REINALDO AUGUSTO DA COSTA BIANCHI

UMA ARQUITETURA DE CONTROLE DISTRIBUÍDA PARA UM SISTEMA DE VISÃO COMPUTACIONAL PROPOSITADA.

Dissertação apresentada à Escola Politécnica da Universidade de São Paulo para a obtenção do título de Mestre em Engenharia.

São Paulo
1998

REINALDO AUGUSTO DA COSTA BIANCHI

UMA ARQUITETURA DE CONTROLE DISTRIBUÍDA PARA UM SISTEMA DE VISÃO COMPUTACIONAL PROPOSITADA.

Dissertação apresentada à Escola Politécnica da Universidade de São Paulo para a obtenção do título de Mestre em Engenharia.

Área de Concentração:
Engenharia Eletrônica

Orientadora:
Anna Helena Reali Costa

São Paulo
1998

Para Mayra.

AGRADECIMENTOS

Aos amigos e companheiros de Pós Graduação de vários laboratórios e departamentos da Escola Politécnica da Universidade de São Paulo: Leliane, Lúcia, Katsumi, Mikael, Gélio, Fábio, Briand, Márcia, Leila, Maurício, que discutiram e sugeriram, revisaram e corrigiram, apoiaram e aturaram, enfim, colaboraram de inúmeras maneiras durante esse período.

Aos técnicos e administradores de sistemas do Laboratório de Sistemas Integráveis (EPUSP), sempre prontos para ajudar na resolução dos problemas da rede: Mateo, Armin, Volnys, Natanael, Luiz e, em especial, a João Kögler.

Aos professores da Escola Politécnica: Márcio Rillo, Jaime Sichman, Tsen Chung Kang, João Martino, Romero Tori, Jaime Cruz, Líria, Selma, João José Neto, que cederam laboratórios, sugeriram melhorias, cobraram resultados, discutiram o conteúdo e estavam sempre dispostos a ajudar. Pelos mesmos motivos, aos professores Alberto Elfes do Centro Tecnológico para Informática e Jacques Waldmann do Instituto Tecnológico da Aeronáutica.

A minha orientadora, Anna Helena Reali Costa, por, mais que orientar este trabalho, ensinar-me a ser pesquisador e professor.

Ao CNPq, pelo financiamento parcial desta pesquisa.

A minha família, pelo suporte e incentivo constante.

E a todos que, direta ou indiretamente, colaboraram na execução deste trabalho.

SUMÁRIO

LISTA DE TABELAS

LISTA DE FIGURAS

RESUMO

ABSTRACT

1. INTRODUÇÃO 1

1.1 VISÃO COMPUTACIONAL E INTELIGÊNCIA ARTIFICIAL 1

1.2 OBJETIVO 3

1.3 DOMÍNIO DE APLICAÇÃO: TAREFA DE MONTAGEM 4

1.4 ORGANIZAÇÃO DO TRABALHO 10

2. HISTÓRICO DA ÁREA DE VISÃO COMPUTACIONAL 11

2.1 INTRODUÇÃO 11

2.2 A TEORIA DE MARR 12

2.3 O PARADIGMA RECONSTRUTIVO 14

2.4 VISÃO ATIVA, ANIMADA E QUALITATIVA 17

2.5 O PARADIGMA PROPOSITADO 19

3. INTELIGÊNCIA ARTIFICIAL DISTRIBUÍDA 23

3.1 INTRODUÇÃO 23

3.2 AGENTES 23

3.3 HISTÓRICO DE DAI E SUAS SUB-ÁREAS 26

3.4 RESOLUÇÃO DISTRIBUÍDA DE PROBLEMAS (DISTRIBUTED PROBLEM SOLVING) 27

3.5 SISTEMAS MULTI-AGENTES 28

3.6 RESOLUÇÃO DISTRIBUÍDA DE PROBLEMAS X SISTEMAS MULTI-AGENTES 30

3.7 OUTRAS ABORDAGENS DISTRIBUÍDAS PARA IA 31

3.8 APLICAÇÕES 34

4. A ARQUITETURA DE CONTROLE PROPOSTA 36

4.1 REGRAS DA SOCIEDADE E ESTRUTURA DE AUTORIDADE 38

4.2 DEFINIÇÃO DOS AGENTES AUTÔNOMOS 39

4.3 MÓDULOS DE COMUNICAÇÃO E PLANEJAMENTO/EXECUÇÃO 41

4.4 LINGUAGEM DE COMUNICAÇÃO DOS AGENTES AUTÔNOMOS 41

4.5 AGENTES PRIMITIVOS 45

5. TRABALHOS CORRELATOS 47

6. A IMPLEMENTAÇÃO DA ARQUITETURA 55

6.1 IMPLEMENTAÇÃO DE UMA ESTRUTURA PARA A ARQUITETURA PROPOSTA 55

6.2 IMPLEMENTAÇÃO DA COMUNICAÇÃO UTILIZANDO A BIBLIOTECA PVM 58

6.3 A IMPLEMENTAÇÃO EM UM DOMÍNIO DE MONTAGEM 61

6.3.1 Descrição da Célula de Montagem 61

6.3.2 Agentes Autônomos 64

6.3.3 Recursos, Regras de Comportamento e Estrutura de Autoridade 65

6.3.4 A Decomposição de Tarefas dos Agentes Autônomos 66

6.3.5 Agentes Primitivos Implementados 68

6.3.6 Considerações sobre a implementação 76

7. TESTES E RESULTADOS 78

7.1 SIMULAÇÃO 78

7.2 RESULTADOS DA APLICAÇÃO 82

8. CONCLUSÃO E TRABALHOS FUTUROS 89

9. REFERÊNCIAS BIBLIOGRÁFICAS 96

APÊNDICE A: TESTES DO SISTEMA IMPLEMENTADO E EXEMPLOS DE FUNCIONAMENTO

LISTA DE TABELAS

TABELA 2.1 - OS TRÊS NÍVEIS NOS QUAIS QUALQUER MÁQUINA REALIZANDO UM PROCESSAMENTO DE INFORMAÇÃO DEVE SER COMPREENDIDA. [MARR, 1982, P.25] 13
TABELA 3.1 - RESUMO DE DEFINIÇÕES DE AGENTES EM [FRANKLIN; GRAESSER,1996]. 24
TABELA 3.2 - RESUMO DAS CARACTERÍSTICAS ENVOLVIDAS NOS SISTEMAS MULTI-AGENTES. 25
TABELA 3.3 - TAXONOMIA DOS AGENTES PROPOSTA POR TALUKDAR (1994) PELA PARTIÇÃO DO ESPAÇO DE EXISTÊNCIA DOS AGENTES EM QUATRO REGIÕES NEBULOSAS. 34
TABELA 4.1 - NÍVEIS COMPORTAMENTAIS DO SISTEMA E A ABORDAGEM MAS PROPOSTA. 37
TABELA 4.2 - TIPOS DE MENSAGENS DE NATUREZA *DECISÓRIA* DEFINIDAS NA LINGUAGEM DE COMUNICAÇÃO PARA INTERAÇÃO ENTRE AGENTES NA SOCIEDADE. 42
TABELA 4.3 - TIPOS DE MENSAGENS DE NATUREZA *DE CONTROLE* DEFINIDAS NA LINGUAGEM DE COMUNICAÇÃO PARA INTERAÇÃO ENTRE AGENTES NA SOCIEDADE. 43
TABELA 4.4 - TIPOS DE MENSAGENS TROCADAS ENTRE OS MÓDULOS DE COMUNICAÇÃO E PLANEJAMENTO/EXECUÇÃO DE UM AGENTE AUTÔNOMO. 44
TABELA 5.1 - COMPARAÇÃO ENTRE OS NÍVEIS COMPORTAMENTAIS, O MODELO MAS E A *SUBSUMPTION ARCHITECTURE*. 49
TABELA 6.1 - CLASSES IMPLEMENTADAS PARA A ARQUITETURA. 56
TABELA 6.2 - EQUIPAMENTO UTILIZADO. 62
TABELA 7.1 - TAXA DE TRANSMISSÃO MÉDIA DO PVM COM SISTEMA *OF LINE* E *ON LINE*. 85
TABELA 7.2 - VELOCIDADES MÁXIMAS E MÍNIMA PARA UM OBJETO EM MOVIMENTO. 85
TABELA 7.3 - TEMPO DE REAÇÃO EM SEGUNDOS PARA O AGENTE EVITADOR DE COLISÃO. 86
TABELA 7.4 - TEMPO DE REAÇÃO DO AGENTE DESOBSTRUTOR. 87
TABELA 7.5 - TEMPO MÉDIO DE PROCESSAMENTO PARA O AGENTE DESOBSTRUTOR (EM SEGUNDOS). 88

LISTA DE FIGURAS

FIGURA 1.1 - ESQUEMA DO SISTEMA UTILIZADO NA CÉLULA FLEXÍVEL DE MONTAGEM. 6
FIGURA 1.2 - EXEMPLO DE DECOMPOSIÇÃO DE TAREFAS. 7
FIGURA 1.3 - O MANIPULADOR UTILIZADO DA CÉLULA DE MONTAGEM. 8
FIGURA 2.1 - DISCIPLINAS QUE INVESTIGAM A VISÃO [TRIVEDI; ROSENFELD, 1989]. 11
FIGURA 3.1 - O MODELO DE UM AGENTE GENÉRICO EM [DEMAZEAU; MÜLLER,1990, P.5]. 29
FIGURA 4.1 - TOPOLOGIA DA ARQUITETURA. 37
FIGURA 4.2 - O MODELO DO AGENTE AUTÔNOMO. 40
FIGURA 5.1 - AGENTES AUTÔNOMOS RELACIONADOS ÀS CAMADAS DA ARQUITETURA DE BROOKS. 48
FIGURA 6.1 - DIAGRAMA DE CLASSES RESUMIDO DA BIBLIOTECA IMPLEMENTADA. 57
FIGURA 6.2 - UM DOS MANIPULADORES DA CÉLULA DE MONTAGEM. 61
FIGURA 6.3 - A CÂMERA UTILIZADA. 63
FIGURA 6.4 - FLUXO DE DADOS DO AGENTE AUTÔNOMO MONTADOR. 67
FIGURA 6.5 - FLUXO DE DADOS DO AGENTE AUTÔNOMO DESOBSTRUTOR. 68
FIGURA 6.6- FLUXO DE DADOS DO AGENTE AUTÔNOMO EVITADOR DE COLISÃO. 68
FIGURA 6.7 - IMAGEM COMPOSTA CAPTURADA. 69
FIGURA 6.8 - IMAGEM BINARIZADA PARA A COR VERMELHA. 70
FIGURA 6.9 – DEFINIÇÃO DOS ELEMENTOS DE BORDA DO OPERADOR UTILIZADO 71
FIGURA 6.10 - DETERMINAÇÃO DE CONTORNOS 72
FIGURA 6.11 - VETORIZADOR 74
FIGURA 6.12 - PADRÃO DE CALIBRAÇÃO. 75
FIGURA 6.13 - IMAGEM CAPTURADA. 76
FIGURA 6.14 - RESULTADO DA SEGMENTAÇÃO. 76
FIGURA 7.1 - ALOCAÇÃO DE 1 RECURSO POR 4 AGENTES. (SIMULAÇÃO) 80
FIGURA 7.2 - ALOCAÇÃO DE 1 RECURSO POR 4 AGENTES DURANTE 5 MINUTOS. (SIMULAÇÃO) 81
FIGURA 7.3 - ALOCAÇÃO DE 5 RECURSO POR 4 AGENTES. (SIMULAÇÃO) 81
FIGURA 7.4 - MONTAGEM COM INTERRUPÇÃO E DESOBSTRUÇÃO 84
FIGURA 7.5 - ALOCAÇÃO DO RECURSO MANIPULADOR DURANTE O EXEMPLO. 84
FIGURA 7.6 - TEMPO MÉDIO DE REAÇÃO PARA COLISÃO VERSUS O NÚMERO DE AGENTES NO SISTEMA 86
FIGURA 7.7 - TEMPO DE REAÇÃO MÉDIO PARA A DESOBSTRUÇÃO VERSUS O NÚMERO DE AGENTES NO SISTEMA. (DESVIOS NÃO SIGNIFICATIVOS) 87

RESUMO

Esta dissertação apresenta uma arquitetura de controle distribuída para um sistema de visão computacional propositada, modelada segundo uma abordagem de Inteligência Artificial Distribuída com ênfase em Sistemas Multi-Agentes.

O principal problema abordado é o da integração de diversos módulos com tarefas e comportamentos diferentes em um sistema robótico, que utiliza visão computacional para perceber o mundo. Esta integração surge da organização de diversas soluções dedicadas a diferentes tarefas visuais, resultando na proposta de uma arquitetura constituída por uma sociedade de Agentes Autônomos (AAs), que comunicam entre si através de uma rede de comunicação descentralizada e totalmente conectada, sendo cada AA responsável por um comportamento independente, organizados segundo regras de comportamento e uma estrutura de autoridade. Essa arquitetura integra diversos módulos da cognição, como percepção visual, planejamento, controle e atuação.

Como tentativa de validação experimental da proposta, essa arquitetura foi implementada em um sistema dotado de um manipulador robótico e uma câmera, trabalhando em um domínio de montagem.

A estrutura básica da arquitetura foi desenvolvida segundo uma metodologia de projeto orientada a objetos e consiste em uma biblioteca de classes que definem o modelo dos agentes e as estruturas para as interações entre esses em uma sociedade.

O sistema robótico proposto foi implementado na Célula Flexível de Montagem da Escola Politécnica da USP, tendo por objetivo realizar tarefas de montagens simples e visualmente guiadas.

O sistema foi testado exaustivamente e os resultados experimentais indicaram algumas vantagens e desvantagens da metodologia empregada, possibilitando a definição de algumas diretrizes importantes para o projeto e implementação de sistemas robóticos inteligentes atuando no mundo real.

ABSTRACT

This dissertation presents a distributed control architecture for a purposive computer vision system, modeled according to a Distributed Artificial Intelligence approach, with emphasis in Multi-Agents Systems.

The main problem addressed is the integration of several modules with different purposes and behaviors in a robotic system, which uses computer vision to perceive the world. This integration arises from the organization of several dedicated solutions to different visual tasks, resulting in the proposal of one architecture that consists of a society of Autonomous Agents, communicating with each other through a decentralized and fully connected network, where each AA is responsible for an independent behavior. This society is organized according to behavior rules and an authority structure. Furthermore, this architecture integrates several cognitive modules, like visual perception, planning, control and action.

As an attempt of an experimental validation of the proposal, this architecture was implemented in a system composed of a robotic manipulator and one camera, working in an assembly domain.

The basic structure of the architecture was developed according to an object oriented methodology and consists of a library of classes which defines the agents' model and the structures needed for the interactions among these agents in a society.

The proposed robotic system was implemented in the Flexible Assembly Cell of the Escola Politécnica da USP, with the objective of accomplishing simple and visually guided assembly tasks.

The system was thoroughly tested and the experimental results indicated some advantages and inconveniences of the methodology, allowing the definition of some important guidelines for the design and implementation of intelligent robotic systems acting in the real world.

1. INTRODUÇÃO

Neste capítulo é apresentada uma introdução ao trabalho proposto, descrevendo os seus objetivos, justificativas e contribuições. Inicia-se o capítulo com a discussão da relação entre Visão Computacional e Inteligência Artificial, que é fundamental para a inserção do trabalho na área de Inteligência Artificial.

1.1 Visão Computacional e Inteligência Artificial

Nos últimos anos vários pesquisadores da área de Visão Computacional (VC) têm dado muita ênfase às técnicas puramente matemáticas, como geometria e análise funcional, afastando-se das investigações básicas em Inteligência Artificial (IA), que geralmente buscam métodos mais gerais para a solução de problemas. Por um lado isto aconteceu porque as metodologias que estavam sendo transferidas de IA para VC mostravam-se grosseiramente inadequadas e, por outro, por existir uma certa confusão de conceitos entre ambas as áreas, incluindo ainda Robótica.

Esse enfoque puramente matemático para VC pode ser problemático, uma vez que tende a compartimentar o estudo, esquecendo outros aspectos que compõem um sistema inteligente, e a ignorar o que muitos autores - como por exemplo, ALOIMONOS (1994), BROOKS (1991b), ELFES (1986), FISHER (1994) e FIRBY (1997) - concordam ser um problema fundamental em IA: a integração dos diferentes aspectos da inteligência.

Muitos pesquisadores da área de VC estão hoje direcionando seus estudos de modo a reaproximá-la de IA, em um contexto bastante diferente do existente há 10 anos atrás, porque existe hoje um embasamento matemático e de engenharia muito mais sólido para a solução dos problemas de VC. Reafirmar que VC e IA possuem muitos aspectos em comum é uma conseqüência dessa linha de pesquisa.

Assim, FISHER (1994) discute a relação entre VC e IA, mostrando que ambas partilham objetivos, metodologias, ferramentas, suposições de domínios (*domain assumptions*) e embasamento biológico, psicológico e filosófico. Afirma que “as diferenças são em grande parte ilusórias, e que VC ainda possui e vai continuar possuindo uma forte conexão com IA” [FISHER, 1994, p.21]. Observa que os objetivos de VC são especializações dos objetivos de IA e que podem ser correlacionados. Entre

as comparações apresentadas por FISHER (1994), tem-se:

- num dado aspecto, IA procura compreender os processos computacionais humanos e VC procura compreender os sistemas visuais biológicos - incluindo o humano, ambos desenvolvendo metodologias, teorias e módulos que possam ser testados;
- IA também procura desenvolver ferramentas que são inteligentes, no sentido de precisar de pouca atenção de um ser humano para atuar, sendo flexíveis e possuindo conhecimento compilado. VC procura construir ferramentas que tenham habilidades de extração de informação e que sejam autônomas;
- IA procura aprimorar as habilidades dos seres humanos, possibilitando o controle do seu ambiente, enquanto VC procura desenvolver ferramentas que ampliem as habilidades perceptivas humanas;
- IA pesquisa o desenvolvimento de representações do conhecimento para domínios diferentes e VC procura determinar qual a melhor representação para um domínio específico.

Desse modo, fica evidente que VC e IA estão profundamente relacionadas, tornando o estudo dos outros aspectos de cognição de fundamental importância para os pesquisadores de VC.

A solução para o problema de como fazer com que um robô se comporte de maneira inteligente é, há muito tempo, o sonho dos cientistas do campo de Inteligência Artificial e engenheiros da área de Robótica.

Sistemas que solucionam esse tipo de problema encontram aplicações na indústria - por exemplo, em células flexíveis de montagem ou manufatura - e em robôs móveis autônomos, que podem se locomover sem serem controlados por uma pessoa e que encontram sozinhos seus caminhos, podendo assim atuar em ambientes não-estruturados. Por exemplo, no caso de uma linha de montagem, espera-se que um robô móvel reconheça o ambiente que o cerca; localize uma peça sobre uma mesa de trabalho; identifique sua forma e orientação para poder manipulá-la; locomova-se ao longo da linha de montagem; posicione-se adequadamente para a tarefa de montagem; etc.

Esse comportamento exige que o robô interaja com o mundo à sua volta,

processando uma grande quantidade de informações sensoriais (visão, tato, distância, força, etc.), e as incorpore ao seu processo de atuação em tempo real. Por isso, a utilização de robôs autônomos requer o desenvolvimento de sistemas avançados, de alto desempenho, baseados em flexibilidade e confiabilidade.

1.2 Objetivo

O objetivo desta dissertação é apresentar uma Arquitetura de Controle Distribuída para um Sistema de Visão Computacional Propositada, modelada com base na abordagem de Inteligência Artificial Distribuída (DAI), com ênfase em Sistemas Multi-Agentes (MAS) e a sua implementação em um sistema dotado de um manipulador robótico e uma câmera, trabalhando em um domínio de montagem.

Desse modo, o principal problema abordado é o da integração de diversos módulos com tarefas e comportamentos diferentes em um sistema robótico e que utiliza Visão Computacional Propositada para perceber o mundo.

A arquitetura proposta consiste em uma sociedade de Agentes Autônomos (AAs), que comunicam entre si através de uma rede de comunicação descentralizada e totalmente conectada, sendo cada qual responsável por um comportamento diverso, organizados segundo regras de comportamento e uma estrutura de autoridade. Essa arquitetura integra diversos módulos de cognição, como percepção visual, planejamento, controle e atuação.

Nesta arquitetura, a definição da sociedade de agentes é dinâmica, permitindo inserir ou retirar AAs da sociedade durante a operação do sistema. Essa inserção ou retirada de agentes não é autônoma no sistema implementado, sendo realizada pelo projetista. Mas a arquitetura permite que essa inserção e remoção seja feita segundo o comando de um ou vários agentes desta sociedade, quando da troca de propósitos do sistema. Isto permite que o sistema atue em múltiplos domínios, com apenas a troca dos agentes necessários a sua adaptação. Construir um sistema que possa atuar em múltiplos domínios implica em determinar os conjuntos de agentes necessários para cada domínio e as condições nas quais cada conjunto deve estar ativo. Finalmente, dois domínios parecidos podem ter agentes iguais em seus conjuntos, possibilitando a reutilização dos agentes.

Deste modo, essa arquitetura permite a criação de sistemas multi-propósito auto-gerenciados. Caso se queira criar um sistema com propósitos diferentes, basta definir os agentes necessários para cada tarefa e inseri-los na sociedade.

Um exemplo de um sistema com múltiplos propósitos é o composto por um robô em uma linha de montagem automobilística: ele pode ter que pintar um carro, realizar uma tarefa de soldagem, adicionar componentes etc. Para cada uma dessas tarefas é definido um agente autônomo ou um conjunto de agentes, que, a partir de dados do ambiente ou mesmo da interação com um operador, irá decidir se deve atuar ou se outros agentes estão ativos. Nesse caso, todos os agentes do sistema são executados simultaneamente, esperando a situação na qual sejam necessários. Assim, vários propósitos podem conviver em um mesmo sistema.

A solução do problema da integração de diversos módulos correspondentes a comportamentos distintos, feita através da definição de uma arquitetura e a implementação dessa em um sistema real envolve vários aspectos, entre eles:

- a definição do modelo de uma sociedade de agentes autônomos, que inclui a definição de um agente, da comunicação entre os agentes e da organização destes em uma sociedade;
- a implementação do modelo acima em uma biblioteca que permita a criação de novos agentes de maneira simples e rápida;
- a criação de uma biblioteca de agentes que encapsulam diversos algoritmos, incluindo Visão Computacional;
- a definição e implementação do controle do sistema robótico, distribuído entre os agentes;
- o estudo de um sistema que trata os problemas na medida em que eles surgem em tempo real e que trabalhe com informações incompletas e eventos não previsíveis.

1.3 Domínio de aplicação: tarefa de montagem

HENDLER e MUSLINER (1994) afirmam que “mesmo agora que o custo dos robôs tornou-se razoável, poucos pesquisadores estão tentando implementar estes sistemas em robôs reais”. Assim, como tentativa de validação experimental da

arquitetura proposta, este trabalho a implementa em um sistema de visão computacional propositada dotado de um manipulador robótico e uma câmera, trabalhando em um domínio de montagem.

Na maioria das linhas de montagem reais, um robô localiza previamente cada peça através de um sistema de visão que reconhece e dá informação sobre as posições dessas peças. Com essa informação sensorial o robô pode começar a executar um plano para satisfazer a meta de sua tarefa de montagem original. Essa tarefa deve ser executada continuamente, de modo que novas peças para a montagem possam ser colocadas na mesa por um humano ou por outro robô. Porém, se o sistema de visão descobre peças que não pertencem ao artefato que está sendo montado, isto é, objetos não desejados que possam perturbar a execução da tarefa, o robô deve limpar a área de trabalho antes de continuar.

Uma capacidade essencial do robô deve ser descobrir e evitar possíveis colisões entre o manipulador robótico e um humano (ou outro robô), enquanto está executando uma tarefa de limpeza ou de montagem. Para evitar colisões, ambas as tarefas, de limpeza e de montagem, podem ter a execução interrompida até que a área de trabalho fique livre da possibilidade de colisão.

Um exemplo de aplicação foi implementado na Célula Flexível de Montagem da Divisão de Automação e Inteligência Artificial do Laboratório de Sistemas Integráveis da Escola Politécnica da USP [RILLO, M. et al, 1992].

Essa célula começou a ser construída em 1989, num projeto conjunto com o “Institut fuer Prozessrechentechnik und Robotik” (IPR) da Universidade de Karlsruhe (Alemanha) e atualmente é composta por: três estações de trabalho, uma das quais com uma placa digitalizadora, que permite a captura de imagens de 640 x 480 pixels com 24 bits de cores; uma câmera CCD e 3 microcâmeras; dois manipuladores robóticos articulados verticalmente, de 5 graus de liberdade, cada um controlado por um microcomputador padrão IBM-PC; um microcomputador padrão IBM-PC dotado de uma placa de aquisição de imagens. Todos os computadores que formam a célula estão interligados por uma rede ethernet local. A Célula de Montagem é utilizada como plataforma de desenvolvimento de diversos sistemas, sendo que a Figura 1.1 mostra o esquema de equipamentos utilizado para esta aplicação.

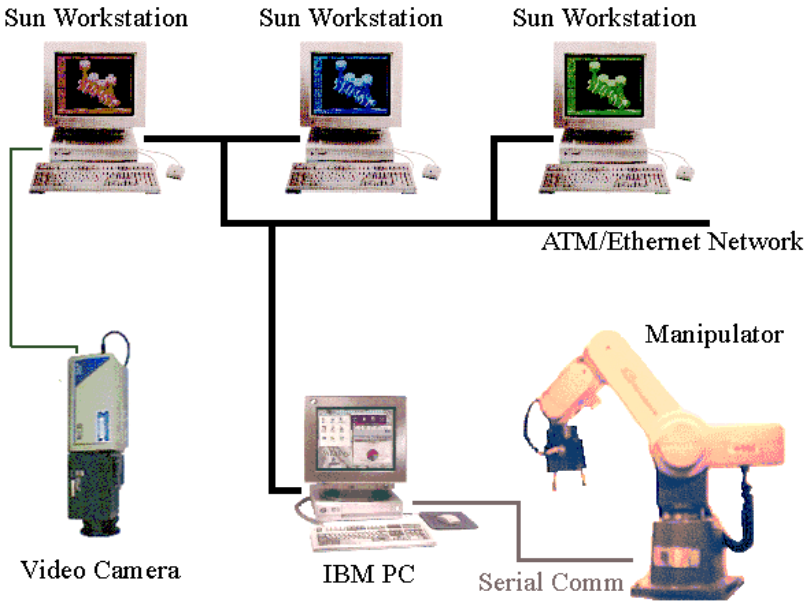


Figura 1.1 - Esquema do sistema utilizado na Célula Flexível de Montagem.

No domínio de montagem implementado, o sistema tem por objetivo realizar tarefas de montagens simples e visualmente guiadas e utiliza um manipulador e uma câmera CCD, ligada à placa digitalizadora colorida da estação de trabalho.

A aplicação consiste de: dados alguns objetos sobre uma mesa, peças e encaixes, o objetivo é encontrar os pares correspondentes e os unir. Como pares da montagem podem ter tamanhos diferentes, a correspondência tem que ser feita entre pares do mesmo tamanho. Enquanto a tarefa principal está sendo executada, interações inesperadas podem acontecer: um humano pode mudar a configuração da mesa, adicionando novos objetos para serem montados ou algum objeto não desejado que force o robô a executar a tarefa de limpeza.

A principal característica do sistema não é sua habilidade para raciocinar sobre a montagem, mas sua habilidade para escolher ações de maneira rápida e efetiva, sendo capaz de atuar em um ambiente incerto e variável. Esse domínio pode ser caracterizado como o de uma tarefa de planejamento reativo complexa, já que possui algumas características importantes:

- **Gerar e executar planos complexos para resolver objetivos específicos da tarefa de montagem**

O domínio de montagem é um caso típico de uma tarefa de planejamento. O

domínio é representado por uma hierarquia de tarefas, a qual descreve como decompor tarefas de um nível mais complexo em níveis mais simples. As tarefas de nível mais simples correspondem à percepção e ação. Para reduzir a complexidade do problema, foi assumido que cada tarefa que participa da solução de um problema pode ser descrita de maneira independente. Dessa maneira, foi possível modelar o domínio como três hierarquias de tarefas separadas: a de montagem propriamente dita, a de limpeza e a de evitar colisões.

A Figura 1.2 ilustra a decomposição de tarefas para um domínio de Montagem. A tarefa Montagem pode ser dividida nas sub-tarefas *procurar peças livres, selecionar pares de peça e encaixe que combinem, pegar objeto, mover objeto, unir par peça-encaixe*. Já a tarefa *Procurar peças livres* pode ser subdividida em *capturar imagem, detectar objetos e cálcular tamanho e posição dos objetos*. Finalmente, a detecção de objetos pode ser separada nas sub-tarefas *reconhecer peça* e *reconhecer encaixe*.

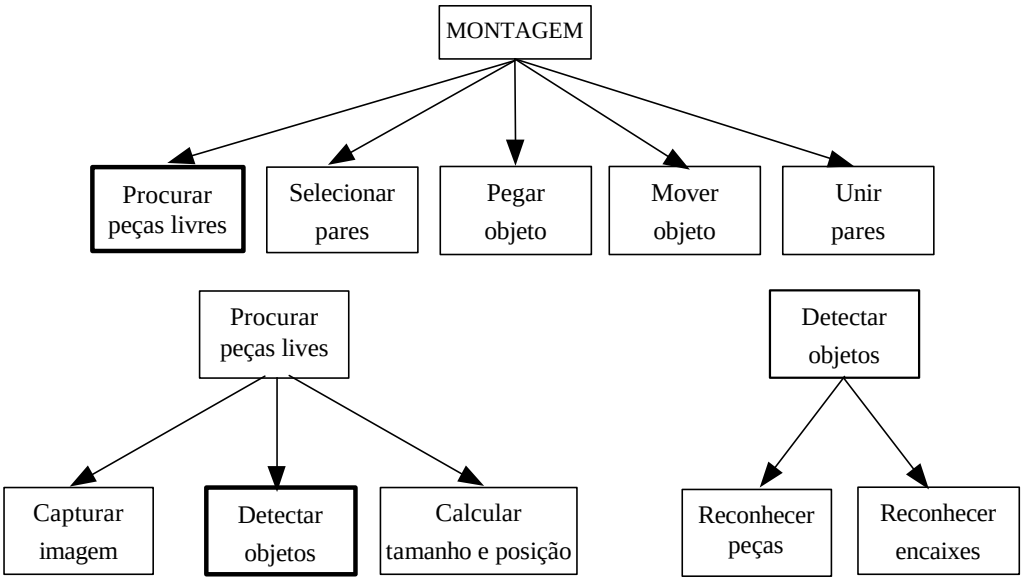


Figura 1.2 - Exemplo de decomposição de tarefas.

A dificuldade da execução dessa tarefa se encontra no processamento e compreensão da imagem; nas interrupções para evitar colisões, nas interações humanas com a configuração da mesa de trabalho. O robô tem que ser capaz de lidar com problemas como: ter objetivos previamente resolvidos desfeitos, i.e., ter pares desunidos; ter adicionadas mais peças para serem montadas; ter objetos não desejados ocultando peças da montagem; ter o manipulador interrompido durante um movimento,

antes de ter alcançado seu objetivo.

- **Realizar alocação de recursos *on-line***

Um recurso é definido como uma parte do sistema que pode ser compartilhado por processos diferentes e que pode ser controlado por somente um processo em um determinado momento. Os recursos que têm que ser compartilhado na aplicação de montagem são a câmera e o manipulador (Figura 1.3). Ambos são compartilhados por três tarefas concorrentes: a montagem, a desobstrução e a tarefa que evita colisões. Nessa aplicação, para qualquer pedido das três tarefas, a câmera pode prover uma imagem sem criar qualquer conflito entre as tarefas. Por outro lado, o manipulador é altamente disputado e, conseqüentemente, as tarefas têm que obedecer uma política específica para resolução de conflito: só uma tarefa pode controlar o manipulador em um determinado momento; a tarefa que evita colisões deve ter prioridade mais alta que as outras, para evitar possíveis acidentes; a tarefa de limpeza deve ter uma prioridade mais alta que a tarefa de montagem, pois objetos indesejados podem ocluir peças relativas à montagem.

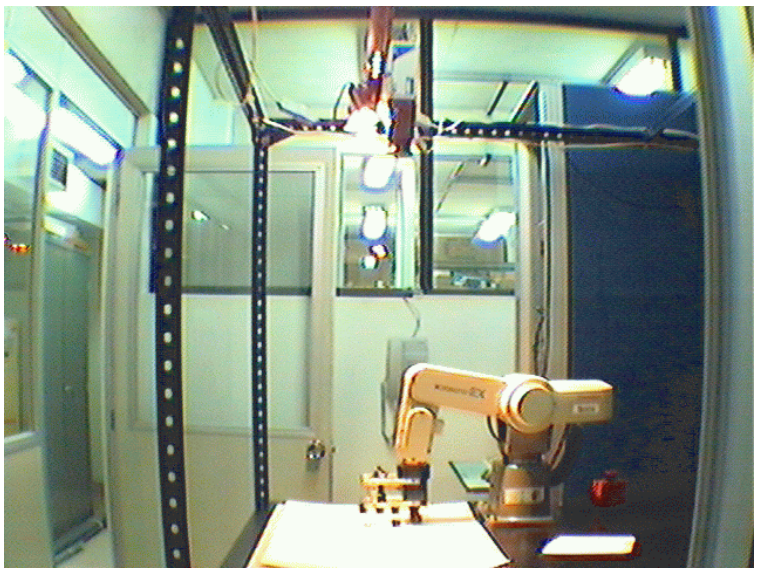


Figura 1.3 - O manipulador utilizado da Célula de Montagem.

- **Perceber o mundo**

O robô deve possuir capacidades visuais para perceber o mundo: reconhecer e localizar partes da montagem e objetos não desejados e descobrir movimentos na cena.

- **Lidar com problemas que surgem em tempo real**

A tarefa que evita colisões tem que ser realizada em tempo real para proteger o humano e preservar os dispositivos.

- **Raciocinar com informação incompleta e eventos imprevisíveis**

Interações humanas podem acontecer a qualquer hora interferindo na tarefa de montagem de modos diferentes e imprevisíveis: podem ser adicionados objetos não desejados ou novas peças para a montagem e o robô tem que ser capaz de lidar corretamente com esses eventos.

