked
(iii) Obstacle obstructs stacking location approach
(iv) Obstacle encountered on the way to the location above stack.

As a remark to the discussion of part stacking given above, I would like to observe that what I need is basically a guarded motion for driving the manipulator above the stacking site, a further guarded motion for approaching the site itself, and finally an ungrasping module for releasing the part held by the gripper.

As previously showed with **pick-up**, the analysis of **stack** provided in the text above is synthesised in the implementation reported in graphic terms in the diagram of Figure 7.

In order to test such an implementation, I took the same 5 parts considered for **pick-up**: a **plate**, an **L-shape part**, a **peg** with a prismatic head, a large prismatic ring, and hollow tube. Then I set a series of 10 stackings at a specific location on the table for each part and recorded success each time. By opportunely fixing obstacles during part stacking, I also tested the other task outcomes mentioned above.

* Either by opening or closing finger.

4.3 Partial benchmark assembly implementation

As already discussed at the beginning of this section, the three main behavioural components allowing my agent to achieve the assembly of the partial benchmark are **pick-up**, **stack**, and **peg-in-hole**.

As far as the **peg-in-hole** is concerned, I would like to point out that a thorough analysis and discussion is reported in another work of mine.[26] Here, I would like simply to state again the possible outcomes which the execution of its implementation may lead to:

(i) Peg successfully mated.
(ii) Insertion aborted.
(iii) Obstacle detected during search for hole, and
(iv) Hole not found.

Now recalling the plan for accomplishing the partial benchmark assembly and recalling the other possible outcomes due to failure in picking up and stacking (**plate** not found, **stacking** failed, and **peg** not found), which I outlined at the beginning of this section, I can synthesise all the assembly task outcomes with 7 possible states (Table I).

As far as inputs are concerned, two parameters are required in order to perform the first part collection (gripping command and **plate** location), then two more to perform the stacking operation (stacking command and **L-shape** location), then again other two to perform the second collection (**peg**), and finally three more to carry out **peg-in-hole** (peg length, hole distance, and tilt angle). In this respect, I would like to point out that this simplified version of benchmark assembly is not tilted, thus the tilt angle parameter is set to 0°.

As previously done for **pick-up** and **stack**, I show in Figure 8 a flow chart-like graphic representation of the behavioural decomposition of the experimental testbed chosen for my investigations. Notice that such a description omits the measurement of the peg shaft because strictly speaking it is not actually part of the assembly process.

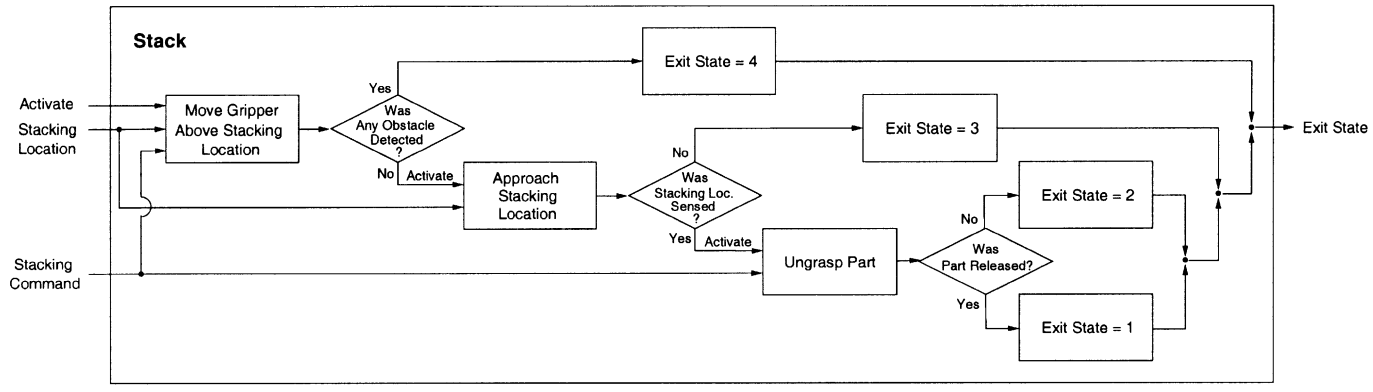


Fig. 7. Stack diagram.

https://doi.org/10.1017/S0263574700003015 Published online by Cambridge University Press

Table I. Partial benchmark outcomes.

Code	Partial Benchmark Exit States
1	**peg** inserted in hole and assembly successfully completed
2	**peg** insertion failed
3	Obstacle detected during search for hole
4	Hole not found
5	**peg** not found
6	**plate** stacking failed
7	**plate** not found

As a last remark, I would like to point out that the line labelled with *activate* and fed as input to the four modules in the aforementioned diagram, has to be interpreted as the flow of the robot manipulator control.

4.4 Partial benchmark assembly experiments

After having successfully presented and discussed the modules required for the partial benchmark assembly as well as its entire behavioural plan (cf. Figure 8), I now report the experiments I carried out concerning the invariance of such a plan. As a matter of fact, I would like to point out that my aim here is not only to show that two specific different kits can be assembled using the same behavioural program, but also to show that, using the modules I developed, if my agent is taught (programmed) to assemble one kit, it can automatically assemble any other similar one.

As done for **peg-in-hole**,[26] I use two different kinds of kits: One made of metal parts (aluminium kit) and one made of wooden ones (wooden kit). The two following paragraphs describes the experiments concerning each of them.

