The VIBRA Multi-Agent Architecture: integrating purposive vision with deliberative and reactive planning.

Anna H. Reali C. Rillo[1] Leliane N. Barros[2] Reinaldo A. C. Bianchi[2]

[1] Department of Computer Engineering
[2] Laboratory of Integrated Systems
University of São Paulo
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

In the view of the inherent difficulty of solving planning tasks in a dynamic world application, AI researchers have claimed that running experiments is a good way to accomplish the goal of understanding the built systems. Experiments can provide: (i) preliminary confirmation of some 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. In this work we propose a multi-agent vision-based architecture to solve complex sensor-based planning tasks. We present a test bed implementation, with skills such as vision and collision avoidance, to run some experiments in the proposed architecture. We demonstrate how the system can successfully execute complex assembly plans while dealing with unpredictable events and imprecise information, with no significant cost in run-time efficiency. Another important result from this work is that the experience on building such robotic system provided some important insights about the vision and planning areas, suggesting a new interpretation and comparative analysis of those two reasoning theories.

Keywords*: purposive vision, agent architecture for planning and execution, deliberative planning, reactive planning.*

1. Introduction

To solve complex problems in a real dynamic world, an intelligent system must be able to interact with its environment by gathering information about its surrounding world through sensing, processing and transforming it into different levels of representation. Such processed information is then used by the system to interact back with the environment through robot actions [Fermüller: 93].

The idea of constructing such intelligent system has been the ultimate goal of the Artificial Intelligence (AI) area and has led the AI 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); to reason with incomplete information and unpredictable events.

In the view of the inherent difficulty of the problem and the limited results obtained from AI in well behaved artificial domains, the AI

researchers have claimed that running experiments is a good way to accomplish the goal of understanding the built systems. Experiments can provide: (i) preliminary confirmation of some 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 need to be conducted to expand and strengthen the original theory [Hanks et al.:93].

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.

Another important result from this work is that the experience on building such robotic system provided some important insights about vision and planning areas, suggesting a new interpretation and comparative analysis of those two reasoning theories.

The remainder of this paper is structured as follows. In Section 2 we present the evolution of the visual perception and planning areas. Section 3 presents the chosen domain application that will serve as a test bed for our proposed architecture. In Section 4 we make a brief discussion about how the choice of a Multi-agent architecture was made. In Section 5 we introduce VIBRA, a multi-agent vision-based reactive architecture. In Section 6 we run some experiments in our test bed depicting the behavior of the VIBRA system when applied to a dynamic environment. Finally, in Section 7 we discuss some important insights acquired in this work and the comparative analysis between the vision and planning theories.

2. The evolution of the visual perception and planning areas

Traditional works on solving complex tasks had led AI researchers to break the view of an intelligent system into a set of independent cognitive modules in order to study them individually. As illustrated in Figure 1, those basic modules are: perception, planning, learning, and execution.

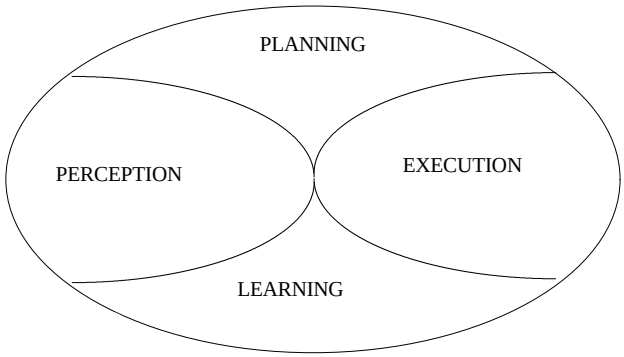

Figure 1: Modular view of an intelligent system.

Within this view, complex problems are solved by the execution of individual modules that can exchange information through well defined interfaces. As we know the supposition 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.

2.1 The vision evolution

One of the most important perceptual sense is vision. According to the modularized view of intelligence, visual perception has to provide an internal and complete description of the world scene to all the other modules. Such approach originated the so called *reconstructive* or *recovery paradigm* [Tarr, Black: 94; Jolion: 94; Marr: 82; Dean et al.:95]. The recovery vision goal 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 [Dean et

al.: 95].

Important results have been achieved with the recovery paradigm in terms of computational theories and algorithms dealing with internal world representations, all of them trying to establish general purpose methodologies and representations preserving as much information as possible. However, the results obtained by implemented working systems when applied to real world domains were not satisfactory. In fact, one important understanding of the recovering approach is 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" [Marr:82; Jolion:94]. That means, the traditional generalized approaches had to be improved: (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* [Aloimonos:94], which embodies the following features:

- visual systems are *active*, i.e., they have to control the image acquisition process, introducing constraints that facilitate the recovery of information about the 3D scene [Aloimonos:94]. Bajcsy [Bajcsy:88] summarizes this idea as "we do not just see, we look".
- a perceptual system has a relationship with the surrounding world. An active observer, that 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 the determination, through the vision system, of what information derived from the image should be used and what corresponding representation is needed [Aloimonos:94]. This depends on the *tasks* the system has to carry out, i.e., on its *purpose*.

This way, according to the purposive approach to build complex perceptual systems, 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 them cooperating to produce a behavior that solve specific tasks. The intelligence should not be divided into isolated cognitive modules, but it should be decomposed in terms of *behaviors* [Aloimonos:94]. Figure 2: Decomposition of a system into behaviors. illustrates the decomposition where each ring corresponds to a specific behavior. For example, a robot that navigates while avoiding collisions shows a behavior that can involve planning, perception, execution and learning.

