9,0 (nove)

Departamento de Engenharia Mecatrônica
e de Sistemas Mecânicos
Escola Politécnica
Universidade de São Paulo

Marco Poli

Visão Computacional Aplicada ao Futebol de Robôs

Orientador:

Prof. Dr. Marcos Barretto

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

São Paulo

2001

Serviço de Bibliotecas
Biblioteca de Engenharia Mecânica, Naval e Oceânica

TF-2001
P758V

Visão computacional
Sistemas de computação
Robôs

ME cad.

1255025

DEDALUS - Acervo - EPMN

31600007569

A meu pai.

Agradecimentos

Gostaria de agradecer aqui a todas aquelas pessoas sem as quais este trabalho nunca teria sido possível. Em especial, ao Professor Doutor Marcos Barretto, por sua dedicação, paciência e preciosos conselhos; ao Professor Doutor Lucas Moscato pelo apoio e confiança; ao amigo Thiago de Castro Martins, pelas críticas sempre bem-vindas e inestimável ajuda; à minha família pelo apoio e suporte nas horas difíceis e à Santa Ifigênia, por todo o indispensável material fornecido.

Resumo

O presente trabalho tem por objetivo desenvolver um sistema que identifique os componentes de um jogo de futebol de robôs da modalidade "MiroSot" da FIRA. Os componentes a serem identificados são os três robôs jogadores do time e a bola. Para isso foram utilizadas técnicas de reconhecimento de componentes em imagens, ressaltando-os e, posteriormente, identificando-os. Técnicas como *gradiente*, *treshold* e *chain-code* foram descritas e utilizadas para localizar os componentes desejadas, e, posteriormente, para que se pudesse determinar o ângulo dos robôs, foi desenvolvido um sistema composto por marcações nos robôs e um algoritmo para identificá-las.

Como é sabido, sistemas que trabalham diretamente com imagens requerem grande processamento computacional. Por esse motivo, o presente trabalho será desenvolvido de forma a executar as operações sobre as imagens utilizando processamento pararelo, com o padrão *MPI*, um método que permite sua utilização em um sistema tipo *cluster* ou em uma máquina isolada, dispondo de múltiplos processadores (SMP).

Abstract

The objective of this work is to develop a system that identifies the components of a robot soccer match, FIRA's "MiroSot" style. The objects for identification are the team's three robot-players and the ball. Computational vision and image processing techniques were applied for object finding and identification. *Gradient Operator*, *Treshold* and *Chain Code* techniques were described and used for finding the desired objects and then, a new system of robot-marks and algorithm was proposed and developed for providing the angles of the robots in the "field".

As known, systems that work directly with images require lots of computational processing. For this reason, this work is developed using paralel-processing techniques, with the *MPI* standard, a method that allows the implemented code to be executed either on a computer *cluster* or on an isolated machine, with multiple processors (SMP).

Sumário

Lista de Figuras

Capítulo 1

Introdução

1.1 Motivação

O Departamento de Engenharia Mecatrônica da Escola Politécnica da USP está, no presente, engajado em desenvolver um protótipo de um sistema de robôs que, sem intervenção externa, realizem uma "Partida de Futebol", em um campo especialmente preparado, onde a detecção da posição dos componentes do sistema deve ser feita utilizando-se um sistema de detecção automatizado de posição. A modalidade em questão chama-se "MiroSot", e é promovida pela FIRA (International Federation of Robot-soccer Associations), como pode ser visto em [7].

O "time" é composto por três "jogadores", robôs autônomos que possuem um receptor de ondas de rádio, motores posicionados de forma a permitirem

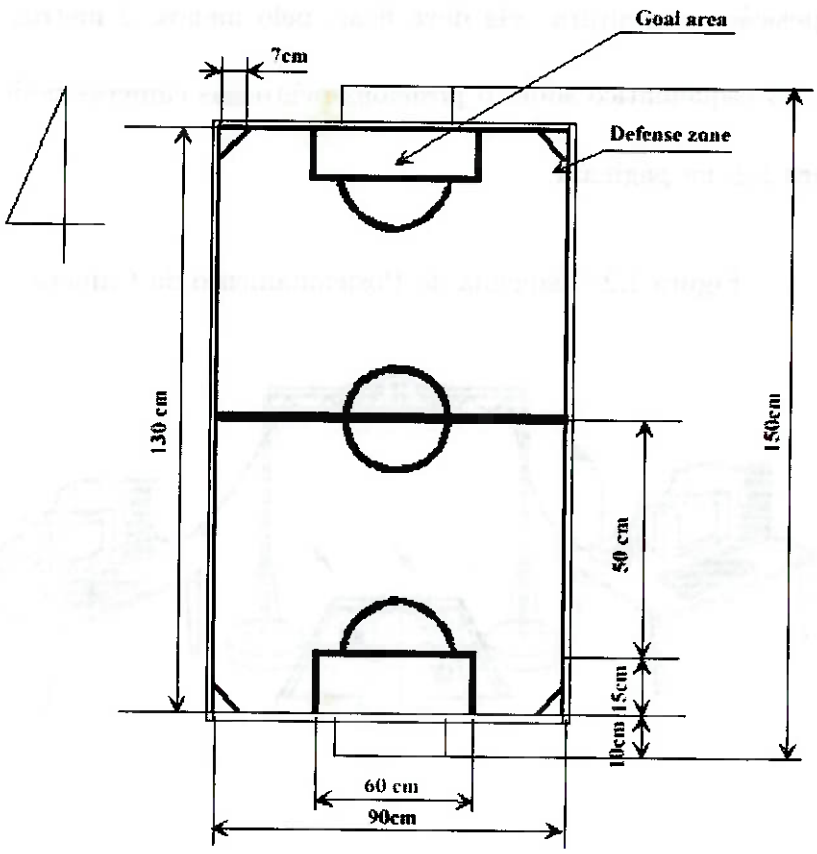
fazer curvas, um sistema de alimentação e marcações na superfície para que se possa identificar cada um através de uma câmera posicionada sobre o campo. As dimensões externas máximas dos robôs são 75x75x75 mm . Cada robô pode, também, conter uma antena, a qual não é considerada no tamanho máximo. A bola é uma bola de golf, na cor laranja, com massa de 46g, e 42.7mm de diâmetro. A câmera deve estar posicionada a uma altura de, pelo menos, 2 metros do campo de futebol. A iluminação deve ser de aproximadamente 1000 lux, e a disposição das luzes deve ser tal que impeça o aparecimento de sombras sobre a mesa.

O campo deve ser de cor verde escuro, de madeira, 1300x900 mm com paredes laterais de 50 mm de altura, na cor branca. A textura da superfície deve ser aquela de uma mesa de ping-pong.

Existem, nessa mesa, certas marcações: as linhas de meio de campo e do gol são linhas com 3 mm de espessura; sobre toda a superfície do campo, existem linhas escuras de 1 mm de espessura a cada 100 mm; círculo central tem um raio de 100 mm; um arco de 200 mm por 5 mm na linha do gol deve ser utilizado para o chute a gol; os quatro cantos da mesa são "cortados" por 70 mm, para evitar que a bola fique presa em algum deles. A figura 1.1 mostra um esquema da mesa.

Dos robôs, dois se movimentam livremente pelo campo, e um pode fa-

Figura 1.1: Esquema do Campo

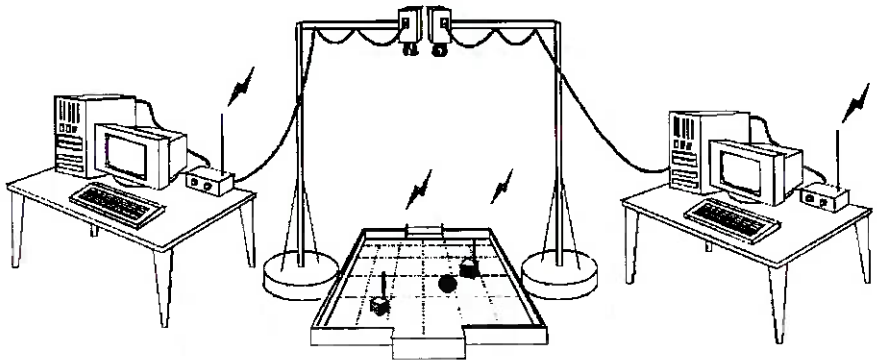


zer o papel de "goleiro", o que significa que ele, a critério da equipe, pode posicionar-se dentro da área do gol, e pode, também, segurar a bola, não podendo, no entanto, encobrir mais de 30% de sua área, como vista por cima.

A câmera a ser utilizada para a obtenção da posição da bola e da posição e da direção dos robôs é de livre escolha dos participantes. As restrições são a posição em relação ao campo—ela deve ficar próxima ao centro do

campo, com variações apenas para permitir a instalação das duas câmeras nessa posição—e a altura—ela deve ficar, pelo menos, 2 metros acima do campo. O esquemático sobre o posicionamento das câmeras pode ser visto na figura 1.2, na página 4.

Figura 1.2: Esquema do Posicionamento da Câmera



1.2 Objetivo

No presente trabalho, foi desenvolvida e implementada uma solução para a identificação de cada um dos objetos na mesa, bem como a obtenção da sua posição e, no caso dos robôs, do seu ângulo.

Esse objetivo será atingido utilizando-se um sistema de processamento de imagem, cujos componentes e métodos serão descritos no decorrer do texto.

Devido à grande quantidade de processamento utilizada por sistemas que trabalham com imagens e a velocidade de processamento requerida pelo pre-

sente projeto, foi decidido que ele seria implementado utilizando-se processamento paralelo. Para isso, diversas opções de implementação foram estudadas.

Os resultados desse trabalho serão, então, fornecidos ao sistema de controle, que está atualmente sendo desenvolvido e implementado em trabalhos paralelos, cujos resultados também serão apresentados a esta Escola.

Capítulo 2

Processamento Paralelo

2.1 Introdução

Processamento paralelo refere-se ao conceito de aumentar a velocidade de execução de um programa dividindo o programa em múltiplos fragmentos que podem ser executados simultaneamente, cada um em seu próprio processador. Um programa executado em n processadores pode ser executado n vezes mais rápido do que se tivesse usando apenas um processador.

Tradicionalmente, múltiplos processadores eram disponibilizados em um "computador paralelo" especialmente desenhado. Nessa linha de computadores, são os mais comuns os sistemas *SMP*, sistemas em que vários processadores dividem uma única memória e interface de *bus*, dentro de um mesmo processador. Também é possível a um grupo de computadores (como, por

exemplo, um grupo de PCs) serem interconectados de maneira a formarem um *cluster* de processamento paralelo . Uma terceira forma razoavelmente comum de computação paralela é o uso das *Multimedia Instruction Extensions*, ou *MMX*, para operarem em paralelo em vetores de dados inteiros.