4.4.1 Partial metal kit. In order to accomplish the first assembly experiment I started by stably fixing the **L-shape** part at a known location on the table, then, after having recorded with a teach pendant both **plate** and **peg** home locations, I finally implemented the task as synthesised in Figure 8 by taking advantage of the implementations of **pick-up**, **stack**, and **peg-in-hole** discussed in Subsections 4.1, 4.2 and[26] respectively.

As observed earlier, the assembly of the partial benchmark requires 9 parameters as input. Two of them (**plate** and **peg** pick-up command descriptions) carried specific information on how to approach and grasp a part such as the speed to be used and the gripping gap between fingers. In this regard, I have to remark that providing gaps which are just enough to clamp these particular parts (metal **plate** and **peg**) would make the assembly program too bound to a specific benchmark kit. Thus, I decided to feed pick-up command descriptions encoding the maximum gap between fingers for both cases.

With regards to **plate** and **peg** home location, it is needless to point out that they are very important in order to accomplish successfully the assembly task, and that an inaccurate placement of them may lead to a failure. Thus, it is crucial to arrange carefully these parts at their home location each time a new assembly trial is performed.

With regards to the three input parameters to **peg-in-hole** (**peg** length, hole distance, and tilt angle), I would like to make a few remarks. First, I need to point out that my partial assembly is carried out on a table which is not tilted with respect to the X-Y plane of the robot frame system, thus the benchmark slant is assumed to be always 0°. Second, the **peg** length is assumed to be determined beforehand according to the technique outlined earlier. Third, the hole distance from which I assume to start the peg insertion is set to be 10 cm above the stacking position of the plate. In this regard, I would like to observe that this particular parameter is actually independent of specific part sizes.

Using the aforementioned premises, I performed a series of 50 experiments and recorded 48 successful assemblies (96% success rate). The two failures were due to **peg** insertion failures (4% failure rate). The first one was brought about by an inaccurate positioning of the **plate** at its home location which caused a misalignment between **plate** and **L-shape** part greater than $\frac{1}{2}$ mm, and the second one by **peg** wedging.

4.4.2. Partial Wooden Kit. The second experiment I report concerns the wooden kit. The aim I set here is to opportunely generalise the modules developed earlier and tested on the metal kit so that they could be re-applied with no modification to the wooden kit, which, I would like to stress, is similar to the metal one but not a mere scale of it.*

As previously done for the metal kit, I first fixed the wooden **L-shape** part at a specific work-cell location on the

* The two kits have different sizes and different proportions despite similar overall shape.

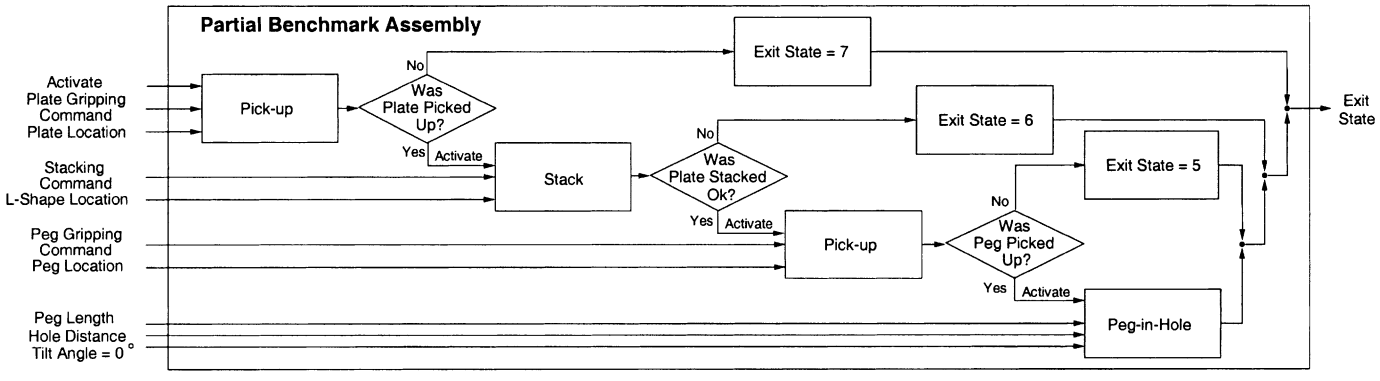


Fig. 8. Partial benchmark assembly diagram.

https://doi.org/10.1017/S0263574700003015 Published online by Cambridge University Press

table and then recorded the home location of both **plate** and **peg**. In order to test the applicability of the behavioural program developed for the metal kit, I set the same pickup and stacking commands used there as inputs to this assembly.

As regards **peg-in-hole** parameters, I kept the same hole distance as before and set the assembly slant to 0°. The length of the peg shaft was then determined as done with the metal kit by following the same technique (cf. Section 3).

Once again, using the premises above, I performed the assembly of the wooden kit by running the behavioural program developed for the metal benchmark containing a new peg shaft length, new **L-shape** part, **plate** and **peg** home locations, and the same other input parameters. As done with the metal kit, I ran a series of 50 assembly experiments and recorded 49 successful assemblies (98% success rate). The only failure observed was due to a **peg-in-hole** failure caused by peg wedging (2% failure rate).

As a final remark, I would like to point out that the assembly of the wooden kit was achieved basically by running the same program developed for the metal kit. This was possible because such a program expressed in behavioural terms is not only independent from specific part sizes, but also from part motions.* However, because of the limited sensing capabilities I initially assumed my agent to be equipped with, I still need to provide it with specific parts feature locations. Moreover, such a program still depends on the assumption that part sizes are not so large to become a hindrance when the initial part collecting movements are performed.

