UNIVERSIDADE FEDERAL DO ESPÍRITO SANTO
CENTRO TECNOLÓGICO
PROGRAMA DE PÓS-GRADUAÇÃO EM ENGENHARIA ELÉTRICA

JAINES DE OLIVEIRA BRAGANÇA

ESTRATÉGIAS PARA DESLOCAMENTO DE CARGAS ATRAVÉS DE COOPERAÇÃO ENTRE ROBÔS MÓVEIS A RODAS

VITÓRIA
2004

JAINES DE OLIVEIRA BRAGANÇA

ESTRATÉGIAS PARA DESLOCAMENTO DE CARGAS ATRAVÉS DE COOPERAÇÃO ENTRE ROBÔS MÓVEIS A RODAS

Dissertação apresentada ao Programa de Pós-Graduação em Engenharia Elétrica do Centro Tecnológico da Universidade Federal do Espírito Santo, como requisito parcial para obtenção do Grau de Mestre em Engenharia Elétrica, na área de concentração em Automação.
Orientadores: Prof. Dr. Teodiano Freire Bastos Filho e Prof. Dr. Mário Sarcinelli Filho.

VITÓRIA
2004

Dados Internacionais de Catalogação-na-publicação (CIP)
(Biblioteca Central da Universidade Federal do Espírito Santo, ES, Brasil)

Bragança, Jaines de Oliveira, 1978-

B813e Estratégias para deslocamento de cargas através de cooperação entre robôs móveis a rodas / Jaines de Oliveira Bragança. – 2004.

122 f. : il.

Orientador: Teodiano Freire Bastos Filho.

Co-Orientador: Mário Sarcinelli Filho.

Dissertação (mestrado) – Universidade Federal do Espírito Santo, Centro Tecnológico.

1. Robôs móveis. 2. Teoria do controle não-linear. 3. Visão por computador. I. Bastos Filho, Teodiano Freire. II. Sarcinelli Filho, Mário. III. Universidade Federal do Espírito Santo. Centro Tecnológico. IV. Título.

CDU: 621.3

JAINES DE OLIVEIRA BRAGANÇA

ESTRATÉGIAS PARA DESLOCAMENTO DE CARGAS ATRAVÉS DE COOPERAÇÃO ENTRE ROBÔS MÓVEIS A RODAS

Dissertação submetida ao programa de Pós-Graduação em Engenharia Elétrica do Centro Tecnológico da Universidade Federal do Espírito Santo, como requisição parcial para a obtenção do Grau de Mestre em Engenharia Elétrica - Automação.

Aprovada em 22 de dezembro de 2004.

COMISSÃO EXAMINADORA

Prof. Dr. Teodiano Freire Bastos Filho
Universidade Federal do Espírito Santo
Orientador

Prof. Dr. Mário Sarcinelli Filho
Universidade Federal do Espírito Santo
Orientador

Prof. Dr. Evandro Ottoni Teatini Salles
Universidade Federal do Espírito Santo

Prof. Dr. Humberto Ferasoli Filho
Universidade Estadual Paulista "Júlio de Mesquita Filho"

DEDICATÓRIA

Dedico este trabalho à minha querida esposa, companheira e amiga, que sempre esteve ao meu lado, acreditando na conclusão deste trabalho, até mais que eu. Agradeço também aos meus pais, minha irmã e meu irmão, pelo amor e compreensão demonstrados, mesmo sob todas as adversidades encontradas ao longo desta caminhada. Fica aqui minha eterna gratidão pelos esforços imensuráveis e pelo carinho destas pessoas, os quais cooperaram muito para que tudo isto fosse possível. “Isto não é um final, mas sim um degrau a menos”.

AGRADECIMENTOS

Quero agradecer a todos aqueles que, de alguma forma, contribuíram para que este projeto fosse concluído. Aos professores Teodiano Freire Bastos Filho e Mário Sarcinelli Filho pela imprescindível orientação, sem a qual não seria possível alcançar os bons resultados deste trabalho. Aos demais professores do departamento desta instituição, fica meu profundo agradecimento, principalmente à professora e amiga Raquel Frizera Vassallo e ao professor Evandro Ottoni Teatini Salles, por terem ajudado na solução dos problemas que surgiram no decorrer deste trabalho.

Agradeço ainda aos alunos dos laboratórios LAI 1 e LAI 2, os quais foram mais que simples companheiros de laboratório, foram e serão verdadeiros amigos. Agradeço também ao aluno de iniciação científica Rafael Leal, que contribuiu muito na construção dos robôs, e principalmente, ao mestrando Paulo André, grande amigo e exímio programador, cuja ajuda foi fundamental para que eu entendesse os trabalhos desenvolvidos pelo grupo de pesquisa do INAUT de San Juan, e posteriormente pudesse desenvolver as soluções para os problemas que queria tratar.

Agradeço ao grupo de pesquisa do INAUT de San Juan, e principalmente, ao aluno de doutorado Carlos Sória, cuja ajuda foi fundamental para que este trabalho fosse executado.

Finalmente, deixo aqui meu eterno agradecimento à FACITEC, pela bolsa concedida durante este mestrado, sem a qual teria sido muito mais difícil concluí-lo.

SUMÁRIO

LISTA DE TABELAS

LISTA DE FIGURAS

RESUMO

Este trabalho trata do controle de robôs móveis cooperativos, usando para isto um controlador não linear de posição. A realimentação para o fechamento da malha de controle é feita por meio de uma câmera estacionária, localizada sobre o ambiente de trabalho dos robôs. Depois de capturar a imagem, a mesma é processada em um computador, o qual é responsável por gerar as diretivas de controle para os robôs. Utilizando apenas um controlador não linear de posição, foi possível desenvolver estratégias para deslocamento de cargas, usando para isto a cooperação entre dois robôs. Estas estratégias consistem em identificar e tratar os possíveis problemas relacionados ao deslocamento da carga, para só depois transporta-la. Além das estratégias para deslocamento de cargas, foram implementadas tarefas que permitem que um robô alcance um ponto destino ou siga uma trajetória pré-definida na Imagem. Outro tipo de tarefa implementada neste trabalho, e que utiliza o controlador não linear de posição, é o movimento dos robôs em formação, a saber: formação em fila e formação lado a lado.

ABSTRACT

This work deals with the control of cooperative robots, using a non linear position controller. The feedback of the control loop is carried out using a camera located over the environment where the robots actuate. After capturing an image, it is processed into a computer, which is responsible for generate the set point to the robots. It was used only a non linear position control and it was possible develop strategies for allowing the displacement of an object using two robots to push it. These strategies consist of analyzing the position and orientation of the object in order to displace it. In addition to these strategies, several tasks were developed by the robots, allowing them reaching a destination point or follow a predefined path. Also, tasks to the robots like formation in a line or side to side were implemented in this work.

1 INTRODUÇÃO

Situações onde cooperação entre dois ou mais elementos é necessária para realizar uma determinada tarefa, ou torná-la mais fácil, são comuns desde os primórdios da civilização humana. Com o crescente aperfeiçoamento da robótica, as aplicações de cooperação têm encontrado cada vez mais espaço entre os robôs móveis, seja em tarefas repetitivas ou situações onde exista algum risco à vida do homem, e até mesmo situações onde exista a impossibilidade da sua presença.

Robôs convencionais, dotados de sistemas sofisticados de sensoriamento e processamento, muitas vezes se tornam desinteressantes para aplicações de cooperação, principalmente pelo custo, que está diretamente ligado à quantidade de robôs necessários à tarefa. Uma abordagem simples para realizar cooperação, no intuito de executar uma determinada tarefa, tem sido o uso de robôs celulares. O fato de utilizar times de robôs celulares oferece redundância ao sistema, e ainda permite que os mesmos sejam pouco sofisticados e, consequentemente, baratos. Estes robôs, trabalhando em conjunto, geralmente dão melhores resultados, quando comparados com um único robô mais sofisticado e, conseqüentemente, mais caro [10].

O que torna a aplicação dos robôs celulares mais interessante, é o fato deles apresentarem baixo custo de construção, o que permite que eles sejam largamente utilizados em situações de alto risco, visto que a perda de um robô celular não representa o mesmo prejuízo que a perda de um robô mais sofisticado. O baixo custo deste tipo de robôs os torna interessantes também para o meio acadêmico. Uma grande vantagem no uso de robôs celulares está na possibilidade do sucesso de uma tarefa, mesmo que haja falha por parte de um ou mais robôs.

Como exemplo de aplicações de robôs celulares podemos citar situações onde seja necessário transportar grande quantidade de pequenos objetos em um determinado ambiente, onde os robôs poderiam trocar informações de obstáculos e objetos a serem transportados. Poderíamos imaginar, ainda, a manipulação ou o transporte de um objeto maior, onde um único robô não tenha condições de transportá-lo sozinho, sendo necessários mais robôs para que a carga possa ser distribuída entre os mesmos. Uma outra situação de cooperação seria o reconhecimento ou construção de um mapa de um

determinado ambiente, onde uma equipe de robôs celulares poderia trocar informações com o objetivo de mapear o ambiente e evitar obstáculos. Uma outra aplicação dos robôs celulares cooperativos está no futebol de robôs, que já conta até com campeonatos mundiais, e que tem sido um grande laboratório para o desenvolvimento e o aperfeiçoamento de novas técnicas de controle.

O uso de robôs celulares tem avançado bastante, desde o campo da indústria e serviços até o uso militar e exploração espacial. A simplicidade e o baixo custo deste tipo de robô fazem dele o preferido em situações perigosas, tanto para humanos como para robôs mais caros e sofisticados.

