MONITORING THE EXECUTION OF SENSORY ROBOT PROGRAMS*

Vincenzo Caglioti[1], presently [3], Massimo Danieli[2], Domenico Sorrenti[3]

1 International Computer Science Institute; 1947 Center Street, 94704-1105 Berkeley, California, U.S.A,

2 ABB SAE Sadelmi s.p.a., P.le Lodi 3, 20137 Milano, Italy

3 Artificial Intelligence and Robotics Project, Dipartimento di Elettronica e Informazione, Politecnico di Milano, P.zza Leonardo da Vinci 32, 20133 Milano, Italy

Abstract - A system is presented, which monitors the execution of an assigned robot program, in order to detect execution errors. The assigned program is supposed to include sensor instructions, which allow to adapt the execution to variable environment conditions. In the presented system, the task information is represented in terms of a relationship between environment conditions and workcell evlution. The selection of the monitoring sensor detctions is based on the accuracy in checking the state variables and in the ambiguity in matching the sensor measures to the variables to be measured.

1. Introduction

Robot programs including sensor instructions are generally aimed at producing a given, desired workcell evolution, *independent* of the environment conditions: e.g., whatever be the entrance position of an input object, this object is assembled to other objects in a predefined workcell location. This situation is in accordance with the "one product" assumption adopted by Fielding et al. [4].

However, this is not the only possibility. Infact a sensorized program could also be aimed at producing a workcell evolution *dependent* on the environment conditions: e.g., the set of the objects assembled with an input object may depend on the geometry of the input object. By this possibility, the robot flexibility can be exploited in order to perform different types of assemblies.

In this work a monitoring system is illustrated, that: (i) plans and executes sensing strategies aimed at assessing the execution correctness of a sensory robot program, (ii) if an execution error is detected, it plans sensing strategies aimed at determining the error state.

Many current approaches to error recovery ([1], [4], [4], [5], [13]) are of limited applicability in most of the practical cases where: (i) task level information, or even object level information, is not directly available, since the robot task is specified by manipulator level programs (AML and VAL are among the most widely used programming languages), (ii) several sensor instructions are often included in realistic programs, in order to let the workcell evolution vary according to the variable environment conditions.

In the system presented, the task information is represented by means of a relationship (*step conditions*) between the environment conditions and the workcell evolution. These conditions have to be satisfied at the end of the execution of each program *step*. The task information is organized in a forest structure (a forest is a set of trees) called *skeleton*. Each node of the forest represents a workcell state. The roots of the forest represent the possible initial states. The leaves of the forest represent the possible final states of the

*This work has been supported by: the Italian National Research Council, C.N.R., PFR 2 "MANUEL"; IBM Semea (contract: *Error Recovery in Assembly Robots*)

program. The task information is automatically extracted, starting from the given program written in the manipulator level language AML [2], [15]. The set of the *correct* (or *nominal*) evolutions, associated to an assigned program, can be obtained by means of the simulation of an errorless excution of the program [16].

The monitoring system selects, for each program step, a set of physical variables that are to be checked in order to verify the step conditions. Each of the selected physical variables has to be checked at least once within a *non-critical sequence*, which is a sequence of steps between two *critical events* for it (for instance, if v is a variable related to a physical object in the workcell, e.g., its position, a critical event for v is the modification of the contacts with the other objects). The possible sensor detections (SDs) are evaluated a priori with respect to their *accuracy* and *ambiguity* in measuring a considered physical variable. The *ambiguity* is related to the risk that the sensed features and the scene features are not matched correctly. Both the coefficients are determined by the simulation of the sensor detections. The best SD, between those capable to check the considered physical variable in the considered *non critical sequence*, is then selected. The chosen SDs, related to the whole set of physical variables, are then combined and give rise to an augmented set of SDs; the new entries of the SD set are built aggregating the SDs that can share some computation (e.g., at the end of step i the two SDs aimed to check the position of *Obj-j* and *Obj-l* can be aggregated because both exploit the actual line drawing of camera-k). By the selection of different sequence of SDs, different *sensing strategies* can be generated. At this point the problem reduces the search of the (time) best sensing strategy. The problem can be viewed as a *weighted set covering* problem [6]; the set of the physical variables to be verified represents the set to be covered, and the SDs represent the covering elements. Each SD covers one or more physical variables, and each physical variable is considered once for each of its non-critical sequences. The weight of each SD is represented by the estimate of its time-cost. We solved the problem by means of a specialization for the zero-one problem of the Branch and Bound algorithm [6].