5. FULL BENCHMARK ASSEMBLY

As pointed out at the beginning of Section 3, the assembly task discussed here implies essentially the same steps as the partial benchmark analysed earlier. The only difference consists in carrying it out on top of a fixture (**jig**) whose upper surface is slightly slanted with respect to the table.† The presence of this fixture implies an extra step with respect to the partial benchmark. Thus, this time the entire assembly requires first the placement of the **L-shape** onto the **jig**, then the placement of the **plate** onto it, and finally the mating of the **peg** with the tandem hole passing through both **plate** and **L-shape** (Figure 9)..

The goal which I aim at achieving by considering this full scale benchmark assembly is twofold. First of all, to show that generalizations of programs expressed in behavioural terms are minimal, and then to show, once again, that if a behavioural agent is taught (programmed) to accomplish one assembly, then it can carry out without further tailoring any other similar one belonging to the same family (i.e. requiring the same steps).

In order to reach these two results, I first analyse the assumptions concerning this specific task (Subsection 5.1), then I outline my implementation of it (Subsections 5.2), and finally I summarise the experimental work I carried out (Subsection 5.3).

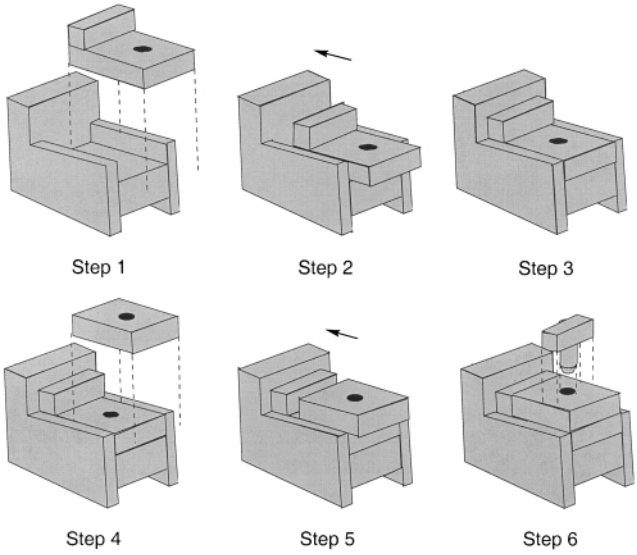


Fig. 9. Full benchmark assembly

5.1 Full benchmark assumption

The first observation I would like to draw concerns the parts composing the assembly. I would like, in fact, to point out that, since this task requires to be performed on a surface slanted with respect to the table, both **L-shape** and **plate** need to be positioned parallel to the slope in order to be stacked on the **jig**.

With regard to sensing, I still expect my agent to rely on the same simple differential touch sensor employed in the case of the simplified version of this assembly.

Concerning testbeds, I consider also in this case two types of kits: One made of metal (aluminium) and one made of wood. Each of them is composed of an **L-shape**, a **plate**, and a **peg** (cf. the partial benchmark) with the addition of a suitable **jig** made of the same material (cf. Figure 1). In this regard, I would like to draw attention to the fact that the full benchmark, as opposed to the partial one, uses the jig as a fixture. Thus, in order to avoid part slippages, this time I need to assume the **jig** to be firmly fixed at a specific location within the work-cell and not the **L-shape** (cf. Section 3).

As far as the execution of the full assembly is concerned, I would like to observe that, since this task involves a **peg-in-hole** operation, my agent needs to cope with two specific kit-dependent information details: specifically, the **peg** shaft length and the tilt angle between table and the slanted upper surface of the **jig**. As easily gathered by a previously referenced work of mine,[26] the former is a crucial piece of information which is used in my **peg-in-hole** implementation in order to allow my agent to realise when the peg is fully mated. As shown later in this section, the latter is instead important for part stacking.

Concerning the determination of the **peg** shaft length, the technique discussed in Section 3 provides my agent with a certain degree of autonomy by allowing it to measure such a length by itself. The idea behind such a technique was

* Notice that this is achieved by not binding motions to specific parts: they are picked up and stacked always following the same path.

† The table is assumed to be parallel to the X-Y plane of the robot frame system.

https://doi.org/10.1017/S0263574700003015 Published online by Cambridge University Press

simply to calculate the shaft length with respect to the height of the table referenced to the robot X-Y plane. In order to do so, the agent was required first to touch the table with one of its fingers and then to record the contact position. After that, it was required to collect the **peg**, to make its tip touch the table, and finally to record this second contact position. The difference between the heights of the two aforementioned contact positions was then assumed to be the length of the **peg** shaft.*

As far as tilt angle is concerned, I employed the same underlying paradigm which consists in letting the executing agent to find out by itself whatever it needs in order to carry out its task. In this specific case, since **peg-in-hole** needs to be opportunely provided also with the aforementioned tilt angle, I let my agent autonomously determine it so that to achieve independence from any specific kit. Before carrying on, I would like to point out that I assume in the rest of this discussion the **jig** to be arranged in such a way that the height of its slope profile decreases when moving along the robot + X-axis.

The algorithm for the angle measurement consists of a sequence of three equidistant close touches of one of the gripper fingertips with the upper face of the **jig** along the +X-axis. Such a sequence of touches† is assumed to start $\frac{1}{2}$ cm upward along the slope with respect to the prestored location of the jig (Figure 10).

