PROGRAMMING LANGUAGE FOR ROBOTIC ASSEMBLY TASKS

Zakaria Toukal - Yacine Amirat - Said Babaei - Jean Pontonnau

Laboratoire d'informatique industrielle et d'automatique (L.I.I.A)
IUT de Créteil
61, Avenue du Général de Gaulle
94010 Créteil Cedex - France
Tel: 45 17 17 51
Fax: 45 17 17 54

Abstract

The work presented in this paper deals with the development of high level reactive macro-language dedicated to robotic assembly tasks. In order to satisfy the following constraints: Openness, modularity, real-time making decision and powerful programming system, a control architecture is proposed. It includes three software layers: user layer, command layer and manage/control layer. This last layer is based on a real-time multitasking executive. This programming language has been applied to drive a flexible assembly cell. In this paper a detailed analysis of the manage/control layer is presented. In order to show that our control system operates satisfactorily several simulations are presented and discussed.

I Introduction

Programming robotic systems in assembly field implies the development of fast, adaptive and intelligent controls and also needs a correct and reliable integration of sensors informations related to the evolution of the environment and of the system itself, particularly in a real-time context [13]. Among programming methods, high level languages have been developed in order to allow shorter programs. These languages are easier to understand by a non-experienced human operator. The pioneering works have concerned telerobotics applications, so several languages have been proposed as:

- LARTS system (Language Aided Robotic Teleoperation System). This system considers the telemanipulated control as a learning method which allows the acquisition of required data for the automated tasks description [9].
- YALTA language (Yet Another Language for Telerobotics Applications) which is an object level graphic language. It allows:

1. To program and to simulate robot actions plan,
2. To carry out and to modify on-line the automatic control,
3. To recover in teles-operated mode when an incident occurs,
4. To visualize the task evolution [15].
- MEISTER language (Model Enhanced Intelligent and Skilful Teleoperation) is an evolution of the LARTS system. It includes a motion interpreter associated with the world model which allows, given the start configuration to predict the goal configuration [11].

Generally, this kind of systems has the following major drawbacks:

- They are not portable for various hardware control architecture [8],
- Insertion or modification of control laws needs user intervention at low level programming. This leads to complex and coupled functions handling [13],
- Semantics of certain languages is not enough rigorous [6],
- Man operator must be experienced.

All these drawbacks along with economic factors have led many researchers to investigate new approaches of robotic programming languages providing high flexibility, powerful instructions and real-time decision capability.

The work presented here deals with the development of a high level and reactive macro-language dedicated to robotic assembly tasks such as: inserting a peg in a hole, or a contour tracking of a part. This macro-language must satisfy the following constraints:

- Openness,
- Modularity,
- Real-time making decision,
- Powerful programming system,

These constraints have led us to develop a control architecture which includes three software layers:
1. User layer,
2. Command layer,
3. Manage and control layer. This layer is based on a real-time multitasking executive.

The programming language which has been developed has been applied to drive a flexible assembly cell which includes a parallel robot.

In this paper, a detailed analysis of the manage and control layer is presented. In order to verify the validity of our control system, several simulations have been performed.

2. Description of the assembly cell

Figure 1 shows a whole view of the assembly cell. This cell includes the following modules:
- a 2D cartesian robot,
- a six degrees of freedom parallel robot which acts as an active force controlled wrist of the cartesian robot.

In our approach, an assembly task consists of the following steps:

- Wide amplitude displacements performed by the 2D catesian robot in order to bring the assembly parts in a close vicinity,
- Very accurate corrective trajectory performed by the parallel
robot under control of an external vision sensor in order to perform
the proper location of the moving part with respect to the receptive
part. The vision system measures the relative positioning of the
parts to assemble.

- assembly or final insertion phase: During this phase contacts
between parts may arise. So it is therefore necessary to implement a
force feedback control of the parallel robot in order to carry out
this task successfully. This force feedback is needed for security
constraints and to insure regularity and quality of assembly.

The photography 1 represents a whole view of the
experimental set up.

3. Software architecture of the flexible assembly cell
control system

When specifying the different actions executable by a robot,
it's necessary to structure the whole set of physical and algorithmic
entities constituting the hardware and software environment,
according to an object-oriented approach [16]. This approach
allows in the same way to develop a modular and structured
software and a very high performance man-machine interface. In
this case, the robot task constitutes an actor characterized by an
autonomous behaviour [6][10]. In the description of this task a
particular attention must be given to signals and variables labeling.
The timing specifications (frequency, delays, synchronization...) of
communication interfaces must also be specified. As a robot is a
reactive system, the description of a task must take into account the
following fundamental aspects:

- the functional aspect related to the aim to reach,
- the environmental aspect which ensures the communication with
the environment by listing reaction causes,
- the informational aspect which translates the entities of the
robotic system into abstracted data [3].