The considered sensor detection are:

1. A *photo* primitive, whose output is a binary signal relative to a photocell mounted between the fingers of the robot gripper. If an object is present between the fingers, then the output is active.
2. A *loc(x, y, theta, phi, psi, Obj)* primitive, which determines position and orientation of a modeled object basing on the linear segments extracted from a single image. The *loc* primitive performs a 3D (i.e., six degrees of freedom) localization [17].
3. A *match(x, y, theta, phi, psi, Obj)* primitive, which compares an expected scene line-drawing with the actually extracted line-drawing, and then performs a 3D localization.

In [14] a criterion is proposed for planning the best sensor configuration in order to check an object feature; it can be seen as a first step toward an *active monitoring system*. On the other hand, the presented system is only capable to determine that some physical variables can be poorly checked (or even not checkable at all), due to the configuration of the workcell objects and sensors.

2. An example of skeleton

Our experimental workcell (Fig. 1) is equipped with a four degrees of freedom SCARA IBM-7547 robot. A photocell is mounted on the robot gripper. Two fixed cameras, indicated by tv1 and tv2, are present. On both these

cameras the primitives *match* and *loc* are implemented.

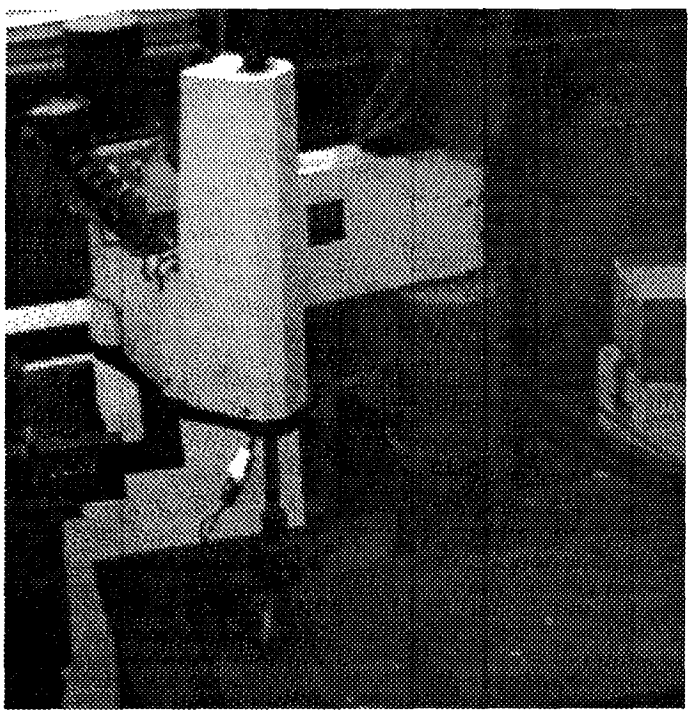


Fig. 1 Experimental Setup

Two objects are involved in the example. The first object (O1), belonging to the OBJ1 class, is a holed parallelepiped with dimensions (11x7x3) along respectively the x, y, z axis of the object centered reference. The origin of this reference is on a vertex of the base of the parallelepiped. The hole is centered on the upper face of the parallelepiped, and its dimensions are (1x3x1.5). The second object (O2), can belong to either of the two classes OBJ2 and OBJ3. Both are parallelepipeds of dimensions (1x3xh). While for the OBJ2 class h=5, for the OBJ3 class h=2. In the initial state of the program execution O2 is present in the workcell in the position: xO2=70, yO2=70, zO2=-50, rO2=0 (rO2 indicates the *roll* angle). The initial position of object O1 is: xO1=-55, yO1=0, zO1=-50, rO1=0. The robot program consists first in determining the class of O2: if its class is OBJ2, then O2 is inserted into O1. Otherwise, the object O2 is put into a basket. The AML/E program ([2]) is:

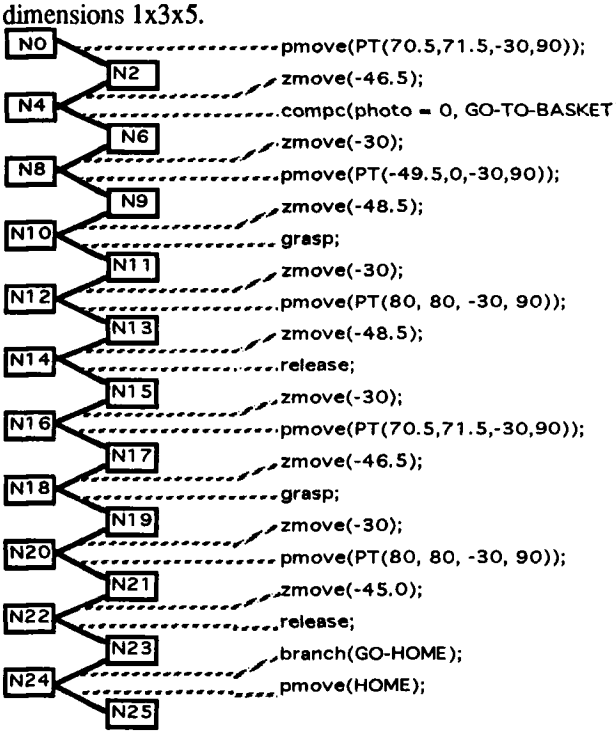
```
HOME: NEW PT(103,0,0,0); -- the parking position
BASKET :NEW PT(0,90,-30,90); -- the basket position
x1,y1,r1: STATIC COUNTER;
        -- cartesian coordinates and roll angle of O1
pmove(PT(70.5,71.5,-30,90));
        -- move to the vertical of O2
zmove(-46.5);
        -- moves to midheight between OBJ2 and OBJ3
compc(photo = 0, GO-TO-BASKET);
        -- if O2 = OBJ3 (h = 2) go to basket
   zmove(-30);
   pmove(PT(-49.5,0,-30,90)); -- vertical of O1
   zmove(-48.5); -- move to grasp O1
   grasp; -- grasp O1
   zmove(-30);
   pmove(PT(80, 80, -30, 90));
        -- move to the vertical of the assembly position
   zmove(-48.5);
        -- move O1 to the assembly position
   release; -- open hand
   zmove(-30);
   pmove(PT(70.5,71.5,-30,90));
        -- move to the vertical of O2
   zmove(-46.5);
   grasp; -- grasp O2
   zmove(-30);
   pmove(PT(80, 80, -30, 90));
        -- move toward assembly position
   zmove(-45.0); -- insert O2 into O1
   release;
   branch(GO-HOME);
:GO-TO-BASKET;
zmove(-49);
grasp; -- grasp O2
zmove(-30);
pmove(BASKET);
release;
:GO-HOME;
pmove(HOME);
:END.
```

The program skeleton, extracted by the nominal case simulator is reported in Fig. 2. The node N0 contains the initial state corresponding to the case in which O2 belongs to the class OBJ2, while N1 contains the initial state corresponding to the case in which O2 belongs to the class OBJ3. The forest actually consists in two linear sequences: one starting from N0, the other starting with N1.

3. An example of monitoring plan

For simplicity, the monitoring example here illustrated involves a skeleton constituted by a single node sequence. Two objects are involved in this example. The first object (O1), which belongs to the OBJ1 class, is a holed

parallelepiped with dimensions 11x7x3. The hole is centered on the upper face of the parallelepiped and its dimensions are 3x1x1.5. The second object (O2), belonging to the OBJ2 class, is a parallelepiped with dimensions 1x3x5.



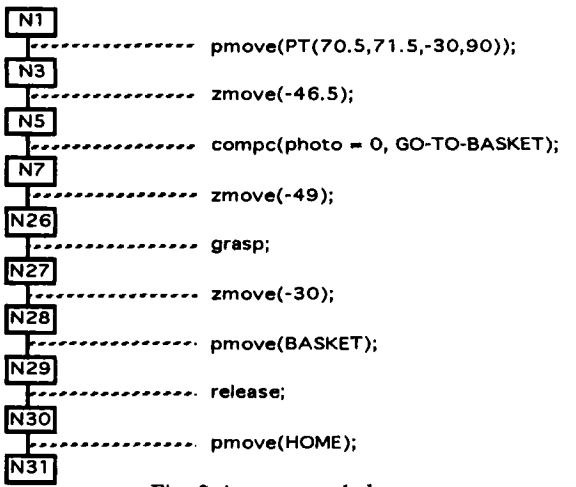


Fig. 2 A program skeleton

In the initial state of the program execution, only O2 is present in the workcell. Its initial position is (70, 70, -50, 0). O1 is introduced into the workcell at the initial position (-55, -2.5, -50, 5.01). The robot program consists first in grasping O2 in its initial position and then in inserting it into the hole of O1. The AML/E program is reported below, the program skeleton in Fig. 2.

```
HOME: NEW PT(103, 0, 0, 0); -- the parking position
O2VERT: NEW PT(70.5, 71.5, -30, 90);
        -- a point along the vertical of O2
O1VERT: NEW PT(-49.826, 1.467, -30, 5.01);
        -- a point along the vertical of O1
pmove(O2VERT); -- move to the vertical of O2
zmove(-46); -- move toward the grasping position of O2
grasp; -- grasp O2
zmove(-30);
pmove(O1VERT); -- a position on the vertical of O1
zmove(-44.5); -- insert O2 into O1
release; -- open hand
zmove(-30);
pmove(HOME); -- return to parking position
:END.
```