Um exemplo de aplicação de robôs celulares no espaço é o sistema projetado pelo Jet Propulsion Laboratory (JPL), do Califórnia Institute of Technology. Este laboratório, que trabalha em conjunto com a NASA, participou, entre outras coisas, do desenvolvimento do famoso *PathFinder*. Eles estão desenvolvendo agora uma série de robôs pequenos, chamados *Space Rovers*, que servirão para a construção de uma base planetária em Marte [13]. A Figura 1 mostra a evolução de um experimento com estes robôs.

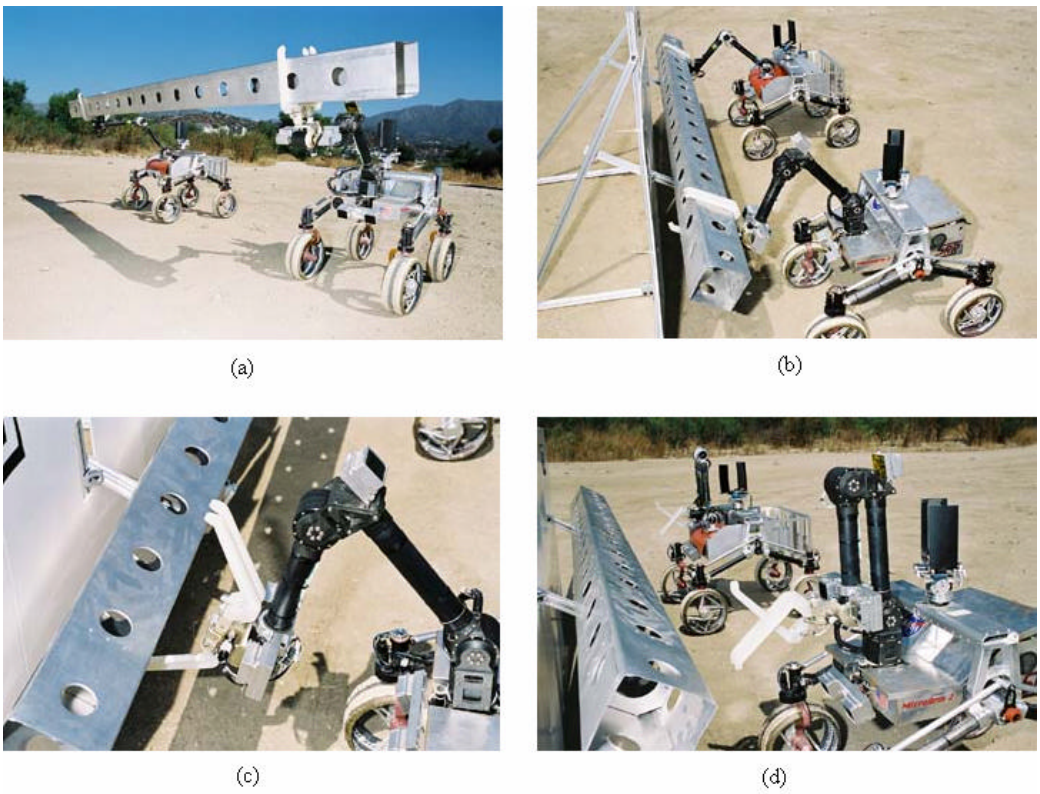


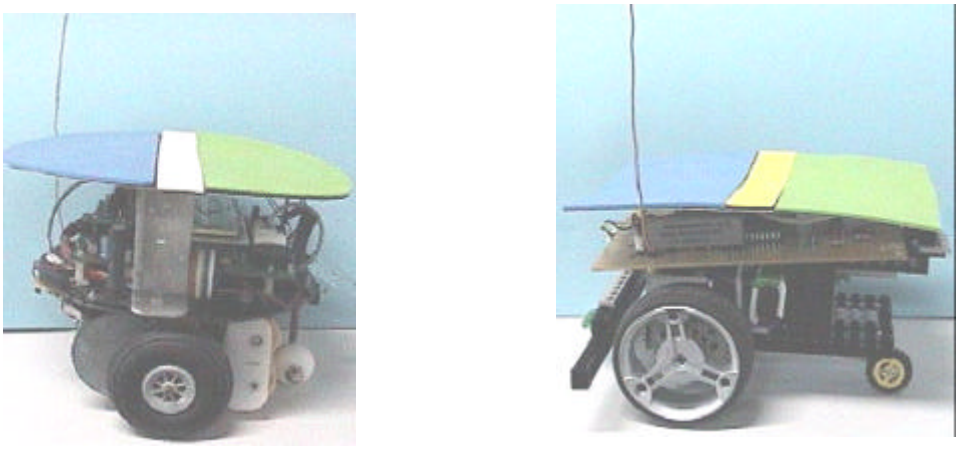
Figura 1 – Cooperação entre robôs do JPL.

O objetivo deste trabalho é controlar dois robôs celulares para realizar o deslocamento de uma carga específica, usando um controlador não linear de posição e realimentação visual. Existem alguns problemas que impedem que seja feito o deslocamento da carga. Estes problemas são tratados à medida que aparecem e, posteriormente, depois de resolvidos, efetuamos o deslocamento da carga propriamente dito. Entre os principais problemas que surgem, quando do deslocamento da carga, está o problema onde os dois robôs estão em lados opostos da caixa. Outro problema comum é quando os dois robôs estão em lados opostos da caixa e não há possibilidade que um dos dois contorne a mesma, pelo fato desta estar com suas duas extremidades próximas às bordas do tablado.

Além das estratégias para deslocamento de cargas, foram implementadas tarefas que permitem que um robô alcance um ponto destino ou siga uma trajetória pré-definida na Imagem. Outro tipo de tarefa implementada neste trabalho, e que utiliza o controlador não linear de posição, é o movimento dos robôs em formação, a saber: formação em fila e formação lado a lado.

1.1 O Sistema de Sensoriamento e Controle Implementado

Para a realização das tarefas de cooperação, nesta dissertação, foram usados dois robôs celulares, um chamado *Rug Warrior*, o qual é mostrado na Figura 2 (a), e outro chamado *Lego*, o qual é mostrado na Figura 2 (b), os quais são descritos com mais detalhes no Apêndice A. Estes dois robôs são identificados por tarjas de cores diferentes posicionadas sobre os mesmos, como veremos no Capítulo 2.



(a) (b)

Figura 2 – Os dois robôs utilizados para cooperação.

O espaço de trabalho destes dois robôs se restringe a um tablado com fundo preto de dimensões 165 cm de largura por 225 cm de comprimento, espaço este delimitado pela resolução de 640x480 *pixels* da câmara webcam, a qual está localizada sobre o mesmo tablado, a uma altura de aproximadamente 3m.

Para não fugir da característica de baixo custo, intrínseca dos robôs celulares, adotamos a idéia de uma única câmara localizada sobre o espaço de trabalho, ao invés de uma câmara para cada robô. Desta forma, com um sistema de visão global, é possível concentrar a aquisição da imagem de todo o sistema em uma só câmara. As imagens capturadas pela webcam são enviadas ao computador, externo aos robôs, através de um cabo USB.

A opção por usar a webcam se deu pela sua relação custo-benefício, pois conseguimos implementar um sistema de controle relativamente complexo e robusto, com bons resultados, sem a necessidade de adquirir uma câmara mais sofisticada.

Depois de processar estas imagens, o computador aplica as leis de controle e calcula os níveis de tensão para cada roda de cada robô. Posteriormente, são enviados pacotes de informações para os robôs, usando para isto a porta serial, um transmissor próximo ao computador e um receptor embarcado em cada robô. Este pacote contém a informação de qual robô deverá receber e processar as informações, bem como os níveis de tensão das rodas, e ainda bytes de verificação de transmissão.

Em função de um convênio promovido pela CAPES entre a Universidade Federal do Espírito Santo - Brasil (UFES) e a Universidad Nacional de San Juan – Argentina (UNSJ), o doutorando Carlos Sória da UNSJ esteve na UFES por seis meses, onde foi dado início às pesquisas na área de robôs celulares cooperativos. Parte do software de controle desenvolvido em C++ pela equipe da UNSJ foi utilizada, principalmente a parte que contempla a aquisição e o tratamento das imagens. O Software de controle inclui as rotinas necessárias para a aquisição e o processamento das imagens, o cálculo das ações de controle para as diferentes estratégias que se queira implementar, a comunicação através da porta serial do computador, através de transmissor sem fio, a coleta dos dados relativos a cada experimento para posterior análise, bem como a interface gráfica com o usuário.