2.2 Por que utilizar processamento paralelo?

Mesmo que o uso de múltiplos processadores possa acelerar a execução de diversas operações, muitos aplicativos não podem beneficiar-se disso. De uma maneira geral, o processamento paralelo pode ser utilizado mais apropriadamente se:

- A aplicação possui paralelismo suficiente para fazer bom uso de múltiplos processadores. Em parte, essa é uma questão de identificar partes do programa que possam ser executadas independentemente e simultaneamente em diferentes processadores, mas deve-se levar também em consideração que algumas tarefas que *podem* se executadas em paralelo podem, na verdade, ter sua velocidade reduzida, se executadas em pararelo em um sistema particular;

- o programa em questão já possui uma versão paralelizada, ou o programa em questão possa ser re-escrito para essa finalidade, ou o projeto

de um novo programa que inclua processamento paralelo.

2.3 Terminologia

Mesmo que o processamento paralelo tenha sido usado em diversos sistemas e por muito tempo, a terminologia ainda é razoavelmente desconhecida para muitos usuários. Alguns dos principais termos são descritos abaixo:

SIMD: (Single Instruction stream, Multiple Data stream) refere-se a um modelo de execução paralelo no qual todos os processadores executam a mesma operação ao mesmo tempo, mas cada processador possui seu próprio conjunto de dados para operar sobre. Esse modelo obviamente se encaixa no conceito de executar a mesma operação em cada elemento de um conjunto de dados, e é, então freqüentemente associado a manipulação de vetores ou conjuntos contínuos de dados. Como todas as operações são sincronizadas, interações entre processadores SIMD tendem a ser implementadas facilmente e de maneira eficaz.

MIMD: (Multiple Instruction stream, Multiple Data stream) refere-se ao modelo de execução paralelo no qual cada processador está essencialmente operando de maneira independente. Esse modelo é o que mais naturalmente se encaixa no conceito de decomposição de um programa

para execução paralela em sua base funcional. Por exemplo, um processador pode estar atualizando um arquivo de dados, enquanto outro processador está gerando uma saída gráfica de um novo dado de entrada. Esse modelo é mais flexível do que uma execução SIMD, mas vem acompanhada do risco de problemas sérios chamados "condições de corrida", nos quais um programa pode intermitentemente falhar devido a variações de tempo na reordenação de operações de um processador em relação às operações de outro.

SPMD: (Single Program, Multiple Data) é uma versão restrita do MIMD na qual todos os processadores estão rodando o mesmo programa. Diferentemente do SIMD, cada processador executando um código SPMD pode tomar caminhos de fluxo de controle diferentes dentro do programa.

Largura de Banda de Comunicação: (*Communication Bandwidth*). A largura de banda de um sistema de comunicação é a máxima quantidade de dados que pode ser transmitida em um unidade de tempo. Largura de banda para comunicações seriais é freqüentemente medida em *baud* ou *bits/segundo (b/s)*, o que geralmente corresponde a 1/10 ou 1/8 da mesma quantidade de *Bytes/segundo (B/s)*. Por exemplo, um modem com 1200 baud transfere cerca de 120 B/s, enquanto que uma conexão

ATM de 155Mb/s é quase 130000 mais rápida, transferindo cerca de 17MB/s. Grande largura de banda permite que grandes blocos de dados sejam transferidos eficientemente entre processadores.

Latência de Comunicação: (Communication Latency). A latência na comunicação em um sistema é o tempo mínimo que se leva para transmitir um objeto, incluindo quaisquer cabeçalhos de transmissão ou recepção. Latência é muito importante em processamento paralelo porque ela determina a mínima *grain size* que pode ser utilizada pelo sistema, isto é, o mínimo tempo de percurso para um segmento de código gerar aceleração pela execução paralela. Basicamente, se um segmento de código é executado por um tempo menor do que seu resultado demora para os dados e os resultados serem transmitidos (por exemplo, latência), executando esse código serialmente no processador que necessita obter o resultado seria mais rápido do que execução em paralelo por diversos processadores. A execução serial evitaria uma sobrecarga na comunicação de dados.

Transmissão de Mensagens: (Message Passing) Transmissão de Mensagens é um modelo para interações entre processadores dentro de um sistema paralelo. No geral, uma mensagem é construída por software em um processador e depois é enviada por uma rede de inter-conexão

para um outro processador, que então precisa aceitá-la e deve agir de acordo com o conteúdo da mensagem. Mesmo que o *overhead*, isto é, o tempo demandado para a interpretação de cada mensagem, ou a latência, pode ser alto, existem tipicamente poucas restrições em quanta informação cada mensagem pode conter. Portanto, a transmissão de mensagens pode prover uma grande largura de banda, sendo, então, uma ótima maneira de se transmitir um grande bloco de dados de um processador para outro. Entretanto, para minimizar a necessidade de custosas operações de transmissão de mensagens, estruturas de dados em um programa paralelo devem ser espalhadas entre os processadores de forma que parte dos dados requisitados por um processador encontrem-se em seus meios locais, e não precisem trafegar de um processador para outro. Essa tarefa é conhecida como *layout de dados*, ou *data layout*.

Memória Compartilhada: (Shared Memory) é um outro modelo para interações entre processadores em um sistema paralelo. Sistemas baratos como *Personal Computers* comumente disponíveis possuem opções para múltiplos processadores, como por exemplo, máquinas Pentium multiprocessadas. Esses processadores dividem *fisicamente* uma mesma memória, de maneira que um dado escrito por um processador na

memória compartilhada pode ser lido por outro, diretamente. Alternativamente, memória compartilhada *logicamente* pode ser implementada em sistemas em que cada processador possua sua própria memória convertendo-se cada instrução de acesso à memória que não esteja armazenada localmente em uma instrução de comunicação para obtenção desse dado de outra memória. Qualquer uma dessas implementações é geralmente considerada de mais fácil implementação do que a transmissão de mensagens. Fisicamente, memória compartilhada pode ter ambos: grande largura de banda e baixa latência, mas somente quando mais de um processador não tenta acessar o bus ao mesmo tempo. Portanto, o layout de dados também pode ter grande impacto na performance, e efeitos de *cache* podem tornar difícil para se determinar qual o melhor layout.

Funções Auxiliares: (Aggregate Functions) em ambos os modelos—transmissão de mensagens e memória compartilhada—a comunicação é iniciada por um único processador; em contraste, a comunicação por funções auxiliares é, intrinsecamente, um modelo de comunicação paralela no qual um grupo todo de processadores age junto. A mais simples das ações é a *sincronização de barreira*, ou *barrier syncronization*, na qual cada processador individualmente espera até que todos os processadores

no grupo tenham chegado à barreira. Fazendo com que cada processador emita um dado como um efeito colateral de se chegar à barreira, é possível fazer-se com que o hardware de comunicação retorne um valor a cada processador, que é uma função arbitrária dos valores coletados de todos os processadores. Por exemplo, o valor de resposta pode ser a resposta à pergunta "algum processador encontrou a solução?" ou ele pode ser a soma de uma valor de cada processador. A latência pode ser muito baixa, mas a largura de banda por processador também tende a ser baixa. Tradicionalmente, esse método é utilizado primariamente para controlar a execução paralela em detrimento a distribuir valores de dados.

Comunicação Coletiva: É um outro nome para funções auxiliares, mais comumente usado quando essas funções são construídas utilizando-se múltiplas operações de transmissão de dados.

SMP: Multi-Processamento Simétrico (Symmetric Multi-Processor) refere-se ao sistema de operação que tem como conceito um grupo de processadores trabalhando juntos como parceiros *(peers)*, de maneira que qualquer trabalho possa ser feito da mesma maneira por qualquer processador. Um sistema SMP típico pode utilizar a combinação do modelo MIMD com memória compartilhada. Na arquitetura Intel 32, SMP

geralmente significa estar conforme o padrão MPS (Intel MultiProcessor Specification).

SWAR: SIMD Dentro De Um Registrador (SIMD Within A Register) é um termo genérico para o conceito de se particionar um registrador em múltiplos segmentos inteiros e utilizar operações de registradores para executar computações paralelas SIMD entre esses campos. Dada uma máquina com k–bit registradores, caminhos de dados e funções de unidades, há muito se sabe que operações podem funcionar como operações paralelas SIMD em tantos quantos n, k/n–bit valores de campos. Mesmo que esse tipo de paralelismo possa ser implementado utilizando-se registradores e instruções inteiras ordinárias. Muitos microprocessadores de última geração já possuem instruções específicas para melhorar a performance dessa técnica para aplicações com orientações multi-media. Existem, nesse estilo, diversos padrões, como o Intel/AMD/Cyrix *MMX* (extensões multi-media — MultiMedia eXtensions), o Digital Alpha *MAX* (MultimediA eXtensions), o Hewlett–Packard PA-RISC *MAX* (Multimedia Acceleration eXtensions), MIPS *MDMX* (Digital Media eXtension), Sun SPARC V9 *VIS* (Visual Instruction Set). Esses sistemas, apesar de muito parecidos, são mutuamente incompatíveis.

2.4 Cluster de Estações de Trabalho

Clusters, que são um conjunto de estações de trabalho operando um sistema de *transmissão de mensagem* entre si, são, atualmente, as mais usadas e mais variadas implementações de processamento paralelo, indo desde estações especialmente projetadas para esse tipo de trabalho, até máquinas que já se tornaram obsoletas para algum trabalho individual.

2.4.1 Motivos para Utilização de um Cluster

Processamento paralelo utilizando-se um sistema em cluster oferece uma série de vantagens importantes, algumas citadas abaixo:

- Cada uma das máquinas dentro de um cluster pode ser um sistema completo, utilizável para toda uma gama de outras aplicações. Isso leva alguns a sugerirem que clusters de computação paralela podem simplesmente utilizar apenas os ciclos não utilizados pela máquina que está operando. Por exemplo a estação de um funcionário da empresa pode "emprestar" ciclos de processamento a um outro funcionário quando seu operador deixá-la ociosa. Não é uma tarefa simples utilizar esses "ciclos ociosos", e talvez faça com que a máquina do usuário por vezes tenha uma resposta mais lenta, mas pode ser feito;

- a grande utilização, atualmente, de sistemas de rede faz com que o

hardware necessário a sua implementação seja vendido em grande volume, o que resulta em preços baixos de aquisição. Deve também ser levado em consideração que, para cada cluster, são necessários apenas um teclado, uma placa de vídeo e um monitor;

- computação em cluster pode ser "escalonada" para sistemas muito grandes. Sistemas de rede disponíveis para aquisição imediata podem formar, facilmente, clusters com 16 ou mais máquinas. Com pequenas alterações, centenas, ou até milhares de máquinas podem ser interligadas;

- o fato de se ter que trocar uma "máquina ruim" dentro do sistema é trivial comparado ao trabalho de se consertar um SMP com problemas, fato que torna a disponibilidade de um sistema em cluster possivelmente muito superior ao de um sistema SMP.