More technically speaking, the measurement requires the manipulator gripper to approach the **jig** so that one of its fingers touches the **jig**'s upper face at point P_1. Once the contact takes place, the position held in that moment by the agent is extracted and the height with respect of the X-Y plane of the robot frame system (z_{P_1}) is recorded. Repeating the procedure further down the slope (P_2), a second height (z_{P_2}) is recorded. Then again, moving further down the slope for an equivalent space (P_3), a final third height (z_{P_3}) is extracted and recorded. By using these three heights, the agent is then able to compute the angle $\mathbf{u}_{12}$ between the slope $\overline{P_1P_3}$ and the table by means of the following trigonometric identity:

$$\mathbf{u}_{12} = \arctan\left|\frac{\zeta_{\Pi_+} - \zeta_{\Pi_-}}{\chi_{\Pi_+} - \chi_{\Pi_\times}}\right| \quad \nabla \quad \arctan\left|\frac{\mathbf{D}\eta_{+-}}{\mathbf{D}\chi}\right| \tag{1}$$

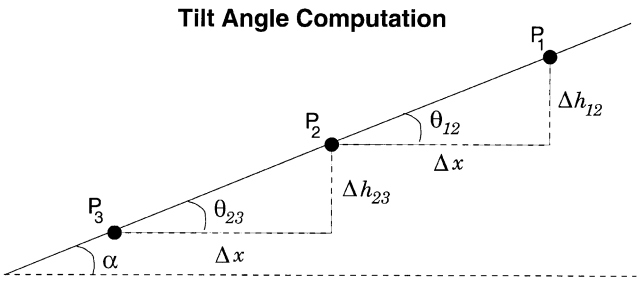


Fig. 10. Procedure to compute the tilt angle.

* Notice that in this technique I assume the **peg** to be gripped so that both fingertips and bottom face of the **peg** head lie on the same plane parallel to the X-Y plane with respect to the gripper (cf. Figure 4).

† The increment between each couple of touches is assumed to be fixed beforehand.

Similarly, the angle $\mathbf{u}_{23}$ between slope $\overline{P_2P_3}$ and table, and the angle $\mathbf{u}_{13}$ between the slope $\overline{P_1P_2}$ and the table can be calculated as:

$$\mathbf{u}_{23} = \arctan\left|\frac{\zeta_{\Pi_-} - \zeta_{\Pi_\times}}{\chi_{\Pi_-} - \chi_{\Pi_-}}\right| \quad \nabla \quad \arctan\left|\frac{\mathbf{D}\eta_{-\times}}{\mathbf{D}\chi}\right| \tag{2}$$

$$\mathbf{u}_{13} = \arctan\left|\frac{\zeta_{\Pi_+} - \zeta_{\Pi_\times}}{\chi_{\Pi_+} - \chi_{\Pi_\times}}\right| \quad \nabla \quad \arctan\left|\frac{\mathbf{D}\eta_{+\times}}{-\mathbf{D}\chi}\right| \tag{3}$$

By averaging on the angles $\mathbf{u}_{12}$, $\mathbf{u}_{23}$ and $\mathbf{u}_{13}$, the agent can finally estimate the tilt angle $\mathbf{a}$ of the slanted **jig** as the mean value:

$$\mathbf{a} = \frac{\mathbf{u}_{12} + \mathbf{u}_{23} + \mathbf{u}_{13}}{3} \tag{4}$$

In this regard, I would like to remark that the third angle $\mathbf{u}_{13}$ is introduced in order to reduce possible uncertainties due to the crudeness of the sensor used for measurement.

5.2 Full Benchmark Implementation

According to the behaviour-based assembly paradigm, any assembly task may be viewed as a single grand behaviour which can be coded with a program module expressed in terms of simpler ones and whose execution leads to the accomplishment of the task. Thus, as previously done for the partial benchmark assembly, the first step is decomposing this task into such terms.

Recalling the description given for the partial assembly in Section 4, I can even in this case immediately identify two modules: one for collecting the **L-shape**, and one for laying it down on top of the **jig**. If both of them are carried out without making the agent detect any obstacle, I need to re-instantiate them for the **plate**. If, once again, they are successfully performed, then the **plate** would lie down at the end of the process stacked on top of the **L-shape**. In which case it would be first required another collecting module to pick up the **peg** from its home location, and then a **peg-in-hole** one to insert it inside the tandem hole passing through both **plate** and **L-shape**.

As pointed out for the partial version of this assembly (Section 4), if all the operations described above are successful, then the outcome of the entire process is "assembly successfully accomplished". All other possible outcomes derive from failures of the operations during the process. In this regard, I would like to recall that **peg-in-hole**, in particular, may terminate in four possible states (cf. Subsection 4.3): **peg** successfully mated, insertion aborted, obstacle detected during search for hole, and hole not found. The first one may be viewed as the outcome of successful assembly, whereas the others can be regarded as three more outcomes of the full benchmark assembly task. With regards to the failures of the other modules composing the assembly process, I may have **L-shape** part not found, **L-shape** stacking failed, **plate** not found, **plate** stacking failed, and **peg** not found. Thus, summarising the observations above, I

https://doi.org/10.1017/S0263574700003015 Published online by Cambridge University Press

Table II. Full benchmark outcomes.

Code	Full Benchmark Exit States
1	**peg** inserted in hole and assembly successfully completed
2	**peg** insertion failed
3	Obstacle detected during search for hole
4	Hole not found
5	**peg** not found
6	**plate** stacking failed
7	**plate** not found
8	**L-shape** part stacking failed
9	**L-shape** part not found

can say that the full benchmark assembly may terminate in one of 9 possible states (Table II).