O sistema completo conta com um computador Pentium IV 2,8GHz, um par transmissor-receptor da Radiometrix que opera na freqüência de 418MHz, um par transmissor-receptor da Radiometrix que opera na freqüência de 433MHz, dois robôs celulares e uma câmara de vídeo tipo webcam da Clone. Por não disponibilizarmos de dois transmissores de RF de mesma freqüência, é que utilizamos dois transmissores operando em freqüências diferentes. A Figura 3 fornece uma visão global do sistema implementado

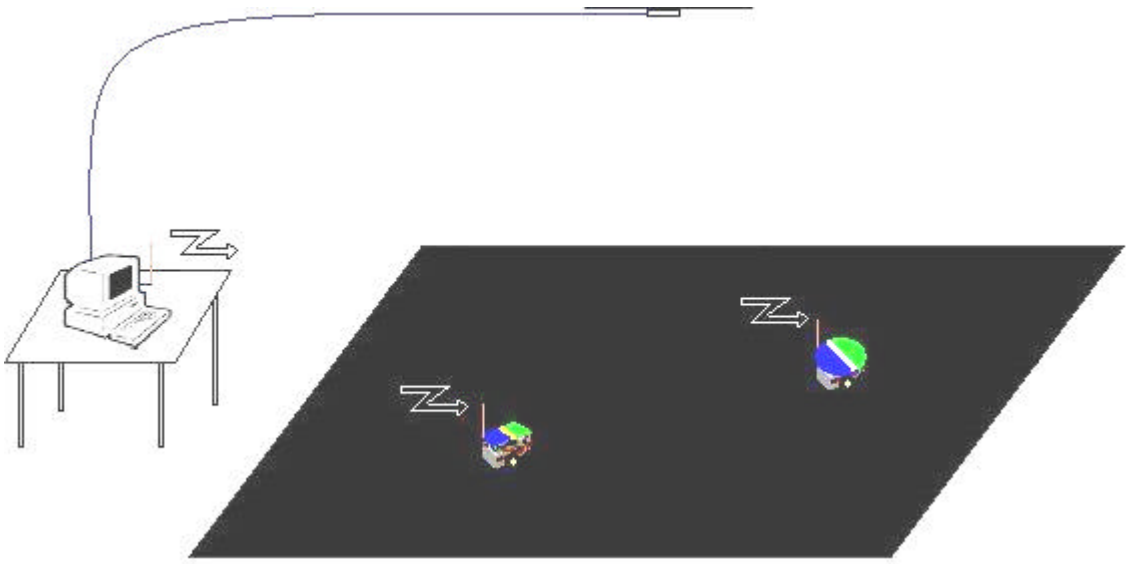


Figura 3 – Visão global do sistema.

1.2 Estrutura da Dissertação

Esta dissertação está organizada da seguinte forma: o Capítulo 2 descreve o sistema de visão artificial implementado, desde a aquisição e tratamento da imagem até a identificação dos parâmetros do sistema. O Capítulo 3 discute todos os aspectos relativos ao projeto do sistema de controle, incluindo ainda as informações do filtro a – ß e as estratégias de cooperação implementadas, utilizando somente um controlador de posição. No Capítulo 4 são mostrados os experimentos realizados e discutidos seus resultados. No Capítulo 5 fazemos as considerações finais e definimos propostas de trabalhos futuros. Além destes capítulos, este trabalho tem três apêndices. No Apêndice A são mostrados os aspectos construtivos dos robôs utilizados neste trabalho. No Apêndice B é feita a formulação geométrica utilizada nas estratégias de cooperação para empurrar a caixa. Finalmente, no Apêndice C são mostradas as características do software utilizado no sistema de controle.

2 O PROCESSAMENTO DE IMAGENS

Por se tratarem de robôs simples e baratos, os robôs celulares usados neste trabalho não são dotados de sensores embarcados para auxílio à navegação, visto que a aquisição dos sensores elevaria o custo para construção dos robôs, adicionando-se ainda a aquisição de um canal de transmissão robô-computador que permitisse comunicação nos dois sentidos (comunicação duplex). Dentro desta proposta de construção de um sistema de baixo custo, surgiu a necessidade de um sistema de realimentação eficiente, robusto, e, principalmente, barato, com o intuito de extrair as informações do plano de trabalho dos robôs para processamento em um computador.

O sistema de realimentação, para a estratégia de controle utilizada neste trabalho, foi visão artificial. O principal atrativo na utilização de visão artificial está na grande quantidade de informação contida na imagem adquirida, aliada ao baixo custo para implantação de um sistema de realimentação visual, que se resume, basicamente, a uma câmara webcam de baixo custo. A webcam está localizada estrategicamente sobre o plano de trabalho dos robôs, com o objetivo de supervisionar e enviar as informações para o computador. Uma limitação do uso de uma câmara global, ao invés de uma em cada robô, é que o espaço de trabalho se resume a um retângulo delimitado pela resolução da câmara. Uma forma de contornar este problema seria trabalhar com mais de uma câmara, o que, porém, não é feito neste trabalho.

2.1 Introdução

A webcam utilizada neste trabalho, mostrada na Figura 4, foi a *Clone Fashion Cam*, modelo 11064, com resolução máxima de 640 x 480, ou seja, 300K *pixels*. Esta webcam é conectada ao computador através de conexão USB (versão 2.0), a uma taxa de 30 fps (*frames per second*) e fornece em sua saída o sinal já no formato digital.

Figura 4 – Câmara webcam utilizada.

A visão artificial pode ser definida como os processos de obtenção, caracterização e interpretação das imagens tomadas do mundo real tridimensional. Estes processos podem ser subdivididos em seis etapas [9]. São elas:
aquisição,
pré-processamento,
segmentação,
caracterização,
reconhecimento e
interpretação.

Neste capítulo discutiremos todos os aspectos relacionados ao processamento de imagens utilizado neste trabalho. Todas as funções de manipulação das imagens foram implementadas utilizando a biblioteca para processamento de imagens IPLv2.5 (*Image Processing Library*) da Intel [14]. Esta biblioteca foi desenvolvida pela Intel para utilização em C e C++, e conta com uma grande variedade de funções para o processamento das imagens.

2.2 Processamento de Imagens Coloridas

A utilização de imagens coloridas nos fornece uma quantidade maior de informações, quando comparadas com representação de imagens em níveis de cinza, visto que nas imagens coloridas é possível discernir entre milhares de matizes e

intensidades de cores distintas, enquanto imagens com representação em níveis de cinza apresentam apenas alguns níveis.

Através da combinação das cores primárias, é possível formar outras cores. As cores primárias são: o vermelho, identificado pela letra R do inglês *red*, verde, identificado pela letra G do inglês *green*, e azul, identificado pela letra B do inglês *blue*. Em 1931, depois de uma convenção, chegou-se a uma normalização dos comprimentos de onda para estas três cores, cujos valores são 700nm para a cor vermelha, 546,1nm para a cor verde e 435,8nm para a cor azul. Vale lembrar que, a menos que se permita uma variação nos comprimentos de onda das cores primárias, não é possível gerar todas as cores do espectro visível.

Existem dois tipos de cores primárias, as cores primárias de luz e as cores primárias de pigmento. As cores primárias de luz são o vermelho, o verde e o azul. Já as cores primárias de pigmento são o amarelo, o cyan e o magenta (padrão de cores utilizado nas impressoras). A Figura 5 mostra as cores primárias de luz, e a Figura 6 mostra as cores primárias de pigmento.

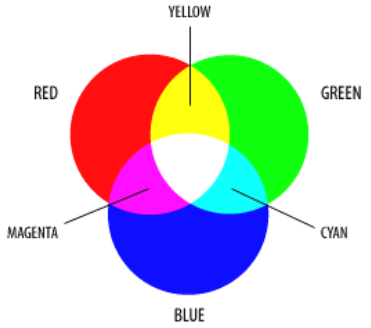


Figura 5 – Cores primárias de luz.

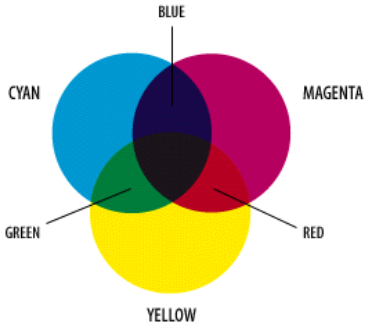


Figura 6 – Cores primárias de pigmento.

2.2.1 Modelo de Cor RGB

A maioria das câmaras coloridas usadas para adquirir imagens digitais utiliza o formato RGB. Além disso, o formato RGB é utilizado também por monitores de vídeo, o que o torna um modelo importante para processamento de imagens [11]. Por este motivo o modelo de cor RGB é o modelo empregado neste trabalho.

O modelo de cor RGB é baseado em um sistema de coordenadas cartesianas tridimensional, onde cada cor é representada por um único ponto no espaço, e consiste em três planos de imagens independentes, um para cada cor primária, os quais se somam para formar a imagem resultante [12]. O subconjunto de interesse é o cubo unitário mostrado na Figura 7. Por conveniência, os valores das cores primárias foram normalizados, motivo este que gerou um cubo unitário.