2.4.2 Limitações dos Clusters

- Com poucas exceções, hardwares de rede não são projetados para serem utilizados em computação paralela. Tipicamente, a latência é muito elevada, e a largura de banda muito restrita, em comparação aos sistemas SMP . Por exemplo, a latência em um sistema SMP é, tipicamente, menor do que alguns poucos microsegundos, mas ela é comumente maior

que centenas ou milhares de microsegundos em um cluster. A largura de banda em uma comunicação SMP é, geralmente, maior que 100MBytes/segundo; mesmo que o hardware de rede mais rápido disponível (por exemplo o "Gigabit Ethernet") ofereça velocidade comparável, as redes mais comuns são usualemente de 10 a 1000 vezes mais lentas.

- existe pouco suporte no mercado para se tratar um cluster como um sistema único de cálculo, apesar de haver muito suporte para implementação.

2.5 MPI (Message Passing Interface)

2.5.1 Descrição Geral

MPI ou *Interface de Passagem de Mensagem* é considerado o novo padrão em protocolo para a implementação de cluster de workstations. É implementado sobre os *Sockets* do protocolo *TCP/IP*, portanto, tudo o que se precisa para montar um sistema utilizando o MPI é o suporte local ao MPI em cada máquina e um meio de comunicação capaz de suportar TCP/IP entre elas.

Outra característica importante do MPI é a de rodar em sistemas muito heterogêneos, isto é, existem implementações no mercado para os mais diversos sistemas, desde persornal computers até máquinas com múltiplos

processadores Alpha.

Por suas características inerentemente heterogêneas, o MPI não possui uma definição própria para o layer físico, ficando essa a critério da implementação.

Outra vantagem significativa do MPI é o fato de se poder usar o mesmo código fonte (sem alterações) para sistemas paralelos em forma de clusters e para máquinas SMP que tenham suporte a parelelismo no sistema operacional. No segundo caso, pode-se informar o MPI que existem, na mesma máquina, n processadores e ele executará as n conexões para essa máquina, e então o sistema operacional pode facilmente alocar cada processo em um de seus n processadores.

Existem diversas implementações para o padrão MPI. Uma das mais populares é o MPICH, da Universidade de Michigan (ver ref. [2]). O MPICH é fornecido como um código fonte, compilável em máquinas executando sistemas operacionais padrão UNIX, o que inclui o Linux. Ele fornece ao usuário, depois de compilado, uma API, isto é, uma *library* contra a qual o software a ser desenvolvido deve ser *linkado*.

A implementação MPICH possui APIs para FORTAN, C e C++. Portanto, o software deve ser desenvolvido em qualquer uma dessas três linguagens de programação, ou até, parte dele em cada uma.

2.5.2 Descrição da API

A implementação do protocolo MPI utilizada nesse projeto, a MPICH, fornece-nos uma API, isto é, um conjunto de funções e rotinas que podem ser chamadas de dentro do código desenvolvido, que implementa as funções de distribuição e comunicação previstas no padrão.

Algumas das funções fornecidas pela API, segundo [2], estão listadas a seguir.

MPI_Init() é a função que inicializa o ambiente de execução do MPI;

MPI_Comm_create() cria um novo canal de comunicação;

MPI_Cancel() cancela um pedido de comunicação;

MPI_Comm_rank() acessa o valor do *rank* do processo atual. Rank é o número que define o processo dentre todos os processos paralelos;

MPI_Comm_compare() compara dois canais de comunicações;

MPI_Get_processor_name() retorna o nome do processador atual, levando em consideção a configuração do MPI, que determina para onde cada processo deve ser enviado;

MPI_Abort() encerra a execuçãao em abiente MPI, retornando uma condição de erro;

MPI_Bcast() só pode ser executado a partir do processo mãe (rank=0), e transmite uma informação ou seqüência de dados a todos os processos sendo executados;

MPI_Barrier() bloqueia o processamento até que todos os processos tenham chegado a essa marca;

MPI_Reduce() combina os valores advindos de todos os processadores para um único valor;

MPI_File_open() abre um arquivo de dentro do ambiente de execução MPI;

MPI_File_set_view() configura o tipo de interação a qual o arquivo aberto estará sujeito;

MPI_File_read() lê dados a partir do arquivo;

MPI_File_write() grava o arquivo em disco;

MPI_File_close() fecha o arquivo;

MPI_Gather() coleta informações de todos os processos;

MPI_Alltoall() distribui dados de todos os processos para todos os processos;

MPI_Bsend() envio de dados utilizando *buffers* definidos pelo usuário;

MPE_Init_log() incializa o ambiente de coleta de dados;

MPE_Describe_state() descreve um estado de execução, para fins de log;

MPE_Start_log() começa a coletar informações enviadas pelos processos;

MPE_Log_event() acrescenta uma informação ao log;

MPE_Finnish_log() termina a coleta de dados;

MPI_Finalize() finaliza o ambiente de execução MPI.

Capítulo 3

Processamento de Imagem

3.1 Introdução

O interesse em métodos de processamento digital de imagens vem de suas aplicações em duas áreas principais: melhoria de informações pictoriais para a percepção humana, e processamento de dados de cenas para percepção autônoma de máquinas. O presente trabalho é focado na segunda, por isso, serão aqui apresentadas as técnicas utilizadas para promover a compreensão de uma imagem por um sistema computacional.

3.2 Processamento e algoritmos

3.2.1 Passos Fundamentais

Processamento digital de imagens engloba uma vasta gama de hardware, software e desenvolvimento teórico. Nessa seção serão discutidos alguns dos passos fundamentais para se realizar uma tarefa de processamento de imagem.

As informações contidas nesse capítulo foram, em grande parte, retiradas de [1].

O primeiro processo envolvido é a *aquisição da imagem* — isto é, adquirir uma imagem digital. Para que se possa fazer essa aquisição são necessários: um sensor de imagem e a capacidade de digitalizar o sinal produzido pelo sensor. O sensor pode ser, por exemplo, uma câmera de televisão, colorida ou preto e branco, que produza uma imagem completa do domínio do problema a cada 1/30 de segundo. Esse sensor poderia ser, também, uma câmera de varredura por linha, que produz uma só linha de imagem por vez. Nesse caso, o movimento do objeto pela linha de varredura produz uma imagem bi-dimensional. Se a saída da câmera ou de qualquer outro sensor de imagem não é em formato digital, um conversor analógico-digital é utilizado para digitalizá-la. A natureza do sensor e da imagem por ele produzida são determinados pela aplicação.

Depois de obtida uma imagem digital, o próximo passo trata sobre o *pré-processamento* dessa imagem. A função-chave do pré-processamento é melhorar a imagem de forma que melhore as chances de sucesso dos outros processos envolvidos. Neste trabalho, o pré-processamento consistirá em uma maneira de melhorar a evidência dos objetos na imagem.

O próximo passo trata da segmentação. Com uma definição ampla, *segmentação* particiona uma imagem de entrada em seus objetos ou partes constituintes. Em geral, a execução de uma segmentação por sistemas computacionais autônomos é um dos passos mais difíceis no processamento digital de imagens. Em uma mão, um processo de segmentação correto pode trazer o problema muito próximo de ser resolvido. Na outra, um processo errático pode levar a uma solução errada, ou, até, a solução nenhuma.

A saída de dados do estágio de segmentação costuma ser pixels de dados "crus", ou raw pixel data, em inglês, constituindo ou a fronteira de uma região, ou a própria região. Em ambos os casos, faz-se necessário converter esses dados em uma forma que seja adequada para se realizar o processamento computacional.

Escolher uma representação é apenas parte da solução para transformar o raw data em uma forma que seja adequada para um processamento computacional subseqüente. Um método deve ser especificado para descrever os

dados de forma que as características de interesse sejam ressaltadas. *Descrição*, também chamada de *seleção de características* trata de extrair características que resultam em alguma informação quantitativa de interesse ou características que são básicas para diferenciar uma classe de objetos de outra.

O último estágio envolve o reconhecimento e a interpretação. *Reconhecimento* é o processo que dá uma rotulação a um objeto baseado na informação obtida de seus descritores. *Interpretação* envolve dar um significado a um grupo de objetos reconhecidos.

Existe também um último tópico a ser considerado, o *domínio do problema*. O domínio é uma das características que devem constar hard-coded na solução do problema, pois ela é inerente à solução.

Mais informações sobre tópicos em processamento de imagem podem ser obtidas em [1, capítulos 1, 2].

3.2.2 Imagem Digitalizada

A imagem, como obtida pelo sensor ou câmera, é captada como vinda de diferentes lugares no espaço. O sistema óptico do sensor ou câmera é responsável por disponibilizar essa imagem em uma superfície de sensoriamento, de forma que ela possa ser digitalizada.

Essa superfície de sensores constitui-se, normalmente, de um conjunto de células chamadas de *Charged-Couple Device* . São células que, quando da incidência de fótons, produzem um sinal elétrico equivalente à quantidade de fótons que incidiram sobre a sua superfície em um determinado intervalo de tempo. Esse intervalo de tempo, geralmente, constitui-se no intervalo de leitura dessa célula.

As células de sensoriamento são distribuídas sobre a superfície que receberá a luminosidade focada do sistema óptico, de tal forma que consiga diferenciar satisfatoriamente os pontos de origem dos fótons que nele incidiram. Como cada célula pode, somente, determinar o número de fótons que nela incidiu, e não a origem desses fótons, faz-se necessário que diversas delas sejam posicionadas de tal forma que se consiga, posteriormente, unificar as informações vindas de cada uma, para que se possa gerar uma representação da imagem original.

O dado de intensidade advindo de cada célula é denominado *pixel*. Para que se possa operar sobre uma imagem, ela deve possuir pixels suficientes para que se possa distinguir, com a precisão necessária, os componentes da imagem original.

Cada pixel, então, contém um dado, que é a intensidade luminosa que atingiu a sua célula geradora, compondo esses dados na forma em que as

células estão posicionadas produz a imagem desejada.