Foram definidos três comportamentos independentes para o sistema, cada qual correspondente a uma hierarquia de tarefas e implementado em um agente autônomo. Esses comportamentos são:

1. Realizar a montagem pegando peças na área de trabalho com o manipulador e colocando-as nas posições desejadas, sobre os respectivos encaixes. O objetivo da montagem e a forma e o tipo das peças podem mudar, por exemplo, de uma montagem de um objeto conhecido para a seleção de peças por forma ou cor.
2. Desobstruir a área de trabalho. A área de trabalho é um espaço previamente definido como importante para a montagem. Objetos não desejados devem ser retirados.
3. Evitar colisões do manipulador com objetos que se movam na área de trabalho, com o objetivo de preservar a integridade física do sistema, evitando acidentes.

Uma característica importante dessa arquitetura é permitir a combinação das abordagens reativa e deliberativa em um mesmo sistema, já que os agentes podem ser capazes de reagir a eventos não previsíveis e assíncronos que ocorram no ambiente, sem se engajar em deliberações complexas, quando devem atuar rapidamente (evitando colisões), assim como desenvolver planos e tomar decisões, para decidir como a montagem será realizada.

Cada agente é executado como um processo independente lógica e fisicamente, uma vez que cada um está alocado a uma estação de trabalho diferente, possibilitando o processamento paralelo. A comunicação entre os agentes é feita através da rede ethernet local.

A proposta desta dissertação surge como resultado do trabalho do autor na Divisão de Automação e Inteligência Artificial do Laboratório de Sistemas Integráveis da Escola

Politécnica da USP. Como estudante de iniciação científica, participou do projeto do sistema RECTRI, cujo objetivo é o reconhecimento tridimensional de peças em uma célula de montagem [RILLO, A., 1994], iniciando sua familiarização com a área de Visão Computacional. Como estudante de mestrado, trabalhou na área de Sistemas de Planejamento Inteligente, na integração do sistema NONLIN em uma célula de montagem [BIANCHI et al., 1995]. Devido às dificuldades encontradas na implementação do sistema de planejamento na Célula de Montagem - sem uma integração com um sistema perceptivo - e juntamente com a influência do trabalho de BROOKS (1991), o trabalho convergiu para uma proposta integrada, onde percepção e ação são dedicadas a comportamentos específicos que cooperam entre si para realizar tarefas complexas.

1.4 Organização do trabalho

O sistema inteligente proposto neste trabalho concentra na visão seu meio de interação perceptual com o ambiente. Desse modo, o segundo capítulo deste trabalho fornece um breve histórico da área de Visão Computacional, apresentando e discutindo os paradigmas existentes. No terceiro capítulo, são introduzidos alguns conceitos referentes à Inteligência Artificial Distribuída (DAI), que é usada para a modelagem da arquitetura proposta. No quarto capítulo é fornecida uma descrição da arquitetura de controle proposta, seguindo a modelagem de Sistemas Multi-Agentes. No capítulo cinco são discutidos alguns trabalhos relacionados às arquiteturas de controle de sistemas inteligentes e a sistemas de visão computacional, que serviram de fundamentação para este trabalho. No sexto capítulo é detalhada a implementação da arquitetura proposta, com a descrição da biblioteca desenvolvida para a construção de agentes e da implementação no domínio de montagem. O sétimo capítulo apresenta os testes realizados, descrevendo os resultados obtidos. No oitavo capítulo apresentam-se as conclusões deste trabalho e discute-se a metodologia utilizada. Finalmente, são apresentadas propostas de extensões que podem ser realizadas em trabalhos futuros.

2. HISTÓRICO DA ÁREA DE VISÃO COMPUTACIONAL

Neste capítulo é apresentado um panorama da área de Visão Computacional, descrevendo os seus paradigmas, discutindo suas definições de visão e a relação entre eles.

2.1 Introdução

As primeiras tentativas de construir um sistema de visão computacional são atribuídas por JOLION (1994) à Levialdi nos anos 50. Foram trabalhos na área de física de altas energias, onde se tentava analisar imagens provenientes de experimentos em câmaras de bolhas, realizando um processamento digital de imagens.

Uma definição tradicional da área de visão computacional como disciplina foi proposta por TRIVEDI e ROSENFELD (1989): “visão computacional é a disciplina que investiga as questões computacionais e algoritmicas associadas à aquisição, ao processamento e à compreensão de imagens”. Somada à Neurofisiologia e à Psicologia Perceptual, formam o conjunto das disciplinas que estudam a visão (Figura 2.1).

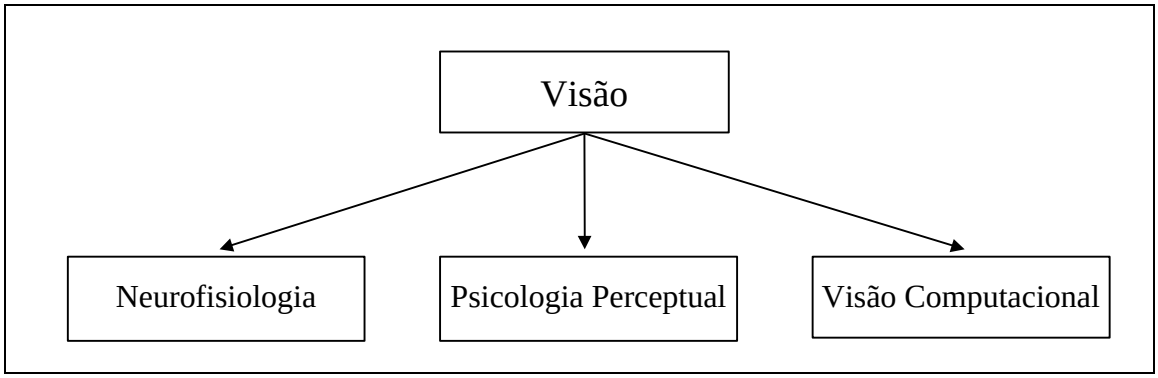


Figura 2.1 - Disciplinas que investigam a visão [TRIVEDI; ROSENFELD, 1989].

Outra definição bem aceita pelos pesquisadores da área é a de MARR (1982), que afirma: “visão é o processo que produz, a partir de imagens do mundo externo, uma descrição que é útil ao usuário e que não é repleta de informações irrelevantes” [MARR, 1982, p.31].

Uma definição mais informal e também mais polêmica da área de visão computacional é dada por TARR e BLACK (1994, p.65), que afirmam que o objetivo da área é a compreensão e modelagem de um sistema de visão de propósito geral em seres

humanos e em máquinas. A crítica mais comum a essa afirmação é que o chamado "Sistema de Visão de Propósito Geral", assim como os termos "Problema Geral da Visão", "Problemas Mal Postos" e "Problema Genericamente Solucionável", não são bem definidos nem tampouco compreendidos [TSOTSOS, 1994] e que a maioria dos problemas em visão são NP-completo ou NP-hard, como provado por TSOTSOS (1989), KIROUSIS e PAPADIMITRIOUS (1985), COOPER (1992), entre outros. Além disso, TSOTSOS (1990) afirma que a visão humana não soluciona esses "problemas genéricos da visão".

Na década de 50 os primeiros pesquisadores da área acreditavam que o problema da visão seria resolvido rapidamente. A área porém se apresentou mais complexa que o imaginado inicialmente. Um dos problemas fundamentais é que visão é um problema mal posto, isto é, a imagem bidimensional de uma cena não possibilita que se construa uma única descrição tridimensional da cena em questão, pois não existem equações geométricas suficientes para encontrar todas as incógnitas necessárias à reconstrução.

Assim, para resolver o problema visual, deve-se encontrar restrições que possam ser usadas para contorná-lo, tornando-o bem posto e possibilitando sua solução. Uma das características fundamentais que diferenciam os paradigmas existentes é a origem e grau das restrições usadas para resolver o problema visual.

A primeira teoria que tentou metodologicamente resolver o problema visual, restringindo-o, foi proposta por Marr em 1982.

2.2 A teoria de Marr

Segundo JOLION (1994), "David Marr foi o primeiro a propor (em seu livro chamado Vision [MARR, 1982]) uma metodologia completa para a visão computacional, que ficou conhecida como o paradigma de Marr".

Antes dele, Gibson foi o que mais se aproximou da proposição de uma teoria computacional para a visão, em 1966. "Sua maior contribuição foi abandonar o debate filosófico e apontar que o fato mais importante sobre os sentidos é que eles são canais para a percepção do mundo real - no caso da visão, as superfícies visíveis" [MARR, 1982, p.29].

A metodologia proposta por Marr consiste na divisão em "três níveis diferentes

nos quais um dispositivo de processamento de informações deve ser compreendido" [MARR, 1982, p.24]. Neles estão contidos a teoria computacional do dispositivo (no nível superior), o algoritmo e tipo de representação usados (no nível intermediário) e os detalhes da implementação física (no nível inferior). JOLION (1994) acredita que esta metodologia de trabalho é uma das principais contribuições de Marr.

Teoria Computacional (nível superior)	**Representação e algoritmo (nível intermediário)**	**Implementação física (nível inferior)**
Qual é o objetivo da computação, por que ela é conveniente e qual é a lógica da estratégia que pode realizá-la?	Como esta teoria computacional pode ser implementada? Em particular, qual é a representação para as entradas e saídas e qual é o algoritmo para a transformação?	Como a representação e o algoritmo podem ser implementados fisicamente?

Tabela 2.1 - Os três níveis nos quais qualquer máquina realizando um processamento de informação deve ser compreendida. [MARR, 1982, p.25]

Marr ainda propôs a divisão da derivação das formas de um objeto a partir de uma imagem em três estágios de representação (de complexidade crescente) [MARR, 1982, p.37]:

1. O primeiro estágio, chamado esboço primário, representa as propriedades importantes da imagem bidimensional, como mudanças de intensidade e distribuição e organização geométrica. Algumas primitivas usadas para a construção desta representação são: os cruzamentos de zero, terminações e descontinuidades, segmentos de bordas, linhas virtuais, grupos, organizações, curvas e fronteiras.
2. O segundo estágio, chamado esboço 2 ½-D, representa algumas propriedades das superfícies visíveis em um sistema de coordenadas centrado no observador. Entre essas propriedades estão a orientação e distância do observador às superfícies visíveis (estimativas), contornos de descontinuidades destas grandezas, reflectância das superfícies e uma descrição aproximada da iluminação. Algumas primitivas usadas nesta representação são orientação local de superfícies, distância do observador, descontinuidades na profundidade

e na orientação das superfícies.

3. O terceiro estágio, chamado modelo 3-D, representa a estrutura tridimensional e a organização espacial das formas observadas, usando uma representação hierárquica modular que inclui primitivas volumétricas e superficiais. As primitivas são modelos tridimensionais das formas, em um sistema de coordenadas centrado nos objetos, organizados hierarquicamente, cada um baseado em um conjunto de eixos no espaço, aos quais formas volumétricas ou de superfície são fixadas, junto com alguma informação sobre as propriedades das superfícies.

A terceira proposição de Marr é a da divisão do problema da visão em subproblemas independentes, tratados como módulos independentes, que é uma aplicação da metodologia tradicional de IA de “dividir para conquistar”. Com isso, diferentes módulos foram definidos, como estereoscopia, *shape from X*, análise de movimento, etc.

Finalmente, Marr realiza várias suposições sobre as características do mundo físico para poder restringir o problema de visão, como a existência de superfícies suaves, a existência de uma organização hierárquica da estrutura espacial, similaridade, continuidade espacial, continuidade das descontinuidades e continuidade do fluxo ótico. Esta assunção de restrições a partir do mundo físico é uma das características da teoria de Marr.

A importância da teoria de Marr para a disciplina de visão computacional é indiscutível. Para ALOIMONOS (1994), estas contribuições estabeleceram as bases que tornaram a área uma disciplina científica. Assim, a partir da teoria proposta por Marr, os pesquisadores da década de 80 começaram a trabalhar no que hoje chamamos de paradigma reconstrutivo [BLACK et al., 1993, JOLION, 1994].

2.3 O paradigma reconstrutivo

O paradigma reconstrutivo pode ser considerado uma interpretação da teoria de Marr, sobre a qual pesquisadores tiveram que fazer adições e outras suposições para poderem construir aplicações reais. Deve-se notar que, apesar do reconstrutivismo ter sua base na teoria de Marr, as duas teorias possuem diferenças.

De acordo com o paradigma reconstrutivo, o objetivo da pesquisa em visão é “a reconstrução em uma representação precisa de uma cena tridimensional observada e suas propriedades, a partir de informações contidas nas imagens, tais como: sombreamento, contornos, estereoscopia, cores, etc” [BLACK et al., 1993; JOLION, 1994]. Assim, sua finalidade principal está voltada para a reconstrução do mundo em uma representação completa. Esta abordagem levou a muitas contribuições (principalmente teóricas), resultando em algumas técnicas matemáticas, por exemplo, relacionadas à descontinuidades e regularização.

Outra definição dos objetivos do paradigma reconstrutivo, formulada por TARR e BLACK (1994), consiste em “construir uma descrição simbólica de uma cena do mundo real em um modelo, que possa ser usado como interface para outros processos cognitivos - raciocínio, planejamento inteligente de atividades -, fazendo isto através de descrições funcionais do mundo visível” [TARR; BLACK, 1994].

O paradigma reconstrutivo justifica a necessidade de representações intermediárias como modo de reduzir o custo computacional da visão, afirmando que comportamentos complexos em ambientes desconhecidos só se tornam possíveis através do uso de representações poderosas. O reconstrutivismo normalmente objetiva a visão genérica (isto é, possibilidade de realizar comportamentos complexos) por uma abordagem de baixo para cima (*bottom-up*), baseada em dados (*data-driven*).

Por ter bases na teoria de Marr, este paradigma recebe todas as críticas feitas àquela [JOLION, 1994], além de outras específicas ao paradigma reconstrutivista [TARR e BLACK, 1994; EDELMAN, 1994]:

- não é possível sempre construir uma representação objetiva de uma cena;
- reconstruir o mundo em um modelo tão complexo quanto ele mesmo é inútil;
- não é sempre necessário construir um modelo tridimensional completo da cena para atingir um objetivo particular;
- a geometria não é o único modelo existente para representar o mundo e para sua compreensão;
- o paradigma de Marr é baseado fortemente na teoria de computação clássica de Von Neumann, com abordagem seqüencial;
- cenas não são tão simples como superfícies regulares;

- restrições podem surgir de outras fontes além do mundo externo - físico;
- devemos pensar também no que podemos obter da cena, não só o que deseja-se.
- os algoritmos de reconstrução não são robustos em presença de ruídos;
- os algoritmos são ineficientes;
- o paradigma reconstrutivista falhou em levar em conta a demanda computacional de um perceptor do mundo real;
- a reconstrução por si só não contribui em nada para a interpretação da cena;
- existem evidências psicofísicas que a hipótese reconstrutivista é inconsistente com a performance humana no reconhecimento;

Algumas destas críticas - as "dependentes de hardware" - estão sendo rebatidas por autores como TARR e BLACK (1994), que acreditam que elas são verdadeiras hoje devido somente ao estado da arte dos computadores, mas que melhoras já estão acabando com esses problemas e que "estamos entrando em um período estimulante no qual todas as críticas serão resolvidas" [TARR & BLACK, 1994]. Apesar disso, eles afirmam que essas críticas foram boas para repensar e redirecionar a pesquisa na área.

Estas críticas ainda merecem comentários:

- quanto à falta de robustez, argumentam que o uso de modelos probabilísticos propicia uma solução melhor para a reconstrução do modelo do mundo;
- para alguns problemas a referida ineficiência dos algorítmos é circunstancial e freqüentemente surgem algoritmos mais eficientes;
- quanto à subestimativa da demanda computacional, pode-se dizer que todos os pesquisadores sistematicamente subestimam o esforço necessário para se extrair alguma informação útil dos sensores;
- a inapropriação para interpretação de cenas só é verdade se é desejada uma interpretação semântica da cena, o que é difícil sob qualquer enfoque teórico;
- apesar do estudo dos modelos naturais ser interessante, pois proporcionam fonte de informações e questionamento, evidências psicofísicas contrárias por si só não invalidam a teoria: existem muitos exemplos onde a solução tecnológica encontrada não é igual à solução biológica e evolutiva, como os

carros, que não possuem patas, e os aviões, que não batem asas.

Muitos pesquisadores acreditam que o paradigma reconstrutivo ainda possui fôlego, e “não só é promissor para atingir os objetivos da visão computacional, mas também possui boa fundamentação computacional e evolutiva” [TARR; BLACK, 1994]. Porém, outros pesquisadores acreditam que algo mais é necessário para resolver o problema de visão. Procurando este algo mais, novos paradigmas foram propostos. O primeiro a surgir foi a chamado visão ativa.

2.4 Visão Ativa, Animada e Qualitativa

Em 1988 surge uma nova proposta para se encontrar restrições a fim de tornar tratável o problema de visão. Nela, a visão controla o processo de aquisição de imagem “introduzindo restrições físicas que facilitam a reconstrução da informação da cena tridimensional” [ALOIMONOS, 1994]. O processo de visão passa a “procurar informação ativamente e pára de considerar que a informação seja captada acidentalmente (passivamente) pelo sensor” [BAJCSY; CAMPOS, 1992, p.31]. A este novo paradigma deu-se o nome de ativo ou visão ativa.

Este paradigma é baseado no trabalho de BAJCSY (1988), que definiu a percepção ativa como o estudo da modelagem e das estratégias de controle para percepção. Ela propôs uma nova metodologia para modelar um sistema perceptivo, fortemente influenciada pela teoria de controle. Nesta metodologia, modelam-se em três níveis os sensores, os módulos de processamento e a interação entre estes módulos. Esta modelagem pode ainda ser global, se ela é aplicada a um sistema como um todo, ou local, se ela modela algo específico, como distorções de uma lente, resolução espacial, filtros e etc.

BAJCSY ainda sugeriu uma nova proposta para a divisão da derivação das formas de um objeto, baseada no controle ativo dos sensores e na proposta original de Marr. Esta possui 5 etapas [BAJCSY, 1988, p.998]:

1. Controle dos dispositivos físicos. Os objetivos nesta etapa são foco grosseiro da cena, o ajuste da abertura, o foco no sujeito e a tentativa de encontrar distâncias a partir do foco.
2. Controle dos módulos visuais de baixo nível e o controle de um aparato

binocular. Os objetivos são a segmentação da imagem 2-D, cálculo do número de bordas e regiões da imagem.

3. Controle de um módulo visual geométrico. O objetivo é a construção do esquema 2 ½-D.
4. Controle de processo de integração de várias vistas da cena e correspondência entre dados e modelos. O objetivo é a descrição de objetos tridimensionais na cena através de categorias volumétricas.
5. Controle da interpretação semântica. O objetivo é a descrição tridimensional da cena em um modelo que pode ter várias complexidades.

Muitos autores, como TARR e BLACK (1994), acreditam que a visão ativa não é um paradigma novo, totalmente diferente do reconstrutivo, mas “uma reformulação do paradigma reconstrutivo tradicional, onde somente se adiciona a exploração dinâmica do ambiente” [TARR; BLACK,1994, p.67], não abandonando todos os outros preceitos da teoria de Marr. Eles acreditam que o paradigma é “uma técnica promissora para desenvolver algoritmos robustos de visão e compreensão da visão biológica, pois aumenta a entrada de dados visuais e pode resolver problemas que são intratáveis no modo tradicional” [TARR; BLACK,1994, p.67].

Os defensores de visão ativa indicam que estas afirmações seriam verdadeiras somente se os sensores se movimentassem aleatoriamente. Mas como o paradigma incorpora na modelagem a intenção explícita do controle dos sensores, ele é fundamentalmente diferente do reconstrutivo. O controle da aquisição da informação para criar mais restrições, adquirindo mais imagens de posições diferentes, escolhidas pelo sistema, é a característica mais importante deste paradigma.

Estas novas idéias foram bem aceitas pela comunidade, que logo em seguida viu que as novas idéias ainda não tinham acabado. Assim, após o surgimento da Visão Ativa, o paradigma Animado foi proposto por Dana Ballard (1991). Neste paradigma, "o objetivo não é construir uma representação interna explícita e completa do mundo, mas exibir comportamentos que resolvam problemas” [BALLARD; BROWN, 1992, p.17].

Segundo Ballard “o comportamento visual mais importante é o controle de fixação (*gaze control*) ” [BALLARD, 1991]. O uso deste comportamento gera restrições adicionais que simplificam enormemente a solução do problema de visão.

Outra característica fundamental da visão animada é o uso exaustivo de aprendizado de comportamentos, pois ele reduz o custo computacional a longo prazo e raramente produz erros irreversíveis.

Algumas vantagens citadas por BALLARD (1991) em favor do paradigma são:

1. “sistemas de visão animada podem usar a procura física: o sistema pode mover as câmeras para chegar mais perto de objetos, mudar o foco ou o ponto de observação;
2. os sistemas podem realizar movimentos na câmera usando aproximações;
3. possibilita o uso de sistemas de coordenadas excêntricos;
4. possibilita o uso de algoritmos qualitativos;
5. o controle de fixação permite a segmentação da imagem em áreas de interesse;
6. possibilita a exploração do contexto ambiental;
7. a visão animada se adapta muito bem a algoritmos de aprendizado que usam referências indexadas” [BALLARD, 1991, p.62].

Neste item que trata das idéias que surgiram para tentar eliminar os problemas do reconstrutivismo, falta citar ainda um paradigma, o qualitativo. Nele, a idéia central é que “os comportamentos podem não necessitar de representações elaboradas do mundo tridimensional” [ALOIMONOS, 1992], aumentando assim o desempenho dos sistemas de visão que não precisam reconstruir todo o mundo físico.

Pode-se perceber que novas abordagem para solucionar o problema da visão foram introduzidas com os paradigmas apresentados acima, e foram se desenvolvendo quase paralelamente no tempo. Uma última abordagem é apresentada a seguir.

2.5 O paradigma propositado

O paradigma propositado (*purposive*) [ALOIMONOS, 1994; BLACK et al., 1993; JOLION, 1994; RIVLIN et al., 1991; SWAIN; STRICKER, 1991] surgiu como uma evolução e integração dos paradigmas ativo, animado e qualitativo, e às vezes, é confundido com algum desses. Além de reunir as idéias dos três paradigmas citados, ele propõe que “a visão deve ser considerada dentro de um contexto de tarefas que um agente deve realizar, e procura retirar dos propósitos do agente as restrições para

solucionar o problema mal posto de visão" [ALOIMONOS, 1994]. Assim, a única justificativa para se selecionar o que inserir em um modelo do mundo real é a finalidade deste modelo.

Outra característica fundamental do paradigma é a grande integração da visão com outras áreas de inteligência artificial, como planejamento inteligente de atividades, raciocínio e aprendizado - resultado da crença que visão não é um problema contido em si, e que pode ser mais facilmente resolvido se for integrado com estes sistemas.

Deste modo, ao tentar solucionar o problema da visão, deve-se questionar o que se deseja reconhecer, levando a uma questão diretamente ligada às tarefas visuais, isto é, ao propósito. Assim, o pensamento propositado coloca questões cujas respostas levam a soluções de tarefas específicas e não de uso geral, tornando-se um paralelo à teoria computacional proposta pelos reconstrutivistas.

Autores como TARR e BLACK (1994), definem que o objetivo do paradigma "é construir sistemas para resolver tarefas específicas" e "o estudo das tarefas que organismos com visão podem fazer".

Segundo ALOIMONOS (1994), "os trabalhos de Brooks e Marr estão nos finais opostos de um espectro onde a visão propositada se encontra no meio" [ALOIMONOS, 1994, p.74]. É evidente a influência dos trabalhos de BROOKS (1986 e 1991) no surgimento da visão ativa, qualitativa, animada e propositada, nas quais a decomposição em tarefas, comportamentos propositados e muitas outras características daquele trabalho são usadas.

A popularidade da abordagem propositada tem crescido bastante, uma vez que ela alcança melhores resultados práticos na interação entre robôs inteligentes e ambientes complexos.

A principal crítica da comunidade reconstrutivista em relação à propositada tem sido a de que soluções propositadas não possibilitam fácil expansão para tratamento de problemas mais sofisticados, questionando a possibilidade de adquirir comportamentos mais complexos com a simples montagem de comportamentos mais simples.

TARR e BLACK (1994) duvidam que qualquer sistema possa ser compreendido em um contexto tão estreito como este, pois concordam com MARR (1982): "Em um sistema geral, restrições que são particulares a uma cena fazem com que o sistema

aumente de complexidade para gerenciar o grande número destas restrições, ou seja, o aumento de módulos independentes - que surgem de restrições diferentes - aumenta a complexidade e que no ser humano esse número é muito grande" [MARR, 1982].

Outra questão que surge é quanto à provável duplicação de esforços ao se tentar construir módulos distintos para comportamentos diversos. Algumas outras críticas ao paradigma propositado, compiladas por TARR e BLACK (1994), são:

- "a visão propositada é enganadora, porque não possuímos ainda um conhecimento adequado de quando processamentos específicos para certas tarefas e críticos em relação ao tempo devem ser implementados e porque não compreendemos as computações que podem surgir da combinação de muitos processos;
- algumas suposições do paradigma propositado são apenas uma reformulação daquelas feitas pelo paradigma reconstrutivo;
- o propositivismo só serve para resolver problemas simples e casos particulares"

TARR e BLACK (1994) acreditam que o paradigma propositado defende uma mudança nos objetivos da área de visão computacional que é simplesmente um desvio para evitar o estudo dos problemas reais da área de visão computacional, que é "explicar e implementar comportamentos visuais complexos". Acreditam que o paradigma defende o abandono da busca por sistemas de visão inteligentes, por não querer tratar problemas como sistema de visão de propósito geral e a visão humana, "desistindo dos objetivos de inteligência artificial, psicologia cognitiva e neurociências". Por isso, eles afirmam que esta não é uma alternativa ao paradigma reconstrutivo.

Porém, já foram apresentadas muitas críticas contra reconstrutivistas e propositivistas. Assim, estabeleceu-se uma discussão entre reconstrutivistas e propositivistas, que tem resultado em artigos onde ânimos exaltados defendem suas abordagens como a mais apropriada para o estudo em Visão Computacional [TARR e BLACK, 1994; TSOTSOS, 1994; ALOIMONOS, 1994].

O propositivismo procura atingir a visão genérica através de uma abordagem de cima para baixo (*top-down*), orientada aos objetivos (*goal-oriented*), dirigida pelas tarefas (*task-driven*). Desta maneira, a visão de propósito geral emergirá de uma organização de diversas soluções dedicadas a diferentes tarefas visuais. Assim, o

problema principal torna-se: como organizá-las e quais são as tarefas primitivas, voltando o enfoque para arquiteturas de integração dos sistemas visuais.

É na busca de soluções para este importante problema que se encontra o maior enfoque desta dissertação. Para tentar resolvê-lo, este trabalho recorre a metodologias empregadas nas áreas de Inteligência Artificial Distribuída e Sistemas Multi-Agentes.

3. INTELIGÊNCIA ARTIFICIAL DISTRIBUÍDA

3.1 Introdução

Com suas raízes no meio da década de 70, o campo de Inteligência Artificial Distribuída (DAI) tem apresentado um grande crescimento nos últimos anos e o termo mais utilizado dessa teoria, “Agentes”, popularizou-se. Esse crescimento se deve em parte ao aumento do potencial de conectividade dos computadores, que somado à explosão da utilização das redes de computadores está permitindo a implementação de aplicações distribuídas a um baixo custo. O crescimento exponencial da World Wide Web, outro aspecto do mesmo crescimento das redes, também vem incentivando as pesquisas em DAI, já que a WWW é um ambiente explicitamente distribuído que necessita de novas aplicações.

Apesar da ampla utilização dos agentes, nem todas as aplicações estão baseadas em DAI. Assim, este capítulo inicia com uma discussão sobre o que são agentes. Em seguida, apresenta-se o campo de Inteligência Artificial Distribuída, a base para a arquitetura apresentada neste trabalho, e suas sub-áreas, Solução de Problemas Distribuída (DPS) e Sistemas Multi-Agentes (MAS). Ao final, apresenta-se outras abordagens distribuídas para Inteligência Artificial, não incluídas na área de DAI.

3.2 Agentes

Uma das questões mais polêmicas do campo de Inteligência Artificial, que há anos está sendo discutida, é a definição de o que é um agente.

Para HUHNS e SINGH (1997), “Existem dois pontos de vistas em extremos opostos sobre os agentes. Uma bem estabelecida tradição considera os agentes como entidades essencialmente cognitivas e conscientes, que possuem sentimentos, percepção e emoção exatamente como os seres humanos. Sob este ponto de vista, todo o trabalho computacional feito sobre agentes atualmente é inerentemente inadequado. O outro ponto de vista assume que agentes são meramente autômatos e se comportam exatamente como foram projetados ou programados. Este ponto de vista admite uma grande variedade de sistemas computacionais, incluindo agentes computacionais. Uma preocupação é que ele pode ser muito permissivo” [HUHNS; SINGH, 1997, p. 1].

Dentro desse amplo espectro, existem inúmeras definições para “agentes”. FRANKLIN e GRAESSER (1996) apresentam uma coleção dessas definições, das quais as mais relevantes estão resumidas na Tabela 3.1.

Autor	Definição
RUSSELL e NORVIG, 1995, p. 33	“Um agente é qualquer coisa que pode ser vista como percebendo seu ambiente através de sensores e agindo sobre este ambiente através de efetuadores.”
MAES, 1995, p.108	“Agentes Autônomos são sistemas computacionais que habitam algum ambiente dinâmico e complexo, percebem e atuam autonomamente neste ambiente e, fazendo isto, atingem um conjunto de objetivos ou tarefas para os quais foram projetados.”
SMITH, CYPHER e SPOHRER, 1994	“Um agente é definido como uma entidade de software persistente dedicada a um propósito específico.”
HAYES-ROTH, 1995	“Agentes inteligentes realizam continuamente três funções: percebem as condições dinâmicas em um ambiente; agem para afetar as condições do ambiente; e raciocinam para interpretar as percepções, resolver problemas, realizar inferências e determinar ações.”
WOOLDRIDGE e JENNINGS, 1995, p. 2	“... um sistema computacional baseado em hardware ou (mais habitualmente) em software que possui as seguintes propriedades: autonomia, habilidade social, reatividade e pró-atividade.”
FRANKLIN, 1995	“Agentes Autônomos são sistemas capazes de ações autônomas e propositadas no mundo real.”

Tabela 3.1 - Resumo de definições de agentes em [FRANKLIN; GRAESSER,1996].

A partir dessas definições, apesar de variadas, algumas características básicas que os agentes devem possuir podem ser definidas. HUHNS e SINGH (1997) discutem profundamente essas características, resumidas na Tabela 3.2. Esta tabela apresenta uma grande variedade de conceitos, alguns dos quais definem áreas de atuações. Por exemplo, o nível de autonomia de um agente é estudado por grupos de IA como o de MAES (1995), que já realizaram dois congressos internacionais exclusivamente sobre agentes autônomos; o nível cognitivo de um agente define se este faz parte do grupo dos sistemas reativos ou dos sistemas deliberativos (ou ambos); os aspectos sociais são estudados principalmente pelos grupos de Sistemas Multi-Agentes. Algumas dessas características serão discutidas com mais detalhes nos próximos itens do capítulo.

Característica	Propriedade	Valores possíveis
Intrínsecas do agente	Tempo de duração	de transiente a de vida longa
	Nível cognitivo	de reativo a deliberativo
	Construção	de declarativo a procedimental
	Mobilidade	de estacionário a itinerante
	Adaptabilidade	fixa - lecionável - autodidata
	Modelagem	do ambiente, dele próprio ou de outros agentes
Extrínsecas do agente	Localidade	do local a remoto
	Autonomia social	do independente a controlado
	Sociabilidade	autista, ciente, responsável, membro de um time
	Amabilidade	cooperativo - competitivo - antagonista
	Interações	logística: direta ou com facilitadores; nível semântico: declarativas ou procedimentais
do Sistema (sociedade de agentes)	Unicidade	de homogêneo a heterogêneo
	Granularidade	de fina a grossa
	Estrutura de controle	hierárquica a democrática
	Autonomia de interface	especifica comunicação - intelecto - habilidades
	Autonomia de execução	independente ou controlado
do Framework	Autonomia de projeto	plataforma - linguagem - arquitetura interna - protocolo de interação
	Infra-estrutura de comunicação	memória compartilhada ou baseado em mensagens; conectado ou não; ponto-a-ponto - multicast - broadcast; push ou pull; síncrono ou assíncrono
	Serviço de mediação	baseado em ontologias; transacional
	Protocolo de mensagens	KQML; HTTP e HTML; OLE; CORBA; DSOM
do Ambiente	Conhecimento	quanto o agente conhece do ambiente?
	Previsibilidade	quanto o agente pode prever sobre o ambiente?
	Controlabilidade	quanto o agente pode controlar o ambiente?
	Historicidade	estados futuros dependem de estados passados?
	Teleologicidade	outras partes do ambiente possuem propósito? (i.e. existem outros agentes?)
	Tempo real	o ambiente se modifica enquanto o agente delibera?

Tabela 3.2 - Resumo das características envolvidas nos sistemas multi-agentes.

Finalmente, RUSSELL e NORVIG (1995) afirmam que “a noção de um agente deve servir como uma ferramenta para a análise de sistemas e não uma caracterização absoluta que divide o mundo em duas categorias: a dos agentes e a dos não agentes” [RUSSELL; NORVIG, 1995, p.33].

Assim, este item é concluído com uma definição que contém alguns conceitos mais habitualmente aceitos: “Agentes são componentes (de software) ativos e persistentes que percebem o mundo, raciocinam, agem e se comunicam” [HUHNS; SINGH, 1997, p.1].

3.3 Histórico de DAI e suas sub-áreas

Segundo GASSER (1997) “A comunidade contemporânea de pesquisadores trabalhando com agentes provém de muitas linhas e grupos diferentes, mas principalmente da família de pesquisadores de Inteligência Artificial Distribuída / Sistemas Multi-Agentes (DAI/MAS), que já conta com 25 anos de trabalhos”. E mais: “No meio da década de 70, pesquisadores na área de DAI/MAS começaram a formular teorias básicas, arquiteturas e experimentos que mostraram, computacionalmente, como interações e divisão de trabalho podem ser efetivos na solução de problemas, como incertezas inerentes introduzidas pela distribuição de conhecimento e ação podem ser gerenciadas e como teorias formais do conhecimento e ação podiam ser estendidas para o contexto localizado, distribuído e multi-agentes” [GASSER, 1997].

Podem ser encontradas nos anos 70 as primeiras tentativas de solucionar problemas de maneira distribuída, com projetos como o HEARSAY-II [ERMAN, 1980] - o sistema que introduziu o modelo do “Quadro Negro” (*Blackboard*) para comunicação - e a pesquisa de HEWITT e outros sobre “Atores” [HEWITT, 1977].