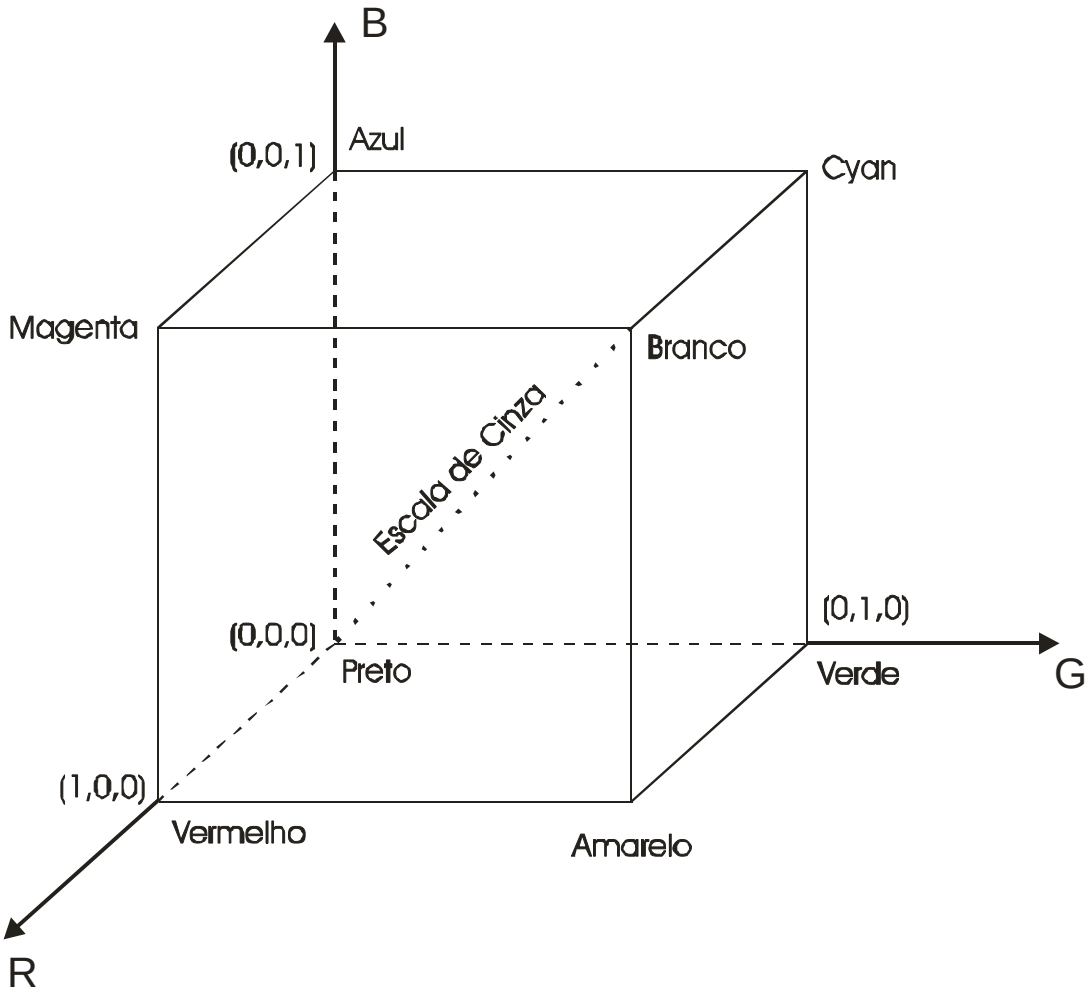


Figura 7 – Cubo unitário RGB.

Podemos observar na Figura 7 que as cores vermelho, verde e azul se encontram em três vértices, as cores amarelo, magenta e cyan em três outros vértices, a cor preto está na origem e a cor branco está no vértice mais distante da origem. Neste modelo, a escala de cinza é a linha pontilhada que se estende desde a cor preto até a cor branco, e as cores são pontos dentro ou na superfície do cubo, definidos por vetores com origem no ponto (0,0,0) [11].

2.3 Aquisição e Tratamento das Imagens

A Webcam é responsável por capturar a imagem e convertê-la para a forma digital no sistema RGB. Uma imagem digital é uma imagem $f(x,y)$ discretizada tanto nas coordenadas espaciais como na intensidade. Uma imagem digital em cores pode ser considerada como um conjunto de matrizes cujo índice de linha e coluna identificam um ponto na imagem, e o valor do elemento correspondente àquela posição indica a intensidade da cor naquele ponto [9]. Os elementos de uma distribuição digital deste tipo se denominam elementos da imagem, ou mais comumente *pixels* ou *pels*, do inglês "*picture elements*".

2.3.1 O filtro de mediana

Depois da etapa de aquisição da imagem, é interessante empregar alguma ou algumas técnicas para melhorá-la, para que ela se torne mais adequada para alguma aplicação específica. Existem duas categorias de técnicas que podem ser empregadas. A primeira categoria são métodos no domínio do espaço, onde as implementações são feitas no próprio plano da imagem, manipulando os *pixels* da imagem de forma direta. A outra categoria consiste em processar a imagem no domínio da freqüência, utilizando para isto a transformada de Fourier da imagem. As técnicas utilizadas neste trabalho são todas técnicas no domínio do espaço.

A imagem capturada sempre apresenta ruídos de quantificação, além de pontos ilhados que podem prejudicar a etapa de segmentação. Para evitar este tipo de

problema optamos por utilizar um filtro com característica de suavizar a imagem. O filtro escolhido foi o filtro de mediana, o qual é um filtro não linear.

O filtro de mediana opera em cada uma das três matrizes R, G e B, substituindo o valor de um determinado *pixel* pela mediana dos seus *pixels* vizinhos. Este tipo de filtro é muito interessante quando o padrão de ruído consiste em picos elevados na imagem, e o que se deseja preservar são os detalhes das bordas.

A mediana *m* de um conjunto de valores é o elemento do conjunto que o separa ao meio, onde a primeira metade é composta por valores menores que *m* e a segunda metade por valores maiores que *m* [11]. Para aplicar o filtro de mediana em um conjunto de *pixels*, devemos ordenar os *pixels* vizinhos em ordem crescente, para depois escolher a mediana, para então atribuir a mediana calculada ao valor do *pixel*. Seja o exemplo a Figura 8, onde queremos aplicar o filtro de mediana em uma vizinhança 3x3 no *pixel* localizado na segunda linha e segunda coluna. Organizando os valores em ordem crescente temos (10, 15, 20, 20, 20, 20, 20, 25, 100), cujo resultado é uma mediana de 20. A principal função do filtro de mediana é forçar os pontos com intensidades diferentes a serem mais parecidos com sua vizinhança, eliminando picos de intensidade que aparecem isolados na área da máscara do filtro [11].

10	20	20
20	15	20
20	25	100

Figura 8 – Aplicação do filtro de mediana usando máscara 3x3.

Na Figura 9 temos uma fotografia típica de Lena contaminada com um tipo de ruído chamado “ruído sal e pimenta”. A Figura 10 apresenta o resultado da aplicação do filtro de mediana a tal imagem.

Figura 9 – Imagem com ruído.

Figura 10 – Imagem resultante da aplicação do filtro de mediana.

Na Figura 11 podemos observar a imagem binária correspondente apenas à tarja branca do robô *Rug Warrior*. A Figura 12 mostra esta mesma imagem depois da aplicação do filtro de mediana.

Figura 11 – Imagem binária com ruído da tarja branca do *Rug Warrior*.

Figura 12 – Imagem binária sem ruído da tarja branca do *Rug Warrior*.

Aproveitando uma característica da biblioteca Intel, não precisamos aplicar o filtro de mediana em toda a imagem, mas somente na parte da imagem que contém o robô, no intuito de reduzir o tempo de processamento, visto que o filtro de mediana é aplicado a cada *frame* capturado pela câmara. A parte da imagem que contém o robô é chamada de ROI (*Region of Interest*), e será discutida nas próximas seções.

2.4 Detecção e Cálculo das Coordenadas dos Robôs e da Caixa

Uma das tarefas em cooperação que os robôs executarão, neste trabalho, é empurrar uma caixa. Para detectar e calcular a posição dos robôs e da caixa a ser transportada, é preciso lançar mão de algumas técnicas para extrair as informações desta imagem.

2.4.1 Segmentação da imagem

O primeiro passo para o cálculo das posições dos robôs e da caixa consiste em segmentar a imagem. Neste trabalho, a segmentação foi utilizada para destacar as partes da imagem que interessavam.

O tipo de segmentação utilizado neste trabalho é o método de segmentação por limiar. Neste tipo de segmentação é definido um valor de limiar para cada canal, e a segmentação é aplicada em um canal de cada vez. Todo *pixel* que tiver intensidade maior que o valor de limiar recebe o valor máximo possível, e todo *pixel* que tiver intensidade menor que o valor de limiar recebe o valor mínimo possível. Neste trabalho os valores de mínimo e máximo são 0 e 255, respectivamente. A equação

$$g(x,y) = \begin{cases} 1 \ se \ \ f(x,y) > T \\ 0 \ se \ \ f(x,y) \le T \end{cases} \tag{2-1}$$

fornece a expressão que define a imagem segmentada, onde $g(x,y)$ é a intensidade do ponto (x,y) após a segmentação, $f(x,y)$ é a intensidade original do ponto (x,y), e T é o valor do limiar.

2.4.2 Cálculo da posição e orientação dos robôs e da caixa

Como pode ser visto na Figura 13, os robôs são identificados com as tarjas azul na parte da frente, e verde na parte traseira, além de uma tarja específica para cada robô. Esta tarja é branca para o *Rug Warrior* e amarela para o *Lego*. A caixa a ser

empurrada é identificada por duas tarjas, uma de cor magenta e outra de cor cyan. A escolha das cores amarelo, magenta e cyan se deu pela facilidade que se tem de manipular os canais RGB para adquirir estas cores.

Depois de segmentar a imagem e separar as diferentes cores das tarjas, é necessário calcular a posição destas, para então sabermos as posições e orientações dos robôs e da caixa. Este cálculo de posição das tarjas é feito utilizando o conceito de momento, como definido na física clássica, substituindo o valor da massa pelo valor da intensidade do *pixel*.