The driving of the assembly cell described in section 2
implies the use of a control system handling various applications
[5]. In order to increase the cell flexibility with respect to the task
and the environment, the control system must carry out the
following characteristics:

- Openness: In order to allow easily the integration of new
functionnalities through the implementation of new sensors,
actuators, and control laws.
- Modularity: Decoupling functions and hierarchizing of
data processing so that a function of a control layer can be easily
modified.
- Real-time making decision: Fast and deterministic response
times are required when exceptional situations occur.
- Powerful processing system: Providing a man-machine
interface allowing the access to all software layers and an easy
writing of programs by a non-experienced human operator. The
programming system must also allow to run simultaneous actions,
to describe the time evolution of the system and to define the
system behavior facing to exceptional situations.

In our case the behavioral modelization of the application
and its elementary actions can be easily obtained by structuring the different software and hardware components of the application in an object-oriented model. From the constraints quoted above and the cell characteristics we have developed a control architecture which includes the following software layers:

- user layer,
- command layer,
- manage and control layer.

![Software architecture of the control assembly cell control system](image)

Figure 2. Software architecture of the control assembly cell control system

### 3.1. User layer

The user layer includes a high level macro-language, its command interpreter and the system allowing its exploitation (editor, procedures calling, robotic functions, etc.). The functionalities of this layer are:
- Writing of application program in high level macro-language
- The user can chain the tasks associated to the different levels listed within libraries signals and by specifying the corresponding parameters.
  - Accessibility to all the layers.
  - Ability to call both the editor and the command interpreter.

The last interpretation phase generates a table (automaton) whose each element contains the codes of controlled entities, the functions to call and the associated parameters and signals. Each element is a state and the controlled entities are sub-states. Here, a state can be: either a sequence of actions or parallel actions or a combination of both. For example, one can cite the simultaneous control of the cartesian robot in velocity mode and the parallel robot in position mode. Transitions from a state to another depends on either signals, or acknowledges which correspond to the completion events of the corresponding sub-states (example: position target reached, mechanical stop, jamming of parts).

The automaton is managed by the real time multitasking executive (RTC). Figure 3a shows an example of automaton dedicated to carry out a picking task. The picking task programm is given in figure 3b.

![Picking application automaton](image)

Figure 3a. Picking application automaton

```
[ MOVE_HANDPOSITION ROB_PAR (i)< position, velocity, acceleration > ]
And
MOVE ROBOT CAR_ROB < trajectory name, % velocity > ]

[ OPEN GRIPPER ROB_PAR < opening > ]

[ MOVE_HANDPOSITION ROB_PAR (ii) < position, velocity, acceleration > ]
And
VISION (object name, gravity center) 

[ CLOSE GRIPPER ROB_PAR < force tightening > ]

[ MOVE_HANDSPEED ROB_PAR (ii) < velocity, velocity, acceleration > ]
End;
```

Figure 3b. Picking task program
3.2. Command layer

The user has access to this programming level to create and manage new applications. So, this layer provides the required tools for the implementation and the programming of elementary actions. It provides the following functionalities:
- Insertion and modification of control laws along with integration of sensors and actuators,
- Definition of time constraints (period, delays and synchronization) in order to obtain a desired control dynamics,
- Combination of control laws in order to obtain an up-level function such as hybrid force position control by calling listed functions within libraries.

The user must specify the features of task implementation such as:
- processors assignment to the different elementary tasks,
- Specification of I/O : Address, number, data structure,
- etc.

3.3. Manage and control layer

This layer manages the application automaton under a real time executive multitasking according to the following steps:
- Analysis of automaton state
- State activation parameters request,
- Orders sending to the different assembly cell entities,
- Signals receiving and exception state building,
- Data bases updating, and next state preparing.

Other modules are associated to the layers, described above we mainly distinguish:

**Man-machine interface**

During the simulation execution phase, the operator chooses the actions to be performed and the values of parameters. From these choices, a simulation is then performed in order to determine which input values must be chosen according to each situation. A set of modules is provided to the user from a menu. These modules require informations which are given by the user.