The node sequence is analyzed, in order to find the physical variables to be verified and to determine the corresponding*non-critical sequences.*

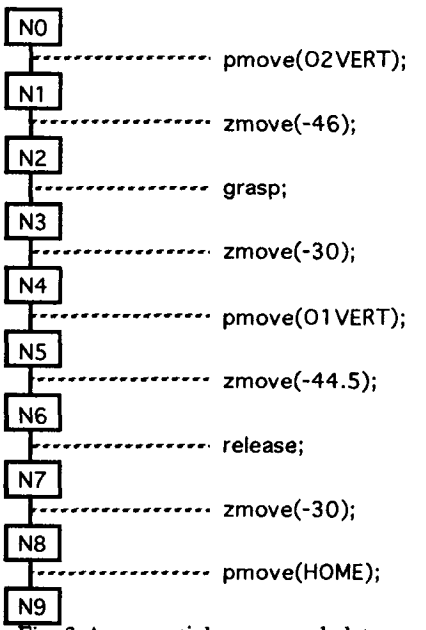


Fig. 3 A sequential program skeleton

For instance, at node N0 the physical variable $x_{O2}=70$ with a related *non-critical sequence* (N0, N1, N2) is found. The *non-critical sequence* for x_{O2} ends at node N2, since in N3 the object O2 is grasped by the robot. After the analysis has been completed, the sensor simulation is executed along the node sequence. The table below reports the *accuracy* σ^2 and *ambiguity* γ relative to the activation

of the *loc* primitive on tv1 at node N8 with respect to the position parameters of objects O1 and O2, as calculated basing on the expected line drawing.

variable	σ^2	γ
x_{O1}	0.5583	0.5
y_{O1}	0.0092	0.5
z_{O1}	0.3791	0.5
r_{O1}	$5.821\ 10^{-7}$	0.5
x_{O2}	0.122	0.25
r_{O2}	$8.735\ 10^{-4}$	0.25

The results relative to the y and z coordinates of the object O2 are neglected, since their *accuracy* is below a given threshold. The plan generated in correspondence to the node sequence is composed of the following sensor instructions, in which the spatial positions are indicated by the roll, pitch, and yaw angles, and by the x, y, z coordinates referred to the frames attached to the cameras tv1 and tv2:

```
(:read (photo1) :correct (value 0) :node N0)
(:read (tv2 (LOC OBJ2)) :correct (value (OBJ2 (0, 0, 0, 20,
      120, 0))) :node N1)
(:read (photo1) :correct (value 1) :node N3)
(:read (photo1) :correct (value 1) :node N4)
(:read (tv1 (LOC OBJ1)) :correct (value (OBJ1 (3.018,
      -0.782, -3.055, 4.246, -2.5, 72.46))) :node N4)
(:read (photo1) :correct (value 1) :node N5)
(:read (tv2 (LOC OBJ1)) :correct (value (OBJ1 (0, 0,
      0.087, -105, 47.5, 0))) :node N5)
(:read (photo1) :correct (value 1) :node N6)
(:read (tv1 (LOC OBJ1)) :correct (value (OBJ1 (3.018,
      -0.782, -3.055, 4.246, -2.5, 72.46))) :node N6)
(:read (tv2 (LOC OBJ2)) :correct (value (OBJ2 (0, 0,
      -1.483, -101.381, 52.095, 1.5))) :node N8)
(:read (tv1 (LOC OBJ1 OBJ2)) :correct (value ((OBJ1
      (3.018, -0.782, -3.055, 4.246, -2.5, 72.46)) (OBJ2
      (-1.644, -0.073, 2.359, 0.614, 1.833, 73.97))))
      :node N9)
(:read (photo1) :correct (value 0) :node N9)
```

where the keyword *:read* indicates the sensor that must be activated; the keyword *:correct* is followed by the predicates to be satisfied by the sensor data. The keyword *:node* indicates the node to which the instruction is associated.