Na década de 80, surgiram alguns dos trabalhos mais conhecidos da área, como a *Contract Net*, um protocolo de comunicação baseado na estrutura do mercado [SMITH, 1980], e os Sistemas Abertos (*Open Systems*) [HEWITT, 1986], onde o trabalho existente sobre “Atores” foi estendido para solucionar problemas como recursos compartilhados e comportamentos complexos. Além disso, foram realizados vários congressos sobre DAI e editadas coletâneas de artigos. [BOND; GASSER, 1988].

No final da década de 80, BOND e GASSER (1988) definem Inteligência

Artificial Distribuída (DAI) como “o campo de IA que tem interesse no uso da concorrência em suas computações” e propõem a divisão de DAI em duas áreas principais: Resolução Distribuída de Problemas (*Distributed Problem Solving - DPS*), que se preocupa com a solução de problemas específicos usando diversos módulos cooperando entre si, e Sistemas Multi-Agentes (MAS), que se preocupa com a coordenação dos comportamentos de diversos agentes inteligentes autônomos para atingir um ou mais objetivos.

HUHNS e SINGH (1997) propõem alguns enfoques a partir dos quais os agentes e seus ambientes podem ser estudados:

- Racional: dividida em racionalidade lógica, que inclui tipicamente conceitos qualitativos, intenções e crenças; e racionalidade econômica (ou de mercado), que retira modelos da teoria econômica, como princípios de negociação e mecanismos de mercado.
- Social: que estuda a modelagem de sociedades de agentes utilizando abstrações provenientes da sociologia e teoria organizacional.
- Interativo: que estuda a interação entre os agentes a partir de teorias como os Atos de Fala (*Speech Acts*), que “teoriza que a linguagem não é usada apenas para comunicar idéias, mas também para realizar ações” [HUHNS; SINGH, 1997, p.16].
- Adaptativo: que adota como princípio que os agentes devem ser persistentes e adaptativos.

Finalmente, a década de 90 apresentou grandes desenvolvimentos, teóricos e práticos, com algumas tentativas de padronização para o desenvolvimento de agentes como a linguagem KQML, “que fornece uma maneira de estruturar as mensagens trocadas entre agentes” [HUHNS; SINGH, 1997, p.9].

3.4 Resolução Distribuída de Problemas (Distributed Problem Solving)

Segundo CHAIB-DRAA (1997), “o trabalho em DPS concentra-se em aplicar a força dos sistemas interligados por redes para a solução de problemas.” “(...) todo trabalho em DPS enfatiza o problema e como construir múltiplas entidades inteligentes que trabalham juntas para resolvê-lo eficientemente” [CHAIB-DRAA, 1997, p.33].

A metodologia utilizada pelos pesquisadores em DPS consiste na decomposição de um problema pré-existente em partes, na solução de cada uma destas partes, que pode envolver a interação entre elas e, finalmente, na integração dos resultados parciais para a composição de um resultado geral.

A abordagem DPS é de cima para baixo (*top-down*), geralmente associada à engenharia, enquanto MAS é uma abordagem de baixo para cima (*bottom-up*). Mas DPS e MAS não são áreas disjuntas: DPS pode ser visto como uma parte da solução de um problema na abordagem MAS. WERNER (1992) discute profundamente a relação entre MAS e DPS e o conceito de sistemas Multi-Agentes.

Enquanto DPS prioriza na solução de um problema utilizando vários agentes, o estudo em MAS é voltado às teorias dos agentes e suas características em ambientes multi-agentes.

3.5 Sistemas Multi-Agentes

MAS estuda os agentes como membros de sistemas multi-agentes, onde características como sociabilidade e amabilidade, autonomia social e interações são essenciais. Assim, normalmente os agentes são vistos como autônomos, possivelmente pré-existentes e tipicamente heterogêneos.

Para os pesquisadores de MAS, um agente pode ser definido como “um processo computacional com um único locus de controle e/ou intenção” [BOND e GASSER, 1988]. DEMAZEAU e MÜLLER usam o termo agente “em um sentido amplo para designar uma entidade inteligente, agindo racionalmente e intencionalmente com respeito aos seus próprios objetivos e o estado do seu conhecimento em um determinado momento” [DEMAZEAU; MÜLLER, 1990, p.4]. Esses autores propuseram um agente genérico que deve possuir no mínimo algumas características para interagir em um mundo Multi-Agentes (a Figura 3.1 apresenta esquematicamente o modelo desse agente). Essas características são [DEMAZEAU; MÜLLER, 1990, p.5]:

- conhecimento: é a representação do mundo e do problema que o agente tem que solucionar. Pode ser implícito ou explícito, inato ou aprendido através de comunicação e percepção;
- objetivos: como o conhecimento, pode ser implícito, explícito, inato ou

aprendido;

- capacidades de raciocínio: que podem derivar soluções possíveis para o problema do agente;
- capacidades de decisão: para definir uma escolha entre as soluções possíveis para o problema do agente.

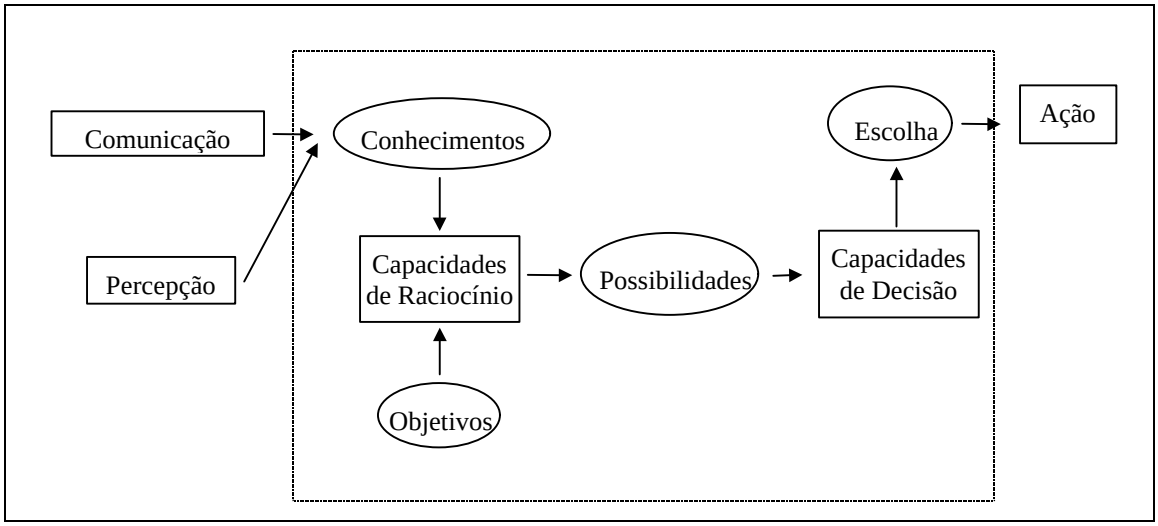


Figura 3.1 - O modelo de um agente genérico em [DEMAZEAU; MÜLLER,1990, p.5].

Para BOND e GASSER (1988), a organização dos agentes em um sistema multi-agentes, que diz respeito a como estruturar e interligar os módulos do sistema, "pode fornecer uma estrutura de restrições e expectativas sobre os comportamentos de agentes." Assim, a organização do sistema define os comportamentos dos agentes, que se refere a como os agentes vão atuar em relação aos outros agentes do sistema. Quanto à organização, um sistema multi-agentes pode ter uma estrutura:

- centralizada e hierárquica: agentes com mais informações, em um ponto mais alto da hierarquia, controlam agentes com menos informações, em níveis inferiores. Os agentes em níveis inferiores enviam dados para os seus superiores e os processos decisórios são centralizados e limitados;
- de autoridade: um agente pode controlar os agentes que estão sob sua autoridade. Esse tipo de organização reduz o trabalho coordenado;
- mercadológica: organizado com base nas leis de mercado, onde agentes competem por tarefas e por recursos;
- de comunidade com regras de comportamento: os agentes são organizados em

uma comunidade onde todos os membros obedecem algumas regras.

Os comportamentos que agentes podem exibir em um sistema multi-agentes são definidos por [DEMAZEAU; MÜLLER, 1990, p.6]:

- coabitação: um agente precisa atingir um objetivo e consegue fazer isso sozinho;
- cooperação: um agente precisa atingir um objetivo mas não consegue fazer isso sozinho, tendo que cooperar com outros agentes;
- colaboração: alguns agentes podem alcançar sozinhos objetivos que dizem respeito a todos os agentes, e apenas um agente é escolhido para realizar as tarefas necessárias;
- distribuição: somente vários agentes trabalhando juntos conseguem atingir alguns objetivos comuns.

Desse modo, DPS pode ser vista como uma abordagem propositada, voltada às tarefas, enquanto MAS, como uma abordagem de uso geral. Os críticos de MAS afirmam que ela é muito antropomórfica, genérica e sem propósito - críticas comuns aos sistemas de propósito geral, enquanto a abordagem DPS é mais adequada para um problema especificado.

3.6 Resolução Distribuída de Problemas x Sistemas Multi-Agentes

Apesar de hoje a maioria dos pesquisadores denominar tanto os módulos de um sistema DPS como os de um MAS de agentes, SICHMAN, DEMAZEAU e BOISSIER (1992) apresentam as principais diferenças entre um sistema de Resolução Distribuída de Problemas (DPS) e um Sistema Multi-Agentes (MAS):

- Em um sistema DPS não existe a representação explícita dos objetivos e habilidades dos outros agentes do sistema, enquanto em um sistema MAS essas estão representadas explicitamente em cada agente. Em um sistema DPS estas habilidades e objetivos são introduzidas por seu programador.
- Em um sistema DPS a decomposição de tarefas é feita por um programador, que evita os possíveis conflitos entre módulos, enquanto em um sistema MAS essa decomposição é feita pelos agentes, através de comunicação. Assim, mesmo que em um sistema DPS exista comunicação entre módulos,

conversações complexas não são necessárias para se atingir um objetivo do sistema.

- Um sistema DPS não é aberto - só um programador pode adicionar ou retirar módulos do sistema -, enquanto um sistema MAS é aberto.
- Nos sistemas MAS, os agentes são autônomos, existindo independentemente do problema, possuindo objetivos próprios, que podem conflitar com os objetivos de outros agentes no sistema e podendo resolver problemas para os quais não foram preparados. Em um sistema DPS os agentes são criados para resolver um problema determinado.
- Nos sistemas MAS os agentes são robustos em relação às mudanças no ambiente, pois não foram construídos para um determinado estado do mundo.

De um ponto de vista mais abrangente, ao analisar um sistema implementado é muito difícil dizer se foi desenvolvido seguindo uma metodologia DPS ou MAS. A diferença entre as duas áreas se refere mais à motivação científica e tecnológica dos pesquisadores: em DPS, são desenvolvidos agentes específicos para cada parte do problema; em MAS, o interesse está em como agentes pré-existentes podem se compor para resolver o problema.

3.7 Outras abordagens distribuídas para IA

Apesar de serem as mais organizadas e bem estudadas, DPS e MAS não são as únicas áreas que estudam a aplicação da Inteligência Artificial de maneira distribuída. Por isso, são apresentadas neste item algumas outras denominações, que em alguns casos são subdivisões do estudo dos agentes onde algumas das características apresentadas na tabela 3.2 são fundamentais e em outros, áreas de estudo de origem diversa.

- ***Inteligência Artificial Descentralizada***

A chamada IA Descentralizada (DzAI) [DEMAZEAU; MÜLLER, 1990] é outro campo de IA relacionado com os sistemas distribuídos.

A característica principal dos sistemas descentralizados é - evidentemente - não existir um controle centralizado, enquanto em sistemas DPS e MAS esse pode existir, como por exemplo, em um agente agindo como controlador em uma sociedade

organizada hierarquicamente. Assim, essa área estuda os sistemas onde a característica intrínseca dos agentes de Autonomia Social é fixa, sempre “independente”.

Atualmente, o termo IA Descentralizada está sendo pouco utilizado, sendo considerado um caso específico de MAS.

- ***Vida Artificial***

Segundo MAES (1995), “o campo de Vida Artificial tem por objetivo estudar e compreender a vida biológica através da síntese de formas de vida artificial”. “Vida Artificial compartilha com Inteligência Artificial o interesse na síntese de Agentes Autônomos Adaptativos, que habitem um ambiente dinâmico, percebam e atuem autonomamente nesse ambiente e, assim fazendo, atinjam um conjunto de objetivos ou tarefas para os quais foram projetados” [MAES, 1995, p.108].

Essa comunidade possui uma abordagem diferente da comunidade de IA para a construção de agentes, dando prioridade ao comportamento reativo, ao invés de conhecimento e raciocínio, e à adaptação e aprendizagem. Assim, esse campo pode ser visto como o dos pesquisadores que estudam agentes, tendo em vista que algumas das características apresentadas na tabela 3.2 foram fixadas: o agente é de vida longa, possui a maior adaptabilidade possível, modela a si e ao ambiente e, além de ser reativo, possui total autonomia de execução. O comportamento inteligente de uma sociedade normalmente surge da interação de agentes básicos reativos.

Esses pesquisadores têm inspiração na natureza, na biologia e principalmente na etologia, disciplina que estuda os mecanismos de comportamentos dos animais e sua organização, como em sociedades de insetos [DROGOUL, 1995].

- ***Times Assíncronos (A-Teams)***

Times Assíncronos *(A-Teams*) é uma metodologia proposta por TALUKDAR et. al. (1993) para a construção de sistemas distribuídos. Nessa proposta, um número arbitrário de agentes completamente autônomos trabalham em um determinado problema de maneira independente e cooperam assincronamente, circulando resultados através de redes fortemente cíclicas.

Segundo KANG e TALUKDAR (1997), o desafio do projeto dos Times Assíncronos é “obter uma melhor compreensão e propor novas estruturas organizacionais que permitirão a colaboração de diferentes agentes de software, cada

qual com suas capacidades especiais, de maneira a formar times para a resolução de problemas difíceis" [KANG; TALUKDAR, 1997, p. 489].

O que diferencia os trabalhos nos Times Assíncronos dos trabalhos da comunidade de DAI é que os *A-Teams* definem precisamente o que são os agentes, como eles interagem e ainda uma metodologia para a construção de sistemas.

Desse modo, um sistema baseado nos *A-Teams* é composto por dois tipos de elementos:

- memórias compartilhadas, que são depósitos de soluções de onde um agente pode ler ou escrever uma solução para o problema trabalhado. A existência destas memórias compartilhadas é a principal diferença dos conceitos básicos da teoria de MAS.
- agentes, que são construídos a partir de algoritmos para o problema e que possuem capacidades de comunicação, através da leitura e escrita nas memórias compartilhadas, de seleção e escalonamento, que decidem sobre qual problema vai trabalhar e quando.

Para a construção de um novo sistema é proposta uma receita, descrita em [KANG; TALUKDAR, 1997, p. 492]:

1. escolha um problema para o qual não exista algoritmos determinísticos eficientes;
2. decomponha esse problema em subproblemas;
3. atribua uma memória para manter um inventário de soluções para cada subproblema;
4. selecione e crie algoritmos para cada subproblema;
5. crie um agente autônomo a partir de cada algoritmo: escolha um escalonador para decidir quando e quão freqüentemente o agente irá atuar; escolha um seletor para selecionar a partir de quais memórias ele irá ler dados e para qual memória escreverá resultados; e implemente o algoritmo;
6. interconecte os agentes e as memórias.

A última característica de um sistema de Times Assíncronos é a existência de agentes iniciadores e destrutores que, respectivamente, geram soluções iniciais dos

subproblemas (geralmente aleatórias) para as memórias e que eliminam soluções ruins destas memórias.

Esse trabalho vem apresentando resultados interessantes e está sendo usado para solução de problemas de chão de fábrica, diagnóstico e controle de sistemas de potência, robôs sob demanda e escalonamento de trens, além de ter ficado mais conhecido por quebrar o recorde mundial para o tempo de solução do Problema do Caixeiro Viajante em 1992 [TALUKDAR et al, 1992]. Nesse último trabalho, os autores provaram que a organização dos algoritmos existentes em um Time Assíncrono é mais eficiente que qualquer um desses algoritmos em separado.

Finalmente, o trabalho do grupo é fortemente baseado em teoria de controle, sendo que muitas análises são realizadas sob esse ponto de vista. A Tabela 3.3 mostra uma análise dos vários tipos de sistemas para a solução de problemas sob o ponto de vista de TALUKDAR (1994).

	Fluxo de dados Cíclico	**Fluxo de dados Acíclico**
Fluxo de controle hierárquico	• Blackboards • Algoritmos genéticos • *Subsumption architecture* • *Simulated annealing* • A maioria das corporações	• Linhas de montagem supervisionadas • A maioria dos controladores em tempo real • A maioria dos pacotes de CAD
Fluxo de controle nulo	• Comunidades celulares • Sociedades de insetos • Cardumes de peixes • Redes neuronais recorrentes • Livre mercado • Comunidades científicas • Times Assíncronos	• Redes neuronais (forward) • *Systolic arrays* • Linhas de montagem não-supervisionadas

Tabela 3.3 - Taxonomia dos agentes proposta por TALUKDAR (1994) pela partição do espaço de existência dos agentes em quatro regiões nebulosas.

3.8 Aplicações

Segundo HUHNS e SINGH (1997), “uma forte indicação de que a tecnologia de agentes amadureceu é a grande quantidade e variedade de aplicações que tem surgido recentemente” [HUHNS; SINGH, 1997, p. 25]. Essas aplicações estão se dando em diversas áreas, mas principalmente nas seguintes:

- Internet e Acesso à Informação, onde os agentes mostraram-se como uma importante interface para procura e organização de informações;
- Assistentes Pessoais, [MAES et al. 1994], que são agentes que auxiliam o trabalho de um usuário, funcionando como um assistente. Esses agentes podem ou não ter uma representação gráfica, mas geralmente possuem tarefas de interface entre o usuário e o sistema;
- Aplicações Industriais, [JENNINGS et al., 1996], onde os agentes têm sido utilizados para controle de informações, controle de um acelerador de partículas, de gerenciamento de distribuição de trabalho, carga elétrica ou de telecomunicações, escalonamento e *job-shop*;
- Comércio Eletrônico, onde agentes são utilizados para realizar contratos [SANDHOLM et al., 1995];
- Controle de sistemas robóticos, onde agentes sempre estiveram presentes, agora sistemas MAS estão sendo usados para coordenação de times de robôs autônomos [MATRIC, 1998] e [VELOSO et al., 1998].

Muitas novas áreas de aplicações estão surgindo. Por exemplo, uma área onde a utilização de agentes está se iniciando é a da Visão Computacional, onde poucos trabalhos foram realizados.

4. A ARQUITETURA DE CONTROLE PROPOSTA

Nos capítulos anteriores foi feita uma revisão das áreas de Visão Computacional e Inteligência Artificial Distribuída, visando apresentar o contexto no qual está inserida a arquitetura de controle para o sistema proposto nesta dissertação. Essa arquitetura é apresentada neste capítulo.

O principal problema abordado neste trabalho é o da integração de diversos módulos com comportamentos diferentes em um sistema robótico que utiliza visão computacional para perceber o mundo.

Para solucionar este problema, propõe-se inicialmente que comportamentos independentes sejam definidos para que o propósito do sistema (tarefa de mais alto nível) seja atingido. Cada comportamento deve ser decomposto em uma hierarquia de tarefas e modelado como um agente autônomo, integrando percepção visual, planejamento, reação e execução.

A possibilidade de combinar agentes reativos e deliberativos em um mesmo sistema oferece a capacidade de atuar em ambientes que exijam atuação em tempo real, com interações imprevisíveis e assíncronas, ainda permitindo que alguns agentes sejam engajados em deliberações complexas, desenvolvendo planos e tomando decisões.

Os termos tarefa e comportamento são geralmente usados com significados similares na literatura, uma vez que são duais. Aqui comportamento é o resultado da interação de um agente com o ambiente ao tentar atingir um objetivo (por exemplo, evitar uma colisão, realizar uma montagem, etc.). A tarefa é o objetivo explícito do agente. O propósito é considerado como sendo uma tarefa de alto nível, que pode ser subdividida em tarefas menores (por exemplo, identificar uma peça, localizar uma peça, planejar uma montagem, etc.).

Para definir adequadamente a interface entre diversos comportamentos do sistema e possibilitar a representação de conhecimento distribuído, adotou-se uma abordagem Multi-Agentes. Assim, na metodologia usada para desenvolver o sistema, um agente autônomo é definido para cada comportamento do sistema. Esses agentes são então inseridos um a um em uma sociedade com regras para as interações, possibilitando a comunicação entre eles para atingir um objetivo. Nessa sociedade, todos os agentes autônomos são conectados entre si por uma rede de comunicação descentralizada e

totalmente conectada. A Figura 4.1 ilustra os aspectos de modularidade e distribuição da arquitetura, composta por Agentes Autônomos, uma rede de comunicação que permite a troca de mensagens entre os agentes e os sensores e atuadores que possibilitam a interação com o mundo.

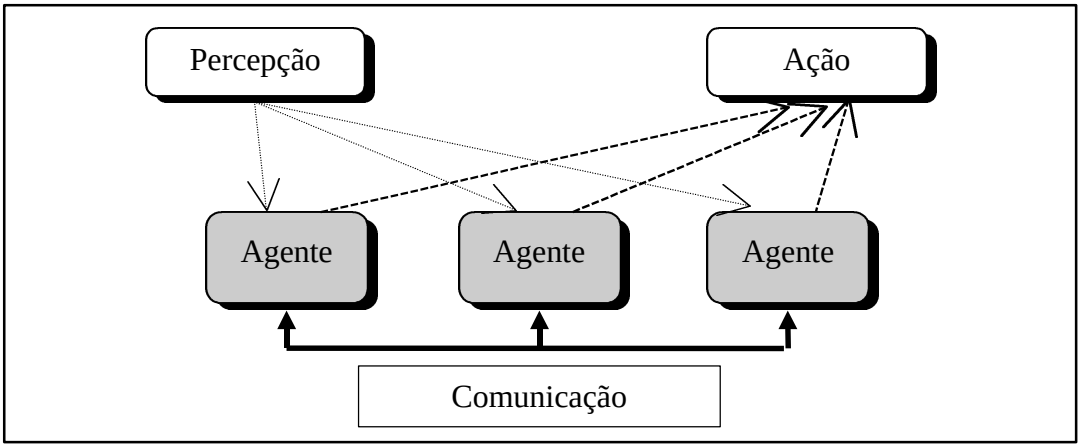


Figura 4.1 - Topologia da arquitetura.

Assim, essa arquitetura pode ser observada a partir de três diferentes níveis de abstração, baseados em seus aspectos comportamentais. No primeiro nível, encontram-se os propósitos globais, o objetivo global do sistema no domínio específico, onde o sistema é visto como uma sociedade de agentes autônomos; no segundo nível estão os comportamentos definidos para se alcançar o propósito do sistema, onde se encontram os agentes autônomos; no terceiro estão as tarefas nas quais os comportamentos podem ser divididos. (Tabela 4.1)

Nível Comportamental	**Descrição**	**Modelo Multi-Agentes**
Propósitos globais	Os Propósitos no domínio de atuação do sistema.	Uma sociedade de agentes autônomos.
Comportamentos	Os comportamentos nos quais o propósito do sistema é dividida.	Um agente autônomo para cada comportamento.
Tarefas	Tarefas nas quais os comportamentos podem ser decompostos.	Módulos que constituem as habilidades dos agentes autônomos.

Tabela 4.1 - Níveis comportamentais do sistema e a abordagem MAS proposta.

A arquitetura proposta implementa os níveis comportamentais em elementos

distintos: os propósitos são decompostos em diversos conjuntos de comportamentos, cada um voltado para uma tarefa específica. Assim, cada sistema é implementado por uma sociedade de agentes autônomos específica para um determinado domínio; cada comportamento específico que o sistema pode ter em um domínio é inserido em agente autônomo; finalmente, o agente autônomo executa diversas tarefas. A abordagem multi-agentes é usada para modelar, além dos propósitos, comportamentos, tarefas, e suas relações.

Outra característica importante da arquitetura proposta é a capacidade de realização de alocação de recursos *on-line*. Como apresentado na introdução, um recurso é definido como sendo parte do sistema que tem seu uso compartilhado pelos agentes e que pode ser controlado por somente um agente em um determinado momento. Por exemplo, o sistema de direção de um robô móvel ou um manipulador robótico em uma célula de montagem são recursos, assim como uma câmera (e seu hardware e software de aquisição de dados).

Existe uma diferença entre o nível de prioridade nos quais os recursos são compartilhados. Isso ocorre porque alguns recursos são mais disputados que outros. Por exemplo, uma câmera que consegue fornecer imagens sem criar conflito entre os agentes é menos disputada que um manipulador robótico. Por outro lado, caso essa câmera faça parte de um sistema de visão ativa, torna-se mais disputada, uma vez que os agentes competem pelo controle do processo de aquisição de dados. Por isso, existe a necessidade de uma política para a solução de conflitos na alocação de recursos entre os agentes.

4.1 Regras da Sociedade e Estrutura de Autoridade

Para gerenciar a interação entre os agentes, a sociedade é controlada por uma política que envolve regras de comportamento e uma estrutura de autoridade. Esta política possibilita que os agentes decidam qual deles deve possuir o controle de um recurso em um determinado momento.

A estrutura de autoridade define o nível de prioridade que um agente possui para usar um determinado recurso. A estrutura de autoridade da sociedade é o que torna possível a decisão sobre como os recursos do sistema devem ser alocados, decidindo

qual agente deve ter o controle de um recurso em um determinado momento, evitando conflitos. A maneira pela qual essa decisão é tomada depende das regras de comportamento definidas para a sociedade em uma implementação específica e pode ser, por exemplo, o resultado de uma competição entre agentes. Desse modo, as regras de comportamento definem como a estrutura de autoridade deve ser usada para o controle das comunicações e da alocação dos recursos.

As regras de comportamento e a estrutura de autoridade são dependentes de domínio: as regras podem mudar e os níveis de autoridade variam para cada agente e cada recurso.

A estrutura de autoridade de um sistema possui uma profunda relação com a precedência e a dependência entre os comportamento dos agentes presentes na sociedade. Em geral, os comportamentos reativos devem ter precedência sobre os deliberativos, uma vez que exigem atuação em tempo real. Dessa maneira, a estrutura de autoridade definida para os AAs é geralmente uma classificação que vai do agente mais reativo ao mais deliberativo.

O projetista de um sistema, após definir o domínio da aplicação e os seus propósitos, necessita implementar os Agentes Autônomos e definir os outros aspectos do sistema que definem a sociedade, como a estrutura de autoridade, as regras de comportamento e uma linguagem de comunicação. Estes aspectos são discutidos a seguir.

4.2 Definição dos Agentes Autônomos

Lembrando que, segundo o paradigma Propositado para Visão Computacional, em um sistema perceptual a visão não deve ser considerada como um módulo contido, mas como uma entidade que possui outras capacidades inteligentes, como planejamento inteligente de atividades, raciocínio e aprendizado, é apresentado agora o modelo dos Agentes Autônomos, que reflete essa característica de grande integração da visão com outros módulos cognitivos de inteligência artificial.

Os Agentes Autônomos são formados por 8 componentes (mostrado resumidamente na figura 4.2):

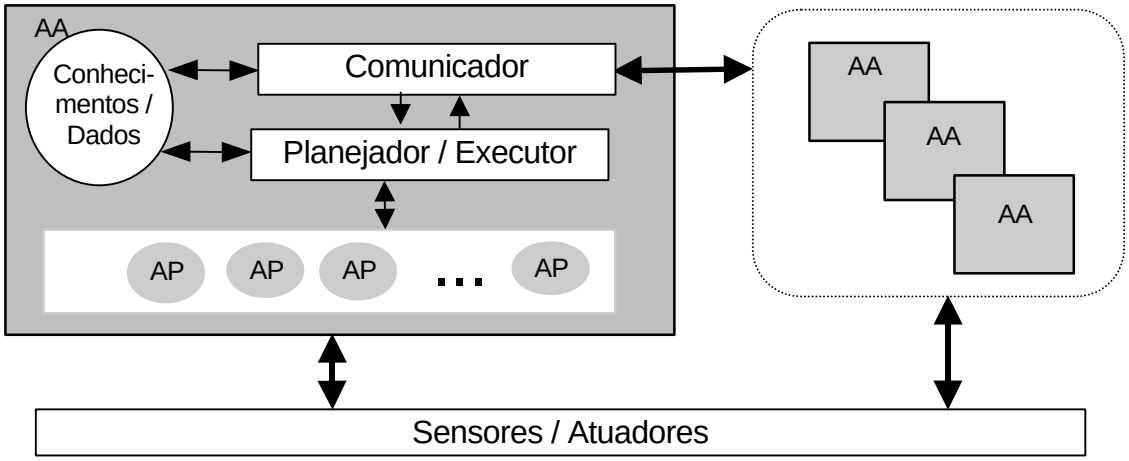


Figura 4.2 - O modelo do Agente Autônomo.

1. *Módulo de Comunicação* (comunicador): responsável pela interação entre os AAs e o módulo de planejamento/execução.
2. *Módulo de Planejamento e Execução* (planejador/executor): responsável pela geração e execução de um plano para atingir a tarefa definida para o AA, exibindo um comportamento esperado, controlando a aquisição de dados sensoriais e a atuação do agente sobre o mundo.
3. *Agentes Primitivos (APs)*: agentes que executam as tarefas das quais um AA depende, como sensoreamento e ação. Por exemplo, um AA recebe informações sensoriais de uma câmera ativando um Agente Primitivo respectivo.
4. *Protocolos e Linguagens de interação*: definem a capacidade de comunicação de um AA.
5. *Estrutura de Autoridade*: consiste na relação entre recursos e o nível de prioridade de um agente.
6. *Regras da Sociedade*: define o nível de prioridade para cada par recurso-agente.
7. *Conjunto de AAs na sociedade*: a lista com todos os Agentes Autônomos presentes na sociedade.
8. *Representação simbólica do mundo*: representa todo conhecimento sobre o ambiente que é necessário a cada agente.

Esse modelo é usado para definir todos os AAs em uma sociedade, não importando o seu comportamento. Alguns desses componentes já foram apresentados e

outros necessitam de uma descrição mais detalhada, que será feita nas seções seguintes.

4.3 Módulos de Comunicação e Planejamento/Execução

O módulo de Comunicação é responsável pela interação entre os AAs e o módulo de planejamento/execução. Esse módulo tem conhecimento dos protocolos, linguagens e políticas existentes na sociedade.

O módulo de Planejamento e Execução é responsável pela geração e execução de um plano para atingir o objetivo definido para o AA, exibindo um comportamento esperado. Como planejamento e a execução são baseados em visão, esse módulo também controla a aquisição de dados sensoriais e a atuação do agente sobre o mundo. A complexidade desse módulo depende da competência do agente, variando desde um módulo reativo, passando por um executor de planos pré-estabelecidos até podendo ser composto por um sistema de planejamento complexo.

A representação simbólica do mundo é armazenada nos módulos de comunicação e de planejamento/execução, sendo que a parte dos conhecimentos simbólicos que é utilizada pelos dois módulos é armazenada em um quadro-negro. Com isso, não é necessário interromper a execução do comportamento do respectivo agente a cada mensagem recebida, já que o quadro-negro contém todas as informações que o módulo de comunicação possa necessitar, como estado dos recursos. Esse modelo também permite que o módulo de comunicação seja definido e implementado da mesma maneira em todos os agentes.

A interação entre esses dois módulos, comunicador e planejador/executor. é feita através do quadro negro e de mensagens internas em um agente.

Uma característica essencial relacionada aos agentes em um ambiente multi-agentes é a sua capacidade de comunicação. Isso torna a definição da linguagem de comunicação a ser usada um tópico fundamental, que será feita na próxima seção.

4.4 Linguagem de Comunicação dos Agentes Autônomos

Com base na linguagem de comunicação descrita em [BOISSIER e DEMAZEAU, 1994], foi definida a seguinte linguagem de interação entre os agentes autônomos do sistema:

<*interação*> ::= <*natureza*> <*tipo*> <*conteúdo*>

A natureza de uma mensagem pode ser decisão, controle ou informação.

As mensagens de decisão são utilizadas para a interação entre os Agentes Autônomos, decidindo qual agente terá o controle sobre um recurso do sistema em um determinado momento. As mensagens de decisão são apresentadas na Tabela 4.2.

Tipo	Conteúdo	Descrição
request	<nome do agente requisitante> <recurso>	Requisição de Recurso: usado por um agente para requisitar o controle de um recurso do sistema a outro agente
transfer	<agente destino> <recurso> <estado do recurso>	Transferência de Recurso: usado para transferir o controle de um recurso de um AA para outro (é um broadcast)
free	<nome do agente> <recurso>	Liberação de Recurso: usado para informar que um recurso alocado por um agente está disponível (é um broadcast)

Tabela 4.2 - Tipos de mensagens de natureza *decisória* definidas na linguagem de comunicação para interação entre agentes na sociedade.

As mensagens de controle são utilizadas durante a iniciação, uma modificação ou o término de uma sociedade. Os tipos de mensagens de controle são apresentadas na Tabela 4.3.

As mensagens de natureza de controle são trocadas entre os Agentes Autônomos e agentes específicos, denominados iniciadores: um ou mais agentes responsáveis por criar, modificar ou destruir a sociedade, pela adição ou remoção de agentes e pelo controle dos recursos nas fases inicial e final da sociedade.

Esses agentes possuem conhecimentos sobre os recursos e a estrutura de autoridade, que é passada a cada novo agente assim que são iniciados. Além de iniciar os agentes da sociedade, os iniciadores não possuem nenhuma diferença dos outros agentes autônomos, exibindo as mesmas características e podendo inclusive apresentar um comportamento específico.

Os tipos de mensagens dessa natureza são bem simplificados. Merecem comentário apenas os tipos *requestAll* e *freeAll*, que facilitam o início e o término de execução do sistema ao permitir aos iniciadores a requisição de todos os recursos ao

mesmo tempo.

Tipo	Conteúdo	Descrição
addAgent	<nome do novo agente> <nova estrutura de autoridade>	Adição de Agente: usado para adicionar um novo agente à sociedade. (é um broadcast)
deleteAgent	<nome do agente>	Remoção de Agente: usada para retirar um agente da sociedade. (é um broadcast)
acknowledge		Confirmação: usada para que todos os agentes presentes na sociedade confirmem a adição ou remoção de outro agente.
inform	<recurso><livre|em_uso> < nome do novo agente >	Informação: usada para o envio de informações sobre os recursos da sociedade para um novo agente.
requestAll		Requisita todos os recursos da sociedade. (é um broadcast)
freeAll		Libera todos os recursos (é um broadcast)

Tabela 4.3 - Tipos de mensagens de natureza de controle definidas na linguagem de comunicação para interação entre agentes na sociedade.