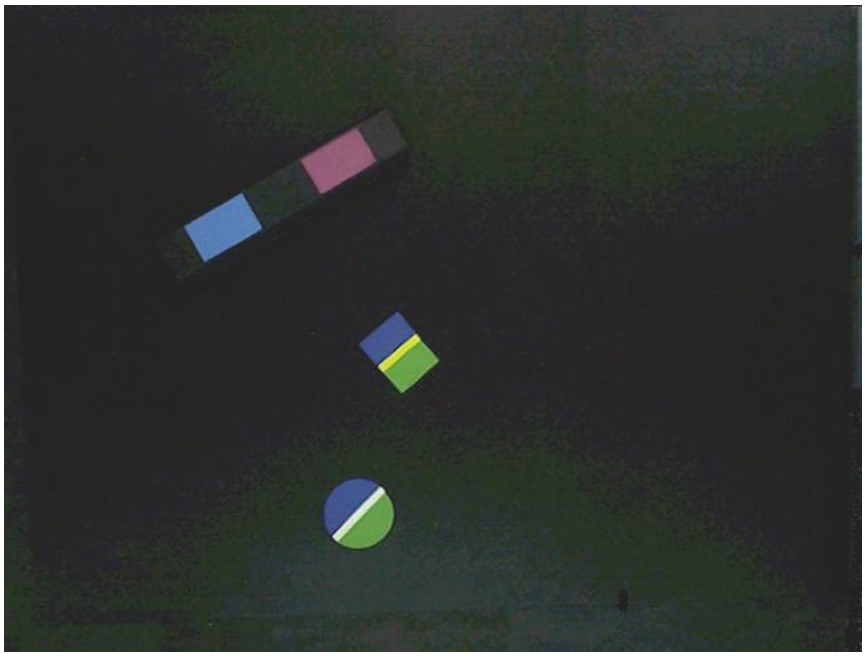


Figura 13 – Robôs e caixa no espaço de trabalho.

Dois importantes tipos de momento são utilizados, o momento espacial $M_U(m,n)$ e o momento central $U_U(m,n)$, os quais são características de uma imagem [14]. Estes dois momentos são definidos, respectivamente, segundo as expressões

$$M_U(m,n) = \sum_{j=0}^{nLinhas-1} \sum_{k=0}^{nCols-1} x_k^m \, y_j^n \, P_{j,k} \tag{2-2}$$

e

$$U_U(m,n) = \sum_{j=0}^{nLinhas-1} \sum_{k=0}^{nCols-1} (x_k - x_0)^m \left(y_j - y_0\right)^n P_{j,k} \qquad (2\text{-}3)$$

onde

$P_{j,k}$ intensidade do *pixel* (j, k),

(x_k, y_j) coordenadas do *pixel* (j, k),

m, n inteiros,

(x_0, y_0) coordenadas do centro de gravidade da imagem.

As coordenadas (x_0, y_0) do centro de gravidade da imagem são dadas em função do momento espacial $M_U(m,n)$, de acordo com as expressões

$$x_0 = \frac{M_U(1,0)}{M_U(0,0)} \qquad (2\text{-}4)$$

e

$$y_0 = \frac{M_U(0,1)}{M_U(0,0)}. \qquad (2\text{-}5)$$

A soma dos expoentes m e n é definida como a ordem do momento. A biblioteca IPL permite o cálculo de momentos até ordem 3.

Com a teoria descrita acima, podemos calcular o centro de gravidade das tarjas e, conseqüentemente, dos robôs e da caixa. Como veremos mais adiante, quando é dado início ao processo, não sabemos onde estão os robôs, sendo necessário calcular os centros de gravidade das tarjas branca e amarela para localizá-los. Uma vez que a ROI foi definida, não precisamos calcular mais os centros de gravidade para estas duas tarjas dentro do *loop* de controle, podendo calcular os centros de gravidade dos robôs, implicitamente, utilizando os centros de gravidade das tarjas azul e verde de cada ROI. A expressão que permite calcular o centro de gravidade de cada robô em função dos centros de gravidade das tarjas azul e verde é

$$Xc = \frac{xAzul + xVerde}{2} \tag{2-6}$$

e

$$Yc = \frac{yAzul + yVerde}{2} \tag{2-7}$$

onde

(*Xc*,*Yc*)	coordenadas do centro de gravidade do robô,
(*xAzul*, *yAzul*)	coordenadas do centro de gravidade da tarja azul,
(*xVerde*, *yVerde*)	coordenadas do centro de gravidade da tarja verde.

Quanto às coordenadas dos centros de gravidade das tarjas magenta e cyan da caixa, estas só precisam ser calculadas uma vez, no início do processo.

Depois de calcular as posições dos robôs e da caixa, podemos calcular a orientação destes. A Figura 14 nos mostra os ângulos a serem calculados.

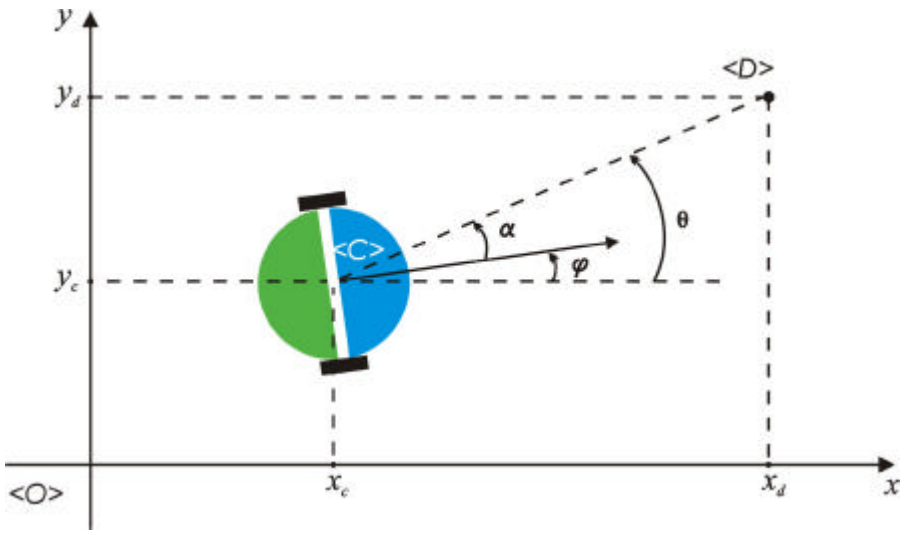


Figura 14 – Ângulos do robô.

As expressões que definem os ângulos da Figura 14 são

$$j = \tan^{-1}\left(\frac{yAzul - yVerde}{xAzul - xVerde}\right), \quad (2\text{-}8)$$

$$q = \tan^{-1}\left(\frac{y_d - y_c}{x_d - x_c}\right) \quad (2\text{-}9)$$

e

$$a = q - j\ , \quad (2\text{-}10)$$

onde

<*O*>	Origem do Sistema de Coordenadas
<*C*>	Centro do Robô
<*D*>	Ponto destino
(x_c, y_c)	coordenadas do centro de gravidade do robô,
(x_d, y_d)	coordenadas do ponto destino do robô,
(*xAzul*, *yAzul*)	coordenadas do centro de gravidade da tarja azul,
(*xVerde*, *yVerde*)	coordenadas do centro de gravidade da tarja verde,
f	ângulo de orientação do robô,
?	ângulo de inclinação da reta que liga o centro do robô ao destino,
a	erro de orientação do robô em relação ao destino.

O cálculo do ângulo de inclinação da caixa, mostrado na Figura 15, é dado pela expressão

$$y = \tan^{-1}\left(\frac{YC - YM}{XC - XM}\right), \quad (2\text{-}11)$$

onde

?	ângulo de inclinação da caixa a ser empurrada,
(*XM*, *YM*)	coordenadas do centro de gravidade da tarja magenta, e
(*XC*, *YC*)	coordenadas do centro de gravidade da tarja cyan.

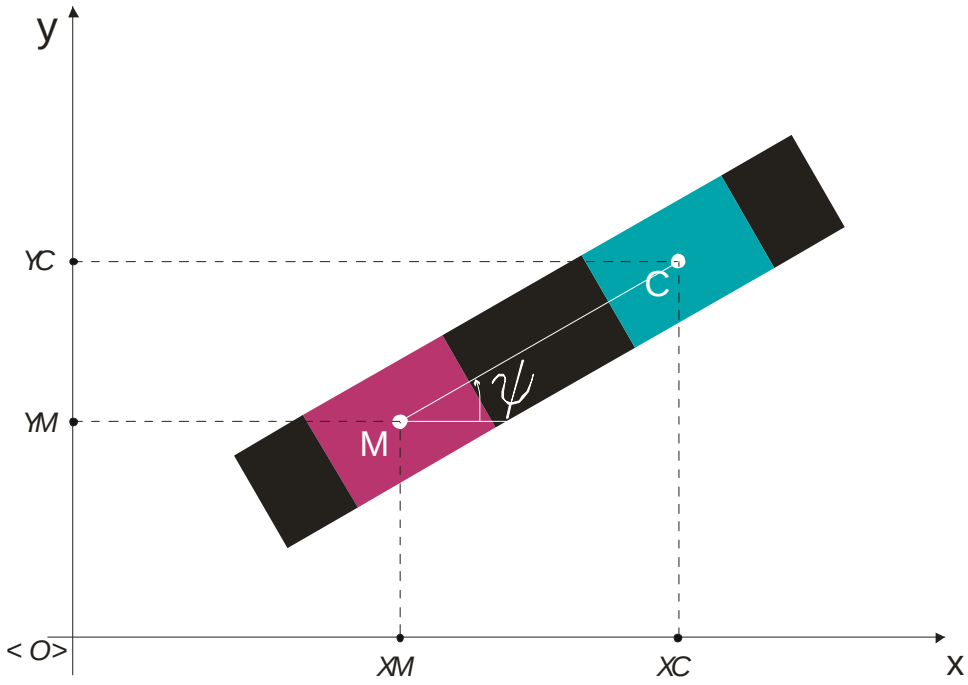


Figura 15 – Ângulo de inclinação da caixa.