4. Current developments

During the program execution, the selected sensor detections may eventually detect an error. In order to proceed with the recovery of the robot operations, the cause of the occurred error must be determined. To this aim a diagnosys systems has been designed. This system will be described briefly in the sequel (experimental results are not yet available).

As an error is detected by the monitoring system, a diagnosis system is called into operation in order to determine the error state. Wrongly positioned objects, and defective objects are among the considered classes of error causes. In order to determine the workcell state, which is the most "likely", the diagnosis system uses information about the past actual execution, e.g., past actions and past sensor detections (both the actual and the expected readings). To consistently integrate the available information and thus discriminate the most likey state hypotheses, an uncertainty management mechanism is adopted, based on evolutions ([7], [8], [9]) of Dempster-Shafer's theory of evidence ([10], [11]).

As a result of the incremental analysis of the error hypotheses, the belief state of each hypotheses is evaluated. The belief state is categorized according to its *support* and its *plausibility*: in [8] six categories are proposed. The hypoteses whose belief state is in either of the groups *believed* and *rather believed than disbelieved* are considered for further validation. For this last task, these hypotheses are handled as correctness conditions to be verified by the monitoring system; namely a sensing strategy is planned and executed in order to verify each hypothesis. If the results of the executed sensor detections allow to confirm one of the considered hypotheses, then the analysis task is concluded and a recovery strategy can be planned. Otherwise, the current set of hypotheses (together with their belief state) is updated in view of the results of the executed sensor detections, and the process is re-iterated

until either a hypothesis is confirmed or no sensor is available to check the error state.

References

[1] R. Smith, M. Gini, "Reliable real-time robot operation employing intelligent forward recovery", Int. Journal of Robotic Systems, Vol. 3, n. 3, pp. 281-300, (1986)

[2] AML/Entry v.4 User's Guide, 2nd ed., IBM Press, Aug. 1985

[4] A. Hoermann, W. Meier, J. Scholen, "A control architecture for an advanced fault-tolerant robot system", Robotics and Autonomous Systems, Vol. 7, pp. 211-225, (1991)

[4] P. Fielding, F. Di Cesare, G. Goldbogen, "Error recovery in automated manufacturing through the augmentation of programmed processes", Int. Journal of Robotic Systems, Vol. 5, n. 4, pp. 337-362, (1988)

[6] G. R. Meijer, L. O. Hertzberger, T. L. Mai, E. Gaussens, F. Arlabosse, "Exception handling systems for autonomous robots based on PES", Robotics and Autonomous Systems, Vol. 7, pp. 197-209, (1991)

[6] H. M. Salkin, K. Mathur "Foundations of Integer Programming", North Holland, (1989)

[7] A. Bonarini, E. Cappelletti, A. Corrao, "Network-Based Management of Subjective Judgements: A Proposal Accepting Cyclic Dependencies", IEEE Transactions on Systems Man and Cybernetics, vol. 22, n. 5, 1992

[8] D. Driankov, "Uncertainty calculus with verbally defined belief-intervals", Int. Journ. on Intelligent Systems, vol. 1, 1986

[9] D. Driankov, "Toward a many-valued logic of quantified belief: The information lattice", Int. Journ. on Intelligent Systems, vol. 6, 1991

[10] G. Shafer, "A Mathematical Theory of Evidence", Princeton University Press, Princeton (1976)

[11] G. Shafer, "Theory and Practice of Belief Functions", Int. Journ. on Approximate Reasoning, vol. 4, n. 5/6, 1990

[13] T. Cao, A. C. Sanderson, "Sensor-based error recovery for robotic tasks sequences using fuzzy Petri nets", Proc. IEEE Intnl. Conf. on Robotics and Automation, (1992)

[14] K. Tarabanis, R. Y. Tsai, P. K. Allen, "Automatic sensor planning for robotic vision tasks", Proc. IEEE Int. Conf. on Robotics and Automation, (1991)

[15] R. H. Taylor, P. D. Summers, J. M. Meyer, "AML: a manufacturing language", The Int. Journ. of Robotics Research, vol. 1, n. 3, 1982

[16] V.Caglioti, M.Somalvico, "Symbolic simulation of sensory robot programs", in B.Torby, T.Jordanides (eds.) "Expert Systems and Robotics", NATO ASI Series, Springer-Verlag, (1990)

[17] V. Caglioti, "The planar three-lines junction perspective problem, with application to the recognition of polygonal patterns", Pattern Recognition, Vol. 26, n. 11, (1993)