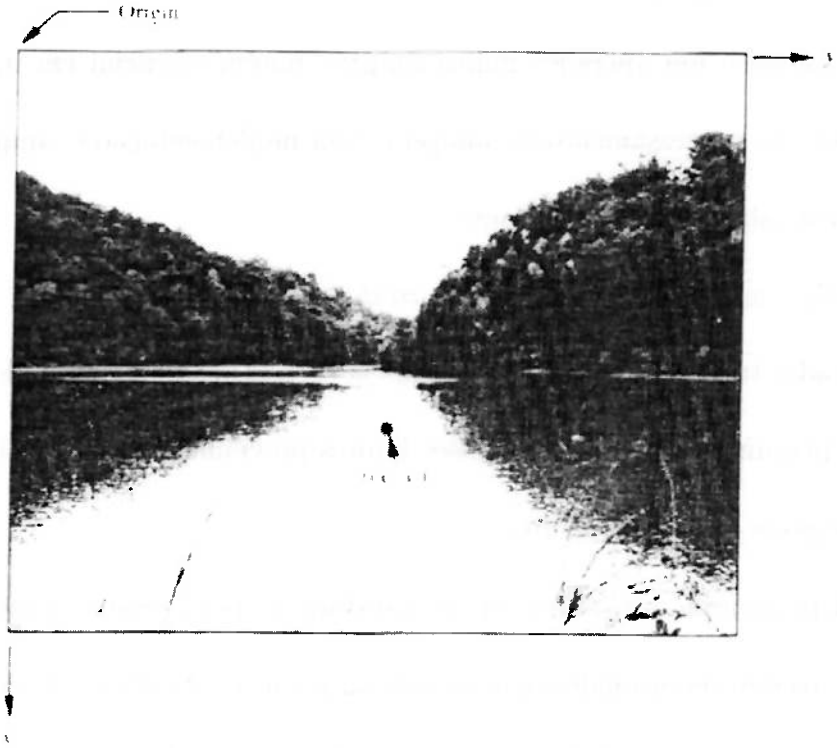
O próximo passo, então, é a conversão dos valores analógicos de dados de cada pixels em valores digitais, para que possam ser operados por computadores digitais. Isso é feito transformando, comparativamente, os valores mais baixos de intensidade em valores menores, e os valores mais altos de intensidade em valores numéricos maiores, dentro de uma escala pré determinada. Um exemplo seria atribuir o número 0 a ausência de luz e o número 255 a maior luminosidade prevista para o ambiente. Isso daria espaço a existência de 256 níveis de intensidade.

O que resulta é um valor $f(x, y)$ =intensidade luminosa no ponto, onde x e y determinam a posição do pixel, como visto na figura 3.1, na página 29.

Depois de convertidos em números digitais, esses valores dos dados, e, conseqüentemente, da luminosidade, são arrumados de forma a gerar uma matriz com o número de linhas e colunas iguais aos pixels, e, portanto, ao número de sensores, em cada uma das direções.

Então, obtém-se uma matriz como

Figura 3.1: A Imagem como uma Função $f(x, y)$



$$f(x,y) \approx \begin{bmatrix} f(0,0) & f(0,1) & \cdots & f(0,M-1) \\ f(1,0) & f(1,1) & \cdots & f(1,M-1) \\ \vdots & & & \vdots \\ f(N-1,0) & f(N-1,1) & \cdots & f(N-1,M-1) \end{bmatrix}$$

Sendo $f(x, y)$ a intensidade de luz obtida no ponto (x, y) da superfície de sensores; M, o número de pixels na horizontal e N, na vertical.

3.2.3 Treshold

O *treshold* é um operador muito simples, porém essencial em muitas aplicações em processamento de imagem. Sua implementação é simples e computacionalmente muito eficiente.

Ele consiste em se estabelecer *tresholds*, ou limites. Quando se aplica o operador treshold a uma imagem, deseja-se, por exemplo, que todos os pixels que possuírem intensidades nesses limites previamente estabelecidos sejam, de alguma forma, marcados.

Um exemplo consistiria em se transformar, por exemplo, todos os pixels que tiverem intensidades maiores que ou iguais a 128 em 1, e todos os pixels que tiverem intensidades menores que 128 em 0. Isso resultaria em uma imagem preto e branco, convertida a partir da imagem em gray scale original.

$$f(x,y) = \begin{cases} 0. & \text{se} f(x,y) < 128: \\ 1. & \text{se} f(x,y) > 128. \end{cases}$$

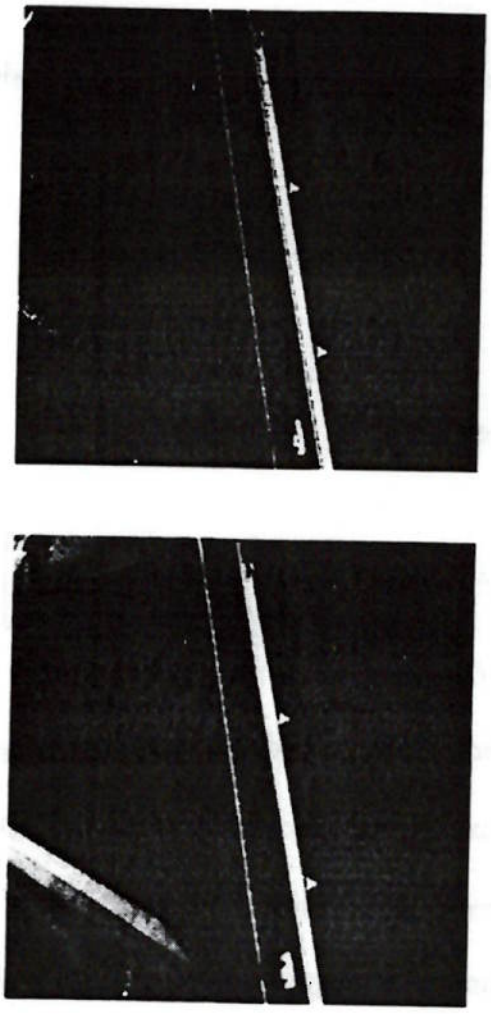
Outro exemplo seria ressaltar uma região de intensidades intermediárias, transformando-se em 1 todos os valores que estiverem dentro desses limites, como, por exemplo:

$$f(x,y) = \begin{cases} 0. & \text{se } f(x,y) < 100 \text{ ou } f(x,y) > 128: \\ 1. & \text{se } 100 \leq f(x,y) \leq 128. \end{cases}$$

Um exemplo de uma imagem e o resultado de um treshold sobre ela, feito

para reessaltar os pontos com maior intensidade luminosa, podem ser vistos na figura 3.2, página 31.

Figura 3.2: Uma Imagem e seu Treshold para Ressaltar Pontos com Maior Luminosidade



3.2.4 Operador Gradiente

O *Operador Gradiente* é, na verdade, um filtro derivativo. Um operador derivativo, como o gradiente, quando aplicado a uma imagem, realça as diferenças de intensidade entre pixels vizinhos. Isso é usado para que se possa, por exemplo, reconhecer objetos em um fundo, quando esses possuem intensidades diferentes.

Para uma função $f(x, y)$, o gradiente de f nas coordenadas (x, y) é definido como o vetor

$$\nabla \vec{f} = \begin{bmatrix} G_x \\ G_y \end{bmatrix} = \begin{bmatrix} \frac{\partial f}{\partial x} \\ \frac{\partial f}{\partial y} \end{bmatrix} \tag{3.1}$$

A magnitude desse vetor,

$$\nabla f = mag(\nabla \vec{f}) = \left[\left(\frac{\partial f}{\partial x} \right)^2 + \left(\frac{\partial f}{\partial y} \right)^2 \right]^{1/2} \tag{3.2}$$

é a base para várias abordagens sobre diferenciação de imagem.

É bem sabido, por análise de vetores, que o vetor gradiente evidencia a direção da maior taxa de variação de f em (x, y). Quando se utiliza esse método para a *detecção de bordas*, ou "edge-detection", em inglês, uma característica importante é a magnitude desse vetor, como visto em 3.2, que costuma ser referida simplesmente como *gradiente*, e simbolizado por ∇f, onde:

$$\nabla f = mag(\nabla \vec{f}) = \left[G_x^2 + G_y^2\right]^{1/2} \tag{3.3}$$

Esse quantidade é igual ao valor máximo da taxa de incremento de $f(x, y)$ por unidade de distância na direção de $\nabla \vec{f}$. É prática comum aproximar-se o gradiente por valores absolutos:

$$\nabla f \approx |G_x| + |G_y| \tag{3.4}$$

que é muito mais fácil de se implementar, além de ser muito mais eficiente em consumo de processamento.

Uma imagem e seu gradiente podem ser vistas na figura 3.3, na página 36.

3.2.5 Chain Code

Chain Codes são utilizados para se representar um contorno através de uma seqüência conexa de segmentos em linha reta de tamanho e direção especificados. Tipicamente, essa representação é baseada na conectividade de graus 4 ou 8 dos segmentos. A direção de cada segmento é codificada utilizando-se um esquema de numeração como o mostrado na figura 3.4, página 37. A primeira imagem mostra uma representação para 4 direções, e a segunda, para 8.

O Chain Code de um objeto em uma imagem digital geralmente é obtido seguindo-se o contorno em alguma direção pré-definida—digamos, utilizando o sentido anti-horário—e rotulando uma direção para cada segmento conectando um par de pixels. Esse método é geralmente inaceitável por dois motivos: (1) a corrente de códigos resultante geralmente é muito grande, e (2) quaisquer pequenas variações ao longo do contorno resulta em ruídos ou segmentações imperfeitas resultam em variações no código que não necessariamente representam a forma do contorno.

Uma solução freqüentemente utilizada para solucionar o problema citado no ultimo parágrafo é utilizar um espaço maior para as divisões mínimas do espaçamento, isto é, na prática, não se utiliza todos os pixels, mas sim, utiliza-se, propositalmente, um pixel a cada n pixels de figura, obtendo, assim, uma representação mais próxima da real para o contorno. Produz-se, então, uma nova representação, onde podem também serem aplicados os métodos de conectividade entre pixels vizinhos.

Na figura 3.5 têm-se em (a) uma fronteira digitalizada de um elemento, com a marcação de um novo grid. Em (b), vê-se a aplicação de uma nova aquisição utilizando-se o novo grid. Em (c) tem-se a aplicação de um chain code de 4 direções sobre a aquisição no novo grid e em (d) o chain code de 8 direções.

Obviamente, a acuracidade do código está diretamente relacionada com o espaço entre pixels utilizado pelo método.

