Integrating purposive vision with deliberative and reactive planning: engineering support for robotic applications

Anna H. Reali C. Rillo[1] Leliane N. Barros[2] Reinaldo A. C. Bianchi[2]
[1] Department of Computer Engineering (PCS)
[2] Laboratory of Integrated Systems (LSI)
University of São Paulo (USP)
Av. Prof. Luciano Gualberto, trav. 3, 158
05508-900 São Paulo SP Brazil
arillo@pcs.usp.br, leliane@lsi.usp.br, rbianchi@lsi.usp.br

Abstract

We propose a multi-agent vision-based architecture to solve complex sensor-based planning tasks. A test bed implementation, with skills such as vision and collision avoidance, was used to run experiments in the proposed architecture. We demonstrate experimentally how the system can execute successfully complex assembly plans while dealing with unpredictable events and imprecise information, with no significant cost in run-time efficiency. Such experiments provided important insights about vision and planning and on how to build real world robotic systems.

Keywords: purposive vision, multi-agent architecture, planning and execution, deliberative planning and reactive planning.

1. Introduction

To solve complex problems in a real dynamic world, an intelligent system must be able to interact with its environment through sensing, processing and transforming information about the surrounding world into different levels of representation. Such processed information is then used by the system to interact back with the environment through robot actions. The idea of constructing such an intelligent system has been the ultimate goal of the Artificial Intelligence (AI) area and has led this community to investigate a number of other robot capabilities, such as: to generate and execute complex plans; to perform online resource allocation; to deal with problems as they arise in real-time (reaction); and to reason with incomplete information and unpredictable events.

In view of the inherent difficulty of the problem and the limited results obtained from AI in well-behaved artificial domains, running experiments is a good way to accomplish the goal of understanding the systems built. Experiments can provide: (i) preliminary confirmation of parts of a reasoning theory; (ii) suggestions of possible modifications to the theory, to the test bed environment, and to the robotic system embedded in the environment. Moreover, they can suggest a large number of additional experiments that in order to expand and strengthen the original theory [1].

In this work we propose a multi-agent architecture that integrates visual perception, planning, reaction and execution to solve real world problems. We run a number of experiments using this architecture applied to an assembly test bed domain. The experience of building the robotic system provided important insights about vision and planning and on how to build real world robotic systems.

2. Evolution of visual perception and planning

Traditionally in AI, an intelligent system is viewed as a set of independent cognitive modules. As illustrated in Figure 1, those basic modules are: perception, planning, learning, and execution. Within this view, complex problems are solved by the execution of individual modules that can exchange information through well defined interfaces. The assumption of independent processes brought up enormous difficulties because of the unexpected interactions of those parts, as we show bellow for the two modules: vision and planning.

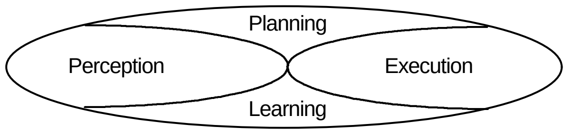

Figure 1: Modular view of an intelligent system.

2.1 Vision

According to the modularized view of intelligence, visual perception has to provide a complete internal description of the world scene to all the other modules. Such approach originated the so called *reconstructive* or *recovery paradigm* [2, 3]. The goal of recovery vision is to derive, from one or more images of a scene, an *accurate* three-dimensional description of the objects in the world and quantitatively recover its properties from image cues such as shading, contours, motion, stereo, color, etc. Thus recovery emphasizes the study of task-independent visual abilities carried out by a passive observer [3].

The recovery paradigm has led to computational theories and algorithms dealing with internal world representations, all of which try to establish general purpose methodologies and representations preserving as much information as possible. However, implementing these ideas has often led to unsatisfactory results. In fact, thanks to the recovery approach we now understand that "vision is an underconstrained problem, i.e., an image does not contain enough information for a complete and unambiguous reconstruction of a 3-D scene" [2]. Traditional approaches have to be improved upon: (i) by increasing run-time efficiency in generating a useful world model, and (ii) by imposing enough constraints to the vision problem. This has led to an alternative approach, called *purposive approach* [4], which embodies the following features:

- Visual systems are *active*, they have to control the image acquisition process, introducing constraints that facilitate the recovery of information about the 3D scene [4];
- Perceptual systems have a relationship with the surrounding world. An active observer, which wants to reconstruct an accurate and complete representation of the world, needs a large amount of computational power. The best way to implement its relationship with the world is by determining, through the vision system, what information derived from the image should be used and what corresponding representation is needed [4]. This depends on the *tasks* the system has to carry out, i.e., on its *purpose*.