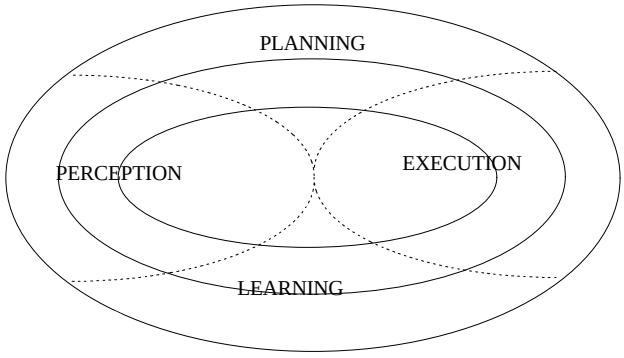

Figure 2: Decomposition of a system into behaviors.

2.2 The AI planning evolution

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. This strategy for problem reduction has been largely explored in the classical planning area: 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 the goal decomposition from the HTN planners. Although those strategies brought up some 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 [Minton et al.: 91; Barret et al.: 94].

The same thing happened in the reactive planning area 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 could arise from the organization of several dedicated behaviors [Brooks: 91]. However, these scalability is questionable, since most works done with strictly behavioral robots are not based on complex sensors and do not go beyond navigation tasks [Medeiros et al.: 97].

These two planning approaches, the deliberative and the reactive planning, can be seen as two independent problem solving methods that, when combined, can solve more complex problems. Some works [Firby et al.: 95; Garcia-Alegre et al.: 97; Medeiros et al.: 97; Wooldridge et al.: 95] have proposed a hybrid architecture to combine the deliberative approach (containing a symbolic world model, developing plans and making decisions), and a reactive one (capable of reacting to events that occur in the environment without engaging in complex reasoning). In this way, a complex system can be built out of many subsystems, arranged into a hierarchy so that the higher levels are supposed to 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 some kind of precedence over the deliberative agents.

However the existing research experience has not yet produced an ultimate paradigm for the distribution and/or coordination of the skills required for intelligent robotic systems when 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 is flexible enough to respond to dangerous situations and failed expectations. An important 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: an Assembly task

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

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 [Rillo et al.: 92]. In Figure 3 we show the configuration of the cell that served as a test bed for our work.

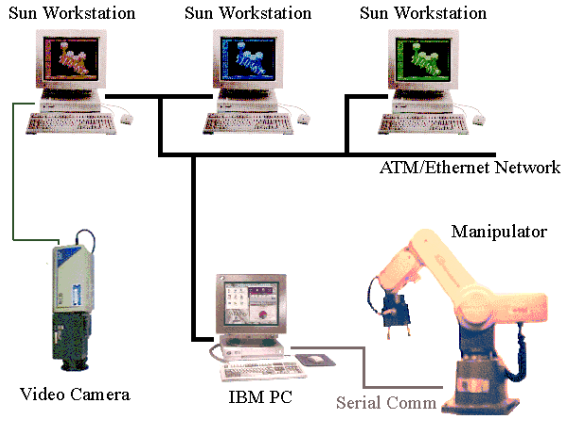

Figure 3: The configuration of the LSI cell used for the Assembly application domain.

The application consists of: 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. This test bed can be characterized as a complex reactive planning task since it carries out a number of important features:

- **To generate and execute complex plans to solve specific task goals**

The Assembly domain, for instance, is a typical case for a planning task. Assume that this domain is represented by a hierarchy of tasks, which describes how to decompose higher level of 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, as previously mentioned in the Assembly domain, corresponds a desired behavior which is independent of any other behavior whatsoever. To each desired behavior o 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 sub-tasks: *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*.

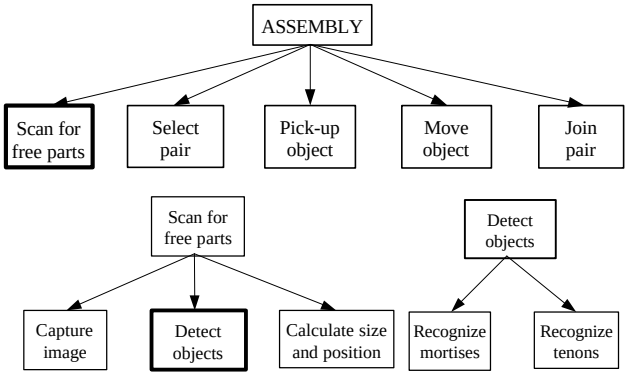

Figure 4: Examples of task decomposition for the Assembly task.

The difficulty in the execution of this 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. The robot has to cope with goals which have been already solved but remain undone, i. e., the existence of disjoined pairs; the addition of more pieces to be assembled; the occurrence of trash objects occluding assembly parts; and having the

manipulator halted during its movements before it achieved the goal.

- **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 arise 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 choice of a problem solving approach

As discussed before, the deliberative approach can not successfully attend the real-time requirements of a dynamic domain. Therefore, rather than building a centralized system to do all the reasoning, some authors ([Firby: 96; Garcia-Alegre et al.: 97; Medeiros et al.: 97; Neves et al.: 97]) have accomplished good results by building a number of computational elements that contains only the data and expertise knowledge necessary to perform their own task allowing the capabilities to be distributed among these elementary building blocks.

The reactive approach follows the same idea of implementing a system as a collection of independent behaviors reminding the distributed approach. However the overall behavior emerging from the organization of several dedicated behaviors do not cope with the requirements of real application problems [Medeiros et al.: 97]. Those systems do not provide symbolic representation for cognitive reasoning, i.e., to reason with a symbolic world model in a deliberative way.