Figura 3.3: Uma Imagem e seu Gradiente

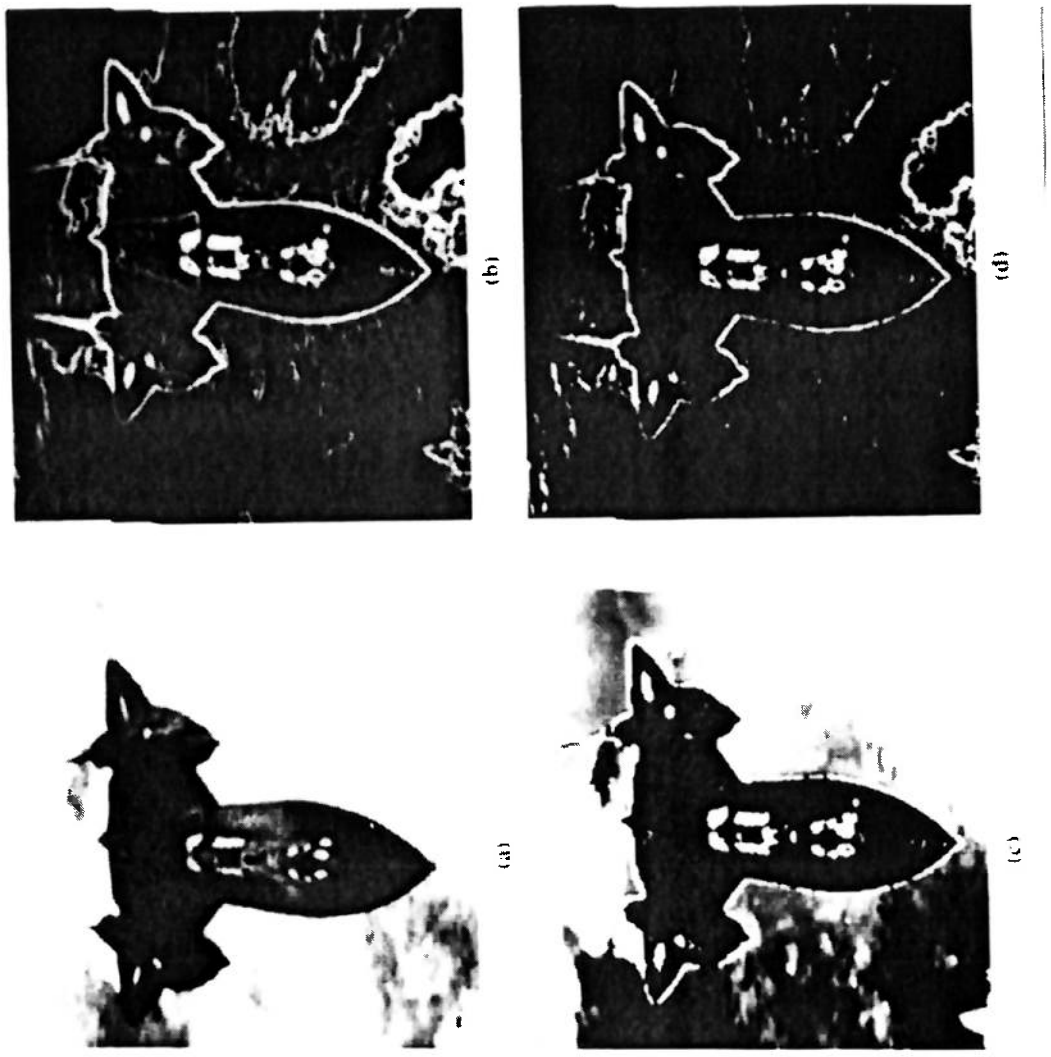


Figura 3.4: Representação da Conectividade de um Pixel, Considerando 4 e 8 Direções

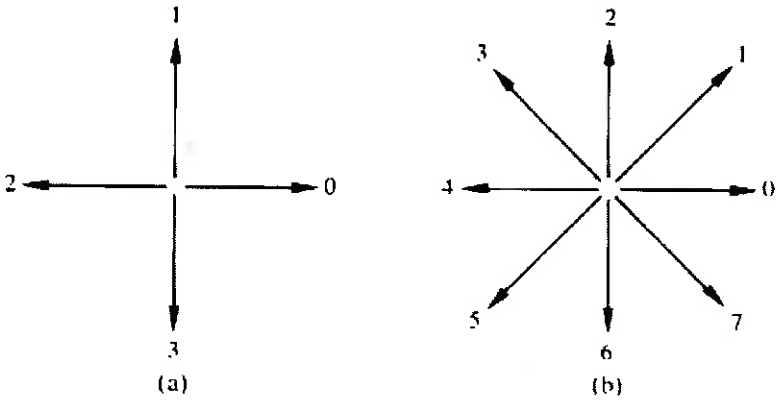
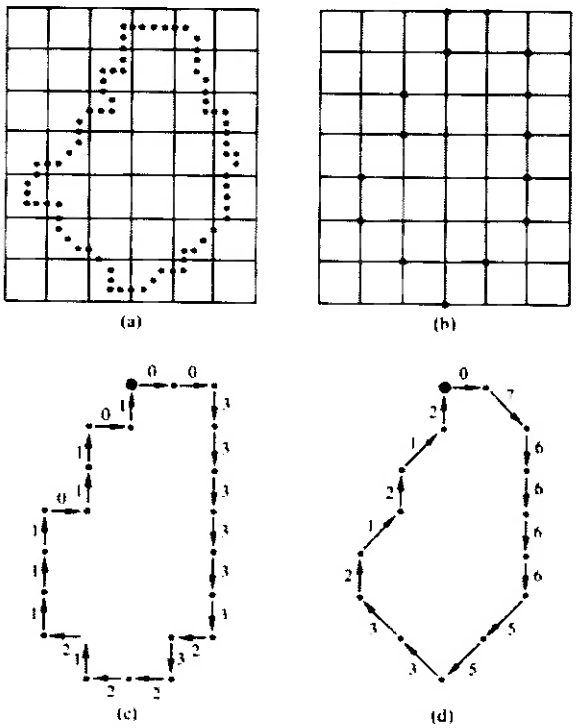


Figura 3.5: Representações da uma Fronteira Digitalizada e a Aplicação de Chain Code



Capítulo 4

Arquitetura de Solução

4.1 Descrição Geral

Para que se possa obter o posicionamento dos objetos na imagem, como mencionado nos objetivos do presente trabalho, citados em 1.2, necessita-se, então, obter, e comunicar ao sistema de controle, a posição da bola, e a posição e ângulo de cada um dos três robôs-jogadores do time em campo.

Os métodos descritos nos capítulos 2 e 3 serão, então, implementados e utilizados, e um procedimento para a identificação e obtenção da direção dos robôs será explicado.

4.2 Definição do Hardware

Para executar as tarefas previstas, existe uma vasta gama de opções possíveis de hardware. Desde computadores pessoais ultrapassados, como os compatíveis com o padrão IBM PC 386, até grandes servidores corporativos, passando por diversos tipos de workstations. Dentre essa gama, encontram-se os Clusters de processamento paralelo e as máquinas com processamento paralelo SMP.

Devido às condições já existentes e disponíveis, optou-se por desenvolver um sistema que fosse executado exatamente em um Cluster de máquinas, mas que possa, também ser executado isoladamente, ou um máquinas que possuam processadores em arquitetura SMP. A solução faz-se, então, razoavelmente independente de hardware, pois, como já citado em 2.5, existem padrões altamente genéricos, e diversas implementações para seus diversos hardwares.

O Hardware pode, deve-se notar, ser trocado por qualquer outro, em que os requisitos listados na seção 4.3 sejam atendidos.

4.3 Definição do Software

Primeiramente deve-se escolher, dado o hardware citado no ítem 4.2, o sistema operacional a ser utilizado. Características desejáveis no mesmo são:

- Grande aproveitamento do hardware, sem desperdícios com processos não relacionados que apenas causariam degradação na velocidade de processamento do programa;
- escalonabilidade com hardwares maiores, ou mais novos, sem que grandes alterações sejam necessárias;
- compatibilidade com versões futuras do próprio sistema operacional;
- facilidade de comunicação em rede;
- disponibilidade e
- baixo preço de aquisição.

Considerando-se as dezenas de sistemas operacionais existentes hoje para a plataforma de hardware escolhida, chegou-se a conclusão de que os sistemas operacionais que possuíam o padrão UNIX seriam os mais indicados. Dentre os opções disponíveis, decidiu-se por se utilizar o sistema Linux , por este possuir diversas implementações funcionais do MPI, possuir suporte a SMP

e por ter suporte a grande parte dos hardwares de aquisição disponíveis no mercado.

Devido às características de velocidade necessárias—o sistema deve ser executado o quanto mais próximo de "real-time" o possível—optou-se por implementar o sistema de processamento paralelo e de imagem utilizando-se unicamente o sistema operacional em modo texto, sem gráficos.

Considerando-se as características da implementação, o sistema de hardware pode ser trocado por quaisquer máquinas que possuam um sistema UNIX compatível com POSIX e uma implementação do MPI funcional.

Para o desenvolvimento do código também foram consideradas diversas linguagens de programação e execução, tais como:

- C;
- C++;
- Pascal;
- FORTRAN e
- ambiente Khoros.

A decisão sobre qual das linguagens utilizar foi baseada em alguns quesitos, a saber:

- Velocidade de processamento;
- compatibilidade com outros sistemas;
- suporte do sistema operacional;
- facilidade de implementação;
- aplicativos de desenvolvimento disponíveis e
- suporte da implementação do MPI.

Chegou-se, então, à conclusão de que a melhor linguagem seria a linguagem C, pelos seguintes motivos:

1. Tem a melhor performance de todas, no sistema Linux;
2. tem uma grande compatibilidade—pode-se facilmente exportar o código para qualquer outro sistema padrão UNIX que utilize-se das diretrizes POSIX;
3. tem suporte pleno do sistema operacional, e de desenvolvedores não relacionados;
4. propicia relativa facilidade de implementação do código;
5. possui diversos aplicativos de desenvolvimento, como IDEs, debuggers e outros e

6. possui diversas implementações do MPI.

Após ter sido efetuada a escolha do sistema operacional, decidiu-se por utilizar a implementação do MPI feita pelo Departamento de Computação da Universidade de Michigan, o MPICH, pois possui uma documentação completa e pode ser facilmente utilizado.

4.4 Sistema de Obtenção e Aquisição de Imagem

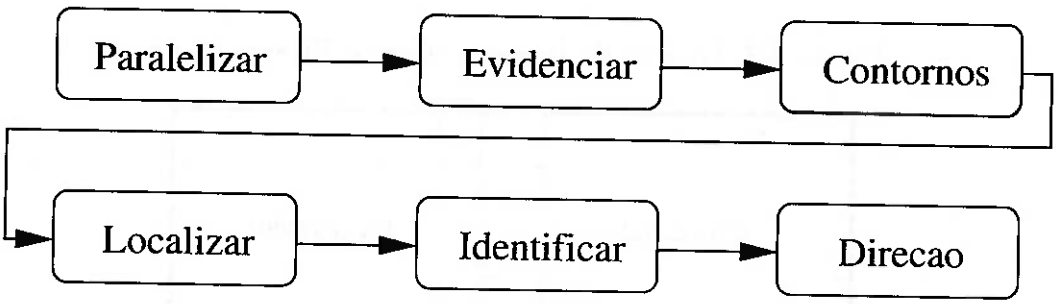
Para que se proceda com a captação da imagem da mesa, será utilizada uma câmera VHS comercial, posicionada 2 metros acima da mesa, no seu centro, como mostrado no esquema da figura 1.2, na página 4.

Após a captação pela câmera, a imagem deve passar pelo sistema de aquisição, para que possa ser utilizada pelo sistema de software. Essas aquisição pode ser feita com qualquer hardware disponível que possua suporte ao sistema operacional escolhido, descrito no ítem 4.3. O equipamento utilizado para esse trabalho foi uma placa modelo VideoBlaster, produzida pela Creative Labs.

4.5 Procedimento

O procedimento a ser adotado para alcançar os objetivos desse trabalho compreende várias etapas, cuja seqüencia está mostrada no diagrama esquematizado que pode ser visto na figura 4.1.