According to the purposive approach vision should not be considered as a self-contained module, but as an entity containing other intelligent capabilities, i.e., planning, reasoning and learning, all of which cooperate to solve specific tasks. Intelligence should not be divided into isolated cognitive modules, but decomposed in terms of *behaviors* [4]. In Figure 2 each ring corresponds to a specific decomposed behavior. For example, a robot that navigates while avoiding collisions shows a behavior that can involve planning, perception, execution and learning.

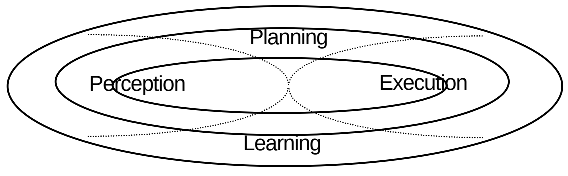

Figure 2: Decomposition of a system into behaviors.

2.2 AI planning

From the early days of AI planning, the idea of dividing the original problem into smaller ones has shown to be an efficient way to obtain some important results. The strategy of dividing a problem into smaller subproblems has been explored in the classical, deliberative, planning literature: first with the use of strategies such as divide-and-conquer and means-ends-analysis from the very early planning systems; next by solving a goal at a time and subgoaling from the partial-order planners; and finally with goal decomposition from the HTN planners [5]. Although those strategies brought on extra problems like non-linearity, goal interaction and constraints inconsistency, they contributed to gradually reduce the complexity of the classical planning problem for different classes of domain applications [6].

The same happened in reactive planning, where the idea of reducing a problem came first with a collection of simple, interacting and dedicated behaviors. The claim of such systems was that the scalability of the system to higher cognitive functions can arise from the organization of several dedicated behaviors [7]. However, scalability is questionable, since most work done with strictly behavioral robots is not based on complex sensors and does not go beyond navigation tasks [8].

We may consider the deliberative and reactive planning approaches as two independent problem solving methods that can be combined to solve more complex problems. Recent contributions [8, 9, 10] have proposed a hybrid architecture to combine the deliberative approach (containing a symbolic world model, developing plans and making decisions), and the reactive one (capable of reacting to events that occur in the environment without engaging in complex reasoning). A complex system can be built out of a hierarchy of subsystems, so that the higher levels process more abstract information than the lower levels. To ensure fast responses to important environmental events, the reactive agents often are assigned to the lower levels of the hierarchy, meaning that they have precedence over the deliberative agents.

Existing research has not yet produced an ultimate paradigm for the distribution and coordination of the skills required for intelligent robotic systems acting in the real world. Some of the desirable features that a distributed planning approach should have are:

1. the robot actions should be of two types: reactive actions and actions selected as a result of a plan generation;
2. the robot should be able to combine at run time sensing and action activities in order to create a complex goal oriented behavior.

We propose a multi-agent architecture that combines both deliberative and reactive planning, capable of taking advantage of environmental and task constraints, and yet flexible enough to respond to dangerous situations and failed expectations. One advantage of this architecture is that the expected behaviors of the system are modularized, allowing the design of different tasks to work in different contexts.

3. The test bed application domain

In most real assembly lines a robot first locates each part through a vision system that recognizes and gives information about the positions of the parts. With the given sensorial information the robot can start to execute a plan to satisfy its original assembly goal. This task should be executed continuously in the sense that new assembly parts can be placed on the table by a human or another robot. However, the human could also place a "trash" object, not related to the assembly task, which can disturb execution of the task. If the vision system detects parts that do not belong in the assembly task, the robot is supposed to clean the work area.

An essential capability of the robot should be to detect and avoid possible collisions between the robot arm and a human (or another robot), while it is executing the cleaning or assembly task. In order to avoid collisions, both the cleaning and assembly tasks can have their execution interrupted until the work area is free of collision contingencies.

An application example was implemented in the LSI Flexible Assembly Cell [11]. In Figure 3 we show the configuration of the cell that served as a test bed for our application domain.