Furthermore, the control system provides to the user the current states of the application. The numerical informations are grouped by themes. The graphical informations are displayed on the monitor screen that allows the operator to visualize the assembly area.

**Data base**

This module allows the storage of usual informations for a specific assembly task. It also saves the state of the universe during the performing of an application scheme. This data base also brings out a time saving and a position computing economy for a given assembly task.

**Application automaton**

The transition rules of the automaton define the language semantics. The execution module is essentially composed of an interfacing between external events and application execution scheme produced signals. It continuously watches for interruptions of different natures.

---

4. Manage and control layer architecture

In this section we detail the manage and control layer architecture which is illustrated in figure 4. We give hereafter the activities of the different tasks.

**Control task : T_Control**

At the initialization, this task is waiting for a message on mailbox MBX_State. This task is activated by the presence of a token in the semaphore SEM_Return and then it executes the following actions:

- It loads the current state of the assembly cell entities (parallel robot, cartesian robot, vision system) from the mailbox MBX_State.
- It sends the modified states of the entities along with the request of the required informations for the execution of the next state. These data are sent to the task T_Update using the mailbox MBX_Update.
- It receives the requested informations via the mailbox MBX_Parameters.
- It elaborates, then sends orders to the different entities via the mailbox MBX_Server.
- It sends to the task T_Return via the mailbox MBX_Return the acknowledges leading to the synchronization of the automaton states.
- It receives the states of the activated entities via the mailbox MBX_State.

**Postman task : T_Postman**

This task is activated after the reception of acknowledges sent by each interface task composing the cell (parallel robot, cartesian robot, vision system). It executes then the following actions:

- It classifies the acknowledges, puts them into the following form: ( component, state ) and sends them in the mailbox MBX_Report
- It decrements the semaphore SEM_Synchro.

**Server task : T_Server**

This task is activated by the presence of a message in the mailbox MBX_Server. Then it executes the following actions:

- It sends the orders to the different entities via their corresponding mailboxes (MBX Parallel Robot, MBX Cartesian Robot, MBX Vision).
- It sends to the semaphore SEM_Synchro a value corresponding to the number of controlled entities.

**Return task : T_Return**

This task is activated by the presence of a message in the mailbox MBX_Return indicating the events to look after. It executes then the following actions:

- It controls the waited events (incident or failure events).
- It sends a token to the semaphore SEM_Return.

Figure 5 illustrates orders distribution mechanism to the different interface tasks for the control of the assembly cell components.
**Figure 4. Manage and control layer architecture**

**Figure 5. Orders distribution**

**Decision task :** T\_Decision

This task is activated by the occurrence of an event in the event word Pnt\_Evt\_Dec. It executes then the following actions:

- It analysis the events occurred and perform the processing associated to the type of event:
  - If the event belongs to the set Ee of emergency events, it sends a token to the semaphore SEM\_Reset in order to activate the task T\_Reset for stopping the application.
  - If the event belongs to the set Eef of the total failure events, it sends a message to the task T\_Cancellation. This message contains the name of the state where the failure has occurred.
  - If the event belongs to the set Epf of partial failure events, it sends a message to the task T\_Resumption. This message contains the name of the entity where the failure has occurred along with the number of messages to be taken in the mailbox MBX\_Report\_2. These messages are:
    - Waiting for the state message from the task T\_Incident.
    - Sending from the task T\_Decision (exception state) to the task T\_Resumption in the mailbox MBX\_Decision.

**Incident task :** T\_Incident

This task waits for a message in the mailbox MBX\_Incident from the task T\_Resumption. This message contains the number of messages to be waited for in the mailbox MBX\_Report\_2. The incident task executes the following actions:
• It resets the semaphore SEM_Report.
• It sends the events to the Return task.
• It gets the messages from the mailbox MBX_Resumption.
• It sends the state to the Decision task via the mailbox MBX_State.

Figure 6 illustrates the synchronization of the tasks by events.

Report task: T_Report

This task is activated by the presence of a message in the mailbox MBX_Report. Then it executes the following actions:
• It stores the received data.
• It sends environment states in the mailbox MBX_State if the counter of the semaphore SEM_Synchrn is equal to zero.

Figure 7 illustrates the synchronization of the tasks in the case where all the actions of an automaton state are successfully executed.

Cancellation task: T_Cancellation

This task is activated by the presence of a message in the mailbox MBX_Cancellation. This message indicates which entity where failure has occurred. Then it suspends a sub-state of the actual state of the automaton.