Figura 4.1: Procedimentos a Serem Executados



4.5.1 Paralelização do Processo

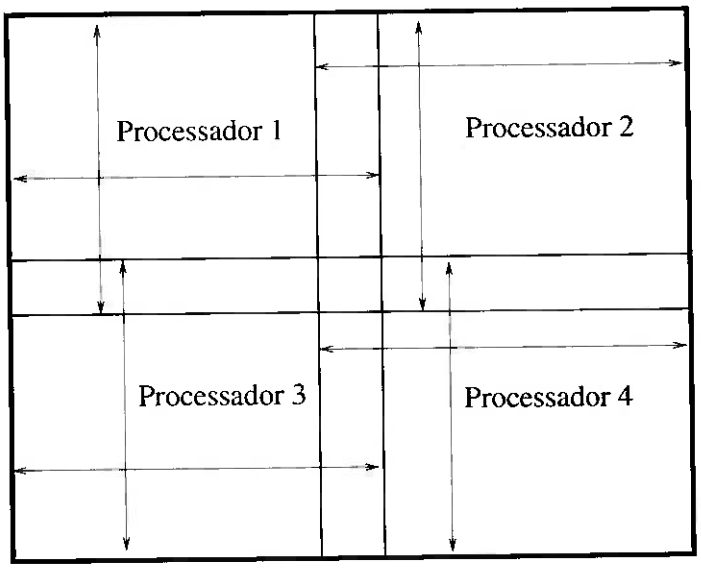
A primeira parte da solução consiste em se desenvolver um método eficiente para executar a paralelização do processo.

O processamento de uma imagem é, intrinsicamente, um processo paralelizável, pois pode-se dividir a imagem em "setores" e cada um pode ser trabalhado por um processador diferente, em paralelo.

Para que se possa paralelizar eficientemente o processo, opta-se por dividir a imagem original em 4 subcomponentes, como mostra a figura 4.2.

Como se pode notar, cada processador ficará com um pouco mais de 1/4 da imagem, pois no caso de termos elementos entre os subcomponentes, seu processamento seria muito complicado. Por isso, o algoritmo deve, também, ser capaz de detectar uma eventual obtenção de um mesmo elemento por dois processadores.

Figura 4.2: Divisão da Imagem entre os Processadores



Após o processamento individual de cada subcomponente, o resultado deve ser enviado ao processador líder, para que a solução final possa ser obtida.

4.5.2 Evidenciação dos Componentes

Devido às condições existentes no sistema (ver ítem 1.1), pode-se simplificar o procedimento de localização dos componentes da imagem aplicando-se uma operação de Treshold sobre a mesma, como descrito no ítem 3.2.3.

Os limites do treshold devem ser tais que façam com que a bola e os robôs sejam mostrados na figura resultante, deve também tentar fazer com que a menor quantidade possível de elementos que não esses sejam evidenciados.

Nos testes no banco de ensaios de laboratório obteve-se os valores limites de 100 e 140 de intensidade. Isto significa que aos valores entre 100 e 140 foram atribuídos o valor 1, e a todos os demais, 0.

$$f(x,y) = \begin{cases} 0, & \text{se} f(x,y) < 100 \text{ou} f(x,y) > 140: \\ 1, & \text{se} 100 \leq f(x,y) \leq 140. \end{cases}$$

Para a aplicação, apesar de as condições luminosas, como descrito no ítem 1.1, serem relativamente constantes, deve-se fazer, no começo de cada jogo, uma calibração dos limites do treshold, para que se garanta que as áreas a serem evidenciadas não estejam ficando de fora da região resultante.

Isso pode, num próximo passo, ser implementado facilmente no software, por exemplo, estabelendo-se uma região inicial onde estarão a bola e dos robôs, de onde o software extrairia a informação sobre as intensidades.

O resultado dessa etapa deve ser uma imagem em preto e branco, onde

constariam os componentes procurados (bola e robôs), e eventuais traços da mesa ou algum tipo de ruído advindo do treshold, como, por exemplo, pixels isolados que, devido às condições de iluminação, caiam na região de evidência.

4.5.3 Obtenção dos Contornos

Com a imagem preto e branco resultante da evidenciação, deve-se agora obter os contornos, ou boundaries dos componentes. Isso pode ser feito, de maneira computacionalmente eficiente, aplicando-se o operador gradiente, descrito em 3.2.4.

O gradiente, para o nosso caso, pode ser simplificado para:

$$\nabla f = \left[\frac{\partial f}{\partial x} + \frac{\partial f}{\partial y}\right] \tag{4.1}$$

$$f(x, y) = \begin{cases} 0, & \text{se} \nabla f = 0; \\ 1, & \text{se} \nabla f \neq 0. \end{cases}$$

Utilizando-se como referência o pixel atual, devem ser checados os pixels diretamente a direita e abaixo, e não os outros. Isso deve produzir uma linha simples, em grande parte dos casos, o que facilitaria o próximo passo do processamento.

A aplicação do operador gradiente deve fornecer como saída uma imagem,

também preto e branco, contendo apenas os contornos das figuras da imagem resultante do treshold, mais algum possível ruído.

4.5.4 Localização dos Objetos

Agora que se tem uma imagem contendo todos os contornos dos objetos na cena, deve-se encontrá-los e catalogá-los. Isso pode ser feito utilizando o Chain Code, descrito em 3.2.5.

Primeiramente, deve-se procurar por pixels não nulos na imagem. Começando do ponto $(x, y) = (0, 0)$, que se refere ao canto superior esquerdo da imagem. Assim que um pixel não nulo for encontrado, ele deve ser seguido, sempre no sentido horário, até que não haja mais pixels, ou que o primeiro tenha sido novamente encontrado.

Seguir no sentido horário significa que devem ser verificados, e seguidos, primeiramente os pixels a direita, na diagonal direita-abaixo, e abaixo, e, apenas se não houver pixel conexo em alguma dessas direções, na diagonal abaixo-esquerda, esquerda, diagonal esquerda-acima, e, finalmente, acima.

Para cada objeto deve-se guardar os limites em x e em y, e o número de pixels. De posse desses valores, pode-se concluir, geometricamente, a que classe de objetos da mesa se refere o objeto encontrado: se é a bola, ou se é algum dos robôs.

Utilizando os limites dos pixels, pode-se, facilmente, concluir se o objeto encontrado é a bola. Isso se dá pois, como visto em 1.1, ela aparecerá na imagem como um círculo com diâmetro de 6 cm. Sabendo qual a altura de posicionamento da câmera no sistema, tenho a quantidade de pixels que a bola ocupará na figura digitalizada. Para as configurações de laboratório, onde, devido a limitaçõoes na altura disponível a câmera está fixada um pouco abaixo dos 2 metros, o diâmetro da bola tem 11 pixels.

Quando concluir-se que o objeto encontrado é a bola, deve-se tirar a média dos pixels máximos e mínimos encontrados, nos dois eixos, para se obter a coordenada central da mesma, que deve ser fornecida ao sistema de controle.

$$x_{bola} = x_{max} - x_{min} \tag{4.2}$$

$$y_{bola} = y_{max} - y_{min} \tag{4.3}$$

Os três robôs pertencentes ao time devem possuir marcas de identificação, a fim de que eles possam ser diferenciados. Foram analisadas várias possibilidades de marcação, e uma foi escolhida.

A marcação escolhida foi a impressão na superfície branca superior de cada robô de uma bola preta, com 2 cm de diâmetro afastada 3 mm da borda do lado do robô considerado "frente", assim consegue-se identificar, a

partir da imagem do robô, como ele está posicionado. Para diferenciar um robô do outro—pois deve-se saber qual o número do robô encontrado, para que o mesmo possa ser acionado—deve-se inserir, no robô número 2, mais uma marca de controle: um outro círculo, mas desta vez localizado na quina oposta ao lado onde a marcação de posicionamento foi feita, como é mostrado na figura 4.4, página 53.

Para o robô número 3, ao invés de ser inserida mais uma marca, serão inseridas duas outras marcas, círculos de mesmo diâmetro da marcação de posicionamento, mas nas duas quinas opostas ao lado considerado "frente". A figura 4.5, na página 54, mostra como ficou essa marcação.

O robô número 1 não deve conter outra marca, a não ser a marcação de posicionamento já citada. Pode-se ver na figura 4.3, na página 52 como ficou sua face superior.

O procedimento para a identificação de um dos objetos encontrados como sendo o um robô deve ocorrer da seguinte maneira:

1. Primeiramente, já deve ter sido executado o processo de seguir os pixels;

2. deve-se, utilizando características geométricas, identificar os possíveis candidatos a serem robôs. Cada robô da equipe tem 7,5 cm de lado. Tem-se, então, como limites de $\delta x = x_2 - x_1$, 7,5 e 10,6 cm, sendo o maior valor o caso em que o robô esteja com a sua diagonal alinhada

ao eixo. Considera-se, então que tudo que possuir essa característica constitui um robô;

3. agora deve-se filtrar os objetos obtidos para evitar que, por exemplo, a marcação da mesa seja processada. Para isso, usa-se uma técnica muito simples: checar a intensidade do pixel central de cada objeto obtido. Sabendo-se que a superfície superior de cada robô é branca no seu centro, procuro na imagem original o pixel central, e checo se pode ser branco;

4. por último, deve-se encontrar o ângulo de inclinação do robô. Isso deve ser feito utilizando-se o algoritmo descrito no ítem 4.5.5.