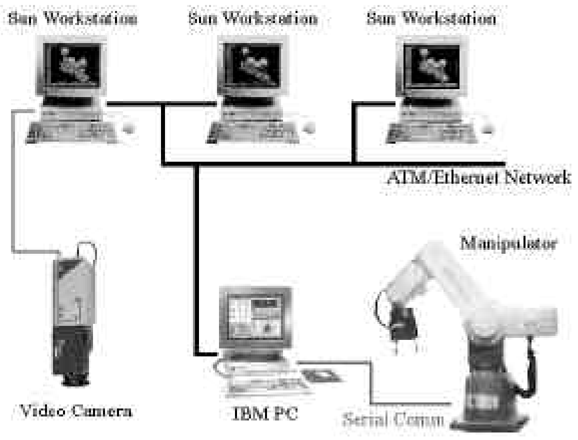

Figure 3: Configuration of the LSI cell used for the assembly domain.

Given a number of parts on a table, mortises and tenons, the goal is to find the matching pairs in order to join them. The assembly pairs can have different sizes, consequently the matching has to be done between pairs of the same size. While the main task is being executed, unexpected human interactions can happen. A human can change the configuration on the table by adding new parts to be assembled or some "trash" objects which cause the robot to perform the cleaning task.

The key feature of the assembly robot is not so much its ability to reason about assembly, but rather its ability to choose timely and effective actions to cope with an uncertain and changing environment. Assembly on this test bed can be characterized as complex and reactive planning task since a number of subtasks are carried out:

- **To generate and execute complex plans**

The assembly domain is a typical case of a planning task. Assume that this domain is represented by a hierarchy of tasks, which describes how to decompose higher level tasks into lower level ones. The lower level tasks correspond to sensing and action. From the point of view of reducing the complexity of the problem, we assume further that to each domain of the solution corresponds a desired behavior which is independent of any other behavior whatsoever. To each desired behavior a hierarchy of tasks is defined. Therefore, we can partition the solution into three domains with a task hierarchy assigned to each, namely the assembly task, the cleaning task and finally the collision avoidance task.

Figure 4 illustrates some of the decomposition tasks from the assembly domain. The assembly task can be decomposed in the subtasks: *Scan for free parts, Select pair, Pick-up object, Move object* and *Join pair.* The subtask *Scan for free parts* can be decomposed into: *Capture image, Detect objects* and *Calculate size and position.* Finally, the *Detect objects* subtask can be decomposed into: *Recognize mortises* and *Recognize tenons*.

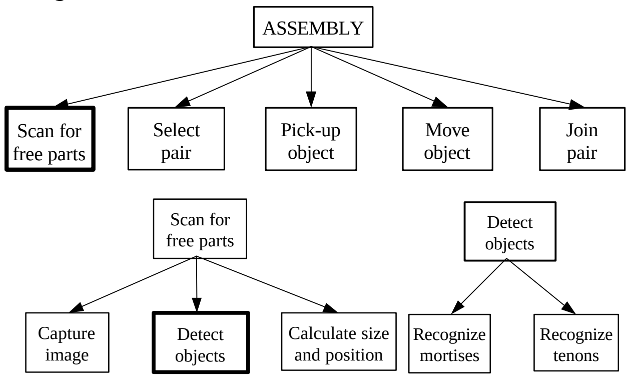

Figure 4: Examples of task decomposition.

The difficulty in the execution of the assembly task rests on possessing adequate image processing and understanding capabilities, appropriately dealing with collision avoidance interruptions and human interactions with the configuration of the work table..

- **To perform online resource allocation**

A resource is defined as a part of the system that can be time-shared by different problem solving processes. The resources to be shared in the assembly application are the camera and the manipulator. both the camera and the manipulator are shared by the three concurrent task hierarchies: assembly, cleaning and collision avoidance. No conflicts arises due to a request for the camera resource by any of the three hierarchies. On the other hand, the manipulator is highly disputed and hence the tasks have to obey a specific policy for conflict resolution, since only one task should control the manipulator in a given moment. The collision avoidance task should have the highest priority in order to prevent accidents, with the cleaning task second with a higher priority than the assembly task.

- **To sense the world**

The robot posses the vision capability to sense the world, i.e., to recognize and locate the assembly parts, trash and to detect movement in the scene.

- **To deal with problems as they arise in real-time**

The collision avoidance task has to be performed in real-time to protect the human operator and preserve the devices.

- **To reason with incomplete information and unpredictable events**

Human interactions can happen at any time interfering with the Assembly task in different and unpredictable ways: the operator can add trash or new assembly parts and the robot has to be able to properly deal with that.