Vale lembrar que todos os cálculos realizados neste trabalho são feitos no plano da imagem, usando o sistema de coordenadas da câmara, e não no plano de trabalho. Esta estratégia é chamada *controle por visão direta* [2], pois não realiza transformações de coordenadas do plano da imagem para o plano de trabalho. A vantagem desta estratégia é evitar problemas associados com a calibração da câmara e cálculos de transformações de coordenadas, pois as medições de erros, utilizadas na realimentação do sistema, são realizadas diretamente nas coordenadas do plano da imagem. Se a opção fosse trabalhar no sistema de coordenadas do plano de trabalho, seria necessário usar uma matriz de transformação (calibração), que contém informações como distância focal, altura e inclinação da câmara, etc., sendo que esta matriz deveria ser recalculada toda vez que a câmara sofresse alguma variação de posição, o que não acontece neste trabalho.

2.4.3 Processamento das imagens dentro e fora do *loop* de controle

Depois de discutidas separadamente as etapas para o processamento de imagens, veremos como estas etapas se juntam para realizar este processamento. O fluxograma da Figura 16 ilustra o processamento de imagens, tanto fora quanto dentro do *loop* de controle.

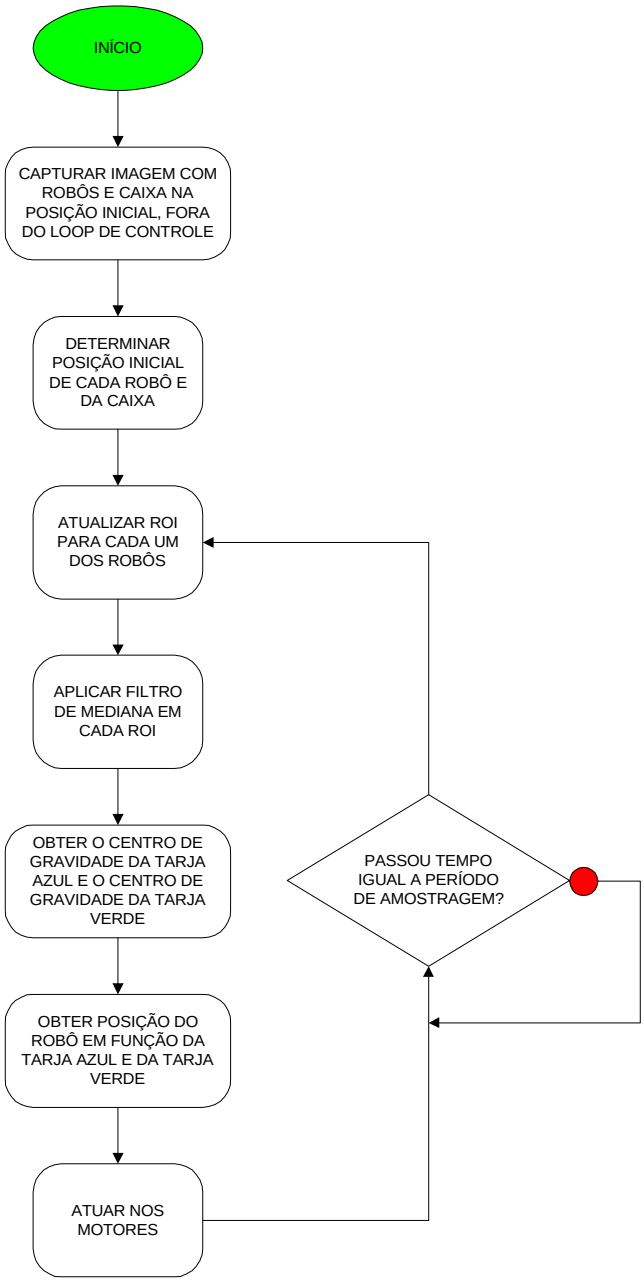


Figura 16 – Fluxograma do processamento de imagens.

Inicialmente, fora do *loop* de controle, capturamos a imagem com os robôs e a caixa nas suas posições iniciais. Uma vez que as tarjas foram detectadas por

segmentação, procedemos com o cálculo da posição inicial dos robôs e da posição das duas tarjas da caixa.

Em função da posição inicial dos robôs, definimos uma ROI (*Region of Interest*) para cada um, para que não seja necessário procurar pelos robôs por toda a imagem, a cada nova imagem adquirida, reduzindo drasticamente o tempo de processamento dentro do *loop* de controle. Para que isto aconteça é necessário que a ROI de cada robô seja atualizada a cada passagem no *loop* de controle, para que o robô sempre se encontre dentro dela.

Agora que temos duas pequenas imagens, uma para a ROI de cada robô, aplicamos o filtro de mediana, descrito anteriormente, no intuito de minimizar ruídos quando da captura da imagem.

Como o tamanho da ROI é relativamente pequeno, não existe a possibilidade de termos dois robôs dentro de uma mesma ROI de um dado robô. Com isso, não precisamos mais levar em consideração as tarjas que diferenciam os robôs, somente as tarjas azul e verde localizadas dentro da ROI, exceto quando um dos robôs não for localizado, sendo necessário reinicializar o sistema e procurar pelos robôs e pela caixa por toda a imagem novamente. A partir da média dos centros de gravidade das cores azul e verde, é possível determinar a posição do robô, em função da qual são calculados os níveis de tensão para cada motor. Depois de calculados, estes níveis de tensão são enviados aos motores, fechando assim o fluxograma da Figura 16.

3 O SISTEMA DE CONTROLE

Para projetar o sistema de controle é necessário levantar o modelo matemático do robô. O modelo matemático é dividido em duas partes, um modelo cinemático e um modelo dinâmico, os quais diferem, basicamente, pelo tipo de abordagem no equacionamento das variáveis que regem o movimento do robô.

3.1 Modelo Cinemático do Robô

O modelo cinemático fornece o conjunto de equações que regem o movimento do robô. Para obter o modelo cinemático do robô devemos considerá-lo como um ponto de massa e dimensões desprezíveis, e ainda admitir que não haja interação de qualquer espécie entre componentes do robô.

Consideremos um robô posicionado a uma distância diferente de zero de um ponto destino <D>, de forma que seu movimento seja estabelecido através da combinação de sua velocidade linear *v* e de sua velocidade angular *?*. O ponto escolhido para representar o robô foi o ponto médio do eixo que liga as duas rodas, onde está o seu centro de gravidade. A Figura 17 mostra as variáveis utilizadas no modelo cinemático, e é o ponto de partida para a dedução das equações cinemáticas. Em tal figura tem-se

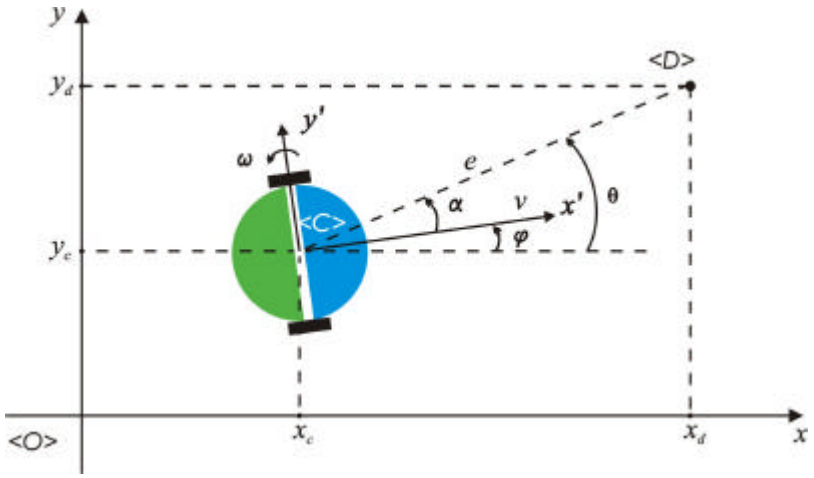


Figura 17 – Obtenção do modelo cinemático de um robô móvel a rodas.

$\{x, y\}$ sistema de coordenadas do plano de trabalho, com origem em <*O*>

$\{x', y'\}$	sistema de coordenadas do robô, com origem em <C>
(x_c, y_c)	coordenadas do centro de gravidade do robô
(x_d, y_d)	coordenadas do ponto destino
v	velocidade linear do robô no sistema de coordenadas $\{x, y\}$
?	velocidade angular do robô no sistema de coordenadas $\{x, y\}$
e	erro de posição do robô em relação ao ponto destino
f	ângulo de orientação do robô
?	ângulo de inclinação da reta que liga o centro do robô ao ponto destino
a	erro de orientação do robô em relação ao ponto destino.

De acordo com [3], o conjunto de equações que relacionam a posição e o ângulo de orientação do robô é

$$\dot{x}_c = v\cos j\ , \tag{3-1}$$

$$\dot{y}_c = vsenj\ , \tag{3-2}$$

e

$$j = w\ . \tag{3-3}$$

Através de tais equações cartesianas podemos chegar ao modelo cinemático expresso em coordenadas polares. A representação polar facilita o projeto dos controladores em malha fechada, que serão usados no controle dos robôs. Antes de chegarmos às equações em coordenadas polares, podemos obter facilmente algumas relações a partir da Figura 17, que são

$$e = \sqrt{(x_d - x_c)^2 + (y_d - y_c)^2}\ , \tag{3-4}$$

$$q = \tan^{-1}\left(\frac{y_d - y_c}{x_d - x_c}\right), \tag{3-5}$$

e

$$a = q - j\ . \tag{3-6}$$

Depois de algumas operações com tais expressões, chegamos ao modelo cinemático expresso em coordenadas polares, a saber

$$\dot{e} = -v\cos a\,, \quad e > 0, \tag{3-7}$$

$$\dot{q} = v\,\frac{sen a}{e}\,, \tag{3-8}$$

e

$$\dot{a} = v\,\frac{sen a}{e} - w\,. \tag{3-9}$$

A equação (3-8) apresenta uma particularidade, pois, para obtê-la, imaginamos um observador no ponto <*D*> que vê o robô se movimentar com velocidade radial igual a *vsena*. A velocidade angular é a diferença das velocidades lineares nos dois pontos, dividida pela distância. Como a velocidade no ponto <*D*> é zero, a expressão da velocidade angular fica como expresso na equação de $\dot{q}$.