Um conjunto de mensagens diferente é utilizado para a interação interna entre os módulos de comunicação e planejamento/execução. A natureza dessas interações é de informação e seus tipos estão definidos na Tabela 4.4.

As mensagens sobre recursos são necessárias para que o módulo de planejamento/execução acione o módulo de comunicação para requisitar um recurso, já que é o módulo de execução que decide se necessita de um recurso ou não.

Desses tipos, os que necessitam de uma descrição mais profunda são *enableInterruption* e *disableInterruption*.

Devido à característica assíncrona do sistema (requisições de recursos são assíncronas) e da limitação de velocidade máxima existente em qualquer rede de comunicações, é necessária a definição de Ações Primitivas que não podem ser interrompidas. Um dos motivos desse bloqueio é a possível perda do estado de um recurso entre as mensagens. Deste modo, uma AÇÃO qualquer é subdividida em um conjunto de ações primitivas:

AÇÃO = {Ações Primitivas},

onde a AÇÃO global pode ser interrompida, mas as primitivas não. Assim, é necessária a existência de mensagens para a habilitação e desabilitação da interrupção da ação de um agente, usada internamente pelo agente que possui o recurso, e as mensagens de confirmação do sucesso dessa habilitação ou desabilitação. Vale notar que a granularidade do conjunto de ações primitivas define a granularidade de atuação de um sistema.

Tipo	Conteúdo	Descrição
do módulo de planejamento/execução para o de comunicação		
request	<nome do recurso>	Requisição de Recurso: informa ao módulo de comunicação que o agente necessita de um recurso
free	<nome do recurso>	Liberação de Recurso: informa que um recurso está disponível
information	<nome do recurso> <estado do recurso>	Atualiza o estado de um recurso
enable-Interruption	<nome do recurso>	Aviso para habilitar a interrupção de um agente que utiliza um recurso. Esta mensagem desbloqueia o recurso, permitindo que este seja transferido para outro agente
disable-Interruption	<nome do recurso>	Aviso para desabilitar a interrupção de um agente que utiliza um recurso. Ele bloqueia o recurso, não permitindo transferências
do módulo de comunicação para o de planejamento/execução		
received	<nome do recurso> <estado do recurso>	Confirmação do recebimento de um recurso
lost	<nome do recurso>	Perda de um recurso
halt	-	Aviso para término de execução
acknowledge	-	Confirmação de habilitação/desabilitação
not-Acknowledge	-	Não confirma a habilitação/desabilitação: usado quando o agente perdeu um recurso que o executor deseja utilizar.

Tabela 4.4 - Tipos de mensagens de natureza informação trocadas entre os módulos de comunicação e planejamento/execução de um Agente Autônomo.

Exemplificando a interação entre os módulos, temos: caso necessite de um recurso, o módulo de planejamento/execução o pede para seu módulo de comunicação. Esse primeiro decide se pode requisitar o recurso, baseado nas regras da sociedade e na estrutura de autoridade. Se puder, requisita o recurso ao agente que possui o seu controle. O agente que tem o recurso recebe o pedido em seu módulo de comunicação, que verifica se pode interromper seu trabalho, e cede o recurso caso positivo. Vale notar que o módulo de planejamento/execução de um agente é que decide se a a utilização do recurso pode ser interrompida, apenas avisando o módulo de comunicação quando isto é possível. Quando o agente requisitante conseguir o recurso, avisa o seu módulo de planejamento/execução, que então passa a utilizá-lo.

A cada utilização do recurso, o módulo de planejamento/execução desabilita a interrupção ao início da ação primitiva e a habilita ao final. Caso o recurso tenha sido requisitado (por um agente de maior autoridade) durante esse tempo, ao ter a interrupção habilitada o módulo de comunicação já transfere o recurso para o agente requisitante e avisa o planejador/executor que o recurso foi perdido.

Ao requisitar a desabilitação da interrupção, o executor aguarda uma confirmação por parte do módulo de comunicação: se ela for positiva (ACK), significa que o agente não perdeu o recurso; se for uma confirmação negativa (NACK), o módulo de planejamento/execução fica sabendo que o recurso foi perdido e toma as providências necessárias para continuar seu trabalho.

Outro elemento importante da arquitetura proposta é a existência de Agentes Primitivos, explicada a seguir.

4.5 Agentes Primitivos

Os Agentes Primitivos (APs) são módulos, encapsulados em agentes, que executam as tarefas das quais um AA depende, como sensoreamento e ação.

Por exemplo, um AA recebe informações sensoriais de uma câmera ativando um Agente Primitivo respectivo que tem o controle do processo de aquisição de imagens. Assim, um AA requisita uma imagem para um AP e esse a captura e envia para o AA.

Para enviar um comando a um manipulador, o AA apenas envia uma instrução para o AP responsável pelo controle direto do manipulador, que, se possível, executará a

instrução e retornará uma mensagem de confirmação.

Além de sensoreamento e ação, agentes primitivos podem ser utilizados para encapsular algoritmos muito utilizados. No caso de um sistema de Visão Computacional, agentes filtros de imagens são um exemplo de APs muito utilizados.

Os Agentes Primitivos são definidos na implementação de cada sistema e, se implementados segundo uma metodologia bem estruturada, possibilitam a reutilização de código fonte para tarefas semelhantes.

São apresentados a seguir alguns trabalhos correlatos significativos, sendo realizada uma comparação entre a arquitetura proposta e outras arquiteturas e abordagens conhecidas.

5. TRABALHOS CORRELATOS

São discutidos neste capítulo os trabalhos que serviram de base e inspiração para o aqui proposto, dos quais muitas idéias e críticas tiveram origem. Entre os trabalhos que influenciaram esta proposta, dois tiveram maior peso: o de BROOKS (1986, 1991) e o de BOISSIER e DEMAZEAU (1992, 1994).

Este capítulo é iniciado com o trabalho mais controverso e conhecido, o de Rodney Brooks, que tem influenciado muitas áreas de IA, como a de Planejamento Inteligente, onde o paradigma reativo surgiu decorrente destas idéias. Ainda, como já afirmado ao ser apresentada a área de Visão Computacional, o paradigma propositado para VC surge como uma conseqüência do seu trabalho [ALOIMONOS, 1994]. Assim, são apresentadas resumidamente as idéias mais importantes do trabalho de Brooks.

BROOKS (1986, 1991) tem sido um entusiasta de uma abordagem baseada em comportamentos em AI. Ele propôs a chamada *Subsumption Architecture* [BROOKS, 1986], onde um sistema é construído pela composição de camadas dedicadas a tarefas específicas, com um comportamento específico, e onde cada camada interage diretamente com o mundo através de percepção e ação. Nesta arquitetura, “camadas de um nível podem assumir o papel de uma camada em um nível mais baixo suprimindo as saídas desta”, e ainda, “camadas em níveis inferiores continuam a funcionar enquanto adiciona-se camadas de níveis mais alto” [BROOKS, 1986, p.152]. Ainda, cada camada é implementada em uma Máquina de Estado Finita (FSM), sendo decomposta em unidades de processamento de informação paralelas e independentes, interligadas por hardware, e que trocam mensagens entre si.

Deve-se enfatizar que a principal característica da *Subsumption Architecture* é permitir que todas as camadas percebam e atuem diretamente no mundo, um método diferente do que se fazia até então, onde só uma camada na base do sistema percebia e atuava, e a decomposição era feita em níveis de processamento, como percepção, navegação e planejamento.

O trabalho de BROOKS (1991) possui muitos outros aspectos importantes, desejados para os sistemas criados com a arquitetura aqui proposta. Entre elas, BROOKS acredita que é necessário que os sistemas:

1. “Dominem apropriadamente e de maneira rápida as mudanças em seu ambiente

dinâmico, sejam robustos em relação ao seu ambiente e tenham um propósito em sua existência.

2. Sejam construídos para o mundo real e nele testados.
3. Sejam construídos incrementalmente: os módulos devem ser adicionados ao sistema um de cada vez, e o sistema deve ser testado exaustivamente antes da adição de um novo módulo". [BROOKS, 1991, p.140 e p.145]

Entre as semelhanças da arquitetura apresentada aqui e a de Brooks, a primeira é notada na relação dos Agentes Autônomos com as camadas da Subsumption Architecture de Brooks, pois ambos os modelos percebem e agem diretamente no mundo e podem suprimir as ações uns dos outros. A Figura 5.1 ilustra esta relação.

Vale notar que a Estrutura de Autoridade e as Regras de Comportamento relacionam a arquitetura proposta à interligação das camadas de Brooks, permitindo que os agentes autônomos inibam e suprimam uns aos outros, retirando os recursos que necessitam uns dos outros.

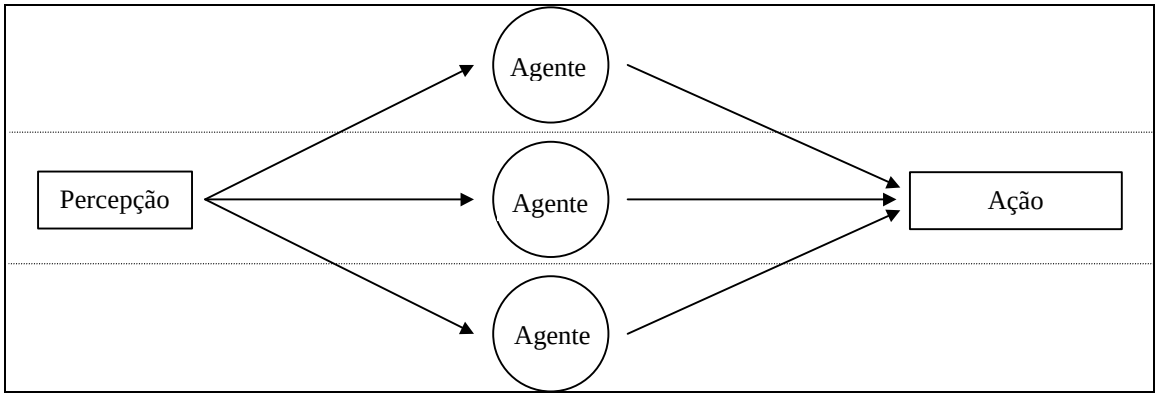


Figura 5.1 - Agentes autônomos relacionados às camadas da arquitetura de Brooks.

A Tabela 5.1 mostra a relação dos níveis comportamentais da arquitetura apresentada, o modelo Multi-Agentes usado para implementar cada nível e a sua similaridade com a arquitetura de Brooks.

A decomposição de agentes em seus respectivos comportamentos é similar à decomposição em tarefas (*task decomposition*) proposta por BROOKS (1991), onde um sistema é construído pela composição de camadas dedicadas a tarefas específicas, com um comportamento específico.

Nível Comportamental	**Modelo Multi-Agentes**	**A *Subsumption Architecture* de Brooks**
Propósitos Globais	Uma sociedade de agentes autônomos.	O propósito do sistema.
Comportamentos	Um agente autônomo para cada comportamento.	As camadas da arquitetura.
Tarefas	Módulos que constituem as habilidades dos agentes autônomos.	A Máquina de Estado Finito (FSM) que compõe cada camada.

Tabela 5.1 - Comparação entre os níveis comportamentais, o modelo MAS e a *Subsumption Architecture*.

Mas existem alguns problemas com a proposta de Brooks que se deseja evitar e modificar. Algumas diferenças importantes entre a proposta do presente trabalho e a dele são:

1. Enquanto as capacidades decisórias do sistema na *Subsumption Architecture* estão escondidas nas FSM, deseja-se que, nesta arquitetura, elas sejam explícitas e distribuídas entre os módulos, para saber exatamente onde uma decisão é tomada.
2. Deseja-se possibilitar que o sistema tenha algum conhecimento simbólico (qualitativo) explícito em cada agente/camada, enquanto a arquitetura de Subsumption não usa representações simbólicas.
3. Deseja-se que, na arquitetura proposta, a interface entre as camadas seja mais bem definida, e por isso cada camada é modelada como um agente.
4. Deseja-se que um agente autônomo possua dentro de si todos os agentes primitivos necessários aos seu funcionamento, assim, pode-se ter a certeza que a sua eliminação não estará afetando outros agentes autônomos.

A proposta de Brooks foi importante como uma reação às arquiteturas robóticas que estavam sendo desenvolvidas até então, que tinham a tendência de se tornar cada vez mais complexas. ELFES (1986) é um exemplo dos sistemas que estavam sendo construídos, onde muito tempo era gasto com o seu próprio gerenciamento. Como reação a esta tendência, um sistema simples e funcional como o de Brooks foi bem recebido.

O segundo conjunto de trabalhos que influencia esta proposta é o apresentado por BOISSIER e DEMAZEAU (1992) e (1994).

Inicialmente, BOISSIER e DEMAZEAU (1992) propõem uma análise de alguns sistemas de visão computacional baseada em DAI-MAS. Nesta, cada sistema é dividido em agentes focais (focus-agents), agentes de nível (level-agents) e agentes básicos (basic-agents). Os agentes focais surgem com uma divisão vertical do sistema em suas tarefas, e os agentes de nível surgem com a divisão horizontal do sistema, em seus níveis de representação. Os agentes básicos são o resultado da interseção dos agentes de foco e de nível, sendo os blocos básicos do sistema. Pode-se relacionar estes agentes aos paradigmas de Visão Computacional notando que os agentes de nível seguem uma abordagem reconstrutiva e os agentes focais são propositados.

Depois deste trabalho de análise, eles apresentaram em 1994 a ASIC (*Architecture for Social and Individual Control*): “uma arquitetura de controle baseada na abordagem Multi-Agentes, que foi usada para construir o sistema MAVI, que integra diferentes módulos visuais em cooperação” [BOISSIER; DEMAZEAU, 1994, p.107]. O modelo proposto por eles para os agentes serviu como base para a modelagem feita neste trabalho, principalmente na parte de comunicação entre agentes.

A principal diferença entre a arquitetura proposta e a ASIC é que esta tem uma abordagem tradicional e o sistema MAVI pode ser considerado como um sistema de visão ativa, enquanto a arquitetura deste trabalho é propositada e comportamental, isto é, se preocupa com integração do sistema de visão com outros módulos de IA, como planejamento de atividades. O sistema proposto nesta dissertação se preocupa com o controle de um sistema baseado em visão que atua no mundo, enquanto o objetivo de BOISSIER e DEMAZEAU é “implementar um sistema de visão composto por módulos independentes que mantenha uma descrição de uma cena em um certo nível de representação para um determinado foco” [BOISSIER; DEMAZEAU, 1994, p.108].

A descrição do funcionamento do sistema MAVI mostra como ele tem uma abordagem reconstrutiva: “o primeiro agente alimenta o sistema com uma seqüência de imagens a serem interpretadas e controla a câmera; o segundo reconstrói as características bidimensionais (bordas de constrastes e agrupamento perceptual) a partir das imagens e um terceiro agente constrói uma descrição simbólica da cena a partir das características bidimensionais” [BOISSIER; DEMAZEAU, 1994, p.113]. Como

exemplo das primeiras aplicações do sistema MAVI em BOISSIER e DEMAZEAU (1994b) é mostrado como o sistema pode ser utilizado para a realização de detecção de contornos em imagens.

Finalmente, deve-se notar que os Agentes Autônomos propostos neste trabalho podem ser classificados como os agentes focais descritos em BOISSIER e DEMAZEAU (1992), pois cada agente possui um comportamento determinado.

Um outro trabalho importante é o de ELFES (1986), que concorda com BROOKS no sentido em que acredita ser “essencial investigar como os subsistemas são integrados em uma arquitetura global” [ELFES, 1986, p. 135].

ELFES (1986) propõe uma arquitetura de controle onde “processos independentes comunicam através de mensagens em um *blackboard* e guardam informações relevantes no mesmo” [ELFES, 1986, p.144]. O sistema é dividido em níveis de processamento (controle robótico, interpretação sensorial, integração sensorial, ...) e utiliza representações para a modelagem do mundo real. Pode-se notar ainda uma característica da abordagem tradicional em seu sistema de teste, chamado Dolphin, que tem como propósito estudar a extração de características do mundo com sonares e não apresenta um comportamento ou realiza uma tarefa específica.

Nos últimos dois anos surgiram vários trabalhos que utilizam a abordagem baseada em agentes tanto para controle de sistemas robóticos como para sistemas de visão computacional. Entre os trabalhos que utilizam agentes para o controle de sistemas robóticos que influenciaram esta dissertação estão os abaixo.

NEVES e OLIVEIRA (1997) propõem uma arquitetura multi-agentes para o controle de um robô móvel autônomo. Neste trabalho, o controle de um agente possui três níveis distintos, o cognitivo, o reativo e o reflexivo, onde em cada nível existe uma comunidade de agentes. Assim, os agentes do nível cognitivo orientam os agentes do nível reativo, para que o sistema apresente uma “reação orientada”, enquanto os agentes no nível reflexivo são responsáveis pelos comportamentos mais elementares. Uma característica muito forte neste trabalho é o estudo da utilização de aprendizado para a inserção de novos comportamentos no sistema.

GARCIA-ALEGRE e RECIO (1997) apresentam as diretrizes utilizadas na implementação de Agentes de Comportamento Básicos para o domínio de navegação

visualmente guiada. Neste sistema existe uma separação entre o sistema de visão e o sistema de locomoção, e os vários agentes necessários para cada sistema são implementados separadamente. Por exemplo, no sistema de locomoção existem agentes básicos como “Pare”, “Ande” e “Volte”, enquanto no sistema de visão existem agentes básicos chamados de “Saccadic”, “Contorno” e “Centralize”. Estes dois sistemas interagem através de um agente que coordena a utilização dos agentes básicos para que o sistema apresente um comportamento.

Tanto MATRIC (1998) quanto VELOSO et al. (1998) utilizam DAI-MAS para modelar a coordenação em sistemas compostos por múltiplos robôs. O primeiro trabalho se preocupa com o aprendizado de comportamentos nesta classe de sistemas, enquanto o segundo estuda a percepção distribuída e a colaboração de agentes autônomos múltiplos em um ambiente incerto e dinâmico.

Finalmente, FIRBY et al (1995) testaram a abordagem de visão propositada em um robô móvel que realiza tarefas no ambiente de um escritório real, e em particular a tarefa de coleta de lixo, e analisam a adaptação deste tipo de sistema a novas situações. A arquitetura proposta integra um planejador reativo e um sistema de controle baseado em capacidades inteligentes que utiliza rotinas visuais para perceber o ambiente. Este trabalho trata um robô como um agente, e conclui que a abordagem é viável para tal tarefa e que este tipo de sistema leva vantagem ao integrar ação e percepção.

Os trabalhos que aplicam a teoria de multi-agentes para sistemas de visão, além dos propostos por BOISSIER e DEMAZEAU já apresentados, incluem:

ARCELLI et al (1998) realizam uma discussão preliminar sobre a utilização de agentes de software para visão computacional e apresentam uma arquitetura heterogênea para o uso de agentes em sistemas de VC. A arquitetura por eles proposta é composta por três camadas de agentes adaptativos cooperativos. Os agentes da primeira camada se comportam como um assistente pessoal e atuam como interface do sistema com o usuário; os agentes da segunda e da terceira camada são chamados agentes de processamento e gerenciam as tarefas visuais.

CHENG et al (1995) descreve uma abordagem para a construção de sistemas de visão computacional para robôs utilizando uma arquitetura baseada em agentes. Eles apresentam um sistema onde os agentes são executados em paralelo e cooperam para

realizar a segmentação e rastreamento 3D baseado em atributos de objetos em movimento a partir de imagens esteroscópicas.

Finalmente, SPINU et al. (1996) implementaram um sistema multi-agentes para a detecção de contornos tratada como um problema de otimização distribuído; e BOUCHER et al. (1996) descrevem um sistema para a segmentação de objetos deformáveis em movimento, utilizado para a segmentação de uma seqüência de imagens de células.

Outros trabalhos que foram importantes para a realização do trabalho proposto e que devem ser citados são :

Os trabalho de SHNEEBELI (1992) e sua continuação em XAVIER (1996) apresentam uma abstração para controladores baseados em agentes onde estes são distribuídos em 3 categorias: Agentes Sensores, Agentes de Comportamento e Agentes Atuadores. Assim, um sistema pode ser construído utilizando classes de agentes sensores e atuadores. Na seqüência deste trabalho, FREIRE et al (1997) utiliza esta arquitetura para construir um sistema sensorial ultra-sônico para um robô móvel.

Como exemplo de utilização da *Subsumption Architecture* objetivando o controle de um sistema de Visão Computacional, PINHANEZ (1994) propõe o seu uso em um sistema com um foco de atenção móvel objetivando a detecção de áreas de alta concentração de bordas em uma cena. O artigo descreve atentamente o uso de métodos baseados em comportamentos para o projeto e a implementação da estrutura de controle de um sensor parecido com a fóvea, que apresenta resultados interessantes.

RIVLIN et al. (1991) estudam o problema do reconhecimento de objetos quando este é considerado no contexto de um agente operando em um ambiente. Eles tratam principalmente da tradução das intenções do sistema em um conjunto de comportamentos, propondo uma metodologia para a construção de sistemas com intenções, comportamentos e tarefas em um ambiente determinado.

FERGUSON (1992) propõe uma arquitetura para o controle de agentes móveis autônomos, na qual um sistema é dividido com base nos seus níveis de representação em três camadas, uma reativa, uma de planejamento e uma de modelagem.

E finalmente, NOREILS e CHATILA (1995) também introduzem uma arquitetura de controle para robôs móveis, e ainda fazem um bom levantamento dos trabalhos relacionados a arquiteturas de controle de robôs.

6. A IMPLEMENTAÇÃO DA ARQUITETURA

Este capítulo descreve a implementação da arquitetura proposta em um sistema de Visão Computacional Propositada. A primeira seção descreve a implementação da estrutura básica da arquitetura, que é composta por classes que definem a comunicação, as regras da sociedade e a estrutura de autoridade e independe de hardware específico, como câmeras e manipuladores. A segunda seção descreve o sistema para a Célula de Montagem da Escola Politécnica da USP implementado.

6.1 Implementação de uma Estrutura para a Arquitetura Proposta

A Estrutura Básica da Arquitetura foi desenvolvida segundo uma metodologia de projeto orientado a objetos (PRESSMAN, 1995) e consiste em uma biblioteca de classes que serve de base para a implementação de novos sistemas. Na construção desta biblioteca foi utilizada a linguagem C++ (compilador GNU C/C++) em um ambiente UNIX, que foi testada com sucesso em várias plataformas (SunOS, Solaris, Irix e Linux).

Esta biblioteca implementa as estruturas básicas necessárias à arquitetura, como os recursos, as regras da sociedade e estrutura de autoridade, os agentes primitivos, autônomos (seus módulos) e iniciadores e a comunicação entre os agentes. A Tabela 6.1 apresenta as classes da biblioteca. Destas, as mais importantes são as que definem os agentes:

- A classe *primitiveAgent* é utilizada por todos os módulos e agentes do sistema, pois implementa a comunicação, com a criação de um sistema de *white pages*, que permite que um agente comunique com outro através de um nome pré-definido. Isto é importante pois possibilita que os agentes se tratem em um nível declarativo, escondendo detalhes de hardware como em qual máquina estão os agentes e quais suas identificações (*task id*). Esta classe é usada para criar os Agentes Primitivos e é classe base para os agentes autônomos.

Classe	Descrição
agentInHierarchy	Esta classe define atributos para um agente autônomo necessários para definir a estrutura de autoridade da sociedade.
authorityStructure usa: *agentInHierarchy*	Esta classe define a estrutura de autoridade da sociedade de agentes autônomos.
resource	Esta classe define os atributos dos recursos do sistema.
resourceInExecution	Esta classe define os atributos dos recursos do sistema que são necessários apenas aos módulos de execução.
resourcesInblackboard	Esta classe define os atributos dos recursos do sistema que são guardados no *blackboard*, necessários aos módulos de comunicação e planejamento/execução.
taskId	Esta classe é usada para relacionar o nome de um agente com seu número de identificação.
primitiveAgent usa: *taskId*	Esta classe define os atributos e métodos necessários para todos os agentes do sistema. Ela é usada diretamente pelos agentes primitivos e as outras classes de agentes são suas herdeiras. Ela define os métodos necessários para a comunicação entre qualquer modulo do sistema, encapsulando as funções de comunicação e criando o sistema de *white pages*.
autonomousAgent-ExecutionModule usa: *resourceInExecution* herda de: *primitiveAgents*	Esta classe define os atributos e métodos para o modulo de planejamento/execução de um agente autônomo.
autonomousAgent-BlackboardModule usa: *resourcesInBlackboard* herda de: *primitiveAgent*	Esta classe implementa o *blackboard* de um agente autônomo.
commonAgent usa: *authorityStructure, resource* herda de: *primitiveAgent*	Esta classe define argumentos e métodos comuns ao modulo de comunicação dos agentes autônomos e do agentes autônomo que pode iniciar o sistema. Nesta classe são implementadas as regras da sociedade e a estrutura de autoridade.
autonomousAgent-CommunicationModule herda de: *commonAgent*	Esta classe define o módulo de comunicação dos agentes autônomos.
initializingAgent-CommunicationModule herda de : *commonAgent*	Esta classe define o módulo de comunicação dos agentes autônomos que podem iniciar a sociedade. Existem poucas diferenças entre esta classe e a acima

Tabela 6.1 - Classes implementadas para a arquitetura.

- Como descrito no capítulo 4, um agente autônomo é constituído por vários módulos. Um AA é implementado a partir de 3 classes: *autonomousAgentCommunicationModule*, *AutonomousAgentExecutionModule* e *autonomousAgentBlackboadModule*. Estas classes implementam os módulos de comunicação, planejamento/execução e um módulo que gerencia o *blackboard* utilizado pelos dois. Em um sistema, os três módulos são executados em paralelo e comunicam entre si.
- As classes *autonomousAgentCommunicationModule* e *initializingAgent-CommunicationModule* são utilizadas para definir os módulos de comunicação dos Agentes Autônomos e dos Agentes Autônomos Iniciadores (os que podem inserir e retirar agentes no sistema). A diferença entre as duas é pequena, por isso ambas tem a classe *commonAgent* como base. Estas classes implementam ainda as regras da sociedade e a estrutura de autoridade.

A Figura 6.1 apresenta um diagrama de classes resumido para a biblioteca implementada.

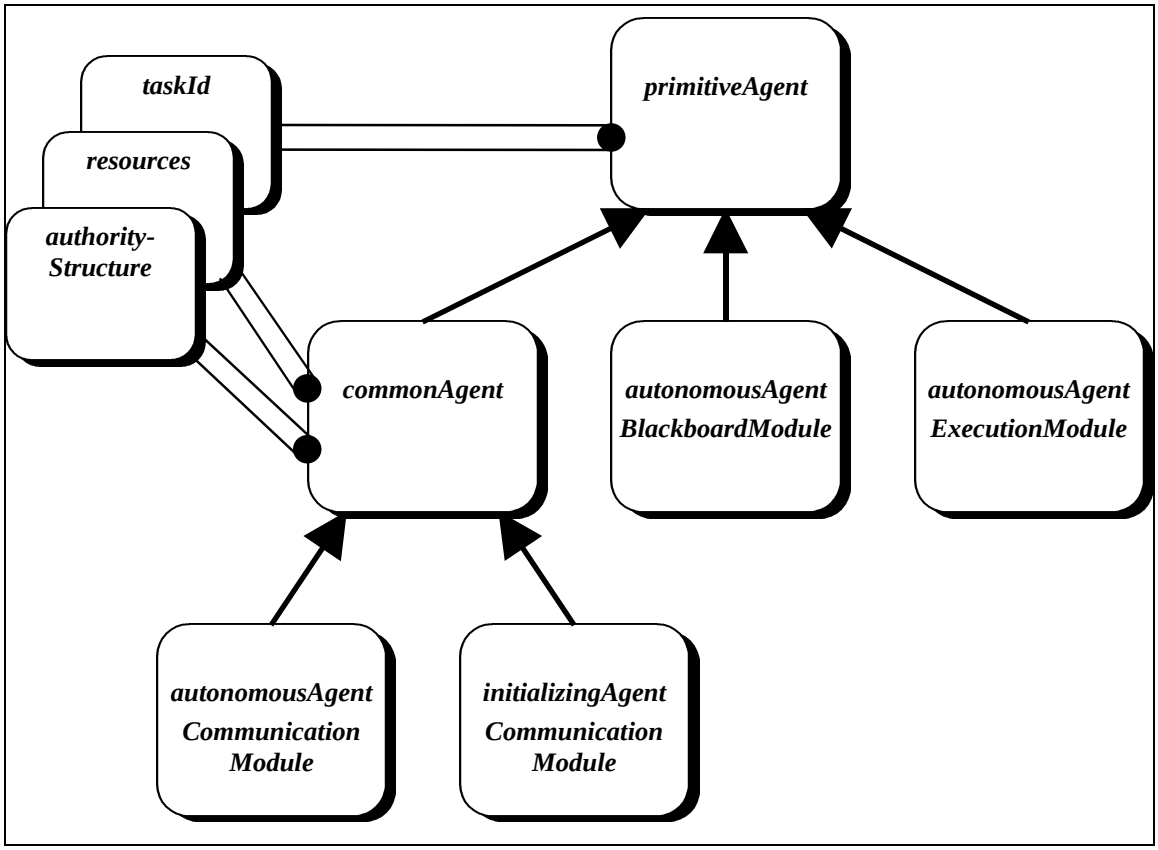


Figura 6.1 - Diagrama de Classes resumido da biblioteca implementada.

Nesta figura, as setas inicam que uma classe herda atributos de outra (por exemplo, *commonAgent* herda de *primitiveAgent*), enquanto as linhas duplas terminadas em círculo indicam que uma classe é instanciada dentro da outra (i. e., *commonAgent* utiliza a classe *resource* para definir seus atributos). Ainda, as regras da sociedade são definidas dentro da classe *commonAgent*.

Outras classes implementadas de importância são:

- *AgentsInHierarchy* e *authorityStructure* definem os objetos da estrutura de autoridade. São utilizadas pela classe do módulo de comunicação dos agente autônomos que, a partir das regras definidas para um sistema, decidem se um recurso pode ser requisitado ou cedido para um outro agente.
- A definição dos recursos do sistema é realizada em 3 classes: *resources*, *resourcesInBlackboard* e *resourcesInExecution*. Esta divisão é feita para diminuir a comunicação entre os módulos do agente autônomo, mantendo no quadro negro apenas os dados que ambos os módulos necessitam e as outras informações separadas em cada módulo.

A comunicação entre os módulos dos agentes definida na classe *primitiveAgents* é básica a todo o trabalho, já que define os sistema pelo qual os agentes poderão trocar mensagens. Assim, sua implementação é descrita em detalhes na próxima seção.

6.2 Implementação da Comunicação utilizando a Biblioteca PVM

O protocolo de comunicação básico entre os agentes foi implementado com base no sistema PVM.

O PVM (Parallel Virtual Machine) é uma plataforma de software que simplifica a programação de sistemas distribuídos em um conjunto heterogêneo de computadores interligados em uma rede. Seu desenvolvimento foi iniciado em 1989 no Laboratório Oak Ridge, nos EUA, por um grupo de pesquisa interessado em computação distribuída em ambientes heterogêneos.

O conceito central do PVM é a noção da Máquina Virtual, um conjunto de computadores hospedeiros heterogêneos conectados por uma rede de comunicação que são “encapsulados” e assim aparecem para o usuário como um único computador paralelo. O PVM possui suporte para quase todos os tipos de arquiteturas de hardware

existente. Por isso, as máquinas individuais podem ser desde supercomputadores vetoriais, estações gráficas multiprocessadas, servidores locais, workstations, até microcomputadores IBP-PC 386 com sistema Windows 95. Quanto a rede que as conecta, a única exigência é o suporte ao padrão TCP-IP. O Exemplo 6.1 mostra a configuração de uma Máquina Virtual do PVM.

A grande vantagem do PVM é a portabilidade e a interoperabilidade permitida. Portabilidade significa que um programa escrito para um sistema com uma arquitetura definida pode ser copiado para uma segunda arquitetura, compilado e executado sem modificações. Interoperabilidade expande esta propriedade, permitindo que os programas escritos e compilados em diferentes arquiteturas possam se comunicar. Assim a Máquina Virtual permite que computadores de arquiteturas diferentes sejam vistos como nós de processamento de um único computador. Além disso, o PVM permite a criação de aplicações tolerantes a falhas, que podem sobreviver a queda de computadores na Máquina Virtual.

```
pvm> conf
5 hosts, 2 data formats
                     HOST     DTID     ARCH   SPEED
                  nausika    40000 SUN4SOL2    1000
                   nikita    80000 SUN4SOL2    1000
                     enya    c0000 SUN4SOL2    1000
                     teco   100000    LINUX    1000
                    hades   140000     SUN4    1000
pvm>
```

Exemplo 6.1 - Configuração de uma Máquina Virtual do PVM.

A plataforma PVM fornece uma biblioteca de funções em C ou FORTRAN para o programador acessar os recursos da plataforma, permitindo a realização de atividades de baixo nível como iniciação de processos, transmissão e recepção de mensagens, sincronização, etc. Segundo GEIST et al. (1996), o “PVM é o padrão mais aceito para a computação distribuída e está sendo vista como o futuro padrão para troca de mensagens.

A escolha do sistema PVM como base para a comunicação entre os agentes foi baseada no fato de permitir uma implementação simples, sem que o autor necessitasse programar utilizando *sockets* do protocolo TCP/IP. Ainda, o suporte ao sistema existente, que conta inclusive com *newsgroups* na Internet, foi decisivo e permitiu a solução de dúvidas rapidamente.

Outras características da comunicação:

- O PVM possibilita a comunicação entre processos, que são criados com números identificadores (*task ids*) que mudam a cada execução: foi necessário adicionar uma camada de *white pages* ao sistema, para permitir o envio de mensagens entre agentes a partir do nome de identificação destes agentes.
- Foi implementado dois tipos de mensagens: as ponto a ponto (entre 2 agentes específicos) e as de *broadcast*, para todos os agentes.
- Na comunicação não foi implementado um tempo de espera máximo (*time-out*). Por isso, ao requisitar uma informação, um módulo do agente espera infinitamente até ela chegar. Isso acarreta na possibilidade do sistema ficar preso em um estado caso ocorra uma perda de uma mensagem.
- Todas as possibilidades de ocorrência de *deadlocks* foram previstas, sendo implementados sistemas de confirmação com *acknowledge* e *notAcknowledge* entre os agentes e seus módulos.