The hybrid architecture combining the deliberative and the reactive approach have been suggested as a promising solution, since it is capable of reacting to events that occur in the environment without engaging in complex reasoning as well as is capable of developing plans and making decisions [Firby et al.: 95; Garcia-Alegre et al.: 97; Haigh et al.: 97; Lyons: 95; Medeiros et al.: 97; Wooldridge et al.: 95].

Some recent results on a Multi-Agent approach for building reactive applications [Firby: 96; Garcia-Alegre: 97] have shown its advantages. Coming from the Distributed Artificial Intelligence field (DAI) with the idea of distributing knowledge and process, this approach proposes models to develop heterogeneous modules, called *agents*, which cooperate and

interact among them. The autonomy of each agent can allow them to intentionally affect the environment in unpredictable ways from other agents point of view, resulting in the uncertainty that may be inherent in the domain.

The term *agent* is generally used in AI, assuming a variety of definitions. For most of this paper, we will rely on the definition of agents as "computational systems that inhabit some complex dynamic environment, sense and act autonomously in this environment, and by doing so perform a set of goals or tasks for which they are designed" [Maes: 95].

Multi-Agent Systems (MAS) define the agent's organization, which are structural entities 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 while not interfering with others. As conditions change, new agents can be implemented and activated in the community 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 [Bond et al.: 88; Stone et al.: 97; Medeiros et al.: 97]:

- Modularity - complex systems can be divided into smaller subsystems that can be designed and implemented separately.
- Parallelism - several independent tasks can be handled by concurrent separate agents, helping deal with limitations imposed by time-bounded reasoning requirements.
- Robustness - greater uncertainty in a task or its components can be handled by spawning redundant subtasks that address a problem from different perspectives.
- Scalability - systems can be built up step by step by adding incrementally new functions, thereby including more and more competencies.
- Simpler programming - from a programmer's perspective the modularity of MAS can lead to simpler programming.
- Flexibility - the design of the system can be easily changed, guided by demonstrated success or failures.
- Reusability - different application tasks can share common subtasks by accessing the same program or reproducing the code.

In the next section we describe a Multi-Agent architecture to solve complex reactive planning tasks, called **VIBRA** - VIsion Based Reactive Architecture. The VIBRA architecture will be used to solve the Assembly domain application described in Section 3. We will show how a multi-agent approach can provide desirable advantages, in special in terms of the modularity and reusability aspects of its components.

5. 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 in order to conduce the society in the process of achieving its goals [Bianchi et al.: 96]. The VIBRA architecture has the ability to process incoming asynchronous goal requests dictated by sensory data, prioritize them, temporally suspend lower priority actions, and interleave compatible behaviors.

An architecture based on Multi-Agent Systems was chosen to facilitate the opportunistic problem-solving needed for complex and ill-structured applications. The Figure 5 illustrates the modularity and distribution aspects of the architecture composed of autonomous agents, a network allowing the communication exchange among them, and sensors and actuators to interact with the world.

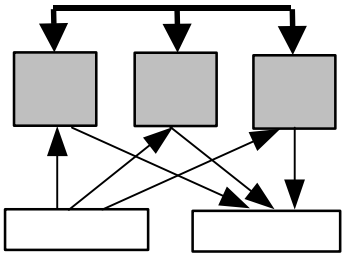

Figure 5: The proposed architecture.

VIBRA is proposed as a flexible architecture to be specially applied to the development of different visually guided robotic tasks once it contains specialized vision knowledge that has been accumulated on previous research work [Rillo: 92; Rillo: 93].

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 one 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. For example, in the Assembly application the manipulator and the camera are shared following different priority levels. As the camera can supply the image without creating any conflict among the agents, the competing tasks have the same priority in their authority structure. On the other hand, to control the allocation of the manipulator, the contingency of the domain imposes the following authority structure: collision has to be avoided before the execution of the cleaning task that again should be done before the Assembly task. In general, reactive tasks should have precedence over the deliberative tasks and in this way the authority structure defined for autonomous agents 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 the sharing of 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 the control of a resource from an agent with lower authority than itself.

Rule # 3: An agent can only request the control of a resource to an agent with higher authority than itself if that agent is releasing the control.

5.1 The structure of an Autonomous Agent

An autonomous agent (AA) is composed of eight important components (Figure 6):

1. *Communicator module*: responsible for the interactions among AAs, reasoning about the society policy; contain knowledge about the protocol and language of interactions among agents (external to the AA) and between planner/executor module defined bellow.
2. *Planner / Executor:* responsible for generating and executing a plan to solve the task that will depict the behavior expected from the AA. This module can vary in its content complexity depending on how complex is the competence of the agent. In fact, the complexity of this module is also a rank ranging from the most reactive agent to the most deliberative one.
3. *Set of Primitive Agents (PAs)*: responsible for very simple tasks, including sensing and acting in the environment. In this way, the AA can only receive sensorial information by activating the respective PA. Their structure is a simpler version of the AA.
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*: defines how to rule the relation resource/agent priority level.
7. *Set of AAs in the society*: lists the AAs that compose the society.
8. *Symbolic representation of the world*: represents the knowledge about the environment that each agent needs in order to perform its behavior.