4. The VIBRA architecture

The VIBRA - VIsion Based Reactive Architecture can be viewed as a society of Autonomous Agents (AAs), each of them depicting a problem-solving behavior due to its specific competence, and collaborating with each other in order to orchestrate the process of achieving its goals [12].

The term *agent* assumes a variety of meanings in AI. We will rely on the definition of agents as "computational systems that inhabit some complex dynamic environment, sense and act autonomously, and by doing so perform a set of goals or tasks for which they are designed" [13].

Multi-Agent Systems (MAS) define the agent's organization in a society, where relationships of authority, communication, control, and information flow are described.

The explicit use of social rules in the definition of an agent enables it to achieve its dynamically-acquired goals without interfering with others. As conditions change, new agents can be implemented and activated in the society during the system execution. On the other hand, those agents that are not performing well can be disabled.

Several advantages of using MAS to build complex systems are listed in [8, 14], wich includes: modularity, parallelism, robustness, scalability, simpler programming, flexibility and reusability.

The VIBRA architecture is a multi-agent system, which processes incoming asynchronous goal requests dictated by sensory data, prioritizes them, temporally suspends lower priority actions, and interleaves compatible behaviors.

VIBRA is proposed as a flexible architecture for the development of different visually guided robotic tasks, as it contains specialized vision knowledge that has been accumulated on previous research work [15].

To deal with interactions among the agents, the society is controlled by a policy involving *conducting rules* and an *authority structure*. This policy enables the agents to decide which agent should have the control of a resource at each moment. The authority structure defines the agent's priority level in the use of a specific resource. The authority structure is domain dependent: the priority levels vary for each agent and each resource. In general, reactive tasks should have a precedence over deliberative tasks, and in this way the authority structure is a rank ranging from the most reactive agent to the most deliberative one.

The conducting rules define how the authority structure can be used to control the communication and share resources among agents. In the VIBRA architecture we adopt the following three simple rules:

Rule # 1: Only one agent can control a resource in a given moment.
Rule # 2: At any moment any agent can request control of a resource from an agent with lower authority than itself.
Rule # 3: An agent can only request control of a resource from a higher authority agent if that agent is releasing control.

4.1 Structure of an Autonomous Agent

An AA is composed of eight components (Figure 5):

1. *Communicator module*: responsible for interactions between AAs and the planner/executor module. It knows about protocols, languages and society policy.
2. *Planner/Executor:* responsible for generating and executing a plan to solve the task that will exhibit the behavior expected from the AA. The complexity of this module depends on the competence of the agent.
3. *Primitive Agents (PAs)*: executes simple tasks, including sensing and acting. The AA can only receive sensorial information by activating the respective PA.
4. *Protocols and languages of interaction:* define the communication capability of the agent.
5. *Authority structure:* consists of the relation resource/agent priority level.
6. *Conducting rules of the society*: define priority levels for each pair resource/agent.
7. *Set of AAs in the society*: lists the AAs of the society.
8. *Symbolic representation of the world*: represents the knowledge about the environment needed by each agent.

This model is used to define all AAs in the society, no matter what their behavior is. A special agent in the society can create, change or destroy the society, by adding or deleting agents, and controlling the resources at the initialization or termination phase of the society.

In the next section we detail the communication language, an essential feature in a multi-agent environment.

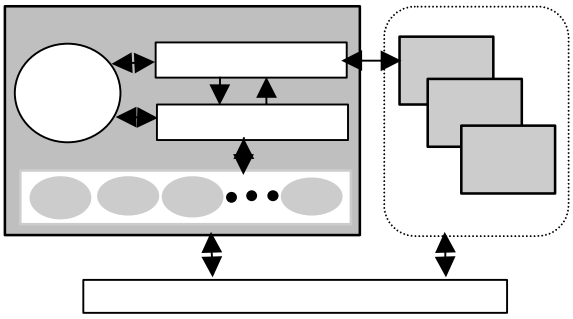

Figure 5: Agent model.

4.2 AA's Communication Languages

The Communication Language among AAs in the society is defined by the following message prototype:

<interaction> ::= *<nature><type><content>*

The nature of the message can be *decision* or *control*. The control messages are used in the initialization, modification and termination of the society and consist of: *addAgent, deleteAgent, acknowledge, inform, requestAll, freeAll.* The decision messages are used to define which agent will take the control of a resource at a time, and they are: *request, transfer* and *free* (see Table 1).

A different set of messages is used for the internal interaction between the planner/executor (EX) module and the communicator (C) module. The *nature* of this communication language is *information*, and they are: *request, free, received, lost, halt, information* (Table 2).