Antes da implementação da linguagem de comunicação entre os agentes, foi realizada uma descrição de cada elemento definido no item 4.4. Para isso foram especificados os seguintes itens:

- Descrição: uma descrição informal da mensagem.
- Condição inicial: especifica quando uma mensagem deve ser enviada.
- Destinatário: qual ou quais agentes devem receber a mensagem.
- Sintaxe: a sintaxe da mensagem, com o formato em que os dados serão enviados, códigos e sinais;
- Algoritmos de envio e recepção: descrevem o processo do envio e da recepção da mensagem, com os diagramas de estado.

Depois desta definição ter sido realizada e os métodos de comunicação básicos definidos e codificados, foi implementado um método para o envio e o recebimento de cada tipo de mensagem definido pela linguagem de comunicação dos agentes.

Para a validação experimental desta estrutura básica da arquitetura proposta foram realizadas diversas simulações. Estas consistiram na criação de sociedades com um número variado de agentes e recursos, sendo utilizados vários sistemas operacionais

(SunOS, Solaris, Linux, Irix e WinNT) e arquiteturas (Sun SparcStation 1, 5, 10 e 20, Silicon Graphics multiprocessados, IBM-PC 486), em máquinas virtuais construídas com computadores de um ou vários laboratórios diferentes. Alguns dos resultados obtidos são apresentados no próximo capítulo.

Depois da validação experimental, foi desenvolvida uma aplicação para demonstrar o funcionamento da arquitetura em um sistema robótico real. Esta aplicação é descrita em detalhes na próxima seção.

6.3 A Implementação em um Domínio de Montagem

Como exemplo de aplicação da arquitetura proposta na solução de problemas inseridos no mundo real, foi implementado um sistema em um domínio de montagem, cujo objetivo é realizar tarefas de montagens simples e visualmente guiadas.

Esta seção descreve em detalhes o hardware utilizado, as regras e a estrutura de autoridade desta sociedade e os agentes autônomos e primitivos implementados.

6.3.1 Descrição da Célula de Montagem

Como já apresentado na introdução, um sistema de Visão Computacional Propositada baseado na arquitetura apresentada foi implementado na Célula de Montagem da Divisão de Automação e Inteligência Artificial do Laboratório de Sistemas Integráveis da Escola Politécnica da USP (RILLO et al., 1992). A Figura 6.2 mostra um dos seus manipuladores. A Tabela 6.2 descreve o equipamento utilizado neste experimento.

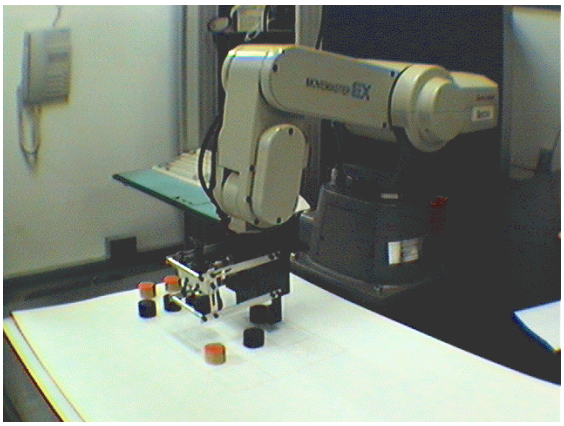


Figura 6.2 - Um dos manipuladores da Célula de Montagem.

Equipamento	Descrição
Duas Estações de Trabalho	Marca Sun, modelo SparcStation 5 de 70 MHz, com 32 Mb de memória RAM,
Uma Estação de Trabalho	Marca Sun, modelo SparcStation 10
Uma Câmera CCD	marca JVC, modelo TK870U. Sinal de Saída: NTSC composto.
Uma Placa de Aquisição de Imagens	Marca Sun modelos SunVideo para SBus de estações SPARCstations e da Biblioteca de Imaging XIL Comporta entrada de sinais NTCS ou PAL não modulados, compostos ou S-Video
Um Manipulador Robótico	Marca Mitsubishi, modelo Movemaster EX RV-M1, articulado verticalmente, com 5 graus de liberdade.
Um Microcomputador	Padrão IBM-PC 486 de 100 MHz, com 32 Mb de memória RAM usando sistema operacional DOS/Windows.

Tabela 6.2 - Equipamento utilizado.

O Sistema de Aquisição de Imagens

O Sistema de aquisição de imagens é composto por uma câmera CCD (Figura 6.3), localizada em uma posição fixa a 1.2 metros acima da área de trabalho do manipulador. A sua montagem é perpendicular à mesa de trabalho, de forma que o plano da imagem seja paralelo ao da mesa. Esta câmera é ligada a uma placa de aquisição de imagens colorida *SunVideo* localizada em uma das estações de trabalho.

SunVideo é um sistema de captura e compressão de vídeo em tempo real, que consiste em uma placa digitalizadora para um barramento *SBus* das estações *Sun SPARCstations* e na biblioteca de tratamento de imagens XIL. Ela é usada principalmente para videoconferência e aplicações multimídia e utiliza várias técnicas de compressão de vídeo, possuindo uma *on-board video compression engine*. Ela captura, digitaliza e comprime sinais NTSC ou PAL, compostos ou S-Video, de vídeo câmeras. No sistema utilizado, o sinal é NTSC composto.

Figura 6.3 - A câmera utilizada.

A placa *SunVideo* captura imagens de até 640 x 480 pixels com 24 bits de cores, colocando à disposição do usuário uma imagem no modelo de cor YUV. Sua velocidade foi testada com o sistema isolado e com carga baixa, sendo que a maior velocidade verificada foi de 0,04 segundos (~25 quadros por segundo) para uma imagem de tamanho igual a 64 x 48 pixeis com 24 bits de cores.

O Manipulador Robótico

O manipulador usado é um Movemaster EX modelo RV-M1, fabricado pela Mitsubishi. Ele é articulado verticalmente possuindo 5 graus de liberdade, com um motor de passo em cada articulação, e uma garra motorizada.

Os manipuladores da célula possuem uma variedade de comandos que permitem controlar os seus movimentos. Estes comandos podem ser classificados quanto à suas funções:

- Posicionamento do manipulador: comandos como *move joint*, *move to position* permitem especificar a posição de cada junta, ou mover a garra de uma certa quantidade de milímetros em uma direção cartesiana.
- Controle da garra: controles para abrir ou fechar a garra e definir a pressão usada na garra.
- Instruções para o controle da porta serial.
- Controle de erro.

Estas instruções são enviadas por um microcomputador através da sua porta serial. Com isso, um programa em um PC controla a atividade de um manipulador.

Ainda existem comandos que permitem a programação de uma atividade residente na unidade de controle, contendo comandos para controle de fluxo de programa, como loops, subrotinas e saltos condicionais.

Finalmente, as instruções são passadas para a unidade de controle através de um *teaching box* ou uma porta de comunicação serial, que foi utilizada no sistema implementado.

6.3.2 Agentes Autônomos

Como foi apresentado no capítulo 4, os Agentes Autônomos são definidos a partir dos propósitos de um sistema. O objetivo global do sistema implementado consiste em: dado uma quantidade de objetos sobre uma mesa, que podem ser peças ou encaixes, o sistema encontre os pares que combinem e os una, sobrepondo as peças sobre os encaixes.

Para a implementação, foram definidos três comportamentos diferentes, cada um correspondendo a um agente autônomo:

1. Agente **Montador** (***Assembler***): ele deve realizar uma montagem pegando peças na área de trabalho com o manipulador e colocando-as nas posições desejadas, sobre os encaixes. Para isso, o agente tem que ser capaz de planejar as atividades envolvidas na montagem. Este planejamento pode ser simples ou complexo, reativo ou temporal, dependendo da complexidade que se deseja do sistema. No estado atual, o agente não possui capacidades de planejamento automático de atividades, sendo o plano, por isso, previamente definido.
2. Agente **Desobstrutor** (***Cleaner***): este agente deve desobstruir a área de trabalho. A área de trabalho é um espaço, previamente definido como importante para a montagem. Assim, objetos não desejados que uma pessoa ou outro manipulador possam ter colocado nesta área devem ser retirados pelo agente.
3. Agente **Evitador de Colisões** (***Collision avoider***): este agente deve evitar colisões do manipulador com objetos que se movam na área de trabalho (outro manipulador, uma mão), com o objetivo, entre outros, de preservar a integridade física do sistema.

Para simplificar a implementação do sistema, visto que o objetivo deste sistema é demonstrar a validade da arquitetura, as peças foram definidos como qualquer objeto cilíndrico de cor preta; os encaixes são anéis de cor azul e qualquer objeto de cor vermelha é considerado lixo, devendo ser removido da área de trabalho. Ainda, o agente Evitador de Colisões verifica apenas o movimento de objetos vermelhos na área de trabalho.

Para controlar a adição e remoção dos agentes na sociedade foi implementado um Agente Autônomo Iniciador. Este agente adiciona os agentes autônomos que o programador definiu como membros da sociedade, informa a cada agente a estrutura da sociedade, o seu estado e os recursos disponíveis. Controla a liberação dos recursos no início e no final da execução da sociedade.

6.3.3 Recursos, Regras de Comportamento e Estrutura de Autoridade

Os recursos foram definidos como sendo parte do sistema que tem seu uso compartilhado pelos agentes e que podem ser controlados por somente um agente em um determinado momento. No sistema implementado o manipulador e a câmera são recursos compartilhados, com diferentes níveis de prioridade.

A estrutura de autoridade define o nível de prioridade que um agente possui para usar um determinado recurso. A estrutura de autoridade da sociedade é o que torna possível a decisão sobre como os recursos do sistema devem ser alocados, decidindo qual agente deve ter o controle de um recurso em um determinado momento.

Como a câmera consegue fornecer imagens sem criar conflito entre os agentes, eles possuem a mesma prioridade. Assim, os agentes autônomos não precisam disputar o controle da câmera e todos podem usá-la ao mesmo tempo, acessando um mesmo agente primitivo que realiza a captura, sempre que necessário.

Já para o controle do manipulador, as contingências do domínio impõem a seguinte estrutura de autoridade: o agente *Evitador de Colisões* é o agente com maior autoridade, o *Desobstrutor* é o segundo e o *Montador* é o agente com menor autoridade. Pode-se notar que a estrutura de autoridade tem por objetivo principal a preservação da integridade física do sistema e, secundária, o objetivo global a ser alcançado, e que os comportamentos reativos tem precedência sobre os deliberativos.

As regras de comportamento definem como a estrutura de autoridade deve ser usada para o controle das comunicações e da alocação dos recursos. A sociedade de agentes proposta possui três regras de comportamento:

- **Regra # 1**: somente um agente pode controlar um recurso em um determinado momento.
- **Regra # 2**: qualquer agente pode requisitar o controle de um recurso a um agente com menor autoridade que ele, em qualquer momento.
- **Regra # 3**: um agente só pode requisitar o controle de um recurso a um agente de maior autoridade caso este esteja cedendo o controle.

O teste desta estrutura de autoridade e regras de comportamento é realizado na mesma simulação usada para tentar validar experimentalmente a arquitetura, já que a simulação exige estas definições. Os resultados são apresentados no próximo capítulo.

6.3.4 A Decomposição de Tarefas dos Agentes Autônomos

A decomposição dos comportamentos em tarefas leva à definição de um conjunto de agentes primitivos para cada agente autônomo, que implementam e executam as tarefas básicas necessárias.

Um dos objetivos que o programador deve ter em mente ao definir um agente primitivo é a reutilização de software que estes agentes possibilitam. Assim, a decomposição dos comportamentos em agentes primitivos deve ser feita de maneira a se criar uma biblioteca de agentes primitivos, que possa ser utilizada por vários agentes autônomos.

No sistema implementado, cada agente autônomo possui um conjunto de agentes primitivos, definidos:

1. Agente **Montador**: Capturador de Imagens, Filtrador de Imagens por Cor, Identificador de Movimentação, Determinador de Contornos, Vetorizador, Selecionador de Pares, Controlador do Manipulador. Pode-se notar que esse agente é o mais complexo, já que é o único não sequencial. Outra nota é que o agente primitivo Identificador de Movimentação é utilizado em um contexto diferente dos outros agentes: aqui ela serve para comprovar se houve modificação na área de trabalho antes de realizar a montagem, já que se alguma

peça se moveu o planejamento da montagem deve ser reiniciado.

2. Agente **Desobstrutor**: Capturador de Imagens, Filtrador de Imagens por Cor, Determinador de Contornos, Vetorizador, Controlador do Manipulador.
3. Agente **Evitador de Colisões:** Capturador de Imagens, Filtrador de Imagens por Cor, Identificador de Movimentação.

Destes agentes primitivos, o Capturador de Imagens e o Controlador do Manipulador são compartilhados por todos os agentes autônomos do sistema, visto que os dois funcionam como interfaces para os recursos do sistema. Para os outros agentes primitivos, atribui-se uma instância para cada agente autônomo. As figuras 6.4, 6.5 e 6.6 apresentam o fluxo de dados entre os agentes primitivos de cada agente autônomo.

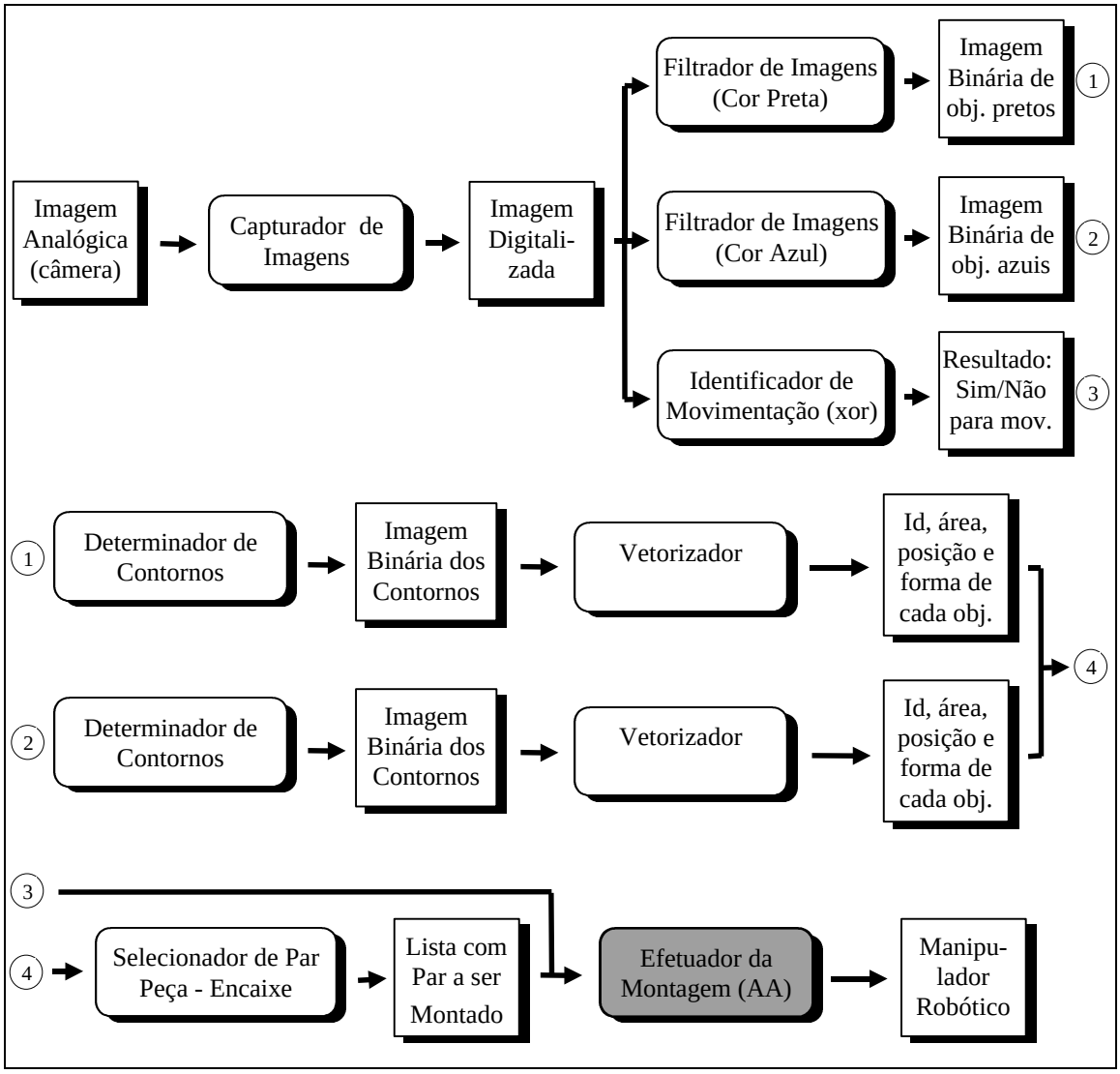


Figura 6.4 - Fluxo de dados do Agente Autônomo Montador.

Nestas figuras, os agentes primitivos são representados pelos retângulos de cantos arredondados e os dados pelos quadrados. Ainda, o retângulo mais escuro representa uma ação que é realizada pelo módulo de planejamento/execução do agente.

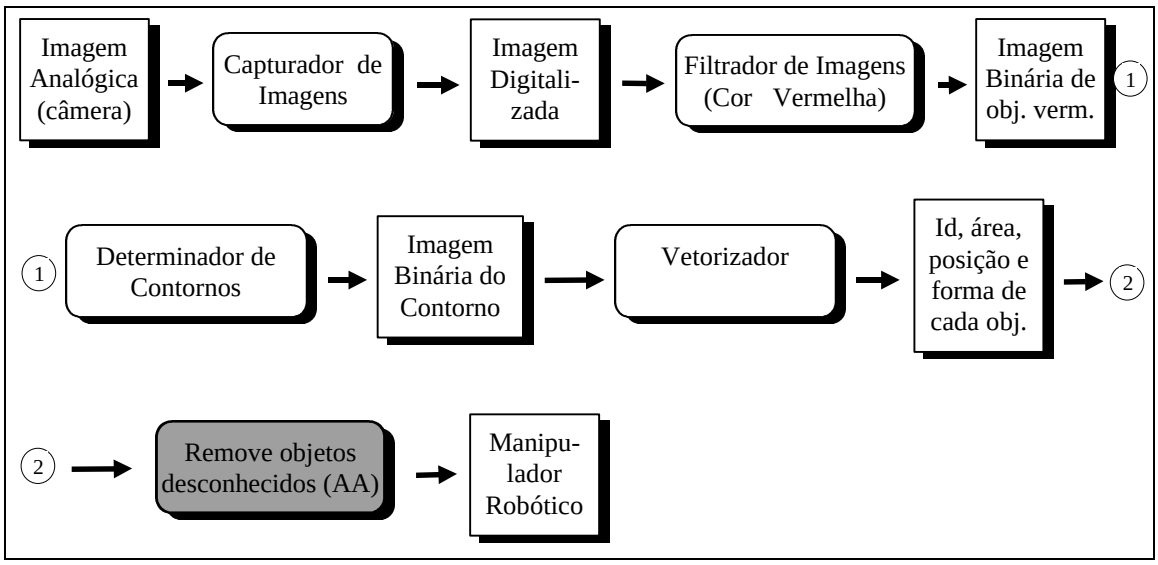


Figura 6.5 - Fluxo de dados do Agente Autônomo Desobstrutor.

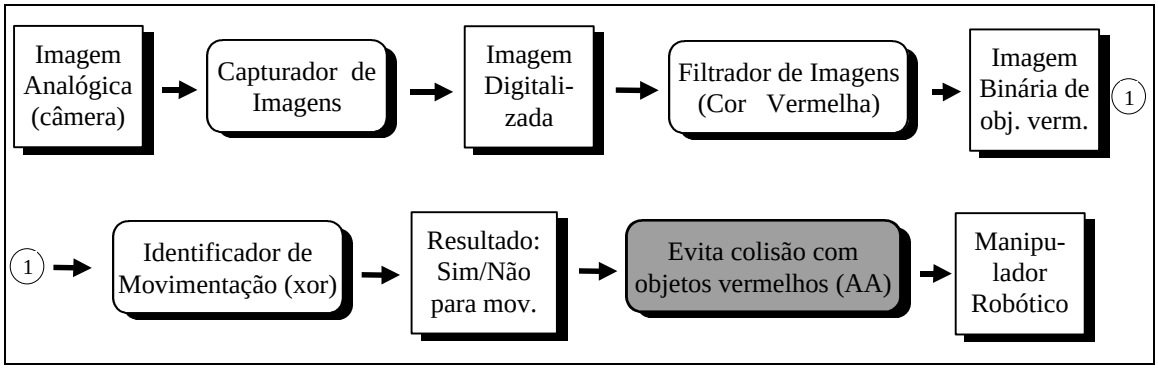


Figura 6.6- Fluxo de dados do Agente Autônomo Evitador de Colisão.

Os agentes primitivos comunicam-se diretamente com o módulo de planejamento/execução do agente autônomo. Assim, para cada agente primitivo foi definida uma linguagem de comunicação própria, que define as mensagens que o agente pode receber e quais as mensagens ele envia.

6.3.5 Agentes Primitivos Implementados

Foram implementados agentes primitivos que realizam diversos tipos de tarefas: captura de imagens; processamento de imagens; e casamento de padrões. Os agentes

utilizados nesta dissertação são descritos abaixo.

6.3.5.1 Capturador de Imagens (Capture)

Este agente captura uma imagem colorida (24 bits), no modelo de cor YUV, de tamanho variável, utilizando uma placa de aquisição de imagens SunVideo.

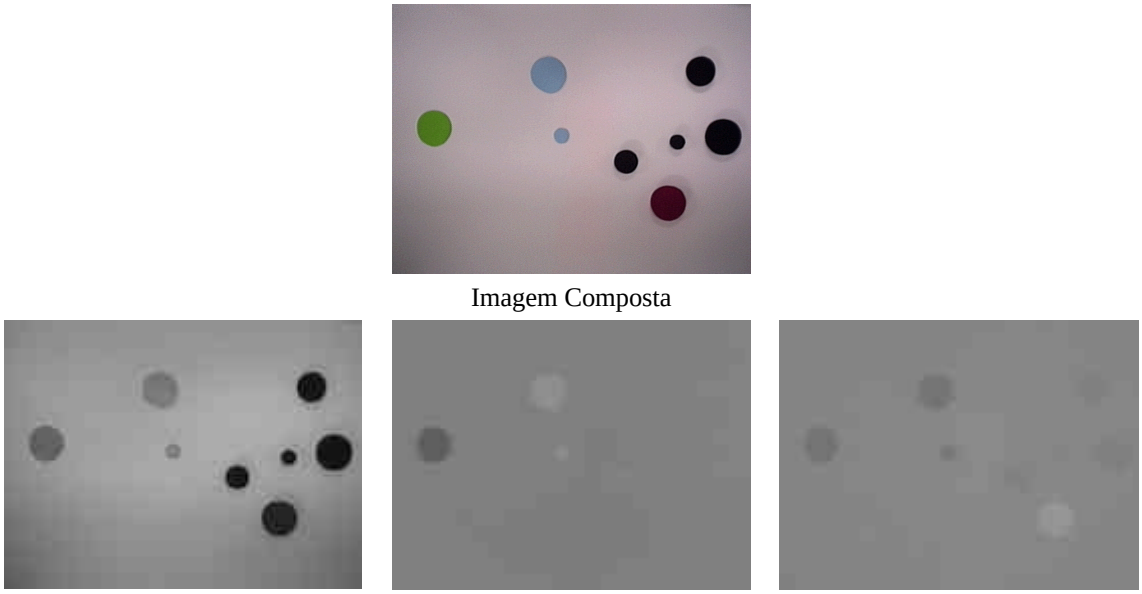


Figura 6.7 - Imagem composta capturada.

O modelo de cores YUV [LI, 1998] foi inicialmente usado no padrão PAL para vídeo analógico e atualmente é usado como padrão para video digital, sendo que o JPEG e o MPEG são baseados em um modelo YUV modificado.

Este modelo define a imagem como sendo uma matriz de pontos, onde a cor de cada ponto é definido por 3 bytes: o primeiro define a luminância da imagem (Y), e os segundo (U) e terceiro (V) definem a crominância.

A luminância Y é definida a partir das cores vermelha, verde e azul, onde:

$$Y = 0.299 * \text{Vermelho} + 0.587 * \text{Verde} + 0.114 * \text{Azul}$$

A crominância é definida como a diferença entre uma cor de referência e um branco de referência para uma mesma luminância. Assim os valores de crominância são definidos para o azul e o vermelho.

$$U = \text{Azul} - Y$$

$$V = \text{Vermelho} - Y$$

Neste modelo de cor uma imagem em níveis de cinza não possui crominância e U e V tornam-se igual a zero. Ainda, V varia do vermelho ao ciano e U do azul ao amarelo.

Este agente recebe mensagens do tipo “(getImage (largura) (altura))”, onde largura e altura definem a dimensão da matriz de imagem, e envia mensagens com a imagem capturada para o agente requisitante. Ele ainda inicia a placa de captura de imagens.

6.3.5.2 Filtrador de Imagens por Cor (colorFilter)

A tarefa primitiva deste agente é filtrar uma imagem colorida, de tamanho variável, construindo uma imagem binária. Ele recebe a imagem colorida e o valor de limiar de uma cor que se deseja filtrar no modelo YUV e atribui 1 às regiões da imagem onde a cor supere o limiar desejado e zero ao resto da imagem.

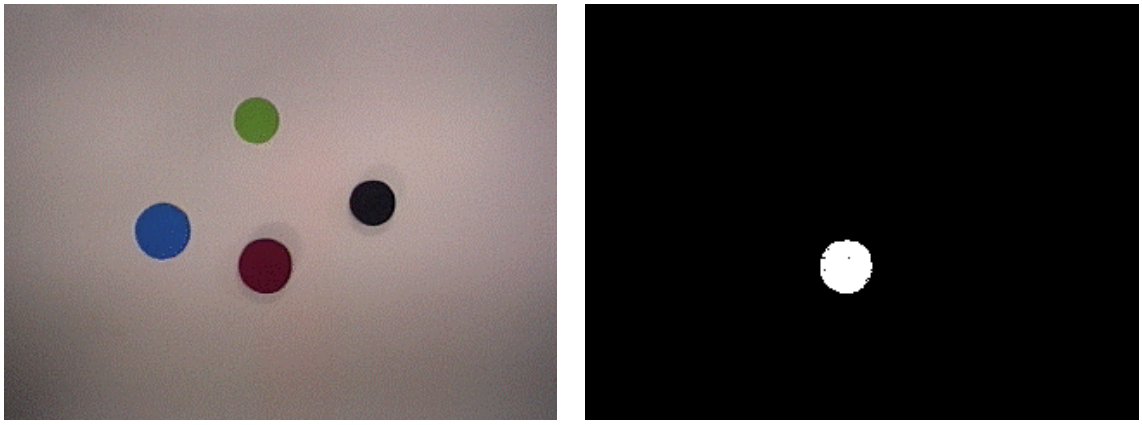


Imagem original Imagem binarizada

Figura 6.8 - Imagem binarizada para a cor vermelha.

6.3.5.3 Identificador de Movimentação (movementIdentifier)

Dadas duas imagens, de tamanho arbitrário, este agente identifica se houve uma movimentação de algum objeto nas imagens calculando o “ou exclusivo” (XOR) entre as imagens e verificando se o número de pontos diferentes é menor que uma tolerância determinada. Caso seja menor que a tolerância, o agente considera que não houve movimentação na imagem. O agente pode ser usado com imagens de qualquer tamanho, coloridas ou não. Caso a imagem seja colorida, é realizada a comparação entre as cores de uma mesma posição nas duas imagens. É atribuido 0 na imagem resultante caso a cor for igual nas duas imagens e 1 caso a cor for diferente. (Definiu-se que as cores são

iguais se a diferença entre os valores YUV de cada pixel esta abaixo de um limite pré-determinado)

6.3.5.4 Determinador de Contornos (edgeDetector)

Dada uma imagem binária, este agente determina os contornos existentes nela. É baseado no algoritmo descrito em (RILLO, A., 1989), que utiliza a definição de contornos proposta por KITCHIN e PUGH (1983). Ele verifica quatro elementos da imagem (máscara 2x2) para determinar se um ponto faz parte de um contorno ou não, criando uma nova imagem onde as bordas encontradas estão entre os elementos da imagem original.

A Figura 6.9 apresenta seis configurações básicas das dezesseis possíveis combinações de quatro elementos de uma imagem binária. As configurações na coluna esquerda não pertencem a regiões de contorno e por isso se atribui zero ao ponto correspondente da imagem de resultante. Os da coluna direita são pontos que pertencem a um contorno e resultam em um. As outras configurações possíveis podem ser obtidas pela rotação destas apresentadas.

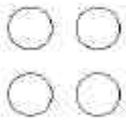
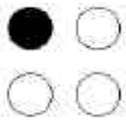
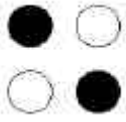
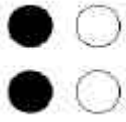
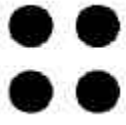
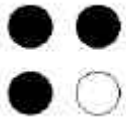
Elementos não pertencentes ao contorno (Resulta 0)	**Elementos pertencentes ao contorno (Resulta 1)**
	
	
	

Figura 6.9 – Definição dos elementos básicos de borda

Como este processo encontra o contorno entre os elementos da imagem, em vez de dobrar a resolução da imagem resultante - o que seria necessário para desenhar as bordas entre os pixels originais- esta é deslocada de meio pixel para a esquerda e para cima da imagem original, visando manter a resolução original.

A Figura 6.10 mostra uma imagem capturada onde existe um anel azul e uma peça preta, a imagem filtrada para a cor azul e o contorno encontrado.

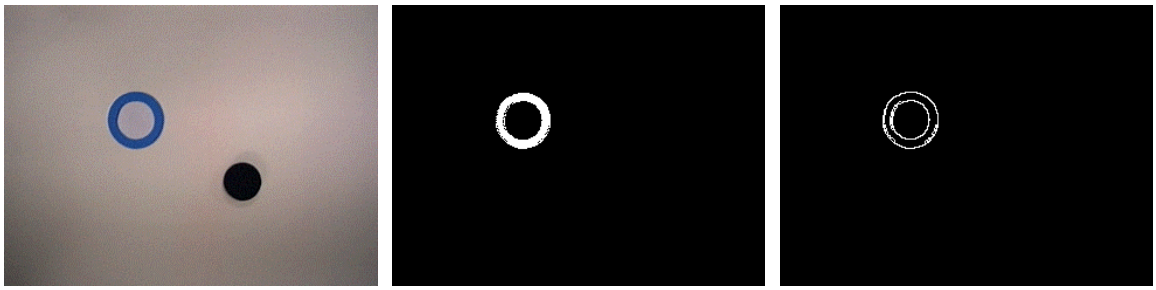


Imagem Capturada | Imagem Filtrada | Imagem dos Contornos

Figura 6.10 - Determinação de Contornos

6.3.5.5 Vetorizador (vetorizer)

Dada a imagem binária, de tamanho determinado contendo os contornos de uma imagem, este agente encontra todas as regiões da imagem que possuem um contorno fechado. Ele gera uma lista com a área, o tamanho do contorno mínimo (*chain size*), a posição do centro de área para todas as regiões fechadas da imagem. Ainda, este agente verifica se cada região possui uma forma circular ou não, utilizando a razão entre a área e o contorno calculado.

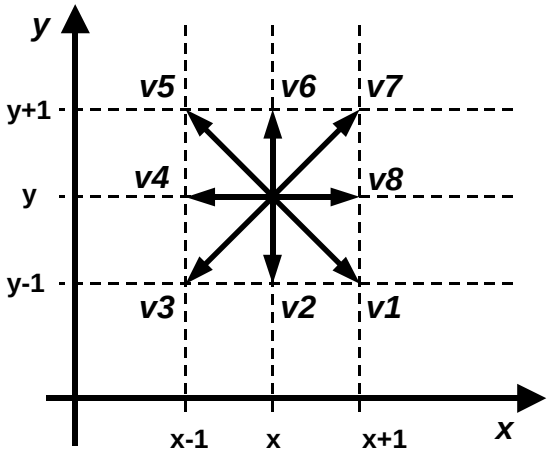


Figura 6.11 – Definição dos vetores elementares

O agente usa um algoritmo do tipo *chain code* (BALLARD, D.; BROWN, C., 1982), baseado no descrito em (RILLO, A., 1989) e em (KITCHIN, P. W.; PUGH, A., 1983). Este algoritmo considera que qualquer ponto em matriz de imagem está conectado aos oito pontos imediatamente a sua volta através de um vetor elementar, cuja

direção foi rotulada (de v1 até v8) e que são apresentados na Figura 6.11.

O contorno de uma região pode ser determinado, assim como sua área calculada, através de um procedimento que segue os contornos de uma imagem, criando uma corrente de vetores ligados. Basta o ponto inicial da corrente e a seqüência dos vetores para se determinar precisamente uma região. Este procedimento segue o contorno mais externo, caso a borda possua mais de um pixel de espessura.

Para calcular a área de uma região é realizado um procedimento semelhante à integração, onde a área total é dada pela somatória da contribuição de cada vetor elementar. A Figura 6.12 mostra as contribuições dos vetores e a Tabela 6.1 apresenta os valores desta contribuição.