Figura 4.3: Marcação do Robô 1

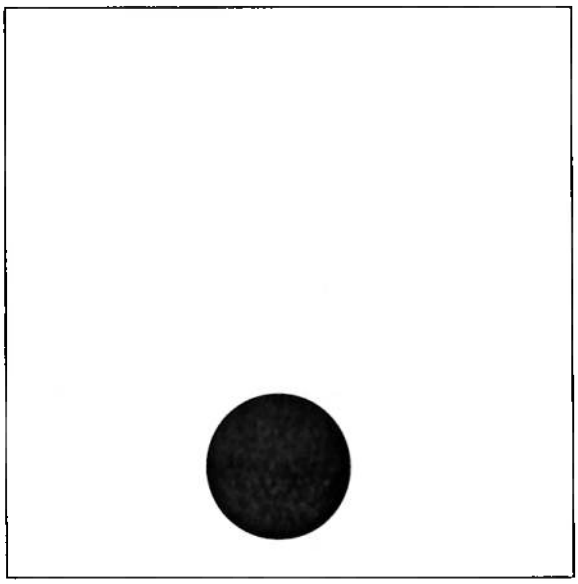
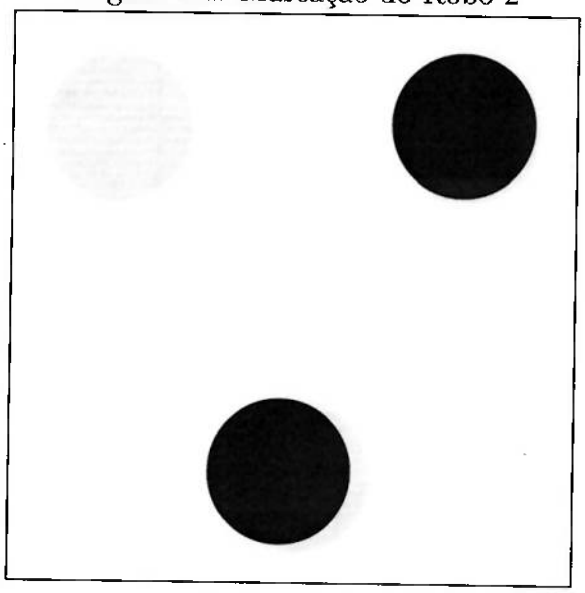


Figura 4.4: Marcação do Robô 2



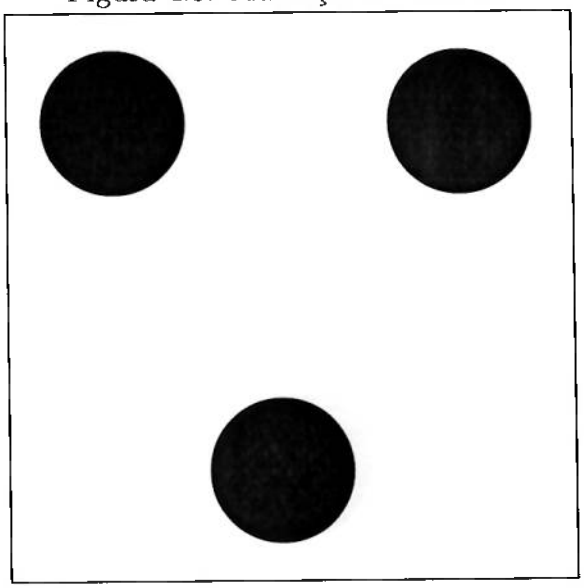
4.5.5 Identificação e Obtenção da Direção de cada Robô

Como já foi descrito em 4.5.4, será agora apresentado o algoritmo para, dadas as posições dos três robôs componentes do time, identificar cada um deles.

A partir do centro do objeto encontrado, deve-se traçar uma circunferência com um raio apropriado, de forma que ela intercepte todas as possíveis marcações em cada robô, perto do centro, como mostrado na figura 4.6, página 57. No caso da configuração utilizada no laboratório para os testes, o raio foi de 8 pixels.

A circunferência pode ser traçada utilizando-se métodos de representação de figuras como os utilizados em aplicações de CAD (Computer Aided De-

Figura 4.5: Marcação do Robô 3



sign).

O método adotado para se traçar a circunferência consiste em dividi-la em 8 partes, e, a partir dessa divisão, estabelecer as regras para a formação dos pixels. O fundamento do método está descrito a seguir:

1. Primeiramente, deve-se ter como entrada o centro e o raio da circunferência;

2. depois, deve-se estabelecer o algoritmo responsável pela divisão da cirfunferência em octantes. A lógica de divisão e traço estão apresentadas a seguir:

   - Primeiro octante: deve-se fazer um laço enquanto $x > y$, onde x é a horizontal e y, a vertical: somar 1 a x e se $x^2 + y^2 > r^2$, deve-se

subtrair y;

- Segundo octante: $y > 0$: deve-se subtrair y e somar x, e se $x^2 + y^2 > r^2$, deve-se subtrair x;
- Terceiro octante: $x > -y$: subtrai-se y e se $x^2 + y^2 > r^2$ deve-se subtrair x;
- Quarto octante: $x > 0$: deve-se subtrair x, subtrair y e se $x^2+y^2 > r^2$, soma-se x;
- Quinto octante: $y < x$: subtrai-se x e se $x^2 + y^2 > r^2$, soma-se y;
- Sexto octante: $y < 0$: soma-se y, subtrai-se x e se $x^2 + y^2 > r^2$, soma-se x;
- Sétimo octante: $-x > y$: deve-se somar y e se $x^2+y^2 > r^2$, deve-se somar x;
- Oitavo octante: $x < 0$: soma-se x, soma-se y e se $x^2 + y^2 > r^2$, subtrai-se y.

3. deve-se, então, contar o número de regiões escuras encontradas, o que nos fornece qual dos robôs está sendo verificado;

4. agora, deve-se encontrar a distância entre as marcas, o que pode ser feito por simples contagem de pixels, e, então, faz-se a verificação de qual delas é a marcação de direção.

(a) No robô 1, ela é a única presente;

(b) no robô número 2, como a verificação descrita anteriormente é feita no sentido horário, é a marca que possui o menor número de pixels brancos antes dela e

(c) no robô 3, é a marca com o maior intervalo de pixels brancos entre as outas duas.

5. encontrada a marca de posicionamento, deve-se, então, encontrar o ponto médio da mesma em relação à circunferência traçada. Faz-se isso tirando a média nos eixos x e y dos pontos escuros encontrados;

6. o último passo é, então, traçar uma reta entre o ponto central do robô e a média do ítem anterior, o que nos dá um vetor com a direção e o sentido.

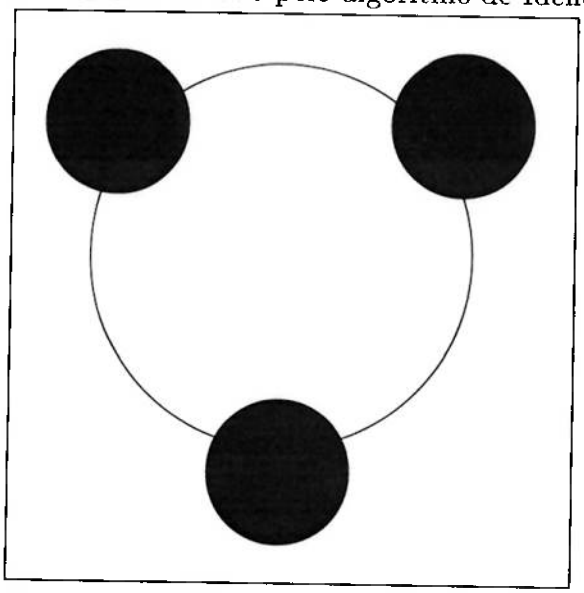
Nesse ponto têm-se, então identificos: a bola, os três robôs e suas direções e sentidos, que é o objetivo do presente trabalho.

Por último, deve-se retornar essas informações ao sistema de controle, para que possam ser tomadas as atitudes apropriadas.

Como pode-se constatar, todos os métodos aqui descritos foram elaborados de forma a propiciar o menor processamento computacional possível, com algoritmos e lógicas simples, para que o objetivo de execução em "real-time"

pudesse ser atingido.

Figura 4.6: Círculo Criado pelo algoritmo de Identificação



Capítulo 5

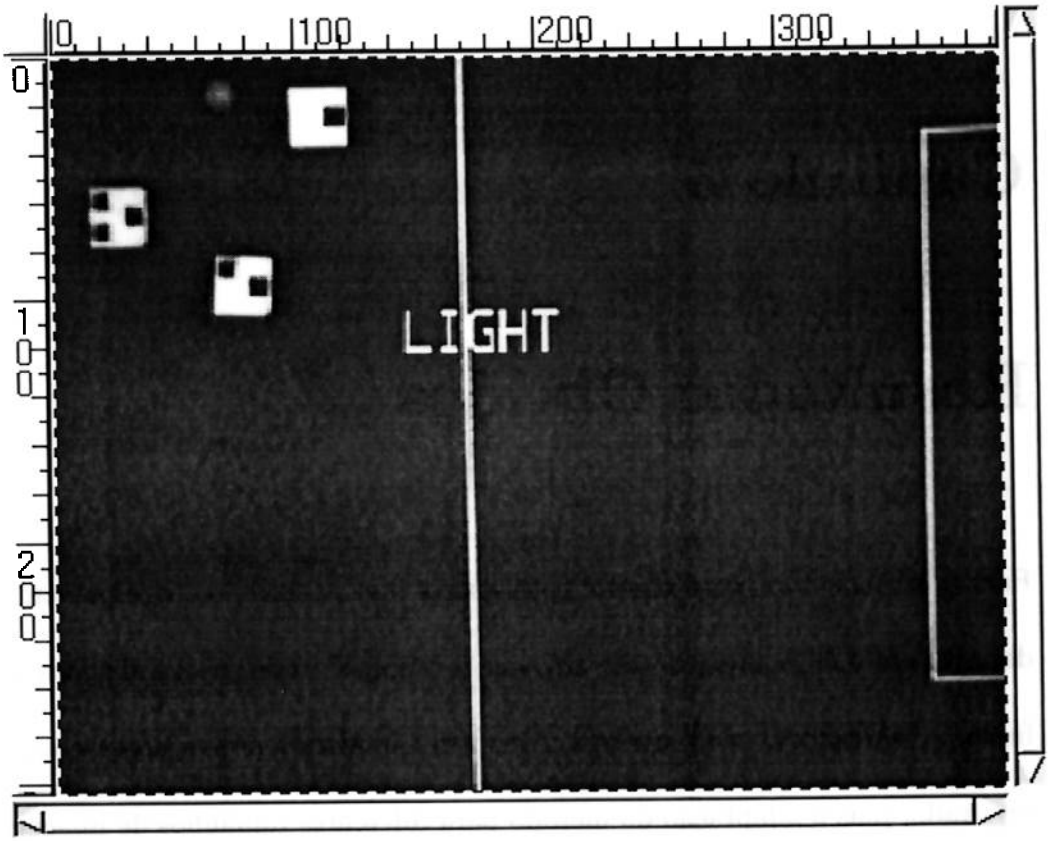
Resultados Obtidos

Para a execução do programa foi obtida uma foto utilizando o equipamento descrito em 4.2. A imagem não abrange o "campo" todo, pois a disposição física do laboratório não o permite. Mas isso não afeta o processamento ou o resultado, pois a adaptação do método para diferentes tamanhos de imagem consiste em se alterar os pixels limites considerados. A imagem pode ser vista na figura 5.1, página 60, com escala em pixels.

O treshold da imagem gerado pela execução do programa pode ser visto na figura 5.2, página 61.

Pode-se notar que a imagem não é perfeita, pois existem variações na iluminação que causam discrepância significativa entre as intensidades luminosas de cada pixel, fazendo com que alguns não sejam reconhecidos como

Figura 5.1: Imagem Utilizada para o Processamento, com Escala em Pixels



deveriam. Isso, no entanto, não constitui grande problema, pois o algoritmo para detecção é robusto a ponto de lidar com esse tipo de empecilho.

O operador gradiente aplicado ao treshold da imagem pela execução do programa pode ser visto na figura 5.3, página 62.