<nature>	**<type>**	**<content>**	**Description**
control	addAgent	<name of the new agent> <new authority structure >	Add a new agent in the society
	deleteAgent	<name of the agent>	Delete an agent
	inform	<resource> <free | in_use> < name of the agent>	Inform about the allocation of the resources in the society
	requestAll		Request all resources to all agents in the society
	freeAll		Free all resources
	acknowledge		Acknowledge the insertion or deletion of an agent
decision	request	<name of the requesting agent> <resource>	Request a resource
	free	<name of the agent> <resource>	Inform when a resource allocated by the agent is not in use
	transfer	<name of the requesting agent> <resource> <state of the resource>	Inform all agents about the new resource controller

Table 1: Types of messages defined in the communication language for interactions among agents in the society.

<nature>: *information*			
From → To	**<type>**	**<content>**	**Description**
EX → C	request	<resource>	Inform that a resource is needed
EX → C	free	<resource>	Inform that the resource is not in use anymore
C → EX	received	<resource> <state of the resource>	Inform that the resource was acquired and its state
C → EX	lost	<resource>	Inform that another AA is taking the resource
C → EX	halt		Halt execution
EX ↔ C	information	<resource> <state of the resource>	Update the state of the resource

Table 2: Types of messages changed between the AA communicator module and its planner/executor module.

5. The assembly application in the VIBRA architecture

In this application, AAs were allocated on several workstations, executed as independent and parallel distributed processes, communicating through the Ethernet/ ATM network. The camera is fixed above the work area.

For the assembly application, we have defined three different behaviors, each one corresponding to an AA:

Assembler agent: to accomplish the assembly task, picking up pieces (tenons) on the work table with the manipulator and putting them in a desired location (mortises);

Cleaner agent: to clean up the work area, taking away trash objects added by humans or another manipulator;

CollisionAvoider agent: to avoid collisions of the manipulator with objects moving in the work area.

5.1 Experimental results

The experiment focus on the interaction between the *Cleaner* and *CollisionAvoider.* A human can place a trash object (a circular piece) on the work table, at any moment, activating the *Cleaner*; each time a moving object is detected, the cleaning task is interrupted and the *CollisionAvoider* is activated; when the moving object disappears from the scene, the *Cleaner* resumes its work, finishing the trash removal.

Different aspects of the experiment results are shown in Figure 6 and Figure 7. Figure 6 shows the image sequence and Figure 7 the exchanged messages between the AAs.