Regarding inputs, I require two parameters to perform the first collection module (an **L-shape** pickup command and the **L-shape** location), two more to perform the first stacking (an **L-shape** stacking command and the **jig** location), other two to perform the second part collection (**plate**), another one to perform the second stacking (a **plate** stacking command), two more to perform the third part collection (**peg**), and finally three to perform **peg-in-hole** (**peg** length, hole distance, and tilt angle).

Extending the diagram of the partial benchmark assembly (cf. Figure 8), I derive the more general behavioural decomposition illustrated in Figure 11. Notice that such a representation omits the measurements of **peg** shaft and **jig** tilt angle because strictly speaking they are not actually part of the assembly process. They are assumed to be carried out following the two techniques discussed earlier.

As a last remark, I would like also in the case of the full benchmark to point out that the lines labelled *activate*, given as inputs to the six modules of the decomposition in the diagram above, have to be interpreted as the flow of the robot manipulator control.

5.3 Full benchmark assembly experiments

Having analysed the assumptions set in order to carry out the full benchmark assembly as well as having described the implementation defined, I need now to report the experiments run in order to show the invariance of the behavioural program accomplishing the assembly of one set of parts with respect to any other kit of the same family. As previously done for the partial benchmark assembly, my first concern was to set the work-cell and develop the assembly program implementing the decomposition shown in Figure 11 which, incidentally, makes use of the same modules developed earlier for the partial assembly.

The goal which I aimed at achieving, even in this case, was not simply to show that a metal kit, as well as a similar wooden one, could be assembled by the same program, but also to show that a task such as the full benchmark assembly, which is close to the partial one examined earlier, might be accomplished by an opportune simple generalization of the very same behavioural program developed for the task akin to the present one (partial assembly).

In order to do so, I set also in this case two different sorts of experiments: Assembly of the metal kit and Assembly of the wooden one. The following two subsections are dedicated to describe and report the experimental results involving each set of parts.

5.3.1 Full metal benchmark assembly. The first assembly experiment I describe involves the metal kit. As already done in the case of the partial benchmark, I started first by firmly fixing the **jig** at a specific location within the work-cell. Subsequently, by means of the robot teach pendant, I recorded the home locations of the various parts of the kit (**L-shape, plate**, and **peg**). Once the work-cell was set and

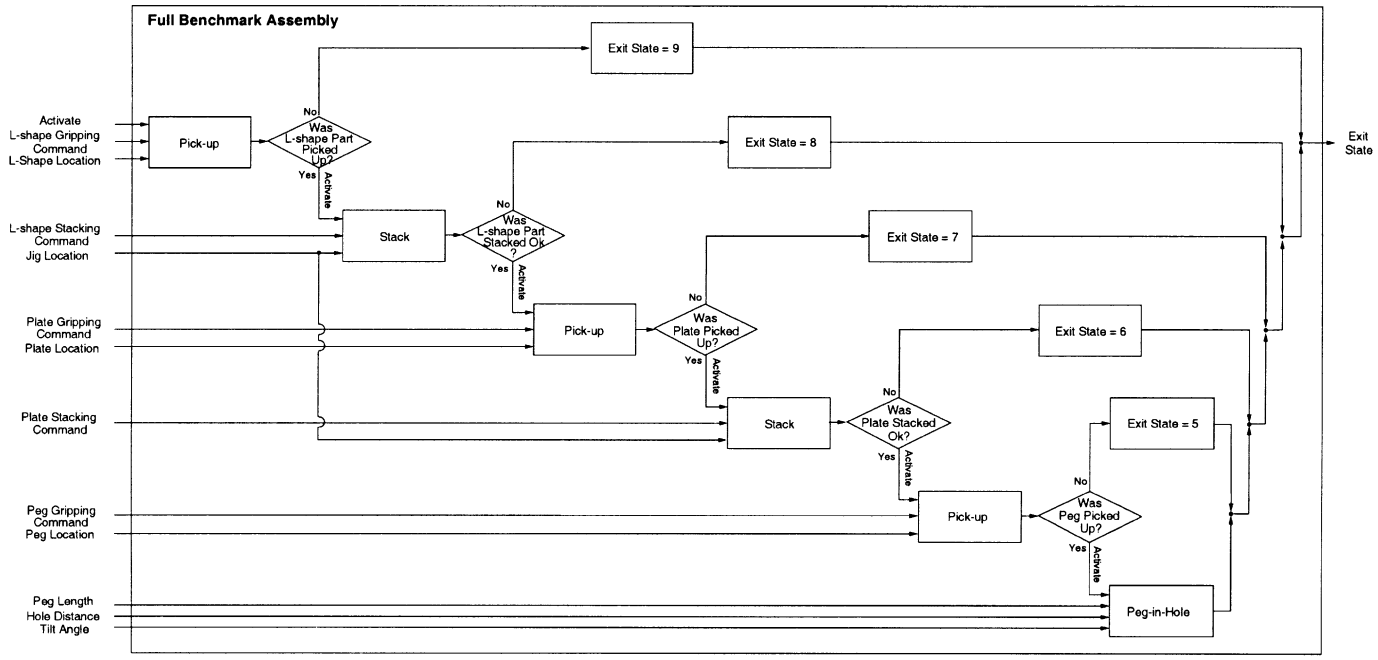


Fig. 11. Full benchmark assembly diagram.

https://doi.org/10.1017/S0263574700003015 Published online by Cambridge University Press

the parts' home locations defined, I implemented the program as outlined in the diagram illustrated in Figure 11. Incidentally, I would like to stress again the point that I used the same **pick-up, stack**, and **peg-in-hole** modules that had been previously developed for the partial benchmark.