Vale lembrar que tais equações só têm valor para erro de posição *e* diferente de zero, uma vez que para um erro de posição igual a zero os valores de *a* e *?* são indeterminados e, conseqüentemente, os valores de $\dot{a}$ e $\dot{q}$ também o são.

3.2 Modelo Dinâmico do Robô

O modelo dinâmico do robô leva em consideração as interações entre os seus componentes, como a dinâmica elétrica e mecânica do motor, a roda e a estrutura mecânica do robô.

Para obtenção do modelo dinâmico do robô, consideramos que o mesmo seja movido por dois motores CC montados em uma estrutura com tração diferencial, e ainda consideramos que o centro de gravidade do robô está sobre o eixo do mesmo. A distância entre as rodas é *D*, o diâmetro das mesmas é 2*R* e as velocidades lineares das rodas esquerda e direita são, respectivamente, v_L *e* v_R, conforme mostra a Figura 18. Levando em consideração que a constante de tempo mecânica do motor é bem menor

que a constante de tempo elétrica, podemos, para fins de controle, desconsiderar a influência do pólo elétrico na resposta do sistema, e considerar apenas o pólo mecânico, o qual é o pólo dominante [2].

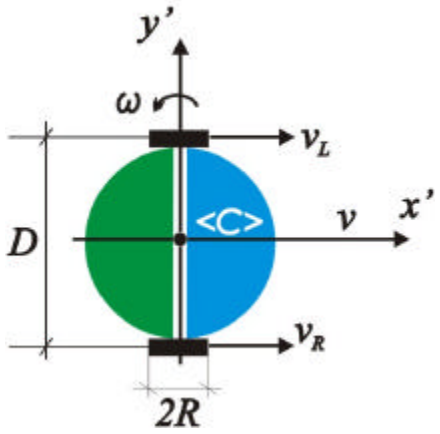


Figura 18 – Obtenção do modelo dinâmico de um robô móvel a rodas.

Para a configuração mostrada na Figura 18, a velocidade linear v e a velocidade angular $?$ do robô são facilmente obtidas através de

$$v = \frac{v_R + v_L}{2} \tag{3-10}$$

e

$$w = \frac{v_R - v_L}{D}, \tag{3-11}$$

onde

$$v_L = Rw_L \tag{3-12}$$

e

$$v_R = Rw_R . \tag{3-13}$$

Sabemos que a relação entre a tensão de entrada $u_i(t)$ e a velocidade angular do eixo do motor $?_i(t)$ é expressa, de acordo com [4] e [5], através das equações diferenciais

$$t\,\dot{w}_L(t) + w_L(t) = k u_L(t) \tag{3-14}$$

e

$$t\,\dot{w}_R(t) + w_R(t) = k u_R(t), \tag{3-15}$$

onde t representa a constante de tempo mecânica do robô, $i = L$ representa o motor da esquerda e $i = R$ representa o motor da direita. Multiplicando-se os dois lados das duas equações por R temos que

$$t\,\dot{v}_L(t) + v_L(t) = k\,R u_L(t) \tag{3-16}$$

e

$$t\,\dot{v}_R(t) + v_R(t) = k\,R\,u_R(t). \tag{3-17}$$

Somando as equações acima, e dividindo por dois, obtemos

$$t\,\dot{v}(t) + v(t) = \frac{k\,R}{2} u_{Comm}(t), \tag{3-18}$$

enquanto que subtraindo as mesmas equações, e dividindo por D, obtemos

$$t\,\dot{w}(t) + w(t) = \frac{k\,R}{D} u_{Diff}(t), \tag{3-19}$$

onde

$$u_{Comm} = u_R + u_L, \tag{3-20}$$

$$u_{Diff} = u_R - u_L, \tag{3-21}$$

$$u_R = \frac{u_{Comm} + u_{Diff}}{2}, \tag{3-22}$$

e

$$u_L = \frac{u_{Comm} - u_{Diff}}{2}. \tag{3-23}$$

Observe que foram definidos dois novos termos, u_{Comm} e u_{Diff} , os quais representam a tensão comum e a tensão diferencial, respectivamente, aplicadas aos motores do robô. Usando o ponto médio do eixo que liga as duas rodas como ponto de referência do robô, chegamos às Equações (3-18) e (3-19), as quais representam dois sistemas SISO (Single-Input-Single-Output) desacoplados, que podem ser controlados separadamente. A partir deste ponto, é possível projetar dois controladores distintos, um para o erro de orientação e outro para a velocidade do robô, usando para isto a tensão diferencial u_{Diff} e a tensão comum u_{Comm}, respectivamente [1].

3.3 Projeto do Controlador de Posição

Em função da natureza dos experimentos que desejamos implementar, precisamos de um controlador que seja capaz de levar um robô de um ponto inicial até um ponto destino. Por isso decidimos usar um controlador não linear de posição. O controlador de posição implementado neste trabalho e todas as suas demonstrações de estabilidade estão descritos em [2]. Este controlador é baseado nas medições que são feitas no plano da imagem. Este método, como vimos no Capítulo 2, é chamado de *controle por visão direta*. O controle de posição é feito através das tensões comum e diferencial, aplicadas aos motores. As equações do controlador são

$$U_{dif} = \frac{tD}{KR}\left(k_a a - k_b w\right), \tag{3-24}$$

$$U_{com} = \frac{2t}{KR}\left(k_c(e)\, e\cos a - k_d v\right) \tag{3-25}$$

e

$$k_c(e) = \frac{k_c}{a+e} \quad , \quad k_d > 0 \tag{3-26}$$

onde

U_{dif}	tensão diferencial
U_{com}	tensão comum
t	constante de tempo mecânica do robô
D	distancia entre os centros das rodas
R	raio da roda
a	erro de orientação do robô em relação ao ponto destino
v	velocidade linear do robô
$?$	velocidade angular do robô
k_a, k_b	parâmetros do controlador
k_c, k_d	parâmetros do controlador
e	erro de posição do robô em relação ao ponto destino.

Analisando as equações do controlador com mais rigor, em especial a da tensão diferencial U_{dif} , percebemos que não se trata de um controlador proporcional derivativo (PD), pois $w \neq \dot{a}$. Este tipo de controlador tem a vantagem de ir atualizando não somente a orientação j do robô, mas também o ângulo q de referência, permitindo uma correção simultânea do erro angular e da posição, evitando assim o problema mencionado em [9] para os controladores PD.

A não linearidade do controlador implementado aparece no cálculo da tensão comum U_{com} . Na expressão de U_{com} podemos observar dois termos. O primeiro regula a ação de controle de modo que quanto menor for o erro de orientação (mais próximo de zero), maior é a saída do controlador, devido à função *cos* (cosseno), fazendo com que a velocidade linear do robô seja relativamente grande. Por outro lado, quanto mais próximo de 90 graus for o erro de orientação, menor será a ação da tensão comum U_{com} nos motores, fazendo com que a velocidade linear seja praticamente nula, e que o robô só apresente velocidade angular. O segundo termo da expressão de U_{com} como no caso da tensão diferencial U_{dif} , é usado para dar mais suavidade ao movimento do robô.

Para que as ações de controle nos atuadores (motores) não fiquem na faixa de saturação nem na faixa de zona morta, é necessário determinar os valores das constantes k_c e a. Para o caso de saturação, o máximo valor de U_{com} é dado por

$$U_{com} = \lim_{e \to \infty} \left(\frac{k_c}{e + a} e \right) = k_c = U_{com\,\max} . \tag{3-27}$$

De posse do valor de $k_c = U_{com\ \max}$, vamos analisar a situação de zona morta, que tem como expressão

$$U_{com} = \lim_{e \to e_{\min}} \left(\frac{U_{com\,\max}}{e + a} e \right) = U_{com\,\min} . \tag{3-28}$$

Neste último caso, e_{min} corresponde à mínima distância que o sistema de visão pode identificar entre a posição do robô e a do ponto destino, e $U_{com\ \min}$ corresponde ao menor valor PWM a partir do qual os motores entram em movimento.

3.4 O Filtro Preditor a – ß

No sistema de controle implementado neste trabalho, a taxa com a qual as informações do espaço de trabalho são obtidas pela câmara é um fator primordial, pois os cálculos que dependem do fator tempo levam em consideração o espaço de tempo entre duas imagens consecutivas capturadas [1]. Como conseqüência desta restrição, é necessário que a captura seja feita freqüentemente. Assim como também é necessário que o valor da taxa seja pré-definido, de forma a permitir todo o processamento necessário entre dois quadros consecutivos, a fim de garantir a estratégia de controle em malha fechada.