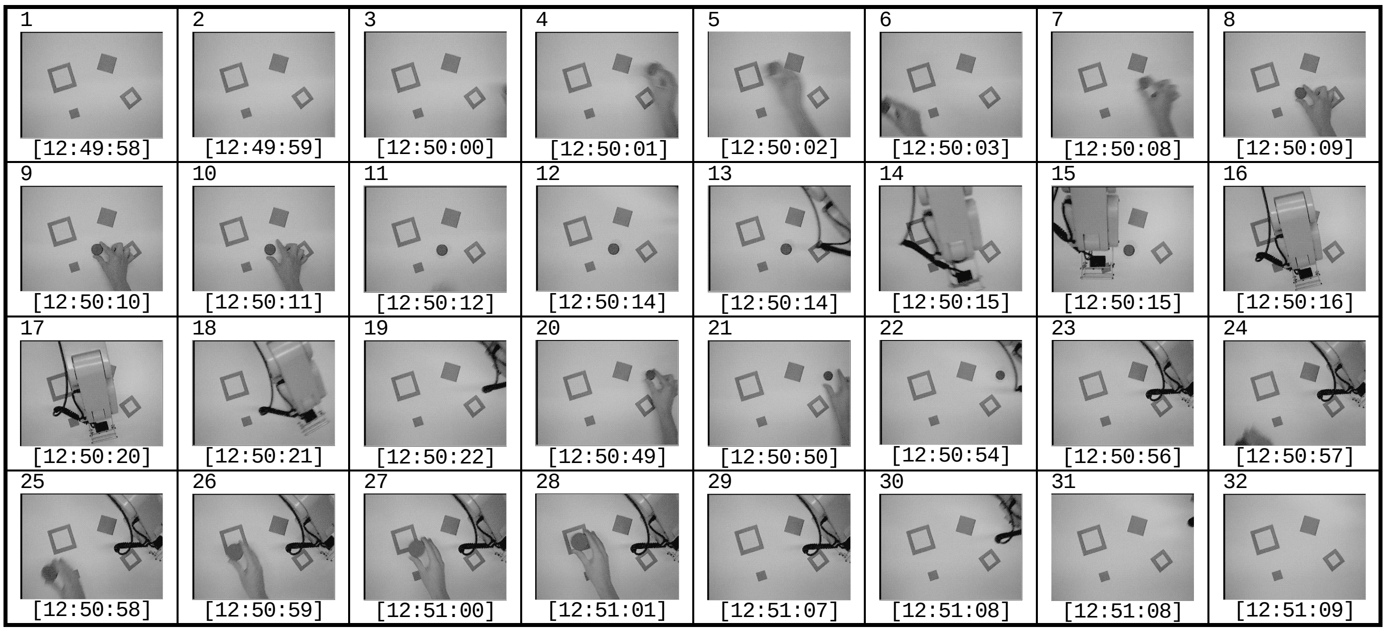

Figure 6: Sequence of images showing 2 unknown objects being placed and removed from the work area.

```
Agent Communication Log Archive
Started at host nausika at Fri Dec 12 12:49:27 1997
[ AGENT NAME ][TIME] and Message
[assembler          ][12:50:01] Message received From: collisionAvoider. Message: (request (collisionAvoider)
(manipulator)).
[assembler          ][12:50:01] (transfer (collisionAvoider) (manipulator) (initial resource working state))
[assembler-execution ][12:50:01] LOG MESSAGE: Resource Manipulator Lost
[collisionAvoider-exe][12:50:01] LOG MESSAGE: Resource Manipulator Received
[collisionAvoider    ][12:50:12] (free (manipulator))
[collisionAvoider    ][12:50:14]  Message Received From:: cleaner. Message: (request (cleaner) (manipulator)).
[collisionAvoider    ][12:50:14] (transfer (cleaner) (manipulator) (initial resource working state))
[cleaner-execution   ][12:50:14] LOG MESSAGE: Resource Manipulator Received
[cleaner             ][12:50:25] (free (manipulator))
...
[collisionAvoider-exe][12:50:47] LOG MESSAGE: Resource Manipulator Needed
[cleaner             ][12:50:47]  Message Received From: collisionAvoider-execution. Message: (request
(manipulator)).
[cleaner             ][12:50:47] (transfer (collisionAvoider) (manipulator) (free working state))
[collisionAvoider-exe][12:50:47] LOG MESSAGE: Resource Manipulator Received
[collisionAvoider    ][12:50:52] (free (manipulator))
[cleaner-execution   ][12:50:54] LOG MESSAGE: Resource Manipulator Needed
[collisionAvoider    ][12:50:54] (transfer (cleaner) (manipulator) (free))
[cleaner-execution   ][12:50:54] LOG MESSAGE: Resource Manipulator Received
[collisionAvoider-exe][12:50:57] LOG MESSAGE: Resource Manipulator Needed
[cleaner             ][12:50:57]  Message Received From:: collisionAvoider. Message: (request
(collisionAvoider) (manipulator)).
[cleaner             ][12:50:57] (transfer (collisionAvoider) (manipulator) (object trash held))
[collisionAvoider-exe][12:50:57] LOG MESSAGE: Resource Manipulator Received
[cleaner-execution   ][12:50:59] LOG MESSAGE: Resource Manipulator Needed
[collisionAvoider    ][12:51:06] (free (manipulator))
[collisionAvoider    ][12:51:06]  Message Received From:: cleaner, Message: (request (cleaner) (manipulator)).
[collisionAvoider    ][12:51:06] (transfer (cleaner) (manipulator) (object trash held))
[cleaner-execution   ][12:51:07] LOG MESSAGE: Resource Manipulator Received
[cleaner             ][12:51:12] (free (manipulator))
```

Figure 7: Message exchange among agents.

The system is started at 12:49:27, and the actions *(scan-for-static-object trash)* and *(detect-moving-object object)* are started by the *Cleaner* and *CollisionAvoider* (Figure 6, images 1 and 2). At 12:50:00 an unknown object enters the work area and keeps moving for 10 seconds. During this time, the actions *(freeze manipulator)* and *(keep-on-moving* *object)* are executed simultaneously by the *CollisionAvoider*, in parallel with *(scan-for-static-object trash)* by the *Cleaner*.