At this point, following the same experimental pattern adopted for the partial benchmark, I achieved the aforementioned program implementation with opportune input parameters. In this regard, I would like to recall (cf. Subsection 5.2) that the assembly of the full benchmark requires 12 parameters as input, three of which (**L-shape, plate**, and **peg pickup** commands) are quite specific to the particular set being manipulated. They carry, as a matter of fact, particular information on how to approach and grasp each part, such as approaching speed and gripping gap between fingers. Thus, in order to be independent from any particular set, also in this case I fed pickup commands encoding the maximum gap between fingers.

With regards to the two stacking commands, they are not bound to a specific kit because they carry information merely about how to stack a part such as the release mode of a part on the stacking site.

As far as **L-shape**, **plate** and **peg** home locations are concerned, I would like to remark that they are even in this case very important for a successful accomplishment of the task. The inaccurate placement of any of them may cause the agent to fail, since, due to the assumptions set, it has no other way to perceive their locations. Thus, it is crucial to rearrange each part carefully at its home location every time a new experiment is restarted.

Concerning the three **peg-in-hole** parameters (**peg length**, hole distance, and tilt angle), I would like to make a few observations. Since I wanted my agent to be both autonomous and independent from particular kits, I needed to compute two of them (**peg** shaft length and **jig** tilt angle) before the whole assembly was started. I achieved this by following the two techniques discussed in Section 3 and Subsection 5.1, respectively. The two measurements resulted in 24.1 mm for the peg shaft (real length 23.5 mm) and 14° for the tilt angle (real slant 15°). In this case, the overestimation of the shaft length was due to an inaccurate gripping of the **peg** which had its head bottom face and fingertips plane not lying on the same plane. The third parameter (hole distance) was set, as with the partial benchmark, to be 10 cm with respect to the stacking position of the **plate**. Incidentally, I would like to point out that this particular parameter is independent of specific part sizes.

Constructing the assembly program with the parameter settings presented above, I performed a series of 50 experiments and recorded 48 successful assemblies (96% success rate) and two failures (4% failure rate). The first of these failures was brought about by a slight misalignment between **peg** and insertion axes caused by a few slippages of the **peg** head between the gripper fingers during the search for hole stage, whereas the second one was due to peg wedging.

After having glued some emery paper to the outer skin of the fingers, I repeated the same series of experiments and this time I recorded 49 successful assemblies out of 50 (98% success rate). Once again, the only failure was due to **peg** wedging.

5.3.2. Full Wooden Benchmark Assembly. The second experiment described concerns the wooden kit for which I performed the same preliminary work-cell setups as reported above for the metal kit. Thus, after having fixed the **jig** and recorded its location together with **L-shape**, **plate**, and **peg** home locations, I used the same program developed for the metal kit opportunely instantiated with the parameters specific to the wooden kit. In this respect, in order to test the generality of the program, I fed the same pickup and stacking commands and the new home locations.

With regards to **peg-in-hole** input parameters, I determined **peg** shaft length and **jig** tilt angle, as described earlier, using the same two techniques outlined in Subsection 5.1. The two relative measurements resulted to be 31 mm for the **peg** shaft (real length 30.8 mm) and 11.5° for the tilt angle (real slant 12°). As far as the hole distance parameter is concerned, I instead kept it as before (10 cm).

Constructing then the same generalised assembly program used for the metal kit with the new parameter settings presented above, I performed a series of 50 experiments recording 49 successful assemblies (98% success rate) and one failure (2% failure rate) due to a **peg** wedging.

As a last remark, I would like to observe that the last series of experiments mentioned above was carried out by using emery papers wrapped around the outer skin of the gripper fingers. Such a trick revealed to be very effective at reducing the occasional part slippages which could always occur during the execution of the behaviours requiring parts contacts.

6. DISCUSSION OF THE RESULTS

At this point, I would like to summarise the results emerging out of the experimentation with the two assemblies considered in this work, i.e. the partial benchmark and the full one. It is interesting to observe, in this regard, that the development of the assembly program was carried out focusing more on the description of the task than on the specificities of the particular sets of parts manipulated. Thus, in line with the *behaviour-based assembly* paradigm, each assembly was defined in terms of sensor-based stacking and mating procedures (behaviours) operating on the assembly component parts (**jig, L-shape**, **plate** and **peg**) at an object level. In other words, the description of, for instance, the full benchmark assembly was given in terms of first placing **L-shape** on top of the **jig**, then laying the **plate** on top of the **L-shape**, and finally mating the **peg** into the tandem hole passing through both **L-shape** and **plate**.

As pointed out for both of the assembly tasks considered, the goal which I aimed at achieving was twofold: first, showing that each assembly task could be expressed in terms of general-purpose task-achieving units (behaviours), and second, showing that the behavioural program so defined could have been used by my behavioural agent for assembling two similar, but not scaled, versions of the same problem* simply by opportunely instantiating the same program.

* Two benchmarks belonging to the same assembly family.

https://doi.org/10.1017/S0263574700003015 Published online by Cambridge University Press

As easily gathered by the experiments I analysed and reported in Sections 4 and 5, both partial and full benchmark were defined in behavioural terms. Indeed, the modules defined and employed in the former were actually re-employed in the latter exactly as they were. Moreover, the generalization from the partial benchmark to its full version was achieved simply by adding the extra stacking behaviour (**L-shape** on top of **jig**) to the initial sequence of steps.