Reset task: T_Reset

This task stops the execution of the application when emergency events occur. This is done when a token is received in the semaphore SEM_Reset.

Resumption task: T_Resumption

This task is activated by the presence of a message in the mailbox MBX_Resumption. This message indicates the number of required acquisitions to the task T_Incident via the mailbox MBX_State. Then it executes the following actions:
• It sends a message to the task T_Incident via the mailbox MBX_Incident.
• It resets the semaphore SEM_Synchrn.
• It waits for a message in the mailbox MBX_Decision.
• It gets the exception state in this mailbox.
• It treats this state in the same way as the task T_Control except the processing associated with the task T_Return.

Figure 8 illustrates the synchronization of tasks in the where an incident occurs.

Emission Task: T_Emission

This task is activated by the presence of a message in the mailbox MBX_Emission. It elaborates the packet of data to transmit along with its sending.
Reception Task : T_Reception

This task is activated by the presence of a message in the mailbox MBX_Reception. It manages the received data and switches them to a mailbox of the task T_Parallel_Robot. Figure 10 shows the communication between the process control and other modules.

Figure 10. Serial link

4.1. Event types

Three types of events are listed which require adequate processing by the task T_Resumption.

Type n°1: It corresponds to events which need modification of orders. A typical event arises for instance when the parallel robot reaches a singular configuration.

Type n°2: It corresponds to events which require the stopping of the application. Such an event may occur when the operator activates the emergency stop.

Type n°3: It corresponds to events which require the stop of an automaton state such as a mechanical stop.

4.2. Incident with local reaction

A certain class of events does not require reaction at the system supervisor level, but are treated at the local level. The interface task of an entity starts running of functions which supervise the control laws execution. Processing are associated with the different kinds of incident which may occur during the execution of the task.

5. Simulations

In order to bring to show that the control system operates satisfactorily, simulations have been performed. So, three canonical situations have been tested and analyzed:

- The first one corresponds to a case of normal running.
- The second one corresponds to a case of total failure.
- The third one corresponds to a case of partial failure. This failure may concerns the one of the controlled entities. Thus, it is the Resumption task which manages the current state of the automaton.

For the examples of simulation reported hereafter, we have chosen as the automaton state, the simultaneous control of the parallel robot, the cartesian robot and the vision system. The chaining of task activities is performed as follows:

Success situation: The control task is activated first. It reads the initial parameters of the cell, then it analyzes the current state of the automaton and sends the different orders to the corresponding interface tasks. Each task is successfully executed and it sends both a signal to the task T_Return acknowledges to the task T_Postman via the mailbox MBX_Report_1. The activation of the next state is conditioned by the presence of a token in the semaphore SEM_Return. Figure 11 shows the chaining of the tasks.

Abbreviations have the following meaning:

Cartesian robot task: TCR, Cancellation task: TCA, Decision task: TD, Postman task: TP, Report task: TREP, Reset task: TRE, Server task: TS, Resumption task: TRES
Return task: TRE, Parallel robot task: TPR, Update task: TU, Vision task: TV

Total failure situation: This corresponds to a failure occurred at the robot interface task. A failure event is then sent to the task T_Decision which sends a token to the semaphore SEM_Reset. Figure 12 illustrates this situation.

Figure 11. Success situation

Figure 12. Total failure situation

Figure 13. Partial failure situation
Partial failure situation at the parallel robot level: In this case, a partial failure occurred during the robot interface task execution, needs a correction of orders. This leads to the activation of the task T_Decision after the reception of the event by the task T_Robot. This task implies the activation of the task T_Resumption which sends to the incident task the required acknowledges. The task T_Incident gets the messages and sends them to the task T_Decision. These messages are required in order to generate an exception automation which will be controlled by the task T_Resumption in the same way as the control task does with the application automation. Finally, the cycle resumes normally until a new incident occurs during the resumption phase. Figure 13 shows this situation.

6. Conclusion

The work presented in this paper concerns the development of a high level reactive macro-language intended to program assembly tasks. We have proposed a programming system which has the following characteristics: openness, modularity, real-time making decision and powerful programming tools. A control architecture with three software layers has been developed. Among these layers, the manage/control layer which is based on a real-time multitasking executive has been finely analysed. Several simulations have been performed in order to study the behavior of the system facing to different situations such as total or partial failure of a state. The programming system which is proposed is actually used to program an assembly cell integrating a parallel robot in order to perform assembly tasks under fuzzy and neuronal control.

References