The *CollisionAvoider* (images 3-12) detects movements in the work area and requestes control of the manipulator to freeze it. The *CollisionAvoider* requests the manipulator at 12:50:00 (image 3) and receives it at 12:50:01 (image 4). In parallel the *Cleaner* agent (images 1-11) keeps on scanning for static objects. The human interaction ended at 12:50:12 (image 11) and the *CollisionAvoider* frees control of the manipulator allowing *Cleaner*, who detects a trash piece on the table, to request the manipulator (image 12). The action *(detect-moving-object object)* restarts.

Cleaner's actions *(pickup-object trash trash-position)* (images 12-17), *(move-object trash trash-can-position)* (images 17-19) and *(drop-held-object)* (image 19) are executed, removing the trash piece from the work table. Then the action *(scan-for-static-object trash)* restarts.

At 12:50:47 another unknown object enters the work area, and is placed on the work table at 12:50:50 (images 20-21) and the *CollisionAvoider* operates. When the object is placed on the table, the *Cleaner* begins to remove it, starting to execute the action *(pickup-object trash trash-position)* (images 22-23). During the execution of this cleaning action, another unknown object enters the work area (at 12:50:57, image 24), and *CollisionAvoider* operates (images 24-28). At 12:51:05, the object leaves the work area and the *Cleaner* finishes its action *(pickup-object trash trash-position)* (image 29) and executes *(move-object trash trash-can-position)* and *(drop-held-object)* (images 30-32). By analyzing the data generated some response times can be determined:

1. Table 3 presents the maximum and minimum speed an object can have so that the *CollisionAvoider* can detect it, when it is alone in the society. Using low resolution image, maximum speed is higher because the agent acquires more images per seconds, and at medium resolution, minimum speed is lower since we can have higher precision images;

Img Resol. (pixels)	Max. Speed (m/s)	Min. Speed (cm/s)
Low (64 x 48)	6.05	39
Medium (120 x 80)	1.1	3.6

Table 3: Maximum and minimum speed for moving object.

2. Table 4 shows the reaction time the *CollisionAvoider* takes when a moving object enters in the work area. The reaction time is dependent on the image resolution and on the network throughput, as images need large bandwidth;

3. Table 5 presents average times for the *Cleaner* to complete its actions, when not interrupted by another agent. Network delay is high because *Cleaner* and the *Image Acquisition* agents are not hosted on the same machine and *Cleaner* works with (320 x 240) images. As expected, the robot working time is the most critical.

Configuration	Proc.	Com.	Net. delay	Total
CA (64 x 48)	0.05	0.05	0	0.1
CA (120 x 80)	0.35	0.05	0	0.4
CA + *CL* (120 x 80)	0.35	0.05	0 to 1.5	< 1. 9

Table 4: Reaction Time in seconds for *CollisionAvoider (CA)* alone and with the *Cleaner (CL)* in different configurations.

Processing	Communication	Net. delay	Working time
less than 1	0.05	2 to 4	10 to 15

Table 5: Average processing time (in seconds) for the Cleaner agent.

6. Discussion and Conclusion

This paper describes the design of an implemented multi-agent architecture, VIBRA, which is a vision-based architecture that can offer a proper framework for building reactive, vision-based planning applications. From a computer vision point of view, VIBRA is a sufficiently flexible tool to create specialized and efficient visual routines to solve specific tasks efficiently. In this way, it can be used as a framework for knowledge acquisition, development and design of new robotic applications.

By applying VIBRA to the design of the assembly application we have learned some important experiences on building a real world robotic system. Some of these important experiences are shared by other researchers, who also used the multi-agents approach in real world applications [9, 14, 16], others resulted from our work. Such knowledge can be used to provide support on the development of new real world planning systems [17] and they consist of:

- To allow agents coordination, it is important to decide about division of labor and organization: defining tasks, selecting which agent does each task, and defining when it executes the task;
- The languages and concepts used for task description and formulation will affect how tasks can be decomposed, and what dependencies explicitly exist among tasks. The same task described from different perspectives may require different partitioning and different skills.
- A distribution of tasks among agents requires the tasks to be formulated and described in a way to provide a better possibility of decomposition, allowing a natural distribution among agents. Tasks requiring more resources or knowledge than an agent can possess must be decomposed. We based our decomposition on the definition of independent cognitive behaviors.
- Dependencies among subproblems affect agent design in terms of predicting possible data flow, decision processes

and actions.