As far as generality of the behavioural program developed is concerned, I would like to make a few comments: First of all, I would like to stress the point that the two kinds of assembly kits (metallic and wooden ones), although similar, were not a mere copy in scale of each other. They held, indeed, some specificity, such as length of the **peg** shaft and tilt angle of the **jig** upper surface, which had to be fed in advance to the program. However, since I wanted to keep my agent as independent as possible from this sort of kit dependent instantiations, I had to devise a way to let it find out the required information by itself. Bearing this in mind, I developed two procedures, which could be viewed as measuring behaviours, enabling my agent to estimate the aforementioned specificities in complete autonomy.

The experiments performed on the two kinds of kits (metallic and wooden ones) described and reported in the previous two sections, showed clearly that programs expressed in behavioural terms, by focussing more on the description of the task than on its actual implementation, transcend assembly particularities. The consequent implication is that if an agent is taught (programmed) to accomplish a certain task, then it can automatically perform the same procedures onto an entire class of them requiring the same sequence of steps, i.e. getting in this way a sort of generalization for free.

Stretching the aforementioned observation, I could say that if two different behavioural agents can perform the same set of basic behaviours, then any task expressed in their terms could be executed independently by each of them.[16] In other words, if they are both knowledgeable in the same domain, then they do not need to tailor any pattern of actions (program) onto such a domain to their specific characteristics. A behavioural program becomes in this way totally portable from one platform to another. In this regard, it is interesting to draw a parallel to the philosophy underlying JAVA™ which makes use of intermediate code that can be run by different hosts.

7. CONCLUSIONS

As experimentally shown in this paper, assembly programs developed within the behaviour-based assembly paradigm hold an important characteristic: generality. A task expressed in goal-achieving terms can, in fact, transcend the great majority of the specificities of a particular set of parts, and consequently be applicable to an entire family without any sort of final tailoring. Indeed, as long as different agents can accomplish the same tasks, programs developed for a specific robot could in theory be executed onto a different one, even from a different maker, without requiring any form of porting.

Acknowledgments

I would like to thank the Department of Artificial Intelligence at the University of Edinburgh for having provided the means for carrying out the research and Mr. Chris Malcolm for his useful comments and suggestions. I would like also to thank ABB Corporate Research for having provided further support to my work. A special thanks in this regard goes to Miss Donna Capararo of ABB Corporate Research for having proofread the paper and provided me with useful remarks on how to improve its readability. Last but not least, I would like to thank the Commission of the European Communities for having provided through the grant EHBCHBICT920012 the initial financial means for carrying out the experimental research in Edinburgh.

References

1. Kenneth M. Moore, Clark Glymour and Patrick J. Hayes, *Android Epistemology* (AAAI Press/The Mit Press, Cambridge, Massachusetts, U.S.A., 1st edition, 1995).
2. L.I. Lieberman and M.A. Welsey, "Autopass: an Automatic Programming System for Computer Controlled Mechanical Assembly," *IBM Journal of Research Development,* **21**(4), 321–333 (1997).
3. R.H. Taylor, P.D. Summers and M.J. Meyer, "AML: A Manufacturing Language," *Int. J. Robotics Research,* **1**(3), 19–41 (1982).
4. Unimation Inc., a Westinghouse Company, Danbury, Connecticut, U.S.A., *Programming Manual – User's Guide to VAL II*, version 1.1 edition (August, 1984), Document 398T1 (Preliminary).
5. S.T. Rock, "Developing Robot Programming Languages using an existing Language as a Base – a Viewpoint," *Robotica* **7** (Part 1), 71–77 (1989).
6. ABB Robotics Products, AB, S-721 68 Västerås, Sweden, RAPID *Reference Manual*, issue for baseware o.s. 2.1 edition, (March, 1997), Article Number 3HAB7537–1.
7. R.P. Paul, "WAVE: A model based language for manipulator control," *Industrial Robot*, **4**(1), 10–17 (March, 1997).
8. R.J. Popplestone, A.P. Ambler and I. Bellos, "RAPT: a Language for Describing Assemblies," *Industrial Robot* **5**(3), 131–137 (1978).
9. Tomás Lozano-Pérez and Rodney A. Brooks, "An Approach to Automatic Robot Programming," *A.I. memo 842* (Artificial Intelligence Laboratory, Massachusetts Institute of Technology, April, 1985).
10. Tomás Lozano-Pérez, Joseph L. Jones, Emmanuel Mazer and Patrick A. O'Donnell, *Handey: a Robot Task Planner* (The M.I.T. Press, Cambridge, Massachusetts, U.S.A., 1st edition, 1992).
11. Bartholomew O. Nnaji, *Theory of Automatic Robot Assembly and Programming* (Chapman & Hall, London, 1st edition, 1993).
12. Rodney Allen Brooks, "A Robust Layered Control System for a Mobile Robot," *IEEE Journal of Robotics and Automation* **1**(2), 14–23 (April 1986).
13. Chris Malcolm and Tim Smithers, "Programming Robotic Assembly in Terms of Task Achieving Behavioural Modules," *Journal of Structural Learning* **10**(2), 137–156 (1989).
14. Mikell P. Groover, Mitchell Weiss, Roger N. Nagel, and Nicholas G. Odrey, *Industrial Robotics: Technology, Programming and Applications*, Industrial Engineering Series (McGraw-Hill, 1st edition, 1986).
15. Tomás Lozano-Pérez, "Robotic Programming," *A.I. Memo 698a* (Artificial Intelligence Laboratory, Massachusetts Institute of Technology, December 1982).
16. Giovanni C. Pettinaro, "Definition of a Behaviour-Based Language for Industrial Robots," *Proceedings of the Inter-*

https://doi.org/10.1017/S0263574700003015 Published online by Cambridge University Press