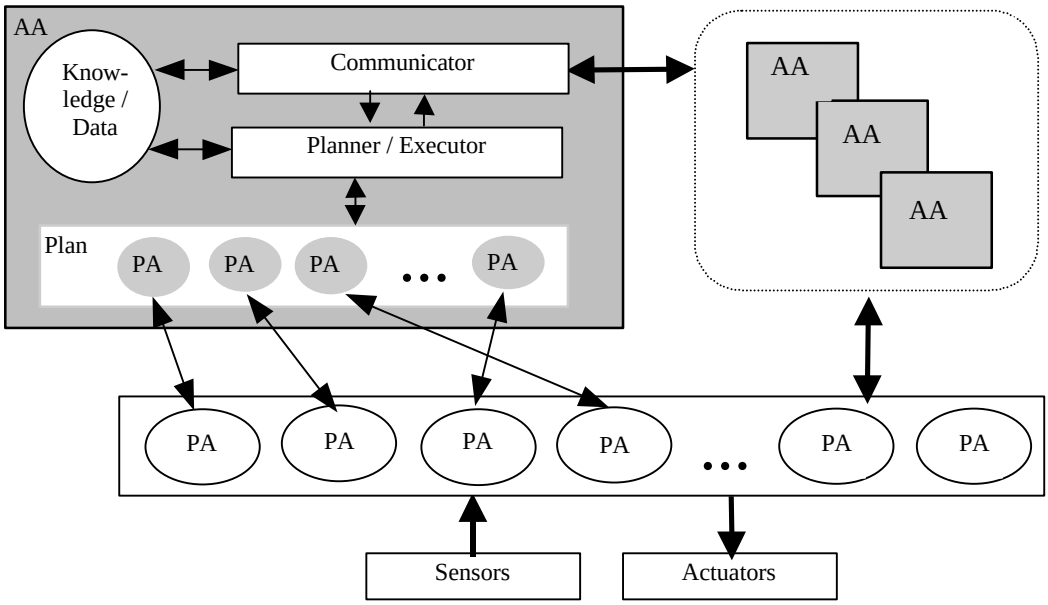

Figure 6: Agent model.

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, since it is an essential feature in a multi-agent environment.

5.2 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 decision messages are used to interact with other agents, deciding which agent will take the control of a resource of the system at a time. There are three types of decision messages: *request*, *transfer* and *free*. The control messages are used in the initialization, modification and termination of the society. There are six types of control messages: *addAgent, deleteAgent, acknowledge, inform, requestAll, freeAll*. Table 1 summarizes the communication language defined for the autonomous agents.

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 there are six types of messages, described in Table 2: *request, free, received, lost, halt, information.*

<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 the society that 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 more in use
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.

6. The Assembly application in the VIBRA architecture

In the present implementation we have decided to create autonomous agents allocated on several workstations, where they are executed as independent and parallel distributed processes, communicating through the Ethernet/ATM network, so we could take advantage of the network availability. The camera is fixed above the work area.

For the Assembly application, we have defined three different behaviors, each one corresponding to an autonomous agent, which are:

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 unwanted objects that humans or another manipulator may have put on this area.

CollisionAvoider agent: to avoid collisions of the manipulator with objects (other manipulators, a human) that move in the work area.

The high level plans used by the defined *Assembler*, *Cleaner* and *CollisionAvoider* agents

in the VIBRA system are described in Figure 9, Figure 8 and Figure 7 respectively.

```
CollisionAvoider
(
     (detect-moving-object object)
     ((freeze manipulator) while (keep-
on-moving object))
)
```

Figure 7: The CollisionAvoider plan.

```
Cleaner
(
     (scan-for-static-object trash)
     (pickup-object trash trash-position)
     (move-object trash trash-can-
position)
     (drop-held-object)
)
```

Figure 8: The Cleaner plan.

```
Assembler

(
     (scan-for-free-parts parts)
     (select-pair tenon mortise)
     (pick-up-object tenon tenon-
position)
     (move-object tenon mortise-position)
     (join-pair tenon mortise)
)
```

Figure 9: The Assembler plan.

6.1 Experimental results

In order to test the VIBRA architecture when performing a real reactive task, we developed an experiment by giving to the robot a cleaning task while avoiding collisions with a human hand. This way we want to test the performance of those two agents, the *Cleaner* and the *CollisionAvoider*, by showing how the interactions were successfully dealt by the VIBRA architecture.

The experiment consists of:

1. A human hand places a trash object (a circular piece) on the work table, activating the *CollisionAvoider*;
2. *Cleaner* commands the manipulator to remove the trash piece;
3. Human hand places another trash object on the work table;
4. During the new cleaning action, a new interference occurs: a moving unknown object enters the work area;
5. When the moving object disappears from the scene, the *Cleaner* resumes its work, finishing the started trash removal.

Different aspects of this experiment are shown in Figure 10, Figure 11 and Figure 12.

Figure 10 shows the image sequence of the experiment. Figure 11 describes the actions executed by the *Cleaner* and the *CollisionAvoider* agents when performing the experiment. Each action shows its corresponding start and finish time so that one can notice the system efficiency while dealing with vision process, reacting to a human hand moving in the scene and executing a planning task as cleaning the table. In Figure 12 we can see the message exchange between the two agents.

Following, a detailed description of the experiment is given.

The system is started at 12:49:27, and the actions *(scan-for-static-object trash)* and *(detect-moving-object object)* are started at *Cleaner* and *CollisionAvoider* (figure 10, 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 at *CollisionAvoider*, in parallel with the *(scan-for-static-object trash)* at *Cleaner*.