Pode-se notar, novamente, que existem diversos erros na formação do gradiente, erros que se devem exatamente à condição de iluminação presente, e que são acentuados por ser a segunda operação realizada à partir da

Figura 5.2: Treshold da Imagem Utilizada no Processamento

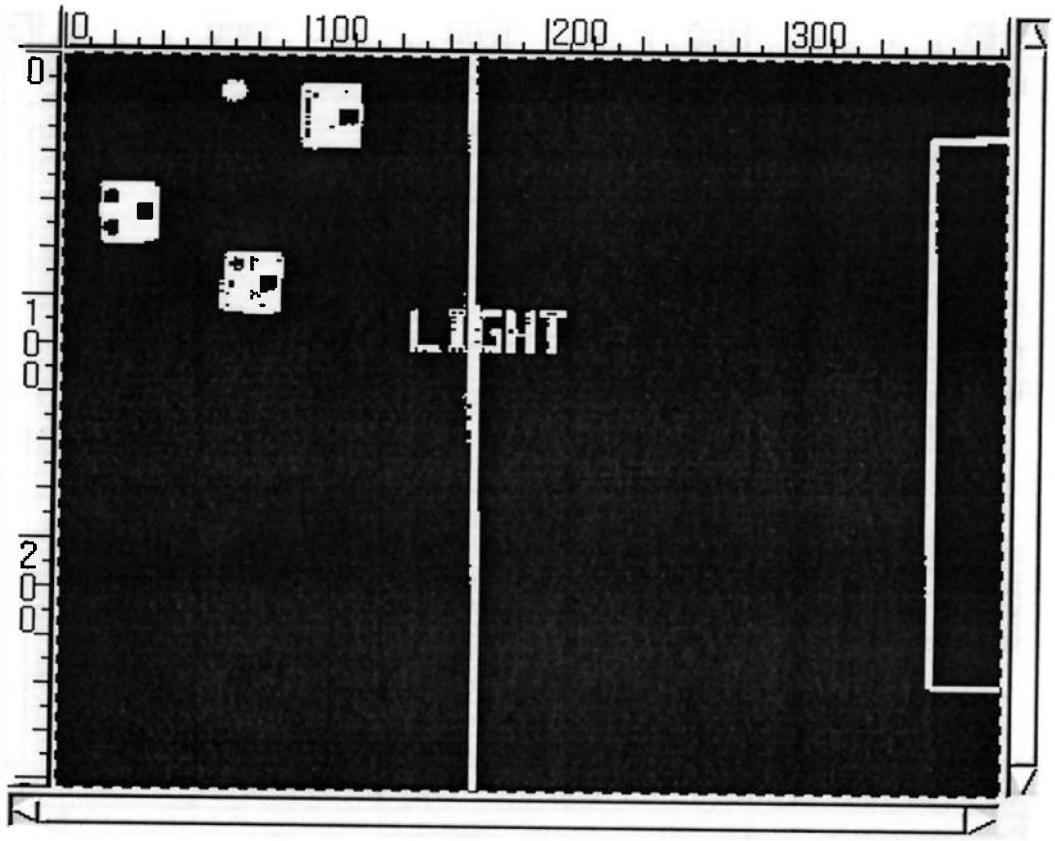


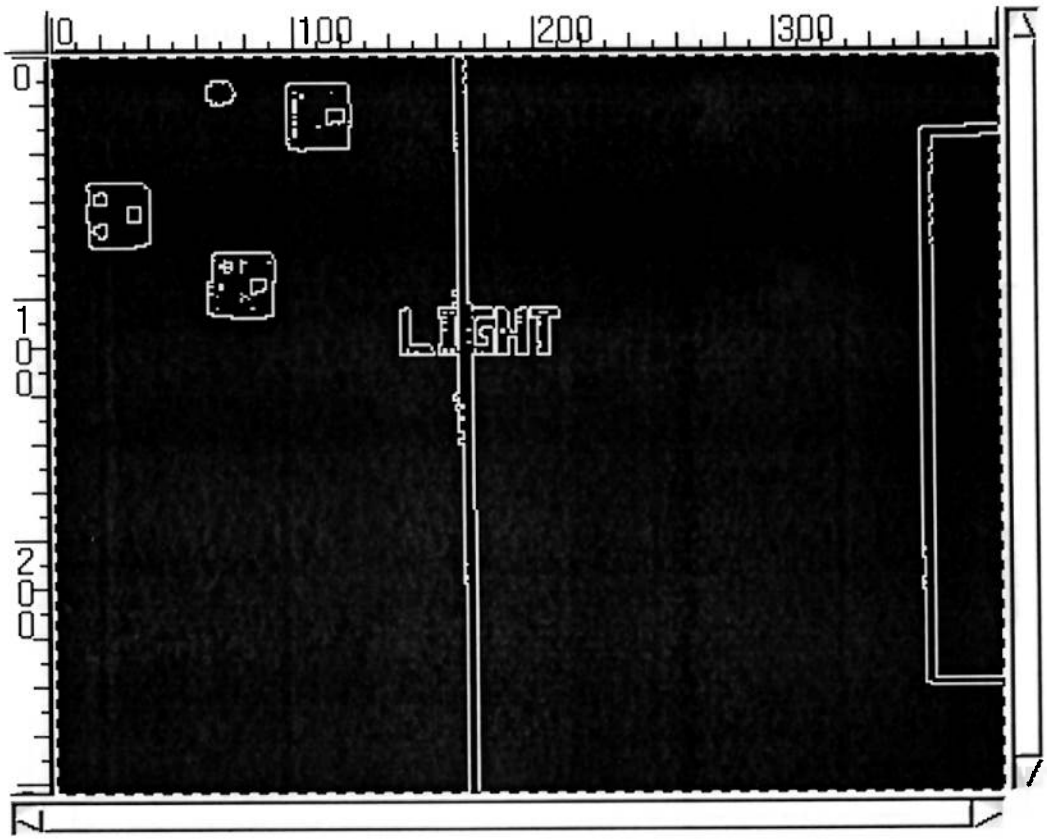
imagem. Novamente, o que nos interessa na figura são apenas os contornos dos objetos. Contornos esses que, como se pode ver, estão bem definidos e perfeitamente conectados.

Partindo desses contornos é feita localização dos objetos e, posteriormente, sua identificação e a obtenção das outras informações necessárias.

A saída fornecida pelo programa foi:

```
Bola: y: 15, x: 71
```

Figura 5.3: Operador Gradiente Aplicado ao Treshold




```
Robo ymin, ymax, xmin, xmax:  11 39 98 124

Centro em: y: 25, x: 111.

Circle(): Marcador: (y,x) 27, 117.

Robo numero 1.


Robo ymin, ymax, xmin, xmax:  52 79 15 40
```

```
Centro em: y: 65, x: 27.
Circle(): Marcador: (y,x) 68, 32.
Robo numero 3.

Robo ymin, ymax, xmin, xmax:  81 108 67 92
Centro em: y: 94, x: 79.
Circle(): Marcador: (y,x) 97, 84.
Robo numero 2.
```

Analisando a imagem original, pode-se constatar que o programa foi bem sucedido em todas as suas funções:

1. encontrou a bola;
2. encontrou todos os robôs;
3. encontrou a numeração correta de cada um e
4. acertou a marcação de controle.

O tempo de execução em uma máquina com processador Celeron 300 e 64Mb de memória foi de 0,313s, em média.

Um comentário cabe sobre esse resultado: a implementação do programa utilizada para efetuar esses testes lê a imagem a partir de um arquivo

em bitmap no formato .pgm (*Portable Gray Map*), pois o sistema de captação direta de imagens do projeto encontra-se em estado de desenvolvimento. Utilizando-se o utilitário *gprof.* do projeto GNU, obteve-se o seguinte resultado:

```
Each sample counts as 0.01 seconds.
  %   cumulative   self              self     total
 time   seconds   seconds   calls  us/call  us/call  name
 39.39      0.13     0.13       1 130000.00 130000.00  Follow
 27.27      0.22     0.09                 pgm_readpgminitrest
 18.18      0.28     0.06       1 60000.00 60000.00  circle
 15.15      0.33     0.05                 pgm_readpgminit
  0.00      0.33     0.00       1     0.00 190000.00  PixelSearch
```

Como pode-se ver, o tempo de processamento gasto pelas funções de inicialização e de leitura do arquivo gráfico demandam 42,42% do processamento, de onde podemos concluir que as funções necessárias à execução do programa em um sistema de processamento e controle completo demandariam apenas 0,190s.

Utilizando duas máquinas, em cluster, obteve-se o tempo total de 0,201s, que não é a metade do tempo de um só processador, como descrito no capítulo 2.

Como pode-se ver, mesmo utilizando apenas uma máquina que, para os padrões atuais já pode ser considerada ultrapassada, o tempo de processamento já pode ser considerado suficiente para que possa ser implementado no sistema de controle.

Capítulo 6

Comentários Finais

Analisando os resultados mostrados no capítulo 5, pode-se observar que o programa cumpriu todos os seus objetivos, encontrando todos os objetos e fornecendo todos os dados necessários para a correta implementação do sistema de controle.

Uma discussão cabe, porém, sobre o processo: sempre que se tem uma máquina executando um reconhecimento autônomo de uma imagem, recai-se sobre o problema da iluminação e das sombras.

A iluminação é um ponto fundamental no algoritmo desenvolvido, pois os processos de treshold e, posteriormente, da identificação dos componentes baseiam-se nesse dado. Isto significa que a iluminação deve ser controlada, pois ainda não existe um sistema computacional adaptativo que consiga elimi-

nar a influência da variação da iluminação em um processamento autônomo.

A ocorrência de sombras é também um fato grave, pois rotinas devem ser inseridas para tentar identificá-las e isolar o sistema original. Também ainda não existem métodos eficientes para isso, mas pesquisas tanto em iluminação quanto em remoção de sombras estão sendo feitas em diversas instituições.

Referências Bibliográficas

[1] Gonzalez, R.C.; Woods, R.E. *Digital Image Processing*. Addison-Wesley Publishing Company, Reading, MA, 1992.

[2] Gropp, W.; Lusk, E. *User's Guide for mpich, a Portable Implementation of MPI*, version 1.2.1, Mathemathics and Computer Science Division, Argonne National Laboratory, 2000.

[3] Gropp, W; Doss, N. *MPICH Model MPI Implementation Reference Manual*, Mathemathics and Computer Science Division, Argonne National Laboratory, 2000.

[4] Página da World Wide Web

http://www-unix.mcs.anl.gov/mpi/, como acessada em 01.06.2001.

[5] Página da World Wide Web

http://www.erc.msstate.edu/misc/mpi/, como acessada em 01.06.2001.

[6] Página da World Wide Web

http://www-unix.mcs.anl.gov/mpi/mpich/. como acessada em 01.06.2001.

[7] Documento da World Wide Web

http://www.fira.net/fira/f2001/docs/overview/98fira_rules.html. como acessado em 01.06.2001.

Índice Remissivo