national Scientific Conference on Artificial Intelligence in Industry (AIII '98), Stara Lesna, High Tatras, Slovakia, 1998 pp. 237–246.
17. James L. Nevins and Daniel E. Whitney, "Computer-Controlled Assembly," *Scientific American*, 62–74 (February, 1978).
18. Peter Balch, "A Software Architecture for Robot Assembly," *D.A.I. Working Paper No. 235* (Department of Artificial Intelligence, University of Edinburgh, 1992).
19. Carlton Byrne, "A Survey of Mechanical and Electro-mechanical Assembly Tasks," *Cardiff Technical Note ARC No. 32* (Department of Mechanical and Manufacturing Systems Engineering, University of Wales Institute of Science and Technology, June, 1989).
20. Giovanni C. Pettinaro, "The Use of Behavioural Modules to Simplify the Programming of Assembly Robots," *PhD thesis* (Department of Artificial Intelligence, University of Edinburgh, 1996).
21. Giovanni C. Pettinaro, "Basic Set of Behaviours for Programming Assembly Robots," *Proceedings of Robotikdagarna 1997*, Linköping Technical University (10–11th June, 1997) pp. 61–72.
22. Cezary Zieliński, "Object-Oriented Robot Programming," *Robotica*, **15** (Part 1), 41–48 (1997).
23. D. Simon, B. Espiau, K. Kapellos and R. Pissard-Gibollet, "Orccad: Software Engineering for Real-Time Robotics – A Technical Insight," *Robotica*, **15** (Part 1), 111–115 (1997).
24. Jiming Liu and Hong Qin, "C^4: a Software Environment for Modeling Self-Organising Behaviours of Autonomous Robots and Groups," *Robotica* **15** (Part 1), 85–98 (1997).
25. Giovanni C. Pettinaro and Chris Malcolm, "Electric Gropper Development," *D.A.I. Technical Report No. 28*, (Department of Artificial Intelligence, University of Edinburgh, September, 1994).
26. Giovanni C. Pettinaro, "Behaviour-Based Peg-in-Hole," *Robotica*, **17** (Part 2), 189–202 (1999).
27. Eric Paulos, "Trends in Grasping," *Robotics Grasping* (Literature Overview, 1993).
28. Göran Lundstrom and Brian W. Rooks, *Industrial Robots-Gripper Review* (International Fluidics Services, Bedford, 1st edition, 1977).
29. Prasad Akella and Mark Cutkosky, "Manipulating with Soft Fingers: Modeling Contacts and Dynamics," *Proceedings of the IEEE International Conference on Robotics and Automation* Resort Scottsdale, Arizona, U.S.A., (14th–19th May, 1989) pp. 764–769.
30. David J. Montana, "The Condition for Contact Grasp Stability," *Proceedings of the IEEE International Conference on Robotics and Automation*, Sacramento, California, U.S.A. (9th–11th April, 1991) pp. 412–417.
31. Antonio Bicchi, Kenneth J. Salisbury and David L. Brock, "Contact Sensing from Force Measurement," *Int. J. of Robotics Research* **12**(3) 249–262 (1993).
32. Marnix van Bruggen, Jan Peter Baartman and Willem F. Bronswoort, "Grips on Parts," *Proceedings of the IEEE International Conference on Robotics and Automation*, Atlanta, Georgia, U.S.A. (2nd–6th May, 1993) **Vol. 2**, pp. 828–833.
33. Stefano Caselli, Eugenio Faldella, Bruno Fringuelli and Francesco Zanichelli, "A Hybrid System for Knowledge-Based Synthesis of Robot Grasps," *Proceedings of the IEEE Conference on Intelligent Robots and Systems (IROS '93)*, Yokohama, Japan (26th–30th July, 1993), **Vol. 2**, pp. 1575–1581.
34. A. Joukhandar, C. Bard, and C. Laugier, "Planning Dextrous Operations using Physical models," *Proceedings of the IEEE International Conference on Robotics and Automation*, San Diego, California, U.S.A. (8th–13th May, 1994) pp. 748–753.
35. Michael Taylor, Andrew Blake and Adrian Cox, "Visually Guided Grasping in 3D", *Proceedings of the IEEE International Conference on Robotics and Automation*, San Diego, California, U.S.A. (8th–13th May, 1994) pp. 761–766.
36. M.A. Rodrigues, Y.F. Li, M.H. Lee, J.J. Rowland and C. King, "Robotic Grasping of Complex Objects without Full Geometrical Knowledge of the Shape," *Proceedings of the IEEE International Conference on Robotics and Automation*, Nagoya, Aichi, Japan (21st–27th May, 1995), pp. 737–742.
37. Nobuhiro Okada, Tadashi Nagata and Tsutomu Hasegawa, "A Reliable Parts-Picking System with an Active and Multi-Sensor Visual System," *Robotica* **15** (Part 6), 693–700 (1997).
38. Giovanni C. Pettinaro, "Behaviour-Based assembly of a Family of Electric Torches," *Proceedings of the 6th IEE International Conference on Mechatronics*, University of Skövde, Mount Billingen, Skövde, Sweden (9th–11th September, 1998) pp. 153–158.
39. Giovanni C. Pettinaro, "Human-Modeled Robot Screw Fastening," *Proceedings of the 1998 IEEE International Conference on Intelligent Robots and Systems (IROS '98)*, Victoria Conference Centre, Victoria, B.C., Canada (12th–16th October, 1998) **Vol. 2**, pp. 1009–1014.

https://doi.org/10.1017/S0263574700003015 Published online by Cambridge University Press