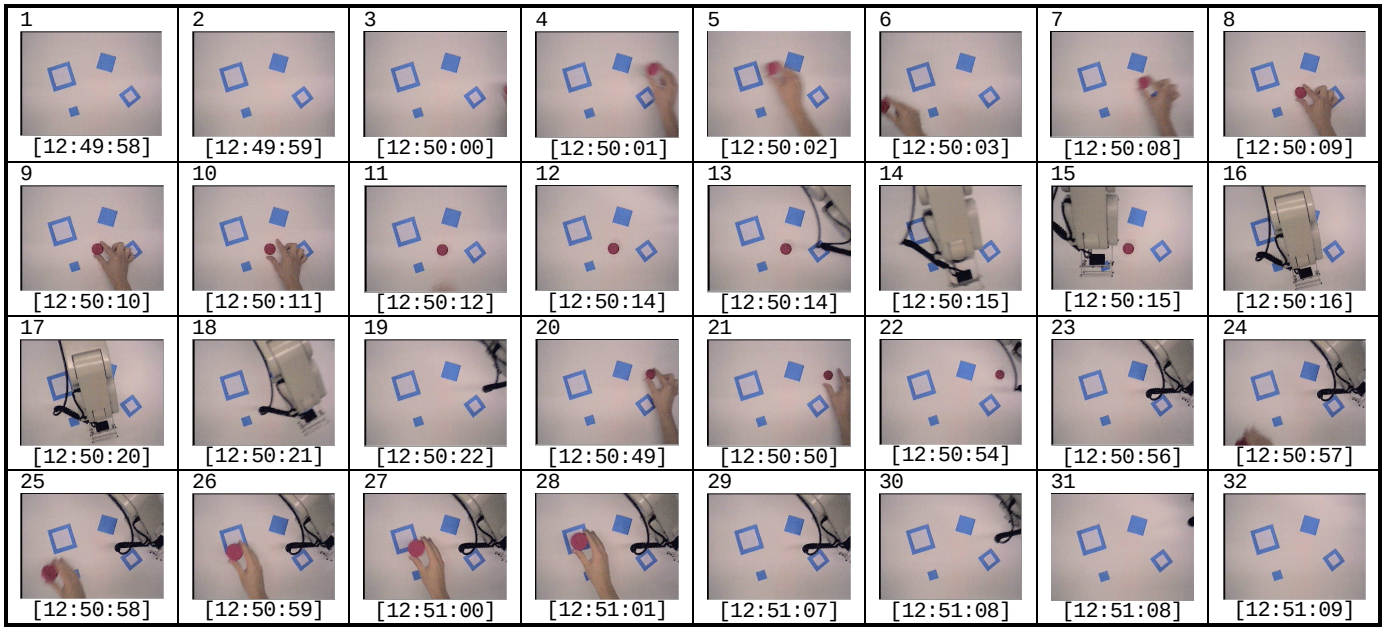

Figure 10: Sequence of images showing 2 unknown objects being placed and removed from the work area.

From image 3 to 12, the *CollisionAvoider* detects movements in the work area and requested the control of the manipulator to freeze its actions. The *CollisionAvoider* requests the manipulator at 12:50:00 (image 3) and received it at 12:50:01 (image 4). In parallel the *Cleaner* agent (images 1 to 11) keeps on scanning for static objects. The human interaction stopped at 12:50:12 (image 11). At this moment the *CollisionAvoider* frees the control of the manipulator allowing *Cleaner* to request the manipulator since it detects a trash piece in the table (image 12). The *CollisionAvoider* 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 *Cleaner* 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 21 and 21). During this time, the respective actions of the *CollisionAvoider* are executed.

```
Agent Communication Log Archive
Started at host nausika at Fri Dec 12 12:49:27 1997
[ AGENT NAME          ][TIME    ]  [TASK AND MESSAGES]
[cleaner-execution    ][12:49:34]  (scan-for-static-object trash) started
[collisionAvoider-exe][12:49:36]  (detect-moving-object object) started
[collisionAvoider-exe][12:50:00]  Resource Manipulator Needed
[collisionAvoider-exe][12:50:00]  (detect-moving-object object) finished
[collisionAvoider-exe][12:50:01]  Resource Manipulator Received
[collisionAvoider-exe][12:50:01]  (freeze manipulator) started
[collisionAvoider-exe][12:50:01]  (keep-on-moving object) started
[collisionAvoider-exe][12:50:12]  (freeze manipulator) finished
[collisionAvoider-exe][12:50:12]  (keep-on-moving object) finished
[collisionAvoider-exe][12:50:12]  (detect-moving-object object) started
[cleaner-execution    ][12:50:14]  (scan-for-static-object trash) finished
[cleaner-execution    ][12:50:14]  Resource Manipulator Needed
```

```
[cleaner-execution   ][12:50:14]  Resource Manipulator Received
[cleaner-execution   ][12:50:14]  (pickup-object trash position((157) (133))) started
[cleaner-execution   ][12:50:20]  (pickup-object trash trash-position) finished
[cleaner-execution   ][12:50:20]  (move-object trash trash-can-position) started
[cleaner-execution   ][12:50:24]  (move-object trash trash-can-position) finished
[cleaner-execution   ][12:50:24]  (drop-held-object) started
[cleaner-execution   ][12:50:25]  (drop-held-object) finished
[cleaner-execution   ][12:50:25]  (scan-for-static-object object) started
...
[collisionAvoider-exe][12:50:47]  Resource Manipulator Needed
[collisionAvoider-exe][12:50:47]  (detect-moving-object object) finished
[collisionAvoider-exe][12:50:47]  Resource Manipulator Received
[collisionAvoider-exe][12:50:47]  (freeze manipulator) started
[collisionAvoider-exe][12:50:47]  (keep-on-moving object) started
[collisionAvoider-exe][12:50:51]  (freeze manipulator) finished
[collisionAvoider-exe][12:50:51]  (keep-on-moving object) finished
[collisionAvoider-exe][12:50:52]  (detect-moving-object object) started
[cleaner-execution   ][12:50:54]  (scan-for-static-object trash) finished
[cleaner-execution   ][12:50:54]  Resource Manipulator Needed
[cleaner-execution   ][12:50:54]  Resource Manipulator Received
[cleaner-execution   ][12:50:54]  (pickup-object trash position((269) (78))) started
[collisionAvoider-exe][12:50:57]  Resource Manipulator Needed
[collisionAvoider-exe][12:50:57]  (detect-moving-object object) finished
[collisionAvoider-exe][12:50:57]  Resource Manipulator Received
[collisionAvoider-exe][12:50:57]  (freeze manipulator) started
[collisionAvoider-exe][12:50:57]  (keep-on-moving object) started
[cleaner-execution   ][12:50:59]  Resource Manipulator Needed
[collisionAvoider-exe][12:51:06]  (freeze manipulator) finished
[collisionAvoider-exe][12:51:06]  (keep-on-moving object) finished
[collisionAvoider-exe][12:51:06]  (detect-moving-object object) started
[cleaner-execution   ][12:51:07]  Resource Manipulator Received
[cleaner-execution   ][12:51:07]  (pickup-object trash trash-position) finished
[cleaner-execution   ][12:51:07]  (move-object trash trash-can-position) started
[cleaner-execution   ][12:51:09]  (move-object trash trash-can-position) finished
[cleaner-execution   ][12:51:09]  (drop-held-object) started
[cleaner-execution   ][12:51:12]  (drop-held-object) finished
[cleaner-execution   ][12:51:12]  (scan-for-static-object object) started
```