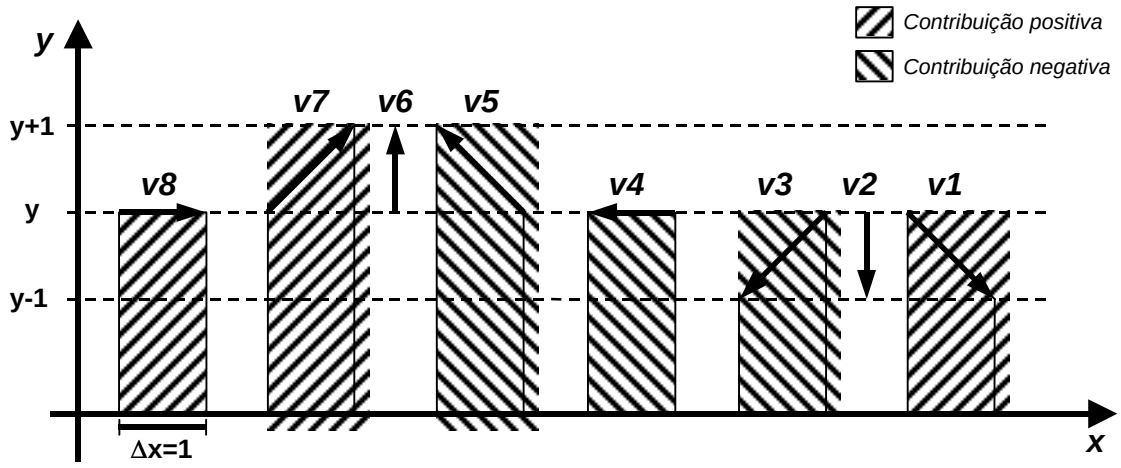


Figura 6.12 – Contribuição de cada elemento para o cálculo da área

Vetor de Direção	**Δ Área**
v8	y · 1
v7	(y + ½) · 1
v6	0
v5	(y + ½) · (-1)
v4	y · (-1)
v3	(y - ½) · (-1)
v2	0
v1	(y - ½) · 1

Tabela 6.1 – Valores da contribuição de área de cada elemento

O perímetro de uma região pode ser encontrado facilmente através dos algoritmos baseados em *chain-codes*. O perímetro total é:

$$Perímetro = PerimetroPar + PerimetroÍmpar \cdot \sqrt{2}$$

isto é, a soma do número de vetores com rótulo par (que estão na horizontal ou vertical e por isso tem tamanho unitário) mais a soma do vetores com perímetro ímpar (que estão nas diagonais) vezes o tamanho destes, que é $\sqrt{2}$. O centro da região também pode ser calculado, utilizando fórmulas para o cálculo de centro de área.

A figura abaixo apresenta um exemplo de imagem vetorizada.

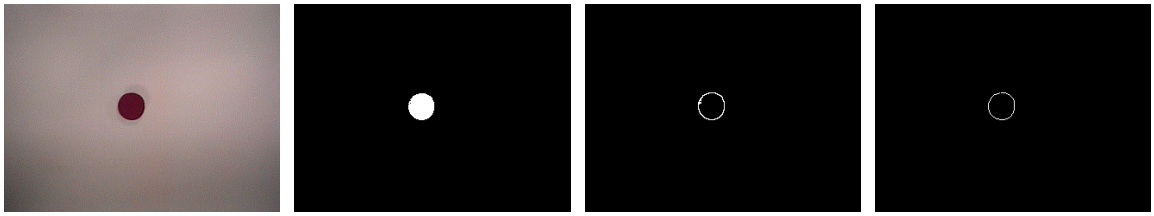


Imagem Capturada | Imagem Filtrada | Contornos | Imagem Vetorizada

Figura 6.13 - Vetorizador

6.3.5.6 Selecionador de Pares (matcher)

Este agente recebe duas listas de regiões da imagem gerada pelo vetorizador, uma para objetos e outra para encaixes, e procura uma peça e um encaixe que casem, isto é, sejam de tamanho (em área) compatível .

6.3.5.7 Controlador do manipulador

Este agente é usado para enviar os comandos para o manipulador robótico. Ele pode receber as seguintes instruções:

- "(pickup-object (tipo_do_objeto) at (posiçãoX, posiçãoY))" - pega um objeto na posição X, Y.
- "(putdown-object (tipo_do_objeto) at (local))" - coloca um objeto em um local determinado.
- "(drop-held-object)" - larga qualquer objeto na posição que o manipulador estiver.
- "(move-object (tipo_de_objeto) to (posiçãoX, posiçãoY))" - move um objeto para a posição X, Y.

- “(join-pair (tenon) (mortise) at (posiçãoX, posiçãoY))”. Junta uma peça e um encaixe, na posição X, Y.

Os valores da posição X e Y que este agente recebe referem-se à posição do objeto nas coordenadas da imagem. Assim, é necessário realizar uma calibração para poder transformar a posição do objeto na imagem para valores de posição nas coordenadas do manipulador.

Para isto é usado um padrão de calibração (Figura 6.14), que é colocado sobre a área de trabalho. Todos os círculos do padrão estão em posições conhecidas do manipulador. Antes de se utilizar o sistema deve ser realizada a calibração *off-line*, que consiste na montagem de uma tabela a partir da imagem do padrão e das posições previamente conhecidas. Esta tabela é usada pelo agente Controlador do Manipulador para encontrar as posições dos objetos durante a operação do sistema, fazendo a correspondência (através da interpolação linear) entre os espaço-imagem e o espaço-manipulador. Durante a operação do sistema, é usada interpolação linear para determinar as posições dos objetos.

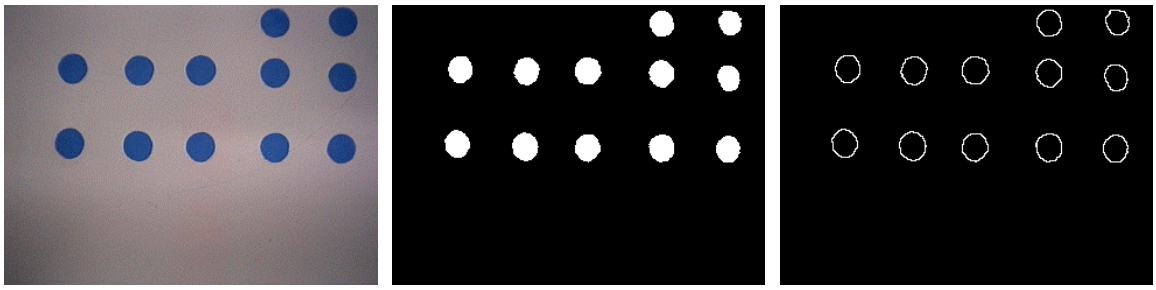


Imagem Capturada | Imagem Filtrada para o Azul | Imagem dos Contornos

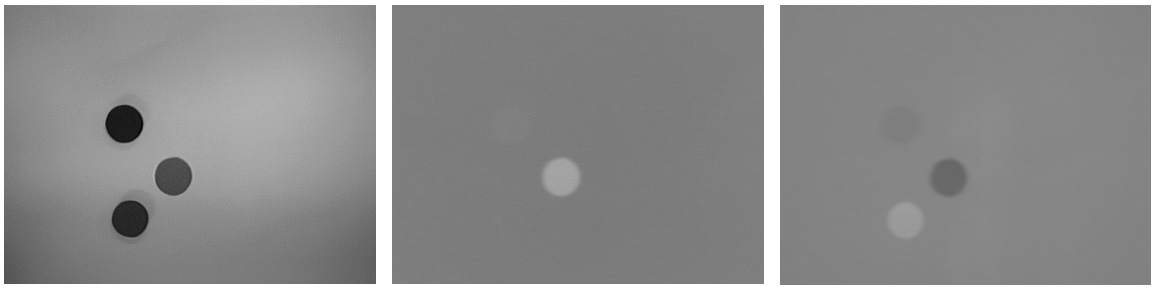
Figura 6.14 - Padrão de Calibração.

Finalmente, este agente é implementado no microcomputador padrão IBM-PC que controla o manipulador robótico e recebe os comandos através da rede da célula de montagem flexível.

6.3.5.8 Segmentador de Regiões Conectadas Coloridas - Rotulação por Cor (colorImageSegmentation)

Este agente segmenta uma imagem colorida, de tamanho fixo (640x480), rotulando as regiões conectadas (da mesma cor). Ele é baseado em um algoritmo de rotulação de regiões em uma imagem do tipo *Blob Coloring*, descrito em (BALLARD,

D.; BROWN, C., 1982), com adaptação para imagens coloridas.

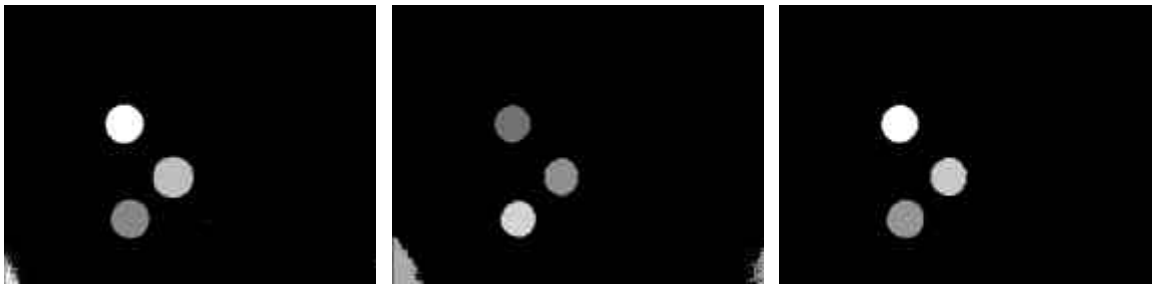


Y U V

Figura 6.15 - Imagem capturada.

Este agente recebe uma imagem e cria uma lista contendo uma descrição da cena, com um “id” para cada objeto, seu tipo (dentre alguns tipos conhecidos), seu tamanho em pixels e a sua posição na imagem.

Ele trabalha em 3 etapas: primeiro rotula uma região; depois agrupa regiões segundo cor e proximidade em objetos; calcula a área e o centro de massa dos objetos; finalmente objetos desconhecidos (como sombras nos cantos da imagem) são retirados. O resultado da execução deste algoritmo sobre uma imagem (Figura 6.15) é mostrado na Figura 6.16. Na atual implementação do sistema ele não foi utilizado.



Regiões Agrupadas Objetos na Cena Objetos de Interesse

Figura 6.16 - Resultado da Segmentação.

6.3.6 Considerações sobre a implementação

Durante a implementação do sistema foi criada uma biblioteca de agentes primitivos, sendo cada um implementado como um processo do sistema que utiliza o PVM para se comunicar, visando possibilitar a reutilização de código para tarefas similares.

Mas durante os testes percebeu-se que a utilização dos agentes primitivos

degradou muito o tempo de resposta do sistema. Isto ocorreu devido ao hardware utilizado (uma rede ethernet de 10 Mbits/segundo que é compartilhada por diversas estações do laboratório) e à maneira de implementar as passagens de mensagens entre os agentes primitivos e o módulo de planejamento/execução dos agentes autônomos, que utiliza o PVM e por isso depende da rede TCP/IP mesmo que os processos estejam localizados em uma mesma estação de trabalho.

Por isso, na tentativa de validação experimental do sistema alguns agentes primitivos foram agregados ao módulo de planejamento/execução dos agentes autônomos, sendo codificados como chamada de funções.

Assim, o sistema deixou de utilizar uma biblioteca de agentes primitivos e passou a usar uma biblioteca de funções primitivas, o que diminuiu a utilização da rede ethernet, permitindo que o sistema apresentasse respostas mais rápidas. Como exceção, os agentes primitivos de captura de imagens e de controle do manipulador continuaram separados em processos independentes, já que são compartilhados pelos agentes autônomos.

Finalmente, para esta implementação, o resultado do compromisso entre a reutilização de módulos e a degradação do sistema não se mostrou interessante, mas a utilização de uma rede de comunicação mais rápida ou a implementação dos agentes primitivos utilizando *threads* (processos “leves” que possibilitam o compartilhamento de memória e mensagens), permitirão a utilização da biblioteca de agentes primitivos, como proposta inicialmente.

Uma série de experimentos foram realizados para uma validação experimental da arquitetura. Os resultados de alguns destes testes são apresentados a seguir.

7. TESTES E RESULTADOS

Devido à dificuldade inerente à construção de sistemas de IA e aos resultados limitados obtida em domínios artificiais bem-comportados, realizar experiências é uma boa maneira de buscar compreensão sobre os sistemas construídos. Experiências podem prover: (i) confirmação preliminar de partes de uma teoria de raciocínio; (ii) sugestões de possíveis modificações para a teoria, para o ambiente onde ela está sendo testada e para o sistema robótico inserido no ambiente. Além disso, elas podem sugerir um grande número de experiências adicionais, possibilitando a expansão e o fortalecimento da teoria original (HANKS et al, 1993).

Neste capítulo são apresentadas várias experiências que usam a arquitetura proposta e que aumentaram a compreensão sobre os Sistemas de Visão Propositada e sobre a metodologia de construção de sistemas robóticos para o mundo real.

7.1 Simulação

Para a validação experimental da estrutura básica da arquitetura proposta foram realizadas diversas simulações. Estas consistiram na criação de sociedades com um número variado de agentes e recursos, sendo utilizados vários sistemas operacionais (SunOS, Solaris, Linux, Irix e WinNT) e arquiteturas (Sun SparcStation 1, 5, 10 e 20, Silicon Graphics multiprocessados, IBM-PC 486), em máquinas virtuais construídas com computadores de um ou vários laboratórios diferentes.

O objetivo destes testes foi o de verificar se as sociedades eram criadas e terminadas da maneira desejada e se os recursos eram passados segundo as regras da sociedade e a estrutura de autoridade.

Nestas simulações o sistema não possuia um propósito global específico nem os agentes autônomos apresentavam comportamentos, sendo que o módulo de planejamento/execução requisitava e liberava aleatoriamente os recursos do sistema, segundo o seguinte algoritmo:

```
espere um tempo aleatório;
escolha 1 recurso aleatoriamente;
se não possui este recurso, requisite-o;
após receber o recurso, espere um tempo aleatório;
verifique os recursos perdidos;
escolha 1 recurso aleatoriamente;
se possui o recurso escolhido, libere-o;
```

Foram realizadas diversas simulações (algumas durando mais de uma hora) com 3 sociedades diferentes:

- 3 agentes e 1 recurso;
- 3 agentes e 5 recursos;
- 20 agentes e 5 recursos.

Inicialmente foram realizados testes com a inserção e remoção dos agentes em uma sociedade, sendo que os resultados obtidos foram os esperados. O Exemplo 7.1 mostra a troca de mensagens durante a inserção de três agentes em uma sociedade com 1 recurso, realizada por um agente iniciador.

```
Agent Comunication Log Archive
Started at host nausika at Fri Apr 10 19:39:49 1998
[ AGENT NAME ][TIME] and Message
[motherAgent         ][19:39:49] LOG: Initializing Agent motherAgent started at nausika.
▪   Adição do agente Montador:
[motherAgent         ][19:39:50] (addAgent (assembler) (assembler motherAgent ))
[assembler           ][19:39:50] LOG: Agent assembler started at enya.
[assembler           ][19:39:50] LOG: MSG Received: Sender: motherAgent, Message:
(addAgent (assembler) (assembler motherAgent )).
[motherAgent         ][19:39:50] LOG: MSG Received: Sender: assembler, Message:
((acknowledge (assembler)).
[assembler           ][19:39:50] LOG: MSG Received: Sender: motherAgent, Message:
(inform ( manipulator motherAgent busy )).
[assembler-execution ][19:39:51] LOG: Execution Module assembler-execution started at
enya.
[assembler-blackboard][19:39:51] LOG: Blackboard Module assembler-blackboard started at
enya.
[motherAgent         ][19:39:51] LOG: MSG Received: Sender: assembler, Message:
((acknowledge (assembler)).
▪   Adição do agente Desobstrutor:
[motherAgent         ][19:39:51] (addAgent (cleaner) (assembler cleaner motherAgent ))
[assembler           ][19:39:51] LOG: MSG Received: Sender: motherAgent, Message:
(addAgent (cleaner) (assembler cleaner motherAgent )).
[motherAgent         ][19:39:51] LOG: MSG Received: Sender: assembler, Message:
((acknowledge (assembler)).
[cleaner             ][19:39:52] LOG: Agent cleaner started at nikita.
[cleaner             ][19:39:52] LOG: MSG Received: Sender: motherAgent, Message:
(addAgent (cleaner) (assembler cleaner motherAgent )).
[motherAgent         ][19:39:52] LOG: MSG Received: Sender: cleaner, Message:
((acknowledge (cleaner)).
[cleaner             ][19:39:52] LOG: MSG Received: Sender: motherAgent, Message:
(inform ( manipulator motherAgent busy )).
[cleaner-blackboard  ][19:39:52] LOG: Blackboard Module cleaner-blackboard started at
nikita.
[motherAgent         ][19:39:52] LOG: MSG Received: Sender: cleaner, Message:
((acknowledge (cleaner)).
[cleaner-execution   ][19:39:52] LOG: Execution Module cleaner-execution started at
nikita.
▪   Adição do agente Evitador de Colisão:
[motherAgent         ][19:39:52] (addAgent (collisionAvoider) (assembler cleaner
collisionAvoider motherAgent ))
[assembler           ][19:39:52] LOG: MSG Received: Sender: motherAgent, Message:
(addAgent (collisionAvoider) (assembler cleaner collisionAvoider motherAgent )).
[cleaner             ][19:39:52] LOG: MSG Received: Sender: motherAgent, Message:
(addAgent (collisionAvoider) (assembler cleaner collisionAvoider motherAgent )).
[motherAgent         ][19:39:53] LOG: MSG Received: Sender: assembler, Message:
((acknowledge (assembler)).
[motherAgent         ][19:39:53] LOG: MSG Received: Sender: cleaner, Message:
((acknowledge (cleaner)).
[collisionAvoider    ][19:39:53] LOG: Agent collisionAvoider started at nausika.
[collisionAvoider    ][19:39:53] LOG: MSG Received: Sender: motherAgent, Message:
(addAgent (collisionAvoider) (assembler cleaner collisionAvoider motherAgent )).
[motherAgent         ][19:39:53] LOG: MSG Received: Sender: collisionAvoider, Message:
```

```
((acknowledge (collisionAvoider)).
[collisionAvoider    ][19:39:54] LOG: MSG Received: Sender: motherAgent, Message:
(inform ( manipulator motherAgent busy )).
[collisionAvoider-exe][19:39:54] LOG: Execution Module collisionAvoider-execution
started at nausika.
[collisionAvoider-bla][19:39:54] LOG: Blackboard Module collisionAvoider-blackboard
started at nausika.
[motherAgent         ][19:39:54] LOG: MSG Received: Sender: collisionAvoider, Message:
((acknowledge (collisionAvoider)).
▪   Liberação dos recursos a partir do agente inicializador:
[motherAgent         ][19:39:54] (freeAll)
[collisionAvoider    ][19:39:54] LOG: MSG Received: Sender: motherAgent, Message:
(freeAll).
[assembler           ][19:39:54] LOG: MSG Received: Sender: motherAgent, Message:
(freeAll).
[cleaner             ][19:39:54] LOG: MSG Received: Sender: motherAgent, Message:
(freeAll).
```

Exemplo 7.1 - Inserção de agentes com comentários em itálico.

Em seguida foram realizados os testes com a passagem dos recursos entre os diversos agentes. A Figura 7.1 mostra o resultado da simulação do funcionamento de uma sociedade com 4 agentes e 1 recurso, onde a linha em negrito indica qual agente está com o recurso em um determinado momento. Pode-se notar que o agente iniciador, de numero 4, inicia com o recurso e depois o libera. Em seguida, o recurso é trocado pelos outros 3 agentes da sociedade, sendo que o número 3 é o de maior autoridade e o de número 1, menor.

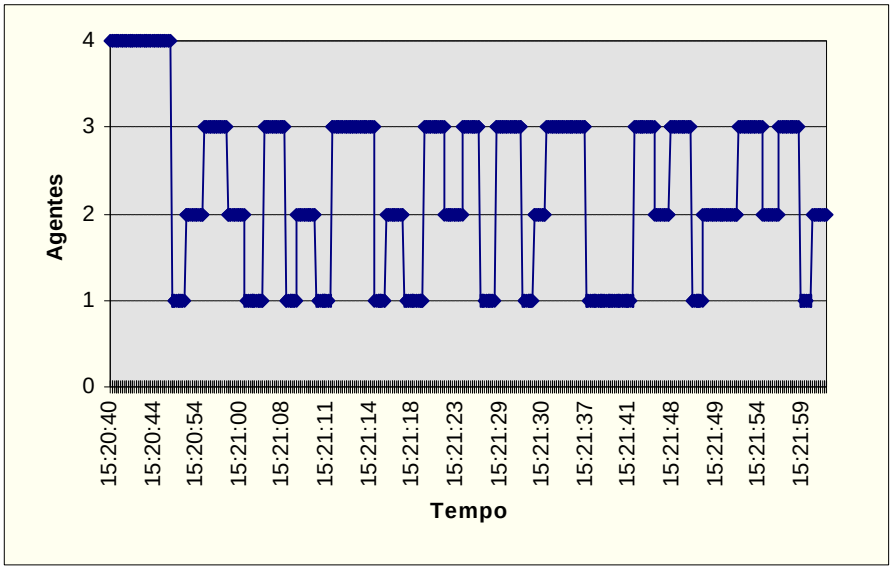


Figura 7.1 - Alocação de 1 recurso por 4 agentes. (simulação)

A Figura 7.2 apresenta a mesma simulação, em um período de tempo maior. Nela pode ser visto que o agente com maior autoridade recebe mais vezes o recurso que os de menor autoridade.

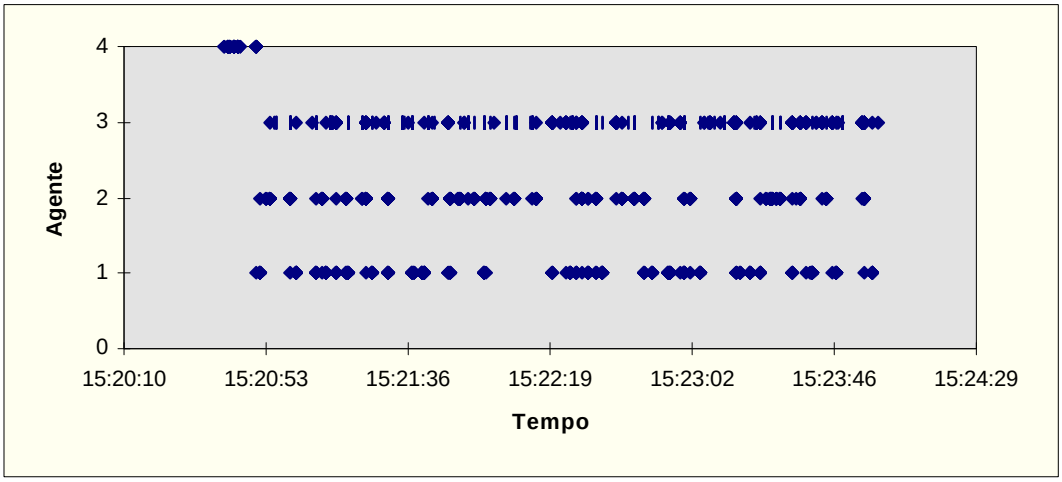


Figura 7.2 - Alocação de 1 recurso por 4 agentes durante 5 minutos. (simulação)

A Figura 7.3 apresenta o resultado da simulação de uma sociedade com 4 agentes e 5 recursos. Pode-se notar que, após os recursos terem sido liberados, eles são passados em tempos aleatórios para os diversos agentes e que o agente número 3 (de maior autoridade) fica em média com mais recursos que o número 2, que por sua vez consegue manter mais recursos que o agente número 1. A estrutura de autoridade nesta simulação é igual à da simulação anterior.

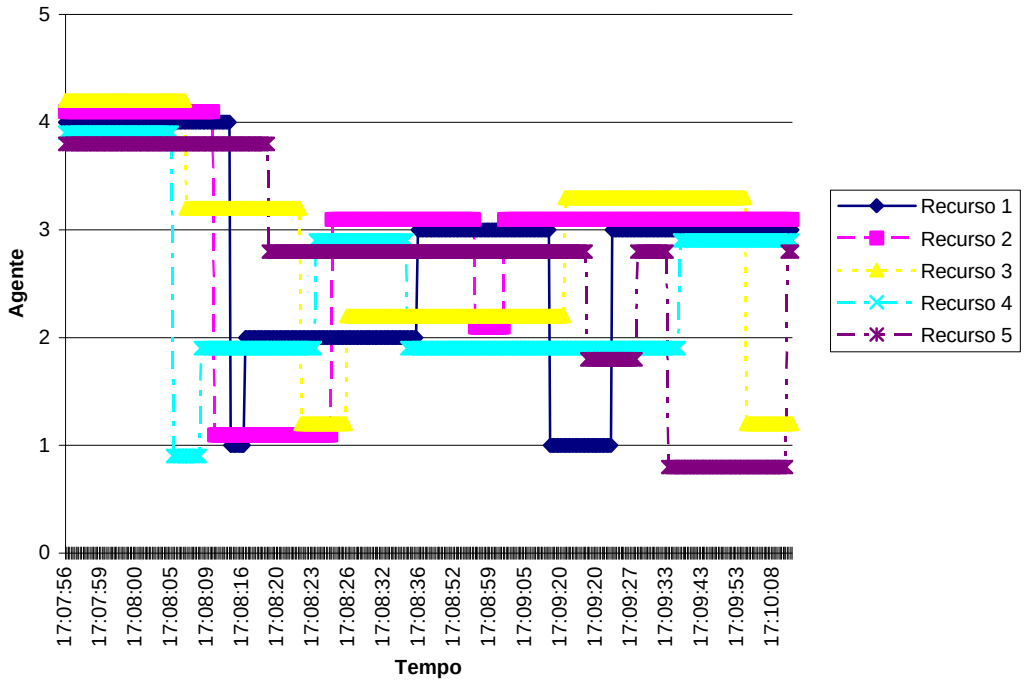


Figura 7.3 - Alocação de 5 recurso por 4 agentes. (simulação)

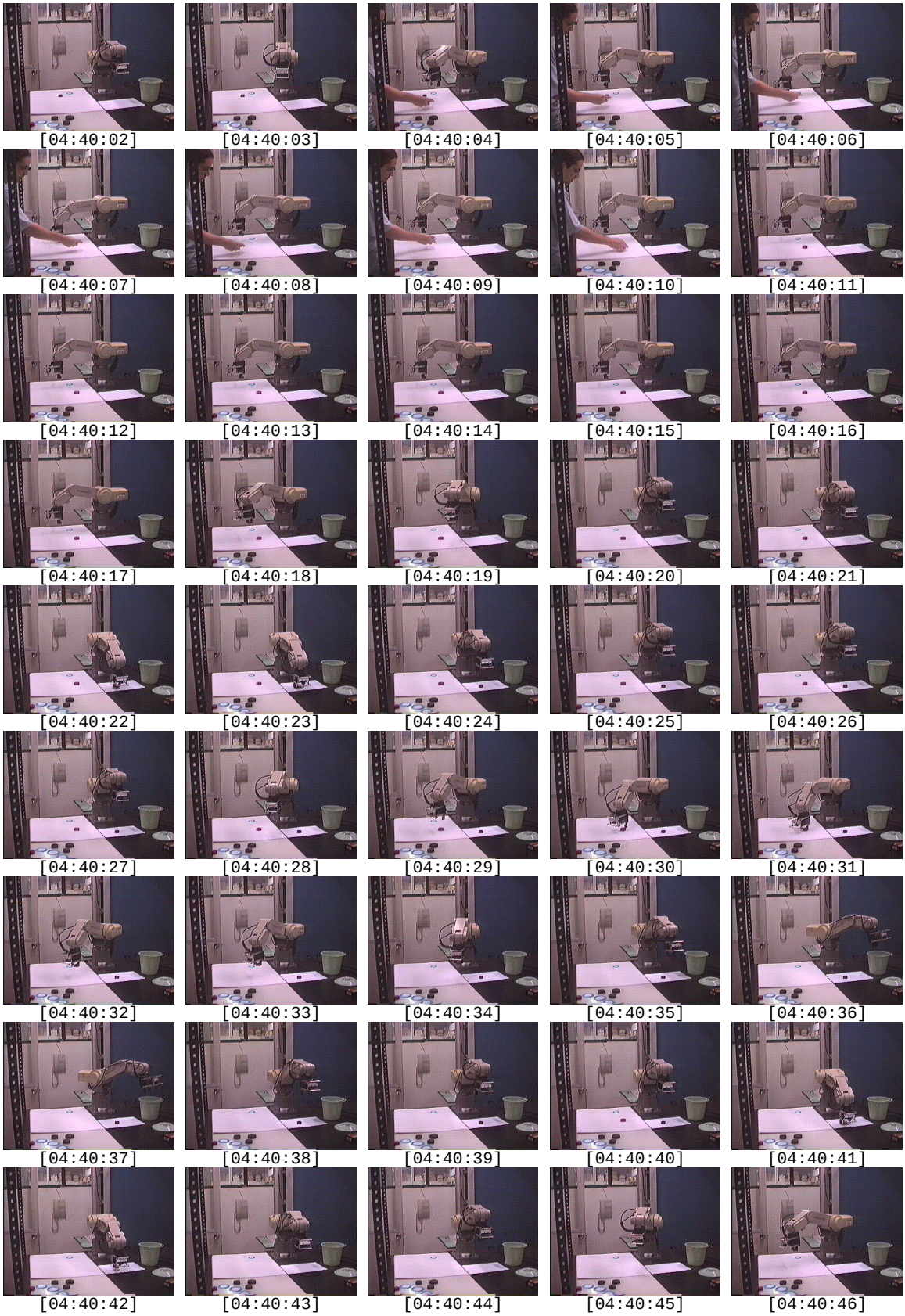
São apresentados agora alguns dos experimentos realizados com o sistema real de Visão Computacional Propositada implementado na célula e resultados.

7.2 Resultados da Aplicação

A aplicação implementada na Célula de Montagem Flexível foi testada extensivamente. É apresentado agora um exemplo do funcionamento do sistema, onde ocorre a interrupção de uma montagem com a colocação de uma obstrução na área de trabalho. A Figura 7.4 mostra o sistema em funcionamento e a Figura 7.5 mostra como o recurso manipulador foi compartilhado pelos agentes durante a interrupção do trabalho de montagem.

O sistema é iniciado às 4:39:27 e os agentes são adicionados um a um. Às 04:40:00 o agente *Assembler* inicia a montagem de uma peça. As 04:40:04 um objeto desconhecido entra na área de trabalho. Às 04:40:06 o agente *CollisionAvoider* descobre que necessita interromper a montagem. Ele requisita o recurso, recebe-o do agente *Assembler* às 04:40:07, parando o manipulador. O objeto se mantém em movimento até 04:04:10. Às 04:40:14 o agente *Cleaner* descobre um objeto desconhecido na área de trabalho, mas não pode requisitar o manipulador, pois este está sendo utilizado pelo *CollisionAvoider*.

Às 04:40:16 o *CollisionAvoider* libera o recurso, que é requisitado pelo agentes *Assembler* e *Cleaner*. A mensagem do *Assembler* chegou em primeiro lugar, assim o *CollisionAvoider* transfere para ele o manipulador. Mas imediatamente o *Cleaner* retira do *Assembler* este recurso, pois tem maior autoridade. Assim, no mesmo segundo, o *Cleaner* recebe o manipulador e inicia a colocação da peça da montagem na mesa, para poder retirar o lixo. Às 04:40:25 ele comanda o manipulador para pegar o lixo da área de trabalho, e às 04:40:31 manda o manipulador de posicionar sobre a lixeira. Às 04:40:33 manda largar o objeto no lixo. (Nota: a ação largar o objeto, apesar de ter sido ordenada às 04:40:33, só ocorre às 04:40:37, devido à maneira na qual a comunicação com o manipulador foi implementada.)


[04:40:02]
[04:40:03]
[04:40:04]
[04:40:05]
[04:40:06]
[04:40:07]
[04:40:08]
[04:40:09]
[04:40:10]
[04:40:11]
[04:40:12]
[04:40:13]
[04:40:14]
[04:40:15]
[04:40:16]
[04:40:17]
[04:40:18]
[04:40:19]
[04:40:20]
[04:40:21]
[04:40:22]
[04:40:23]
[04:40:24]
[04:40:25]
[04:40:26]
[04:40:27]
[04:40:28]
[04:40:29]
[04:40:30]
[04:40:31]
[04:40:32]
[04:40:33]
[04:40:34]
[04:40:35]
[04:40:36]
[04:40:37]
[04:40:38]
[04:40:39]
[04:40:40]
[04:40:41]
[04:40:42]
[04:40:43]
[04:40:44]
[04:40:45]
[04:40:46]

segue →

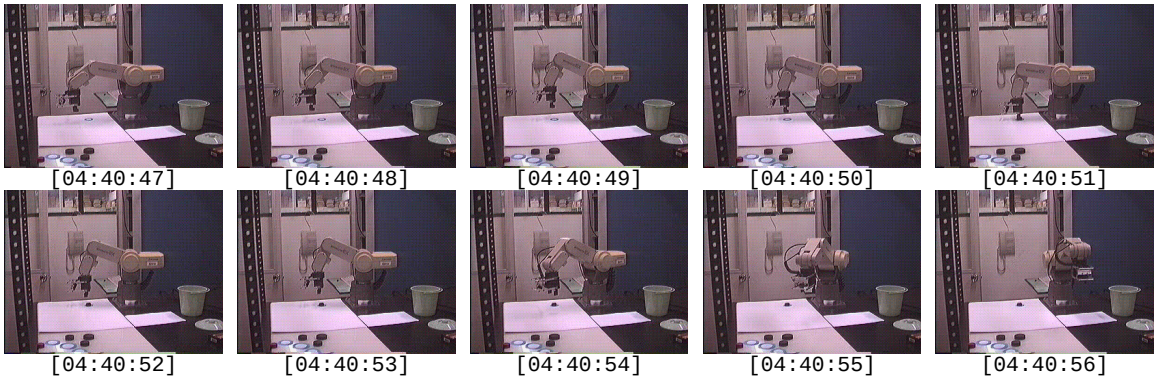


Figura 7.4 - Montagem com Interrupção e Desobstrução.

As 04:40:37, o agente *Cleaner* comanda o manipulador para pegar a peça da mesa e as 04:40:42 envia o manipulador para a posição original. As 04:40:44 o *Cleaner* libera o recurso, sendo transferido imediatamente para o *Assembler*, que no mesmo instante move o manipulador para a posição do encaixe. As 04:40:48 o *Assembler* coloca a peça sobre o encaixe, mandando o manipulador de volta à posição de descanso as 04:40:52.

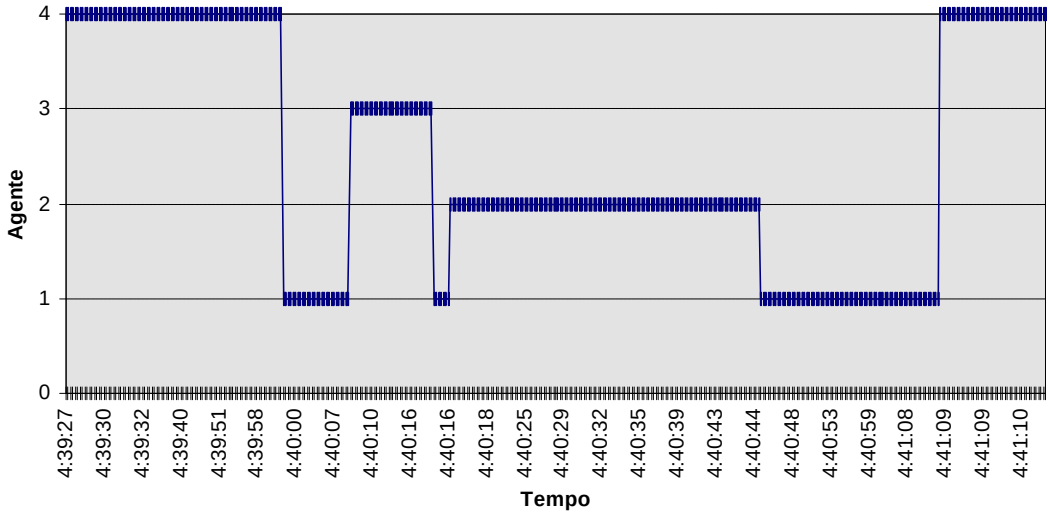


Figura 7.5 - Alocação do recurso manipulador durante o exemplo.