- Conflicts over interacting actions and shared resources may place ordering constraints on agent activities, restricting decomposition choices and creating the need to reconsider decomposition in different dimensions, such as temporal, spatial, or levels of abstraction. Those are the parameters that a designer has to adjust in the society behavior in order to solve the global task goal.
- The society rules depend on the type of the available resources and on what are the needs of the agents over the different resources.
- The authority structure of a society is deeply related to the dependence and precedence among tasks conduced by the AAs. To ensure fast responses to important environmental events, precedence is given to the reactive tasks over the deliberative ones.

Analyzing existing theories on how a robotic system should solve their problems, we have reached some important insights about the vision and planning areas. As we have described in Section 2, by combining planning and perception the knowledge of the situation will be directed, rather than predicted (or previously specified). By doing so most of the uncertainty and some of the incompleteness problems can be solved. In VIBRA this is done by combining reactive behaviors (e.g., the collision avoidance) with the tasks in the application domain (e.g., join pair). The combination of deliberative and reactive planning requires both to be run as independent processes while solving complex tasks. VIBRA is a multi-agent architecture proven to be efficient allowing this combination, fulfilling the AI challenge of building intelligent goal-driven reactive systems.

Acknowledgements

We thank Felipe M. Pait and the anonymous reviewers for their helpful comments, and CNPq for the financial support.

References

[1] S. Hanks; M.E. Pollack; P.R. Cohen. Benchmarks, Test Beds, Controlled Experimentation, and the Design of Agent Architectures. *AI Magazine*, Winter :17-42,1993.

[2] D. Marr. Vision. New York, Freeman, 1982.

[3] T. Dean; J. Allen; Y. Aloimonos. Artificial Intelligence: Theory and Practice. Benjamin/Cummings Publishing Co., Redwood City, CA, 1995.

[4] Y. Aloimonos. What I have learned. *CVGIP: Image Understanding*, 60(1):74-85, July 1994.

[5]. K. Erol, J. Hendler, D.S. Nau. UMCP: a sound and complete procedure for hierarchical task-network planning. In: Conf. on Artificial Intelligence Planning Systems. 1994.

[6] A. Barret, D. Weld. Partial-order planning: evaluating possible efficiency gains. *Artificial. Intelligence*, 67, 1994.

[7] R.A. Brooks. Intelligence without representation. *Artificial Intelligence*, 47: 139-59,1991.

[8] A. D. Medeiros, R. Chatila. Priorities and data abstraction in hierarchical control architectures for autonomous robots. In: WIR, Brasília, BR., 207-220, 1997.

[9] M. Garcia-Alegre, F. Recio. Basic Agents for Visual/Motor Coordination of a Mobile Robot. In Proc. of AGENTS'97, Marina del Rey, CA, USA. ACM, 1997.

[10] M. Wooldridge, N. Jennings. Intelligent Agents: Theory and Practice. *Knowledge Engineering Review*, 1995.

[11] M. Rillo, A.H. Rillo, L.A.R. Costa. LSI Assembly Cell. In IFAC Symposium on Information Control Problems in Manufacturing Technology, Toronto, 361-5, 1992.

[12] R.A.C. Bianchi, A.H.R.C. Rillo. A distributed control architecture for a purposive computer vision system. In: 2nd. Symp. on Intelligence in Automation and Robotics, IEEE, Rockville, MA, USA, p. 288-294, 1996.

[13] P. Maes. Artificial Life meets entertainment: life like autonomous agents. *Communications of the ACM*, 38(11): 108-114, 1995.

[14] A.H. Bond, L. Gasser. Readings in Distributed AI. Morgan Kaufmann, San Mateo, CA, 1988.

[15] A.H.R.C. Rillo. 3D object recognition using a decision hierarchy. In: Proc. SPIE 1771, A.G. Tescher (ed.), Lockheed Palo Alto Research Lab., CA, p.225-33, 1993.

[23] O. Boissier, Y. Demazeau. ASIC: An architecture for social and individual control and its application to Computer Vision. In European Workshop on Modeling Autonomous Agents. p.107-118, 1994.

[16] M.C. Neves, E.A. Oliveira. Control Architecture for an Autonomous Mobile Robot. In Proceedingsof the AGENTS'97 Conference, Marina del Rey, CA, ACM, 1997.

[17] L.N. Barros, J. Hendler, V.R. Benjamins,. Par-KAP: a Knowledge Acquisition Tool for Building Practical Planning Systems. In: IJCAI, Japan, 1997.