Figure 11: The plan generated by the Cleaner and CollisionAvoider.

When the object is placed on the table, *Cleaner* agent begins to remove it, starting to execute the action *(pickup-object trash trash-position)* (images 22 and 23). But during the execution of this cleaning action, another unknown object enters the work area (at 12:50:57, image 24), and *CollisionAvoider* actions are executed again (images 24 - 28). At 12:51:05, the object leaves the work area and the *Cleaner* finishes his action *(pickup-object trash trash-position)* (image 29) and executes *(move-object trash trash-can-position)* and *(drop-held-object)* (images 30 - 32).

```
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 12: Message exchange among agents.

By analyzing the data generated by the described experiment, some response times can be determined:

1. Table 3 presents the higher and lower speed an object can have so that the *CollisionAvoider* agent can detect it, when it is alone in the society. As can be seen, there is a tradeoff between low and high resolution. In low resolution, higher speeds are bigger because the agent acquires more images per seconds, and in medium resolution, lower speed detection is better because we can have higher precision when processing images since the camera is fixed and the resolution is higher.

Image Resolution (pixels)	Higher Speed (m/s)	Lower Speed (cm/s)
Low (64 x 48)	6.05	39
Medium (120 x 80)	1.1	3.6

Table 3: Higher and lower speed for moving object detection by the CollisionAvoider agent.

2. Table 4 shows the time that the *Collision-Avoider* takes to react to a moving object entering the work area, for different configurations. As can be seen, the reaction time is dependent on the image resolution and on the network throughput, as images need large bandwidth.

Configuration	Proc.	Comm.	Network delay	Total
Standalone (64 x 48)	0.05	0.05	0	0.1
Standalone (120 x 80)	0.35	0.05	0	0.4
With *Cleaner* (120 x 80)	0.35	0.05	0 to 1.5	< 1.9

Table 4: Reaction Time in seconds for CollisionAvoider in different configurations.

3. Finally, Table 5: Cleaner Agent average processing time.

4. presents average times for the *Cleaner* agent to complete its actions, when not interrupted by another agent. Here, the 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. Also, the working time of the robot is the most critical time restriction, as expected.

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

Table 5: Cleaner Agent average processing time.

7. 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 [Boissier et al.: 94; Bond et al.: 88; Neves et al.: 97; Garcia-Alegre et al.: 97]. Such knowledge can be used to provide support on the development of new real world planning systems [Barros et al.: 97; Barros et al.:97a]. Some of the important resulting experiences are listed bellow:

- When a community of agents is supposed to work in some coordinated manner, it is important to decide about the division of labor and organization, such as: 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 the agent design, i.e., in terms of predicting possible data flow, decision processes and actions.
- Conflicts over incompatible actions and shared resources may place ordering constraints on agent activities, restricting decomposition choices and forcing the need of reconsidering 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. Often, the more reactive task is given some kind of precedence over the deliberative one, so that it can provide a rapid response to important environmental events.

In terms of analyzing the existing theories on how a robotic system should be built, we could reach some important insights about the vision and planning areas, suggesting a new interpretation and parallels between them, as illustrated in Figure 13.