Para a validação experimental do sistema implementado foram realizadas dezenas de experiências como a descrita acima (ver Apêndice), em diversas configurações do sistema e sempre com a medição dos tempos de reação a partir de um arquivo de *log* de mensagens. Com os dados gerados, alguns resultados puderam ser calculados.

Inicialmente foi verificada a taxa de transmissão média do sistema a partir de um

programa fornecido na plataforma PVM. Pode-se notar que esta taxa, que é definida como o tempo que o PVM leva para preparar uma transmissão (empacotamento) mais o tempo de uso da rede (envio), decai quando o sistema é iniciado. A taxa de transmissão do sistema *off-line* é importante pois estabelece um limite superior para a velocidade que o sistema pode alcançar nesta configuração da rede (*ethernet*).

	Empacotamento Byte/μSegundo	**Envio Byte/μSegundo**	**Total**
off line	11,75	1,14	12,89
3 agentes	6,8	1,32	8,12

Tabela 7.1 - Taxa de transmissão média do PVM com sistema *of line* e *on line.*

Em seguida foi medida a velocidade de aquisição de imagens por um agente autônomo para algumas configurações do sistema. Este tempo inclui a requisição da imagem pelo módulo de planejamento/execução do agente, o tempo de captura pelo agente primitivo Capturador de Imagens e o tempo de transmissão da imagem através da rede. A partir desta velocidade, que depende da taxa de transmissão, puderam ser calculadas as velocidades máxima e mínima que um objeto pode ter para que o agente Evitador de Colisões possa reconhecer o seu movimento (Tabela 7.2). Usando imagens de baixa resolução, a velocidade máxima é maior pois o agente adquire mais imagens por segundo; com imagens de média resolução, a velocidade mínima é menor já que a precisão é maior.

Resolução da Imagem (pixels)	**Velocidade Máxima (m/s)**	**Velocidade Mínima (cm/s)**
Baixa(64 x 48)	6,05	39
Média (120 x 80)	1,1	3,6

Tabela 7.2 - Velocidades máximas e mínima para um objeto em movimento.

Em seguida, a partir dos dados acumulados nos experimentos, alguns tempos de reação e de processamento puderam ser calculados. O tempo de reação é definido como o tempo que o agente leva entre o momento no qual é detectada a necessidade de um recurso e o momento no qual o recurso fica disponível para ser utilizado pelo agente. O tempo de processamento é o utilizado para detectar a necessidade do recurso.

Finalmente, o tempo de comunicação é o intervalo entre o momento que o agente requisita um recurso e o momento que este recurso é transferido.

A Tabela 7.3 apresenta o tempo de reação do agente Evitador de Colisão, que é o tempo que o agente leva para parar o manipulador quando um objeto em movimento entra na área de trabalho. Este tempo foi obtido experimentalmente para o agente em execução sozinho na sociedade e imagens de dois tamanhos diferentes (as duas primeiras linhas), com o agente Desobstrutor (terceira linha) e com os 3 agentes.

O tempo de reação é dependente da resolução da imagem e da taxa de transmissão da rede, já que imagens necessitam de muita largura de banda.

Configuração	**Processamento**	**Comunicação**	**Tempo Total**
solo (64 x 48)	0.15	0	0.15
solo (120 x 80)	0,62 +- 0,26	0	0,62 +- 0,26
+ Desobstrutor (120x80)	1,8 +- 0,33	0,1 +- 0,16	1,9 +- 0,3
3 agentes (120 x 80)	2,4 +- 0,6	0,1 +- 0,16	2,6+- 0,4

Tabela 7.3 - Tempo de reação em segundos para o agente Evitador de Colisão.

O gráfico do tempo de reação para uma colisão versus o número de agentes que estão na sociedade é apresentado na Figura 7.6. Pode-se notar que este tempo cresce linearmente. Isto ocorre pois quanto mais agentes estiverem no sistema, mais concorrido se torna o agente Capturador de Imagens e a rede fica mais congestionada.

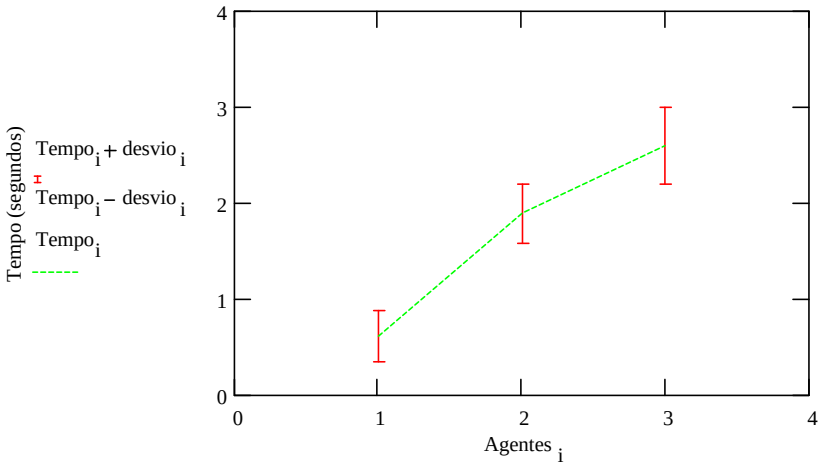


Figura 7.6 - Tempo médio de reação para colisão versus o número de Agentes no sistema.

Seguindo a mesma linha de experimentos, o tempo de reação do agente Desobstrutor pode ser definido como o tempo que ele leva para perceber que existe um objeto que deve ser retirado da área de trabalho. O tempo médio de reação foi obtido experimentalmente para 3 configurações (Tabela 7.4).

Configuração	**Tempo (segundos)**
Desobstrutor com captura na mesma estação	0,88
Desobstrutor em uma estação e Captura em outra	1,46
3 agentes	4,61

Tabela 7.4 - Tempo de reação do Agente Desobstrutor.

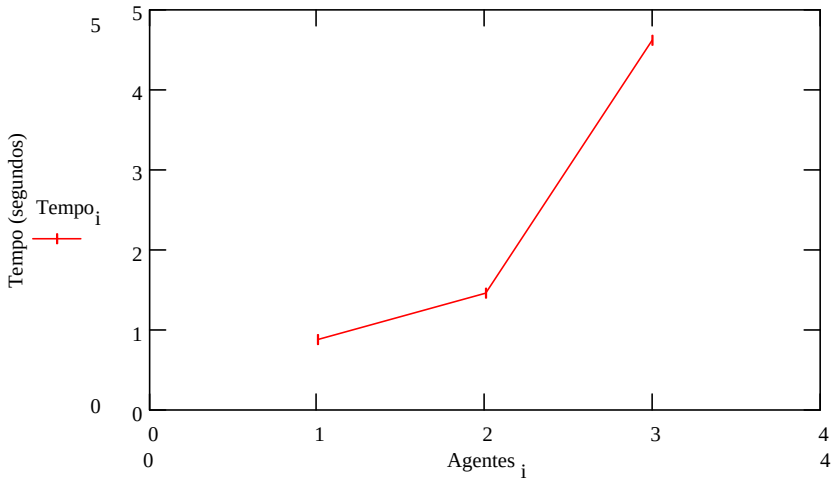


Figura 7.7 - Tempo de reação médio para a desobstrução versus o número de agentes no sistema. (desvios não significativos)

Pelas tabelas acima podemos perceber o atraso que a existência de mais de um agente causa nos tempos de reação do sistema.

O tempo total de retirada de um objeto não desejado é maior que o tempo de reação do Desobstrutor. A Tabela 7.5 apresenta o tempo médio para o agente Desobstrutor completar suas ações, quando não interrompido por outro agente, com o sistema sendo executado com os 3 agentes autônomos. O tempo de processamento inclui o atraso na rede causado pela existência de outros 2 agentes no sistema. O tempo de comunicação é o tempo decorrido do momento que o agente percebe o lixo parado até receber o recurso. Como esperado, o tempo de trabalho do manipulador é o maior.

Processamento	Comunicação	Tempo de Trabalho
5,1 +- 1,05	1,2 +- 0,8	10 a 15

Tabela 7.5 - Tempo médio de processamento para o agente Desobstrutor (em segundos).

Durante o desenvolvimento deste sistema e da realização dos testes muitas novas idéias surgiram, algumas aproveitadas e outras abandonadas. O próximo capítulo discute estas idéias e apresenta críticas, que podem ser incorporadas em trabalhos futuros.

8. CONCLUSÃO E TRABALHOS FUTUROS

Por ser historicamente distribuída e modular, a abordagem de Sistemas Multi-Agentes é uma tendência natural para a modelagem de uma arquitetura de controle distribuída para um sistema de visão computacional propositada, sendo uma solução eficaz para o problema da integração de diversos módulos comportamentos diferentes em um mesmo sistema robótico e que utiliza visão computacional para perceber o mundo.

Entre as vantagens que a modelagem Multi-Agentes propicia, que também são apresentadas por outros autores [BOND; GASSER, 1988; STONE et al, 1997; MEDEIROS et al, 1997], estão:

- Modularidade: sistemas complexos podem ser divididos em subsistemas menores, que podem ser projetados e implementados separadamente, facilitando a decomposição de um comportamento em tarefas mais simples;
- Paralelismo: tarefas independentes podem ser tratadas por agentes concorrentes, o que ajuda a tratar as limitações impostas por restrições de tempo;
- Robustez: incertezas em uma tarefa ou em algum de seus componentes podem ser tratadas com a criação de agentes redundantes que tentam resolver o problema a partir de diferentes perspectivas;
- Escalabilidade: sistemas podem ser construídos passo a passo pela adição incremental de novas funções, com a inclusão gradual de competências;
- Simplificação da programação: do ponto de vista de um programador a modularidade da abordagem MAS pode levar a programação mais simples;
- Flexibilidade: o projeto do sistema pode ser facilmente modificado, sendo guiado por fracassos ou sucessos apresentados;
- Reusabilidade: aplicações diferentes podem compartilhar módulos comuns, acessando o mesmo programa ou com a reprodução do código.
- MAS explicita a interação entre agentes para atingir seus objetivos, tornando a interface entre eles bem definida;
- As capacidades decisórias de um sistema implementado segundo esta

modelagem são explícitas e distribuídas entre os módulos, o que permite saber exatamente onde uma decisão é tomada;

- O conhecimento de um agente sobre os outros agentes pode ser explícito, modelado internamente em cada agente, o que facilita a interação e realização de negociações e contratos entre os agentes.

Entre as desvantagens da abordagem Multi-Agentes pode-se citar que a flexibilidade de um sistema tem um custo que, no caso, é pago na comunicação e na negociação que se dá entre os agentes para atingir um determinado objetivo. Este fato ficou evidenciado no desenvolvimento deste trabalho, quando foi implementado um processo para cada agente primitivo, o que sobrecarregou a comunicação, penalizando a eficiência do sistema.

Este problema tem relação direta com o problema da granularidade na definição das capacidades de cada agente, em que qualquer sistema pode ser visto como um único agente ou como uma sociedade de agentes. Pode-se perceber que quanto maior o número de agentes mais simples, dedicados e reativos que dependam de outros para realizar tarefas, maior é o custo de comunicação e negociação entre os agentes, já que o que poderia ser resolvido internamente em um agente passa a necessitar de uma negociação entre vários agentes.

A determinação da granularidade dos agentes em um sistema deve levar em conta a relação de custo-benefício, flexibilidade versus sobrecarga de comunicação.

De um ponto de vista metodológico, a estrutura criada mostrou ser uma ferramenta flexível e eficiente para a criação de módulos que solucionam problemas específicos, que deve ser usada para o projeto e desenvolvimento de novas aplicações robóticas.

Durante o projeto e desenvolvimento da estrutura básica da arquitetura e a sua aplicação na construção do sistema implementado foram aprendidas algumas lições importantes sobre o desenvolvimento de um sistema robótico. Estes conhecimentos, alguns dos quais são assunto de discussão por outros autores [BOND; GASSER, 1988; GARCIA-ALEGRE, 1997], podem ser usados no suporte ao desenvolvimento de novos sistemas e consistem em:

- Quando uma comunidade de agentes deve trabalhar de uma maneira

coordenada, é importante decidir como ela será organizada e como será realizada a divisão de trabalho. Isso envolve: a definição das tarefas, a seleção de qual agente realizará cada tarefa e a definição de quando um agente executará uma tarefa;

- A linguagem e os conceitos usados para a descrição de uma tarefa afeta o modo como estas são decompostas e quais dependências explícitas existem entre elas. Um mesmo propósito pode ser descrito a partir de diversas perspectivas, o que pode requerer decomposições e capacidades diversas.
- A distribuição de tarefas entre agentes requer que as tarefas sejam formuladas e descritas de maneira a favorecer uma melhor possibilidade de decomposição, permitindo uma distribuição natural entre os agentes. Tarefas que exijam mais conhecimentos ou recursos que um agente pode possuir devem ser decompostas, de maneira a existir comportamentos cognitivos independentes.
- As dependências entre subproblemas afetam o projeto dos agentes, no fluxos de dados, nos processos decisórios e nas ações.
- Os conflitos que surgem a partir de ações incompatíveis e da divisão dos recursos do sistema podem colocar restrições nas atividades dos agentes, restringindo as maneiras de se decompor um sistema e forçando o projetista a considerar a decomposição em diferentes dimensões (temporal, espacial ou níveis de abstração).
- As regras da sociedade dependem dos tipos de recursos disponíveis e das necessidades de uso destes recursos pelos agentes.
- A estrutura de autoridade de um sistema possui uma profunda relação com a precedência e a dependência entre os comportamento dos agentes presentes na sociedade. Em geral, os comportamentos reativos tem precedência sobre os deliberativos para permitir respostas rápidas ao eventos provenientes do ambiente.

A metodologia utilizada, através do desenvolvimento de uma biblioteca para a construção de agentes e de uma biblioteca de agentes primitivos, especializados em tarefas de visuais, apresentou diversos pontos positivos, entre eles:

- Ela permite a implementação de novos sistemas de Visão Computacional

Propositada de maneira fácil e rápida, já que permite a reutilização de módulos que realizam tarefas simples. No sistema desenvolvido, o primeiro agente autônomo levou dias para ser completado, mas através da reutilização dos agentes primitivos, os outros levaram apenas algumas horas para serem implementados.

- A existência de um modelo padrão para os agentes autônomos.
- Uma vez que um agente autônomo possui dentro de si todos os agentes primitivos necessários aos seu funcionamento (com a reprodução do código para reuso), pode-se ter a certeza que a sua eliminação não estará afetando outros agentes autônomos.
- A sociedade de agentes implementada é dinâmica, podendo inserir ou retirar AAs da sociedade, durante a operação do sistema, segundo o controle de um ou vários agentes desta sociedade. Isto permite que um sistema atue em múltiplos domínios, com apenas a troca dos agentes necessários à sua adaptação.
- O processamento distribuído permite que a eficiência necessária ao desempenho de sistemas complexos de ação-percepção seja alcançada.

Uma crítica à metodologia é que a existência de agentes dedicados a tarefas específicas favorece a repetição de processamento. Apesar de isso não ser um fator importante em um sistema com poucos agentes, na medida em que o sistema cresce isso poderá se tornar um problema.

Quanto à implementação da arquitetura em um sistema para a Célula Flexível de Montagem, os exemplos apresentados dão ao leitor uma idéia qualitativa do tipo de sistema que pode ser implementado. Algumas conclusões que podem ser retiradas destes resultados são:

- Apesar dos resultados obtidos com a linguagem de comunicação implementada terem sido excelentes, a comunicação entre os agentes não teve o desempenho esperado. Isto foi causado em parte pela utilização do sistema PVM para o envio de qualquer tipo de mensagem, já que o mesmo funciona como um barramento.
- Quanto mais agentes são inseridos na sociedade, maior o tempo de reação destes agentes. Isto sugere a necessidade de que a rede seja tratada como um

recurso.

- Conforme o número de agentes cresce, maior é a quantidade de imagens que são requisitadas para a câmera. Assim, existe a necessidade dela ser tratada como um recurso, definindo prioridades entre os agentes.

É interessante notar que o sistema implementado na célula pode ser classificado como um sistema DPS, pois a distribuição de tarefas é feita por um programador e não pelo próprio sistema e o sistema é construído para resolver um problema pré-existente. Isto reflete o fato de termos assumido uma postura dedicada e comportamental. Porém, deve-se lembrar que DPS e MAS não são áreas disjuntas: DPS pode ser visto como uma parte da solução de um problema na abordagem MAS.

No sistema implementado, o comportamento dos agentes é o de cooperação ou distribuição, que segundo DEMAZEAU e MÜLLER (1990) é aquele de um agente que precisa atingir um objetivo mas não consegue fazer isso sozinho, tendo que trabalhar com outros agentes. Isto ocorre pois alguns agentes dependem indiretamente de outros para atingir seus objetivos, voltando ao problema de dependência e precedência entre os agentes. Por exemplo, o agente Montador só consegue atingir seu objetivo se a área de trabalho estiver desobstruída e não existirem colisões. Por isso, ele depende dos outros agentes. Já o agente que evita colisões não depende de nenhum outro.

Quanto aos aspectos de Visão Computacional, os algoritmos implementados são simples. Além disso, como o enfoque desta dissertação foi a implementação de uma arquitetura para esta classe de sistemas, foram necessárias maiores simplificações para adequar o tempo de desenvolvimento do sistema ao tempo disponível. Com isso, o sistema de visão implementado trabalha apenas em duas dimensões, não reconhecendo sobreposições de objetos; ele somente reconhece objetos pretos (as peças), azuis (os encaixes) ou vermelhos (as obstruções); finalmente, o sistema só detecta colisões com objetos da cor vermelha.

O sistema foi implementado de maneira distribuída em 3 estações de trabalho Sun SparcStation 5 interligadas por uma rede ethernet de baixa velocidade. Em função desta configuração, foi reduzida a quantidade de agentes primitivos utilizados pelos agentes autônomos para permitir uma melhoria nos tempos de comunicação entre os agentes e de reação do sistema a eventos assíncronos.

Uma questão que não é tratada neste trabalho, e que é principal nos trabalhos de Brooks e de Visão Animada, é a necessidade de um sistema possuir habilidades de aprendizado para ser robusto e capaz de se adaptar ao seu ambiente. Porém, este aprendizado pode ser introduzido no sistema de duas maneiras: na primeira é criado um agente autônomo, cujo comportamento é de aprender novas tarefas e adicionar agentes ao sistema; e na segunda, adiciona-se capacidades de planejamento e aprendizado na área de especialização de cada agente autônomo. Neste segundo caso, os agentes tornariam-se capazes de planejar para alcançar seus objetivos, conhecendo suas habilidades e a dos outros agentes. Porém, estas capacidades possuem um alto preço em complexidade e custo computacional, o que dificultaria sua implementação e sua atuação em tempo real, fugindo do escopo desta proposta.

Durante o desenvolvimento da arquitetura proposta nesta dissertação muitas sugestões de aperfeiçoamentos foram discutidas. As sugestões de trabalhos futuros mais importantes são:

- O grau de cooperação entre os agentes deve ser maior e mais explícito para o melhor aproveitamento de uma sociedade. Para isso, existe a necessidade de que os agentes troquem informações mais relevantes, com a ampliação da linguagem de comunicação. Por exemplo, o sistema ficaria muito mais eficiente se um agente informasse aos outros que uma determinada peça entrou na área de trabalho ou que existe um objeto desconhecido se movendo.
- Outra sugestão de modificação importante é a implementação dos recursos como agentes autônomos que oferecem serviços e que possuem seu próprio conhecimento sobre os estados e informações relevantes.
- Aumentar o nível de inteligência e raciocínio de forma que os agentes possam cumprir tarefas mais deliberativas e menos reativas.
- O aumento do número de agentes para uma aplicação real mais complexa permitiria uma avaliação mais crítica da arquitetura proposta.
- A depuração de um sistema distribuído é uma tarefa muito difícil. Uma sistematização mais apropriada para depuração, com a construção de ferramentas que facilitem este trabalho, permitiria uma melhor avaliação do sistema e uma comparação deste com outros trabalhos.

- Um agente autônomo, que é composto por três módulos (comunicação, planejamento/execução e do quadro negro), deve ser encapsulado fora do PVM: as mensagens entre estes módulos devem independer do barramento do PVM, e as mensagens entre estes módulos e os agentes primitivos também. Uma solução possível é a implementação destes elementos como *threads* (processos “leves” que possibilitam o compartilhamento de memória e mensagens). Isto diminuiria o uso da rede de comunicação e aumentaria a velocidade de reação do sistema.
- Uma opção que foi muito discutida durante a implementação consiste na eliminação do PVM e o uso direto dos pacotes de comunicação entre tarefas com *sockets*. Esta opção não foi adotada, já que depois de alguma pesquisa, chegou-se à conclusão que isto não aumentaria a velocidade do sistema, pois o problema de lentidão na comunicação se encontra nos *sockets*, dos quais o PVM faz uso, e não no PVM propriamente dito.
- Finalmente, deve ser implementada uma maneira de realizar a confirmação sensorial que verifica se uma ação foi realizada. Isso fecharia o laço de controle no nível das ações primitivas.

9. REFERÊNCIAS BIBLIOGRÁFICAS

AGGARWAL, J. K.; MARTIN, W. N. The role of representation and reconstruction in vision: is it a matter of definition? **CVGIP: Image Understanding**, v.60, n.1, p.100-2, July 1994.

ALOIMONOS, Y., ed. Special issue on purposive, qualitative and active vision. **CVGIP: Image Understanding**, v.56, n.1, 1992.

ALOIMONOS, Y. What I have learned. **CVGIP: Image Understanding**, v.60, n.1, p.74-85, July 1994.

ALOIMONOS, Y.; ROSENFELD, A. A response to "Ignorance, myopia and naiveté in computer vision systems" by R. C. Jain and T. O. Binford. **CVGIP: Image Understanding**, v.53, n.1, p.120-24, 1991.

ARCELLI, F.; DE SANTO, M.; DI SALVO, S. Software Agents for Computer Vision: a Preliminary Discussion. In: HAWAII INTERNATIONAL CONFERENCE ON SYSTEM SCIENCES, 31, Hawaii, 1998. **Proceedings**. Los Alamitos, IEEE, 1998. v. 5, p. 9-17.

BAJCSY, R. Active perception. **Proceedings of the IEEE,** v.78, n.8, p.996-1005, 1988.

BAJCSY, R.; CAMPOS, M. Active and exploratory perception. **CVGIP: Image Understanding**, v.56, n.1, p.31-40, July 1992.

BALLARD, D. H. Hierarchic Recognition of Tumor in Chest Radiographs. Basel, Birkhauser-Verlag, 1976.

BALLARD, D. Animate vision. **Artificial Intelligence,** v.48, p.57-86, 1991.

BALLARD, D. H.; BROWN, C. M. **Computer Vision.** Englewood Cliffs, Prentice-Hall, 1982.

BALLARD, D.; BROWN, C. M. Principles of animate vision. **CVGIP: Image Understanding**, v.56, n.1, p.3-21, July 1992.

BIANCHI, R. A. C.; SANTOS, M. V. T.; RILLO, M. Integração de um sistema de planejamento de atividades numa célula de montagem robotizada. In: SIMPÓSIO BRASILEIRO DE AUTOMAÇÃO INTELIGENTE, 2., Curitiba, 1995. **Anais**. Curitiba, SBA, 1995. p.195-200.

BLACK, M. J.; ALOIMONOS, J.; HORSWILL, I.; SANDINI, G.; BROWN, C.M.; MALIK, J.; TARR, M.J. Action, representation, and purpose: re-evaluating the foundations of computational vision. In: INTERNATIONAL JOINT CONFERENCE ON AI, 1993. **Proceedings.** 1993. p.1661-6.

BOISSIER, O.; DEMAZEAU, Y. A distributed artificial intelligence view on general purpose vision systems. In: DEMAZEAU, Y; WERNER, E. (eds.) **Decentralized AI-3.** Amsterdam, Elsevier, 1992. p.311-330.

BOISSIER, O; DEMAZEAU, Y. ASIC: An architecture for social and individual control and its application to Computer Vision. In: EUROPEAN WORKSHOP ON MODELLING AUTONOMOUS AGENTS IN A MULTI-AGENT WORLD, 1994. **Proceedings.** 1994. p.107-18.

BOISSIER, O; DEMAZEAU, Y. Finding Contours Using a Multi-Agent System. In: INTERNATIONAL JOINT CONFERENCE ON ARTIFICIAL INTELLIGENCE, 1994. **Proceedings.** 1994.

BOND, A. H; GASSER, L. **Readings in Distributed Artificial Intelligence**. Morgan Kaufmann, San Mateo, CA, 1988.

BOUCHER, A.; GARBAY, C. A multi-agent system to segment living cells. In: INTERNATIONAL CONFERENCE ON PATTERN RECOGNITION, 13th., 1996. **Proceedings**. Los Alamitos, IEEE, 1996. v.3, p.558-62.

BROOKS, R. A. A robust layered control system for a mobile robot. **IEEE Journal of Robotics and Automation**, VRA-2, n.1, p.14-23, March 1986.

BROOKS, R. A. Intelligence without representation. **Artificial Intelligence**, v.47, p.139-59, 1991.

BROOKS, R. A. **Intelligence without Reason**. MIT AI MEMO No.1293, 1991b.

BROOKS, R. A.; STEIN, L. A. **Building Brains for Bodies**. MIT AI MEMO No. 1439, 1993.

BROWN, C. M. Toward General Vision. **CVGIP: Image Understanding**, v.60, n.1, p.89-91, July 1994.

CHAIB-DRAA, B. Industrial Applications of Distributed AI. In: HUHNS, M. N.; SINGH, M. P. (eds.) **Readings In Agents**. São Francisco, Morgan Kaufmann, 1997.

CHENG, T. K.; KITCHEN, L.; LIU, Z.; COOPER, J. An Agent-Based Approach for Robot Vison System. Technical Report no. 95/341995, Computer Science Department, University of Melbourne, Austrália, 1995.

CHRISTENSEN, H. I.; MADSEN, C. B. Purposive Reconstruction: A Reply to "A Computational and Evolutionary Perspective on the Role of Representation in Vision" by M. J. Tarr and M. J. Black. **CVGIP: Image Understanding**, v.60, n.1, p.103-8, July 1994.

COOPER, M. **Visual Occlusion and the Interpretation of Ambiguous Pictures**. Chicester, Ellis Horwood, 1992.

DEMAZEAU, Y; MÜLLER, J. Decentralized Artificial Intelligence. In: DEMAZEAU, Y; MÜLLER, J. (eds.) **Decentralized AI 1.** Amsterdam, Elsevier, 1990. p.3-13.

DROGOUL, A. When Ants Play Chess (or Can Strategies Emerge from Tactical Behaviors?). In:CALSTELFRANCHI, C; MÜLLER, J. P. (eds.) **From reaction to cognition - Lecture Notes in AI 957**. Berlin-Heidelberg, Springer-Verlag, 1995. p. 13-27.

EDELMAN, S. Representation without reconstruction. **CVGIP: Image Understanding**, v.60, n.1, p.92-4, July 1994.

ELFES, A. A distributed control architecture for an autonomous mobile robot. **Artificial Intelligence**, Computational Mechanics Publications, v.1, n.2, 1986, p.135-44.

ERMAN, L. D. The Hearsay-II speech-understanding system: integrating knowledge to resolve uncertainty. **Computer Surveys**, v.12, n.2, 1980, p.213-53.

FERGUSON, I. A. Toward an architecture for adaptative, rational, mobile agents. **Decentralized AI 3**. Amsterdam, Elsevier, p.249-61, 1992.

FISHER, R. B. Is Computer Vision still AI? **AI Magazine**, p.21-7, Summer, 1994.

FISCHLER, M. A. The modeling and representation of visual information. **CVGIP: Image Understanding**, v.60, n.1, p.98-9, July 1994.

FIRBY, R. J. Opening the door to robotic agents. **IEEE Expert,** v.12, n.2, p.7-9, Mar. 1997.

FIRBY, R. J.; KAHN, R. E.; PROKOPOWICZ, P. N.; SWAIN, M. J. Collecting trash:

a test of Purposive Vision. Submitted to the Workshop on Vision for Robots, Pittsburg, PA, 1995.

FRANKLIN, S. **Artificial Minds.** Cambridge, MA, MIT Press, 1995.

FRANKLIN, S.; GRAESSER, A. Is it an Agent, or just a Program?: A Taxonomy for Autonomous Agents. In: INTERNATIONAL WORKSHOP ON AGENT THEORIES, ARCHITECTURES, AND LANGUAGES, 3., Heidelberg, Alemanha, 1996. **Proceedings.** Berlin, Springer Verlag, 1996.

FREIRE, E. O.; BASTOS, T. F.; DINNIKIV, V.; SHNEEBELI, H. J. Sistema Sensorial Ultra-Sônico para Robô Móvel com Controle Baseado em Agentes. In: SIMPÓSIO BRASILEIRO DE AUTOMAÇÃO INTELIGENTE, 3º. Vitória, 1997. **Anais**. Vitória, UFES-SBA, 1997. p.471-6.

GARCIA-ALEGRE, M. C.; ROCIO, F. Basic Agents for Visual/Motor Coordination of a Mobile Robot. In: INTERNATIONAL CONFERENCE ON AUTONOMOUS AGENTS, 1, Marina Del Rey, 1997. **Proceedings**. Marina Del Rey**,** ACM, 1997.

GASSER, L. Foreword. In: HUHNS, M. N.; SINGH, M. P. (eds.) **Readings In Agents**. São Francisco, Morgan Kaufmann, 1997a. p.v-vi.

GEIST, G. A.; KOHL, J. A.; PAPADOPOULOS, P. M. PVM and MPI: a Comparison of Features. **Calculeteurs Paralleles,** v. 8, n. 2, 1996.

HANKS, S.; POLLACK, M. E. ; COHEN, P. R. Benchmarks, Test Beds, Controlled Experimentation, and the Design of Agent Architectures. **AI Magazine**, v. 17, n. 42, 1993.

HAYES-ROTH, B. An Architecture for Adaptive Intelligent Systems. **Artificial Intelligence (Special Issue on Agents and Interactivity)**, 72, p.329-65, 1995.

HENDLER, J.; MUSLINER, D. **Research in the Autonomous Mobile Robotics Laboratory**, University of Maryland, College Park, Maryland, 1994.

HEWITT, C. Viewing Control Structures as Patterns of Passing Messages. **J. Artificial Intelligence**, v.8, n.3, p. 323-64, June 1977.

HEWITT, C. Offices are Open Systems. In: BOND, A. H; GASSER, L. (eds.) **Readings in Distributed Artificial Intelligence**. Morgan Kaufmann, San Mateo, CA, 1988.

HUHNS, M. N.; SINGH, M. P. Agents and Multiagent Systems: Themes, Approaches,

and Challenges. In: HUHNS, M. N.; SINGH, M. P. (eds.) **Readings In Agents**. São Francisco, Morgan Kaufmann, 1997.

JAIN, R. C. Expensive Vision. **CVGIP: Image Understanding**, v.60, n.1 p.86-8, July 1994.

JAIN, R. C.; BINFORD, T. O. Ignorance, myopia and naiveté in computer vision systems. **CVGIP: Image Understanding**, v.53, n.1, p.112-17, 1991.

JENNINGS, N. R et al. Using ARCHON to develop real-world DAI applications. **IEEE Expert**, v. 11, n, 6, p. 64-70, Dec 1996.

JOLION, J.M. Computer vision methodologies. **CVGIP: Image Understanding**, v.59, n.1, p.53-71, Jan. 1994.

KANG, T, C. TALUKDAR, S. Overview on Designing Autonomous Agents Teams. In: SIMPÓSIO BRASILEIRO DE AUTOMAÇÃO INTELIGENTE, 3º. Vitória, 1997. **Anais**. Vitória, UFES-SBA, 1997. p. 489-93.

KIROUSIS, L.; PAPADIMITRIOUS, C. The complexity of recognizing polyhedral scenes. In: ANNUAL SYMPOSIUM ON FOUNDATIONS OF COMPUTER SCIENCE, 1985. **Proceedings**. Los Alamitos, IEEE, 1985, p. 175-85.

KITCHIN, P. W.; PUGH, A. Processing of Binary Images. In: PUGH, A. (editor) **Robot Vision.** Berlin, Springer, 1983. P. 21–42.

LI, Z. N. **Multimedia Systems Class Notes**. - School of Computing Science - Simon Fraser University, 1998. (http://www.cs.sfu.ca/fas-info/cs/CC/365/li/material/notes/Chap3/Chap3.3/Big.html)

MAES, P. et al. Agents that reduce work and information overload. **Communications of the ACM**, v.37, n.7, p.31-40, July 1994.

MAES, P. Artificial Life Meets Entertainment: Lifelike Autonomous Agents. **Communications of the ACM**, v.38, n.11, p.108-14, Nov 1995.

MARR, D. **Vision**. New York, Freeman, 1982.

MATRIC, M. J. Coordination and learning in multirobot systems. **IEEE Intelligent Systems**, v.13, n.2, p.6-8, April 1998.

MEDEIROS, A. A. D.; CHATILA, R. Priorities and data abstraction in hierarchical control architectures for autonomous robots. In: WORKSHOP ON INTELLIGENT

ROBOTICS, 1º, Brasília, 1997. **Proceedings.** Brasília, 1997. p. 207-220.

NEVES, M. C.; OLIVEIRA, E. A Control Architecture for an Autonomous Mobile Robot. In: INTERNATIONAL CONFERENCE ON AUTONOMOUS AGENTS, 1, Marina Del Rey, 1997. **Proceedings**. Marina Del Rey, ACM, 1997.

NOREILS, F. R.; CHATILA, R. G. Plan execution monitoring and control architecture for mobile robots. **IEEE Transactions on Robotics and Automation**, v.11, n.2, April 1995.