Vision tasks can traditionally solve recognition problems through two different approaches: the recovery and the purposive. In the recovery vision approach, a precise 3D world model is constructed, gathering as much information about the world as possible. However, to every change in the world a new model has to be constructed implying high computational cost. On the other hand, the purposive approach decomposes the visual task into, what is usually called, *recognition behaviors* according to the different types of expertise required by a given domain task. By *recognition behaviors* we should understand that the recognition capability is driven by the behavior required by the task. Computational efficiency is gained with selective and directed vision processes by discarding image information not relevant to the task.

The planning area has two basic approaches: the reactive and the deliberative. In the reactive planning, the task is decomposed according to specialized *autonomous behaviors*. In this case, autonomous behavior means the robot behavior while executing a given task, involving *perception and action*. We make a parallel between purposive vision and reactive planning: both approaches involve perception and action but from different points of view. In the reactive planning the robot generates an *autonomous behavior* by executing a number of actions corresponding to possible world state changes with use of perceptual capabilities (involving perception plus action). On the other hand, the purposive vision, *directed* by those world actions, provides specialized vision processes, defining different recognition capabilities according to the desirable robot behavior and therefore they are called *recognition behaviors* (involving only perception). An important point to be noticed is that, since the reactive planning requires real-time responses, the idea of vision-based specialized recognition processes shows a suitable matching between these two areas.

The goal of a deliberative planning task is to: generate a plan; perform plan refinement and/or task decomposition; solve interactions conflict; allocate resources; perform variable codesignation, etc. In order to plan in a dynamic world, planning has to alternate execution, reaction and perception. By combining planning and perception the knowledge of the situation will be directed, rather than predicted (or previously specified) this way, most of the uncertainty and some of the incompleteness problems can be solved. This is shown with the dotted links between the recognition behaviors and the primitive tasks in the domain application, and/or the links between the 3D World Model and the primitive tasks (the alternative approaches to do vision recognition depends on the nature and requirements of the primitive task). The combination of the deliberative and reactive planning requires both of them running as independent processes while solving complex tasks. With VIBRA we have proposed a suitable multi-agent architecture which has proven to efficiently allow this combination, attending the desirable AI challenge of having intelligent goal-driven reactive systems.

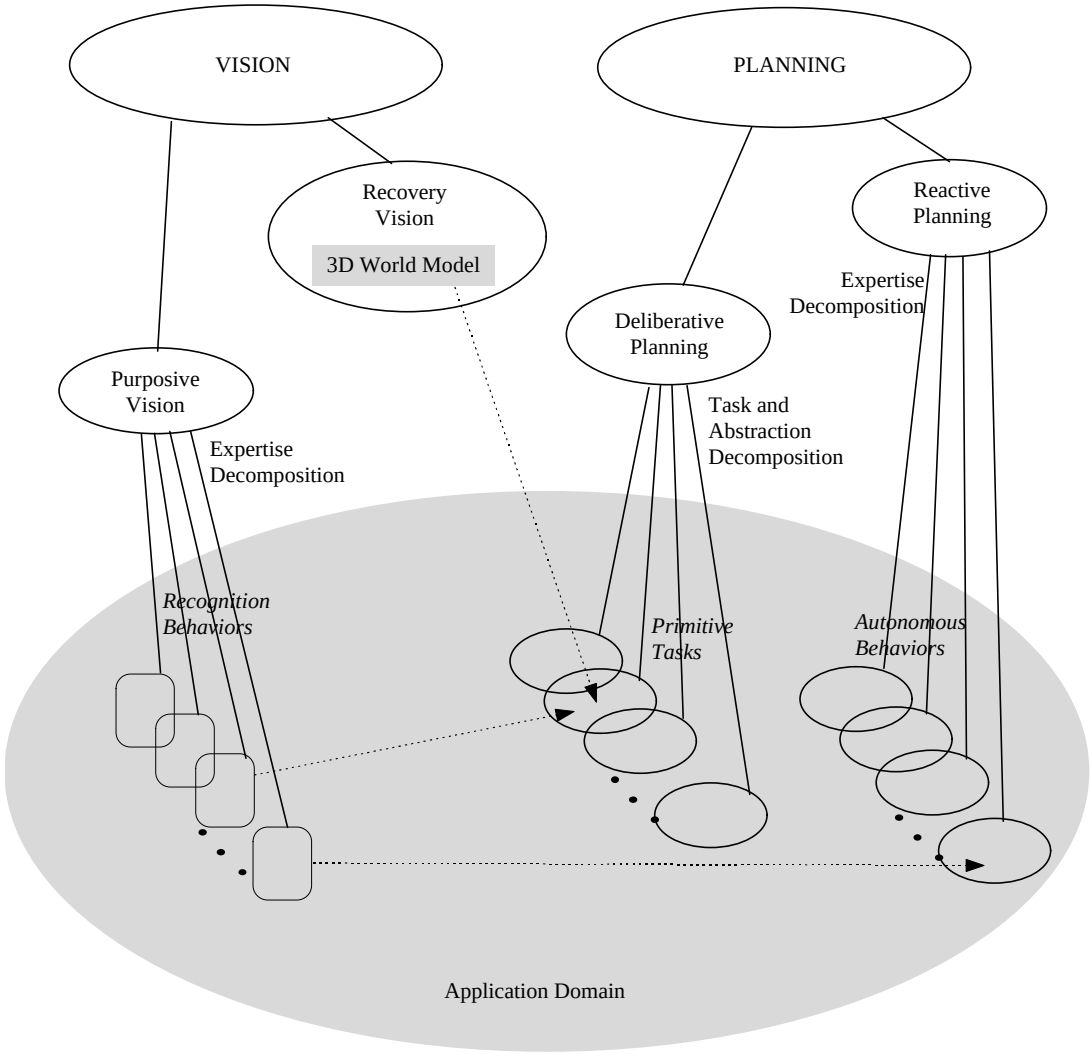

Figure 13: Integration of AI planning and vision according to different approaches.

8. References

[Aloimonos:94] Aloimonos, Y. What I have learned. CVGIP: Image Understanding, 60:1, p.74-85, July 1994.

[Bajcsy: 88] Bajcsy, R. Active perception. Proceedings of the IEEE, 78:8, p.996-1005, 1988.

[Barret et al.: 94] Barret, A.; Weld, D. S. Partial-order planning: evaluating possible efficiency gains. Artificial Intelligence, 67. 1994.

[Barros et al.: 97] Barros, L.N.; Hendler, J.; Benjamins, V.R. Par-KAP: a Knowledge Acquisition Tool for Building Practical Planning Systems. In: International Joint Cconference On Artificial Intelligence (IJCAI), Nagoya, Japan, 1997. Proceedings.

[Barros et al.: 97a] Barros, L.N.; Hendler, J.; Benjamins, V.R. Par-KAP: a Knowledge Acquisition Tool for Building Practical Planning Systems. In: **Proceedings** of Ninth Dutch Conference on Artificial Intelligence, NAIC'97, K. van Marcke, W. Daelemans, Eds, University of Antwerp, Belgium, p. 137-148, 1997.

[Bianchi et al.: 96] Bianchi, R.A.C.; Rillo, A.H.R.C. A distributed control architecture for a purposive computer vision system. 2nd. Symposium on Intelligence in Automation and Robotics (IAR'96), IEEE, Rockville, Maryland, USA, 4-5 November, 1996. p. 288-294.