PINHANEZ, C. S. Behavior-based Active Vision. **International Journal of Pattern Recognition and Artificial Intelligence**, v.8, n.6, p.1493-1526, 1994.

PRESSMAN, R. **Engenharia de Software.** São Paulo, Makron Books, 1995.

RILLO, A.H.R.C. **Sistema de Visão Binária com Reconhecimento de Peças Parcialmente Oclusas.** São Bernardo do Campo, 1989. Dissertação (Mestrado) - Faculdade de Engenharia Industrial.

RILLO, A.H.R.C., **RECTRI: um sistema de reconhecimento tridimensional a partir de uma única imagem de intensidade luminosa.** São Paulo, 1994. Tese (Doutorado) - Escola Politécnica, Universidade de São Paulo.

RILLO, A. H. R. C., BIANCHI, R. A. C., MOREIRA Jr, B., FERRAZ, F. "Integrando Visão e Comportamento: Uma aplicação de reconstrução propositiva". In:. CONGRESSO BRASILEIRO DE AUTOMÁTICA, 11., São Paulo, 1996. **Anais.** São Paulo, SBA, 1996. pp. 573-8.

RILLO, M.; RILLO, A.H.R.C.; COSTA, L.A.R. The LSI assembly cell. In: IFAC/IFIP/IFORS/IMACS/ISPE Symposium on information control problems in manufacturing technology, 7º, Toronto, 1992. **Proceedings.** IFAC, 1992. p. 361-5.

RIVLIN, E.; ALOIMONOS, Y.; ROSENFELD, A. **Purposive Recognition: a Framework.** Technical Report CAR-TR-597, CS-TR-2811, DACA76-89-C-0019, IRI-9057934, University of Maryland, Dec. 1991.

RUSSELL, S. J.; NORVIG, P. **Artificial Intelligence: A Modern Approach.** Englewood Cliffs, Prentice Hall, 1995.

SANDHOLM, T.; LESSER, V. Issues in automated negotiation and eletronic commerce: extending the contract net framework. In: INTERNATIONAL

CONFERENCE IN MULTIAGENTS SYSTEMS, 1995, **Proceedings,** 1995. p. 328-335.

SANDINI, G.; GROSSO, E. Why purposive vision? **CVGIP: Image Understanding**, v.60, n.1 p.108-12, July 1994.

SHNEEBELI, H. A. **Die Steuerung von Mehrfinger-Greifersystemen**. Karlsruhe, Alemanha, 1992. Tese (Doutorado) - Universidade de Karlsruhe.

SICHMAN, J. S.; DEMAZEAU, Y; BOISSIER, O. When can knowledge-based systems be called agents? In: SIMPÓSIO BRASILEIRO DE INTELIGÊNCIA ARTIFICIAL, 9., Rio de Janeiro, 1992. **Anais.** Rio de Janeiro, SBC, 1992. p.172-85.

SMITH, R. G. The Contract Net Protocol: high-level communication and control in a distributed problem solver. **IEEE Transactions on Computers**, v.29, n.12, p.1104-13, Dec 1980.

SMITH, D. C.; CYPHER, A.; SPOHRER, J. KidSim: Programming Agents Without a Programming Language. **Communications of the ACM**, v.37, n.7, p.55-67, 1994.

SPINU, C.; GARBAY, C.; CHASSERY, J. M. A multi-agent approach to edge detection as a distributed optimization problem. In: INTERNATIONAL CONFERENCE ON PATTERN RECOGNITION, 13th.,1996. **Proceedings**. Los Alamitos, IEEE, 1996. v.2, p. 81-5.

STONE, P.; VELOSO, M. Multiagent Systems: A Survey from a Machine Learning Perspective. CMU Internal Report, February, 1997.

Sun Microsystems Computer Corporation **Sun Video User's Guide**. Revision A, August, 1994.

SWAIN, M.J.; STRICKER, M. **Promising Directions in Active Vision.** Technical Report CS 91-27, University of Chicago, Nov. 1991.

TALUKDAR, S. Designing Super-Agents. In: WORKSHOP ON COMPUTATIONAL ORGANIZATION DESIGN. Stanford, AAAI Spring Symposium Series, 1994.

TALUKDAR, S. N.; SOUZA, P. S. Scale Efficient Organizations. In: 1992 IEEE INTERNATIONAL CONFERENCE ON SYSTEMS, MAN AND CYBERNETIC, Chicago, 1992. **Proceedings.** Los Alamitos, IEEE, 1992.

TALUKDAR, S.; SOUSA, P.; MURTHY, S. Organization for computer-based agents. **Eng. Int. Syst.**, v.2, p. 75-87, 1993.

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.

TARR, M. J.; BLACK, M. J. Reconstruction and purpose**.** **CVGIP: Image Understanding**, v.60, n.1, p.113-8, July 1994b.

TRIVEDI, M.M.; ROSENFELD, A. On making computers "See". **IEEE transactions on systems, man and cybernetics,** v.19, n.6, p.1333-6, Nov. 1989.

TSOTSOS, J. K. Complexity of Perceptual search tasks. In: INTERNATIONAL JOINT CONFERENCE ON ARTIFICIAL INTELLIGENCE, 11, Detroit, 1989. **Proceedings**. AAAI, 1989. 1571-7.

TSOTSOS, J. K. A Complexity Level Analysis of Vision. **Behaviour and Brain Sci**, v.13, p.423-55, 1990.

TSOTSOS, J. K. There is no one way to look at vision. **CVGIP: Image Understanding**, v.60, n.1, p.95-7, July 1994.

VAN DE VELDE, W. Toward learning robots. **Robotics and Autonomous Systems**, v.8, n.1, pp 1-6, Nov. 1991.

VELOSO, M.; STONE, P; HAN, K. The CMUnited-97 Robotic Soccer Team: Perception and Multiagent Control. In: INTERNATIONAL CONFERENCE ON AUTONOMOUS AGENTS, 2, Minneapolis, 1998. **Proceedings.** AAAI, 1998.

WALTZ, D. I. "*Generating semantic descriptions from drawings of scenes with shadows*". In **The Psicology of Computer Vision**. New York, McGraw-Hill, 1975.

WERNER, E. The design of Multi-Agent Systems. In: DEMAZEAU, Y; WERNER, E. (eds.) **Decentralized AI-3.** Amsterdam, Elsevier, 1992. p.3-28.

WOOLDRIDGE, M.; JENNINGS, N. R. Agent Theories, Architectures, and Languages: a Survey. In: WOOLDRIDGE, M.; JENNINGS, N. R. (eds.) **Intelligent Agents.** Berlin, Springer-Verlag, 1995. p.1-22.

XAVIER, J. E. M. **Uma Estrutura para a Construção de Sistemas de Controle Baseados em Agentes para Robôs Móveis**. Vitória, 1996. Dissertação (Mestrado)

- Universidade Federal do Espírito Santo.

APÊNDICE A: TESTE DO SISTEMA IMPLEMENTADO E EXEMPLOS DE FUNCIONAMENTO

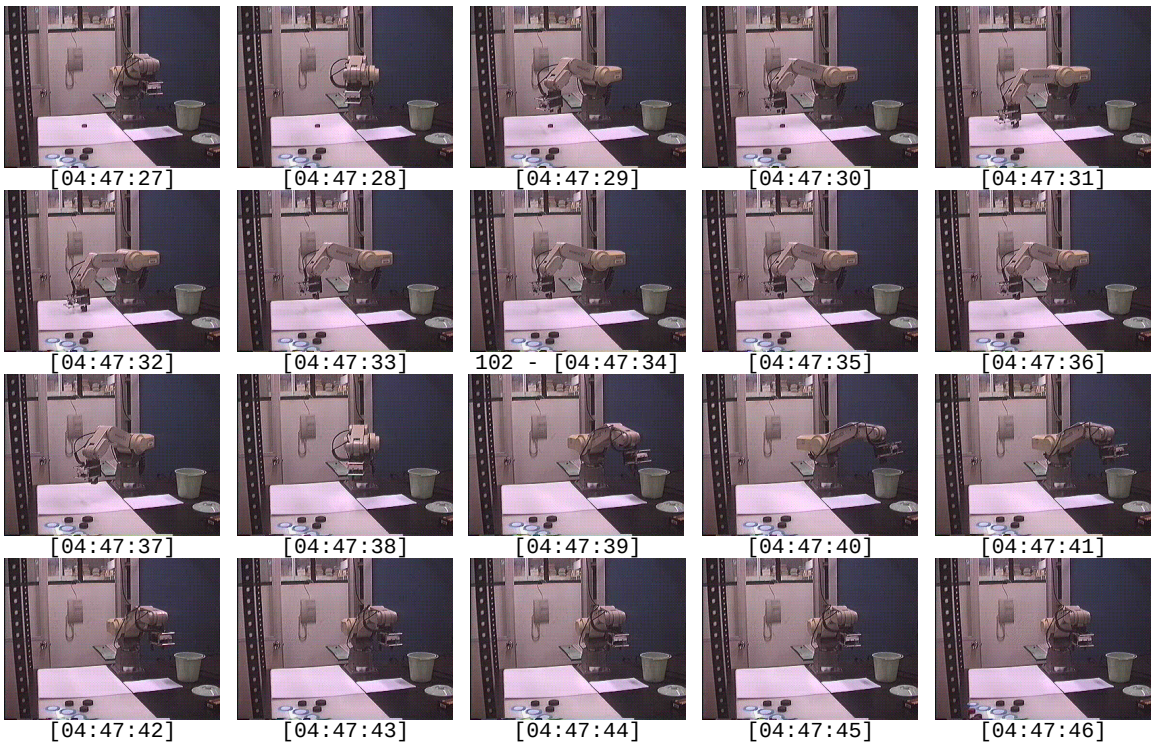
Como dito anteriormente, foram realizados dezenas de testes com o sistema implementado, que levaram desde modificações no modelo do agente até melhorias no sistema implementado. Nos experimentos foi verificado:

- O gerenciamento dos recursos pelos agentes (em simulações e no sistema implementado na Célula);
- A transferência de recursos e o funcionamento das regras da sociedade e da estrutura de autoridade (em simulações e no sistema implementado na Célula);
- o funcionamento de cada agente individualmente, com a realização de diversas montagens e desobstruções;
- A interação entre os 3 agentes, com a interrupção de uma montagem com devido à necessidade de evitar uma colisão e à necessidade de desobstruir a área de trabalho;
- A reação a obstrução de peças;
- A reação ao aparecimento de uma peça em uma posição que o manipulador não alcança;

São apresentados a seguir alguns exemplos do funcionamento do sistema implementado na Célula de Montagens.

UMA DESOBSTRUÇÃO:

A figura abaixo apresenta o sistema realizando uma desobstrução, apenas retirando uma peça da área de trabalho.



A seguir se encontra o arquivo das mensagens de comunicação entre os agentes durante a realização da limpeza, com comentários em itálico.

```
•  requisita o recurso
[cleaner-execution   ][04:47:25] LOG: Resource Manipulator Needed
[cleaner             ][04:47:25] LOG: MSG Received: Sender: cleaner-
execution, Message: (request (manipulator)).
[motherAgent         ][04:47:25] LOG: MSG Received: Sender: cleaner,
Message: (request (cleaner) (manipulator)).
[motherAgent         ][04:47:25] (transfer (cleaner) (manipulator) (free))
[collisionAvoider    ][04:47:25] LOG: MSG Received: Sender: motherAgent,
Message: (transfer (cleaner) (manipulator) (free)).
[assembler           ][04:47:25] LOG: MSG Received: Sender: motherAgent,
Message: (transfer (cleaner) (manipulator) (free)).
[cleaner             ][04:47:25] LOG: MSG Received: Sender: motherAgent,
Message: (transfer (cleaner) (manipulator) (free)).
[cleaner-blackboard  ][04:47:25] LOG: MSG Received: Sender: cleaner,
Message: (updateResource (manipulator) (free)).
[cleaner-execution   ][04:47:26] LOG: Resource Manipulator Received
[cleaner-blackboard  ][04:47:26] LOG: MSG Received: Sender: cleaner-
execution, Message: (updateResource (manipulator) (free)).
•  vai pegar o lixo
[cleaner             ][04:47:27] LOG: MSG Received: Sender: cleaner-
execution, Message: (disableInterruption (manipulator)).
[cleaner-execution   ][04:47:27] LOG: (acknowledge)
[manipulator-primitiv][04:47:27] LOG: Manipulator Received: (pickup-object
(trash) at (131, 122)).
[cleaner-blackboard  ][04:47:33] LOG: MSG Received: Sender: cleaner-
execution, Message: (updateResource (manipulator) (object-held trash)).
[cleaner             ][04:47:33] LOG: MSG Received: Sender: cleaner-
```

```
execution, Message: (enableInterruption (manipulator)).
[cleaner-execution   ][04:47:33] LOG: (acknowledge)
```

- *leva o lixo para lixeira*

```
[cleaner             ][04:47:33] LOG: MSG Received: Sender: cleaner-
execution, Message: (disableInterruption (manipulator)).
[cleaner-execution   ][04:47:33] LOG: (acknowledge)
[manipulator-primitiv][04:47:34] LOG: Manipulator Received: (move-object
(trash) to (trashcan-position)).
[cleaner-blackboard  ][04:47:34] LOG: MSG Received: Sender: cleaner-
execution, Message: (updateResource (manipulator) (object-held trash)).
[cleaner             ][04:47:35] LOG: MSG Received: Sender: cleaner-
execution, Message: (enableInterruption (manipulator)).
[cleaner-execution   ][04:47:35] LOG: (acknowledge)
```

- *mandou largar o lixo.*

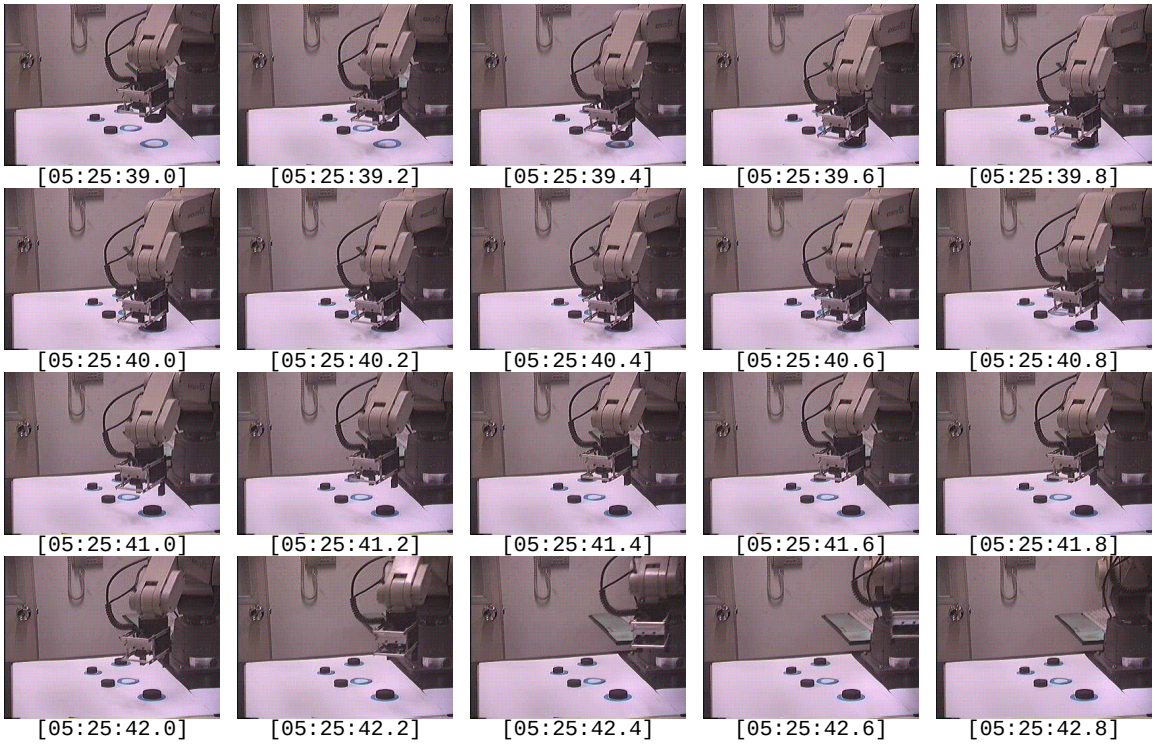
```
[cleaner             ][04:47:35] LOG: MSG Received: Sender: cleaner-
execution, Message: (disableInterruption (manipulator)).
[cleaner-execution   ][04:47:35] LOG: (acknowledge)
[manipulator-primitiv][04:47:35] LOG: Manipulator Received: (drop-held-
object).
[cleaner-blackboard  ][04:47:39] LOG: MSG Received: Sender: cleaner-
execution, Message: (updateResource (manipulator) (free)).
[cleaner             ][04:47:39] LOG: MSG Received: Sender: cleaner-
execution, Message: (enableInterruption (manipulator)).
[cleaner-execution   ][04:47:39] LOG: (acknowledge)
[cleaner             ][04:47:39] LOG: MSG Received: Sender: cleaner-
execution, Message: (free (manipulator)).
```

- *liberou o manipulador.*

UMA MONTAGEM:

A figura abaixo apresenta uma montagem vista de perto, com a sobreposição de uma peça sobre um encaixe.

segue →



A seguir se encontra o arquivo das mensagens de comunicação entre os agentes durante a realização da montagem, com comentários em itálico.

```
Agent Comunication Log Archive
Started at host nausika at Sun Jun  7 05:24:26 1998
[ AGENT NAME ][TIME] and Message

. . .

[assembler-execution ][05:25:27] LOG: Resource Manipulator Needed
[assembler-execution ][05:25:27] LOG: I'll request it.
[assembler           ][05:25:27] LOG: MSG Received: Sender: assembler-execution,
Message: (request (manipulator)).
[cleaner             ][05:25:27] LOG: MSG Received: Sender: assembler, Message:
(transfer (assembler) (manipulator) (free)).
[assembler           ][05:25:27] (transfer (assembler) (manipulator) (free))
[assembler-execution ][05:25:27] LOG: Resource Manipulator Received
[motherAgent         ][05:25:27] LOG: MSG Received: Sender: assembler, Message:
(transfer (assembler) (manipulator) (free)).
[collisionAvoider    ][05:25:27] LOG: MSG Received: Sender: assembler, Message:
(transfer (assembler) (manipulator) (free)).
[assembler-blackboard][05:25:27] LOG: MSG Received: Sender: assembler-execution,
Message: (updateResource (manipulator) (free)).
[assembler-execution ][05:25:31] LOG: (pickup-object (tenon) at (90, 93)) started
[assembler           ][05:25:31] LOG: MSG Received: Sender: assembler-execution,
Message: (disableInterruption (manipulator)).
[assembler-execution ][05:25:31] LOG: (acknowledge)

•   manda pegar a peça

[manipulator-primitiv][05:25:31] LOG: Manipulator Received: (pickup-object (tenon) at
(90, 93)).
[assembler-blackboard][05:25:37] LOG: MSG Received: Sender: assembler-execution,
Message: (updateResource (manipulator) (object-held tenon at position 90 93)).
[assembler           ][05:25:37] LOG: MSG Received: Sender: assembler-execution,
Message: (enableInterruption (manipulator)).
[assembler-execution ][05:25:37] LOG: (acknowledge)
 [assembler           ][05:25:37] LOG: MSG Received: Sender: assembler-execution,
Message: (disableInterruption (manipulator)).
[assembler-execution ][05:25:37] LOG: (acknowledge)

•   mandou mover
```

```
[manipulator-primitiv][05:25:37] LOG: Manipulator Received: (move-object (tenon) to
(246, 76)).
[assembler-blackboard][05:25:39] LOG: MSG Received: Sender: assembler-execution,
Message: (updateResource (manipulator) (object-held tenon at position 246 76)).
[assembler           ][05:25:39] LOG: MSG Received: Sender: assembler-execution,
Message: (enableInterruption (manipulator)).
[assembler-execution ][05:25:39] LOG: (acknowledge)
[assembler-execution ][05:25:39] LOG: (move-object (tenon) to (246, 76)) finished
[assembler-execution ][05:25:39] LOG: (join-pair tenon mortise) started
[assembler           ][05:25:39] LOG: MSG Received: Sender: assembler-execution,
Message: (disableInterruption (manipulator)).
[assembler-execution ][05:25:39] LOG: (acknowledge)
```

- *mandou unir a peça e o encaixe*

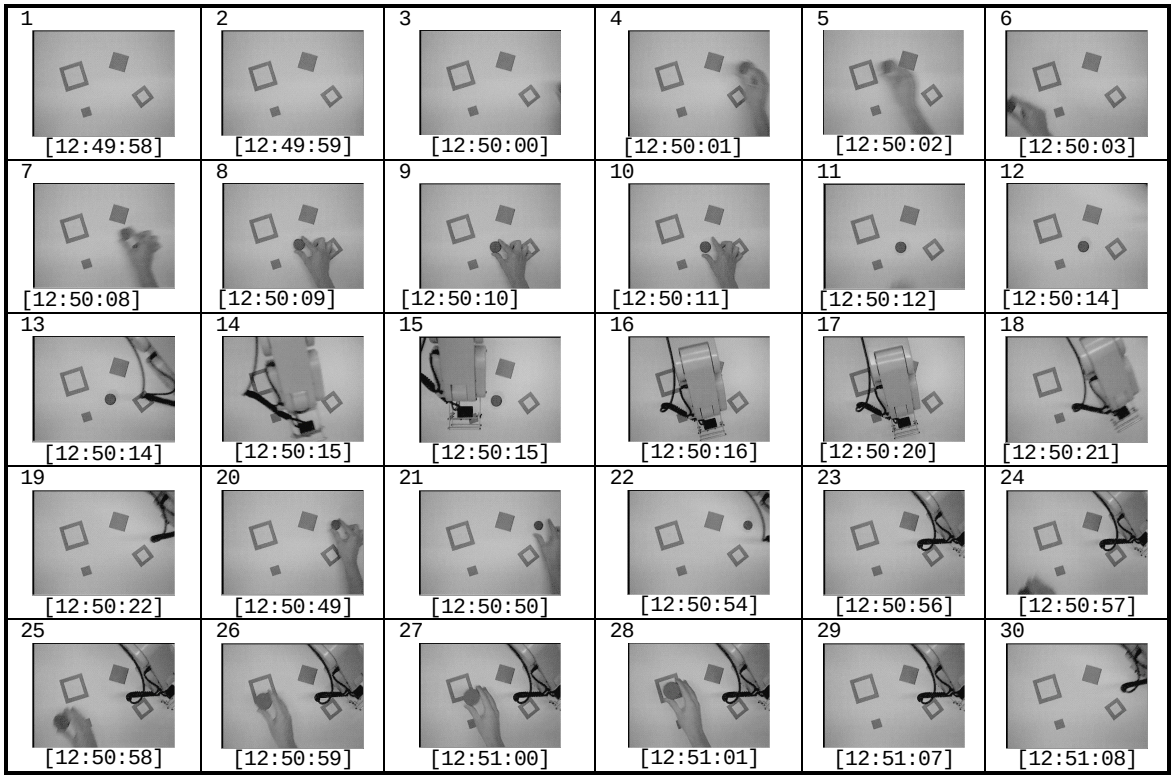
```
[manipulator-primitiv][05:25:39] LOG: Manipulator Received: (join-pair (tenon) (mortise)
at (246, 76)).
[assembler-blackboard][05:25:42] LOG: MSG Received: Sender: assembler-execution,
Message: (updateResource (manipulator) (free)).
[assembler           ][05:25:42] LOG: MSG Received: Sender: assembler-execution,
Message: (enableInterruption (manipulator)).
[assembler-execution ][05:25:42] LOG: (acknowledge)
[assembler-execution ][05:25:42] LOG: (join-pair tenon mortise) finished
[assembler           ][05:25:42] LOG: MSG Received: Sender: assembler-execution,
Message: (disableInterruption (manipulator)).
[assembler-execution ][05:25:42] LOG: (acknowledge)
```

- *mandou para a posição inicial*

```
[manipulator-primitiv][05:25:42] LOG: Manipulator Received: (move-object (null) to
(home)).
[assembler-blackboard][05:25:43] LOG: MSG Received: Sender: assembler-execution,
Message: (updateResource (manipulator) (free)).
[assembler           ][05:25:44] LOG: MSG Received: Sender: assembler-execution,
Message: (enableInterruption (manipulator)).
[assembler-execution ][05:25:44] LOG: (acknowledge)
[assembler-execution ][05:25:44] LOG: (scan-for-static-object object) started
[assembler           ][05:25:44] LOG: MSG Received: Sender: assembler-execution,
Message: (free (manipulator)).
[assembler           ][05:25:44] (free (manipulator))
[cleaner             ][05:25:44] LOG: MSG Received: Sender: assembler, Message: (free
(manipulator)).
[motherAgent         ][05:25:44] LOG: MSG Received: Sender: assembler, Message: (free
(manipulator)).
[collisionAvoider    ][05:25:44] LOG: MSG Received: Sender: assembler, Message: (free
(manipulator)).
```

UMA DESOBSTRUÇÃO VISTA DE CIMA

A seqüência de imagens abaixo mostra dois objetos não desejados sendo colocados e removidos da área de trabalho, do ponto de vista da câmera usada pelo sistema.



A seguir se encontra o arquivo das mensagens de comunicação entre os agentes durante a realização da desobstrução.

```
Agent Communication Log ArchiveStarted 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))
```

LISTA DE TESTES REALIZADOS

São apresentados a lista de testes utilizada para a validação experimental do sistema implementado na célula de montagens, com os 3 agentes autônomos funcionando ao mesmo tempo. Todos os testes resultaram como esperado.

COLLISION AVOIDER

a) Sem nenhum objeto sobre a área de trabalho: para verificar funcionamento do sistema. Resultado esperado: o collisionAvoider não deve pedir o recurso;

b) Com lixo vermelho passando pela área de trabalho sem parar. 10 passagens de 5 segundos, iniciadas de 10 em 10 segundos, a uma velocidade constante, aproximadamente 0.5 metro/segundo. Resultado esperado: o agente deve requisitar e liberar o recurso manipulador. Pode-se calcular o tempo de reação médio com este teste;

c) Com o lixo vermelho passando pela área de trabalho e parando em um lugar fora da área que manipulador alcança, o que evita que o cleaner requisite o recurso. 10 movimentações de 5 segundos, com parada de 5 segundos, iniciadas de 10 em 10 segundos e a uma velocidade constante, aprox. 0.5 metro/segundo. Resultado esperado: o agente deve requisitar e liberar o recurso manipulador, sem o cleaner remover o lixo. (NOTA: igual item c do cleaner)

d) Iniciar o sistema com o lixo parada na área de trabalho. Resultado esperado: o agente não deve requisitar o recurso;

e) Mover lentamente o lixo. Resultado esperado: Para velocidades muito baixas, não deve haver requisição do manipulador;

f) Teste de oclusão. Retirar uma peça com o manipulador. Resultado esperado: não deve pedir o recurso. (NOTA: igual teste b do cleaner)

CLEANER

a) Sem nenhum objeto sobre a área de trabalho: para verificar funcionamento do sistema. Resultado esperado: o cleaner não deve pedir o recurso; (nota: igual teste 1 do collisionAvoider)

b) Colocar o lixo vermelho na área de trabalho em um lugar que o manipulador alcança. 10 colocações, iniciadas de 10 em 10 segundos, com o lixo andando durante 5 segundos antes de parar e a uma velocidade constante, aprox. 0.5 metro/segundo. Resultado esperado: o agente collisionAvoider deve requisitar e liberar o recurso manipulador, sendo que depois o cleaner deve requisitar o recurso, retirar o lixo e liberar o recurso. Pode-se calcular o tempo de reação médio com este teste, para o cleaner e o collisionAvoider, e ainda testa a interação entre os dois. NOTA: ao receber o recurso o cleaner não deve ser interrompido até o final;

c) Com o lixo vermelho passando pela área de trabalho e parando nela em um lugar fora da

área alcançável pelo manipulador, o que evita que o cleaner requisite o recurso. 10 movimentações de 5 segundos, com parada de 5 segundos, iniciadas de 10 em 10 segundos e a uma velocidade constante e normal, aprox. 0.5 metro/segundo. Resultado esperado: o agente collisionAvoider deve requisitar e liberar o recurso manipulador, sem o cleaner remover o lixo (NOTA: igual item c do collisionAvoider).

d) Iniciar o sistema com o lixo parada na área de trabalho. Resultado esperado: o agente deve requisitar o recurso e retirar o lixo; (NOTA: igual item d do collisionAvoider).

e) Mover lentamente o lixo. Resultado esperado: Para velocidades muito baixas, não deve haver requisição do manipulador; (NOTA: igual item e do collisionAvoider).

f) Lixo se movendo e parando de 5 em 5 segundos, andando de uma posição que o manipulador não alcançe para uma que ele alcance. Fazer 2 testes em direção ao manipulador e 2 testes em direção contrária. Resultado esperado: ao entrar na área que o manipulador alcança o lixo deve ser removido.

g) Colocar uma peça dentro em uma peça fora da área alcançável. Resultado esperado: ele deve remover a peça que o manipulador alcança;

h) Colocar dois lixos na área de trabalho. Resultado Esperado: remoção dos 2 objetos;

i) colocar 3 lixos na área de trabalho. Resultado Esperado: remoção dos 3 objetos;

j) Teste de interrupção pelo collisionAvoider. Colocar o lixo vermelho na área de trabalho em um lugar alcançável. 10 colocações, iniciadas de 10 em 10 segundos, com o lixo andando durante 5 segundos antes de parar e a uma velocidade constante e normal, aprox. 0.5 metro/segundo. Após parar, ao ser iniciada a retirada, entrar outra peça vermelha, que ficara andando por 5 segundos. Resultado esperado: o agente collisionAvoider deve requisitar e liberar o recurso manipulador, sendo que depois o cleaner deve pegar o recurso. Ao tentar retirar o lixo, ele é interrompido pelo collisionAvoider e após 5 segundos termina a tarefa e libera o recurso. Pode-se calcular o tempo de reação médio com este teste, para o cleaner e o collisionAvoider, e ainda testa a interação entre os 2;

ASSEMBLER

Sem objetos na área de trabalho:

a) para verificar funcionamento do sistema. Resultado esperado: o assembler não deve pedir o recurso; (nota: igual teste 1 do collisionAvoider)

Com uma peça:

a) Colocar 1 peça sem encaixe: 10 vezes, peças diferentes. Resultado esperado: o assembler não deve pedir o recurso;

b) colocar 1 encaixe sem peça; 10 vezes. Resultado esperado: idem ao anterior;

c) Colocar 1 peça e 1 encaixe na área de trabalho em uma posição que o manipulador alcançe e deixar ele montar. Ao terminar, esperar 10 segundos. Então, mover a peça e o encaixe

para outra posição. NOTA: ao mover, sempre entrar com uma peça vermelha junto com a mão. Fazer 20 vezes, com peças e encaixes diferentes, sempre os pares. Resultado esperado: ao reconhecer uma peça e um encaixe, ele deve montá-lo. Ao terminar, ele não deve mover a peça encaixada. Testei ainda interação com collisionAvoider e descobri tempo de reação;

d) 10 testes semelhantes ao acima, com 1 peça e 1 encaixe diferente.

e) com 1 peça e 2 encaixes iguais, sendo eles o encaixe certo.

f) com 1 peça e 2 encaixes diferentes, um certo em 1 errado. Mudar a peça e os encaixes.

g) 1 peça e todos os encaixes;

Alcançabilidade. Testa a colocação de peças em uma posição que a câmera a encontra, mas que o manipulador não alcança:

a) 1 peça e um encaixe certo fora da área alcançável. 5 vezes;

b) 1 peça na área alcançável e um encaixe certo fora dela. 5 vezes;

c) 1 peça certa fora da área alcançável e um encaixe dentro dela;

Com 2 peças. 5 testes para cada:

a) e 0 encaixes;

b) e 1 encaixe certo para uma delas;

c) e 2 encaixes, os certos;

d) e 2 encaixes, um certo e um errado;

e) vários encaixes, 2 certos e outros errados;

Com 3 peças. 1 testes para cada:

a) e 0 encaixes;

b) e 1 encaixe certo para uma delas;

c) e 2 encaixes, os certos;

d) e 3 encaixes, os certos;

e) e 3 encaixes, um certo e dois errados;

f) e 3 encaixes, 2 destos e 1 errado;

g) vários encaixes, 3 certos e outros errados;

Nota: com estes experimentos a montagem foi testada, verificando se o sistema está reconhecendo as peças e os encaixes certos e se está deixando os errados no lugar.

Com 4 peças e 4 encaixes: 5 testes para a montagem completa.

Com todas as peças e todos os encaixes: 5 testes para a montagem completa.

Interação com o collisionAvoider:

a) Colocar 1 peça e 1 encaixe na área de trabalho alcançável e deixar ele montar. Ao estar sendo feita a montagem, interrompê-la por 5 segundos com peça vermelha andando pela área de trabalho. Ao terminar, esperar 10 segundos. Então, mover a peça e o encaixe para outra posição, para a realização de outra montagem. Fazer 10 vezes, com peças e encaixes diferentes, sempre os pares. Resultado esperado: ao reconhecer uma peça e um encaixe, o assembler deve iniciar uma montagem. Ao ser interrompido, deve parar e voltar ao trabalho. Ao terminar, ele não deve mover a peça encaixada. Testei ainda interação com collisionAvoider e descobri tempo de reação.

Interação com o cleaner:

a) Colocar 1 peça e 1 encaixe na área de trabalho alcançável e deixar o assembler montar. Interromper no meio a montagem colocando uma peça vermelha na área de trabalho. Anotar horário de entrada da peça e de parada de movimentação. Durante a montagem, interrompê-la por alguns segundos com peça vermelha andando pela área de trabalho, e no final deixar a peça na área alcançável. Ao terminar, esperar 10 segundos. Então, mover a peça e o encaixe para outra posição, com outra montagem. NOTA: ao mover, sempre entrar com uma peça vermelha junto com a mão. Fazer 10 vezes, com peças e encaixes diferentes, sempre os pares. Resultado esperado: ao reconhecer uma peça e um encaixe, ele deve iniciar uma montagem. No momento que o lixo entrar, o assembler deve ser interrompido, com a retirada do lixo pelo cleaner. Ao final o assembler deve voltar ao trabalho. Ao terminar, ele não deve mover a peça encaixada. Testei ainda interação com collisionAvoider e cleaner e descobri tempo de reação;
b) 1 peça. 1 encaixe e 2 lixos; Deixar os 2 ao mesmo tempo na mesa
c) 1 peça 1 encaixe e 2 lixos, colocar 1, ao estar retirando o lixo colocar outro.