[Boissier et al.: 94] Boissier, O; Demazeau, Y. ASIC: An architecture for social and individual control and its application to Computer Vision. In: European Workshop On Modeling Autonomous Agents In A Multi-Agent World, 1994. Proceedings**.** 1994. p.107-18.

[Bond et al.: 88] Bond, A. H.; Gasser, L. Readings in Distributed Artificial Intelligence. Morgan Kaufmann, San Mateo, CA, 1988.

[Brooks: 91] Brooks, R. A. Intelligence without representation. Artificial Intelligence, 47, 1991. p. 139-59.

[Dean et al.: 95] Dean, T.; Allen, J. Aloimonos, Y. Artificial Intelligence: Theory and Practice. Benjamin/Cummings Publishing Co., Redwood City, CA, 1995.

[Fermüller: 93] Fermüller, C. Basic Visual Capabilities. CS-TR-3064, University of Maryland, College Park, MA, 1993.

[Firby et al.: 95] Firby, R. J.; Kahn, R. E.; Prokopowicz, P. N.; Swain, M. J. Collecting trash: a test of Purposive Vision. Workshop on Vision for Robots, Pittsburgh, PA, 1995.

[Firby: 96] Firby, R. J. Modularity Issues in Reactive Planning. Proceedings of the 3rd International Conference on AI Planning Systems, May 1996. p. 78-85.

[Garcia-Alegre et al.: 97] Garcia-Alegre, M. C.; Recio, F. Basic Agents for Visual/Motor Coordination of a Mobile Robot. AGENTS'97 Conference Proceedings, Marina del Rey, CA, USA. ACM, 1997.

[Haigh et al.: 97] Haigh, K. Z.; Veloso, M. High-Level Planning and Low-Level Execution: Towards a Complete Robotic Agent. In Proceedings of the First International Conference on Autonomous Agents, February 1997.

[Hanks et al.: 93] Hanks, S.; Pollack, M. E.; Cohen, P. R. Benchmarks, Test Beds, Controlled Experimentation, and the Design of Agent Architectures. AI Magazine, p. 17-42, Winter 1993.

[Jolion: 94] Jolion, J.M. Computer vision methodologies. CVGIP: Image Understanding, v.59, n.1, p.53-71, Jan. 1994.

[Lyons et al.: 95] Lyons, D. M.; Hendriks, A. J. Exploiting Patterns of Interaction to achieve reactive behavior. Artificial Intelligence, 73, 1995. p. 117-148.

[Maes: 95] Maes, P. Artificial Life meets entertainment: life like autonomous agents. Communications of the ACM, 38:11, 1995, p.108-114.

[Marr: 82] Marr, D. Vision. New York, Freeman, 1982.

[Medeiros et al.: 97] Medeiros, A. A. D.; Chatila, R. Priorities and data abstraction in hierarchical control architectures for autonomous robots. In: Workshop on Intelligent Robotics, Brasília, Brazil, 1997. p. 207-220.

[Minton et.al: 91] Minton, S.; Bresina, J.; Drummond, M. Commitment strategies in planning: a comparative analysis. In: IJCAI, 1991.

[Neves et al.: 97] Neves, M. C.; Oliveira, E. A Control Architecture for an Autonomous Mobile Robot. AGENTS'97 Conference Proceedings, Marina del Rey, CA, USA. ACM, 1997.

[Rillo: 92] Rillo, A.H.R.C. Grouping-based recognition system. SPIE'91 - Advances in Intelligent Robotic Systems - Model-based vision development and tools conference, 11-15 November 1991, BOSTON, Massachusetts, USA, R. M. Larson & H. N. Nasr (eds.), Proc. SPIE 1609, 1992, pp. 274-282.

[Rillo: 93] Rillo, A.H.R.C. 3D object recognition using a decision hierarchy. SPIE'1992 International Symposium on Optical Applied Science and Engineering - Applications of digital image processing XV, 19-24 July 1992, San Diego, California, USA. Proc. SPIE 1771, Andrew

G. Tescher (ed.), Lockheed Palo Alto Research Lab., Saratoga, CA, USA.1993, p.225-33.

[Rillo et al.: 92] Rillo, M.; Rillo, A.H.R.C.; Costa, L.A.R. The LSI Assembly Cell. In: IFAC Symposium on Information Control Problems in Manufacturing Technology, 7º, Toronto, 1992. Proceedings. IFAC, 1992. p. 361-5.

[Stone et al.: 97] Stone, P.; Veloso, M. Multiagent Systems: A Survey from a Machine Learning Perspective. CMU Internal Report, February 1997.

[Tarr, Black: 94] Tarr, M. J.; Black, M. J. A computational and evolutionary perspective on the role of representation in vision. CVGIP: Image Understanding, v.60, n.1, p.65-73, July 1994.

[Wooldridge et al.: 95] Wooldridge, M.; Jennings, N. R. Intelligent Agents: Theory and Practice. Knowledge Engineering Review, 1995.