Como exemplo desta dependência do tempo, podemos observar a equação

$$x_{Cest} = x_{Cant} + \frac{v_x}{T}, \tag{3-29}$$

onde o valor estimado (predito) de posição (x_{Cest}) é igual ao valor anterior (x_{Cant}) mais a variação estimada de posição em x (v_x/T, onde T representa o espaço de tempo entre dois quadros consecutivos).

Porém, todo processo de medição está sujeito a ruído. No nosso caso não é diferente, principalmente pelo fato dos valores de posição possuírem natureza discreta, característica intrínseca do processamento de imagens. O surgimento de ruído se torna mais evidente ainda para pequenas distâncias na imagem, à medida que o robô se aproxima do ponto destino. Este ruído, se não for tratado, se propaga para outras variáveis necessárias ao controle, como a velocidade e o erro de orientação. A solução para este tipo de problema é usar um filtro adequado para estimar a posição e a velocidade, a partir de observações de posição [1].

O filtro escolhido para ser implementado foi o filtro a – ß, uma variação do filtro de Kalman [8]. A principal vantagem do filtro a – ß em relação ao filtro de Kalman é que os ganhos são constantes, não sendo necessário recalculá-los a cada ciclo de captura de imagem, como seria feito em um filtro de Kalman convencional. O fato dos ganhos serem constantes é muito importante em uma implementação onde a janela de tempo entre duas aquisições de imagem representa todo o tempo que se tem para realizar o processamento.

3.4.1 Modelo matemático do ponto destino

Consideremos o modelo de um sistema dinâmico, em tempo discreto, em uma dimensão, descrito na forma de vetor de estados, o qual é composto pela posição e velocidade do robô. Consideremos, ainda, que a observação (medição) é obtida através das medições de posição [7]. Este modelo pode ser descrito como se segue:

Dinâmica:
$$\begin{bmatrix} x(k+1) \\ v(k+1) \end{bmatrix} = \begin{bmatrix} 1 & T \\ 0 & 1 \end{bmatrix} \begin{bmatrix} x(k) \\ v(k) \end{bmatrix} + \begin{bmatrix} T^2/2 \\ T \end{bmatrix} w(k) \tag{3-30}$$

Medição: $$z(k) = \begin{bmatrix} 1 & 0 \end{bmatrix} \begin{bmatrix} x(k) \\ v(k) \end{bmatrix} + h(k) \qquad (3\text{-}31)$$

Para tal conjunto de equações, tem-se que

$x(k)$	posição no instante k
$v(k)$	velocidade no instante k
T	período de amostragem
$z(k)$	medição de posição
$?(k)$	perturbação de manobra do ponto destino
$?(k)$	ruído de incerteza de medição

Tanto a perturbação de manobra do ponto destino quanto o ruído de medição são modelados com média zero (ruído estacionário branco) e variância $s_?^2$ e $s_?^2$, respectivamente.

A perturbação de manobra do ponto destino representa os parâmetros desconhecidos e as não-linearidades do robô.

3.4.2 Modelo do filtro a – ß

O filtro preditor a – ß é um caso especial do filtro de Kalman [8]. Para chegarmos às equações do filtro a – ß é necessário definir o índice de manobrabilidade $?$ [7], o qual é definido como a relação entre a perturbação de manobra do ponto destino e a incerteza de medição, ou seja,

$$l^2 \equiv T^4 \frac{s_w^2}{s_h^2}, \qquad (3\text{-}32)$$

sendo ainda proporcional ao período de amostragem. Uma vez que o índice de manobrabilidade está definido, o filtro ótimo também está implicitamente definido.

No nosso caso, o filtro a – ß se resume a um procedimento recursivo usado para estimar posição e velocidade em uma dimensão, usando para isto medições de posição. O fato dos movimentos nos eixos x e y serem desacoplados nos permite aplicar o filtro a – ß nas duas direções de forma independente. Devido a este desacoplamento, vamos simplificar e desenvolver as equações somente para a direção x, sendo possível aplicá-las na direção y, sem perda de generalidade. As equações do filtro preditor são [6]

Predição:
$$\begin{bmatrix} \hat{x}(k+1|k) \\ \hat{v}(k+1|k) \end{bmatrix} = \begin{bmatrix} 1 & T \\ 0 & 1 \end{bmatrix} \begin{bmatrix} \hat{x}(k|k) \\ \hat{v}(k|k) \end{bmatrix}, \tag{3-33}$$

Filtragem (Correção):
$$\begin{bmatrix} \hat{x}(k+1|k+1) \\ \hat{v}(k+1|k+1) \end{bmatrix} = \begin{bmatrix} \hat{x}(k+1|k) \\ \hat{v}(k+1|k) \end{bmatrix} + \begin{bmatrix} a \\ b/T \end{bmatrix} [z(k+1) - \hat{x}(k+1|k)], \tag{3-34}$$

$$a = -\frac{l^2 + 8l - (l+4)\sqrt{l^2+8l}}{8}, \tag{3-35}$$

$$b = \frac{l^2 + 4l - l\sqrt{l^2+8l}}{4}, \tag{3-36}$$

e

$$l^2 = \frac{b^2}{1-a} \tag{3-37}$$

onde

$\hat{x}(k|k)$: posição predita e filtrada no instante *k*

$\hat{v}(k|k)$: velocidade predita e filtrada no instante *k*

$\hat{x}(k+1|k)$: posição predita no instante *k*+1 em função da posição predita e filtrada no instante *k*

$\hat{v}(k+1|k)$: velocidade predita no instante $k+1$ em função da velocidade predita e filtrada no instante k

$\hat{x}(k+1|k+1)$: posição predita e filtrada no instante $k+1$

$\hat{v}(k+1|k+1)$: velocidade predita e filtrada no instante $k+1$

$z(k+1)$: posição medida no instante $k+1$

a, $ß$: constantes do filtro a – ß

l : índice de Manobrabilidade

3.5 Estratégias de Controle Utilizando o Controlador de Posição

O controle de posição implementado neste trabalho nos permite uma série de aplicações, as quais serão vistas nesta seção. No experimento de ponto destino é necessário que o usuário defina o ponto no qual o robô deverá chegar. No experimento de seguir trajetória o usuário deve definir os pontos que compõem a trajetória a ser seguida pelo robô. Tanto no experimento de ponto destino quanto no de seguir trajetória, o usuário define os pontos na imagem através do mouse.

Uma característica do controlador de posição e da realimentação visual, usados neste trabalho, é que não existe a garantia que o robô alcance o ponto com precisão, e sim uma área delimitada por um círculo em torno deste ponto. Desta forma admitimos que o robô alcançou um ponto toda vez que ele estiver a uma distância deste ponto que seja menor que um erro máximo admitido, valor este estipulado pelo usuário através da interface gráfica.

3.5.1 Alcançar um ponto destino na imagem

Uma vez definida uma posição inicial (x_c, y_c) do robô em uma imagem, a uma distância *e*, diferente de zero, de um ponto destino <*D*>, o objetivo de controle consiste em chegar ao ponto destino através do caminho mais curto, ou seja, uma reta. A Figura 19 mostra esta situação. Neste tipo de aplicação, basta passar as coordenadas do ponto destino <*D*> para o controlador de posição, e o robô então busca este ponto, através do controle de posição, e pára quando o robô apresenta um erro *e* menor que um limite, estabelecido também pelo usuário através da interface gráfica.

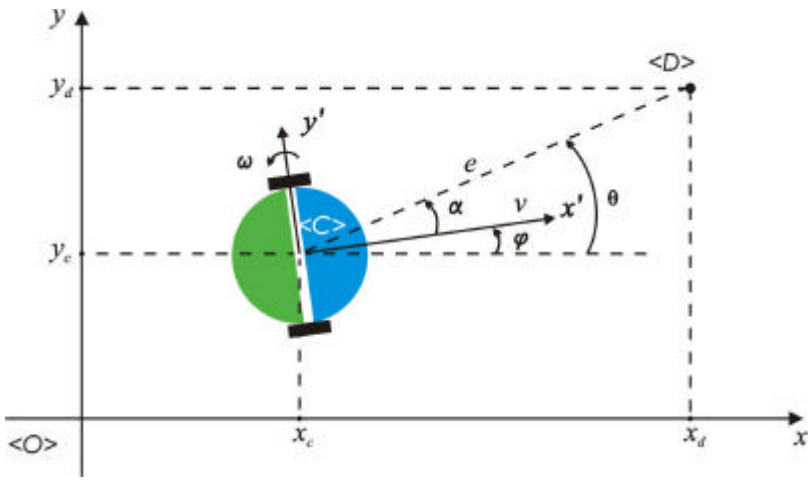


Figura 19 – Tarefa alcançar ponto destino.

3.5.2 Seguir uma trajetória pré-definida na imagem

Uma vez que o usuário tenha definido os pontos que compõem a trajetória a ser seguida pelo robô, como visto na Figura 20 (a), o objetivo de controle consiste em fazer com que o robô percorra a trajetória partindo do ponto inicial, passando pelos pontos intermediários, seqüencialmente, e chegue ao último ponto definido pelo usuário.

A estratégia de controle utilizada para que o robô siga esta trajetória consiste em fazer, inicialmente, com que o ponto destino seja o primeiro ponto da trajetória, como visto na Figura 20 (a). Á medida que o robô for entrando na zona deste ponto destino, incrementa-se o ponteiro que aponta para o ponto destino, de forma que o

novo ponto destino seja o próximo ponto da trajetória, como visto na Figura 20 (b). Esta idéia se repete até que o último ponto da trajetória seja alcançado, como visto na Figura 20 (c). Desta forma, podemos dizer que o robô chega ao destino final passando por destinos intermediários.

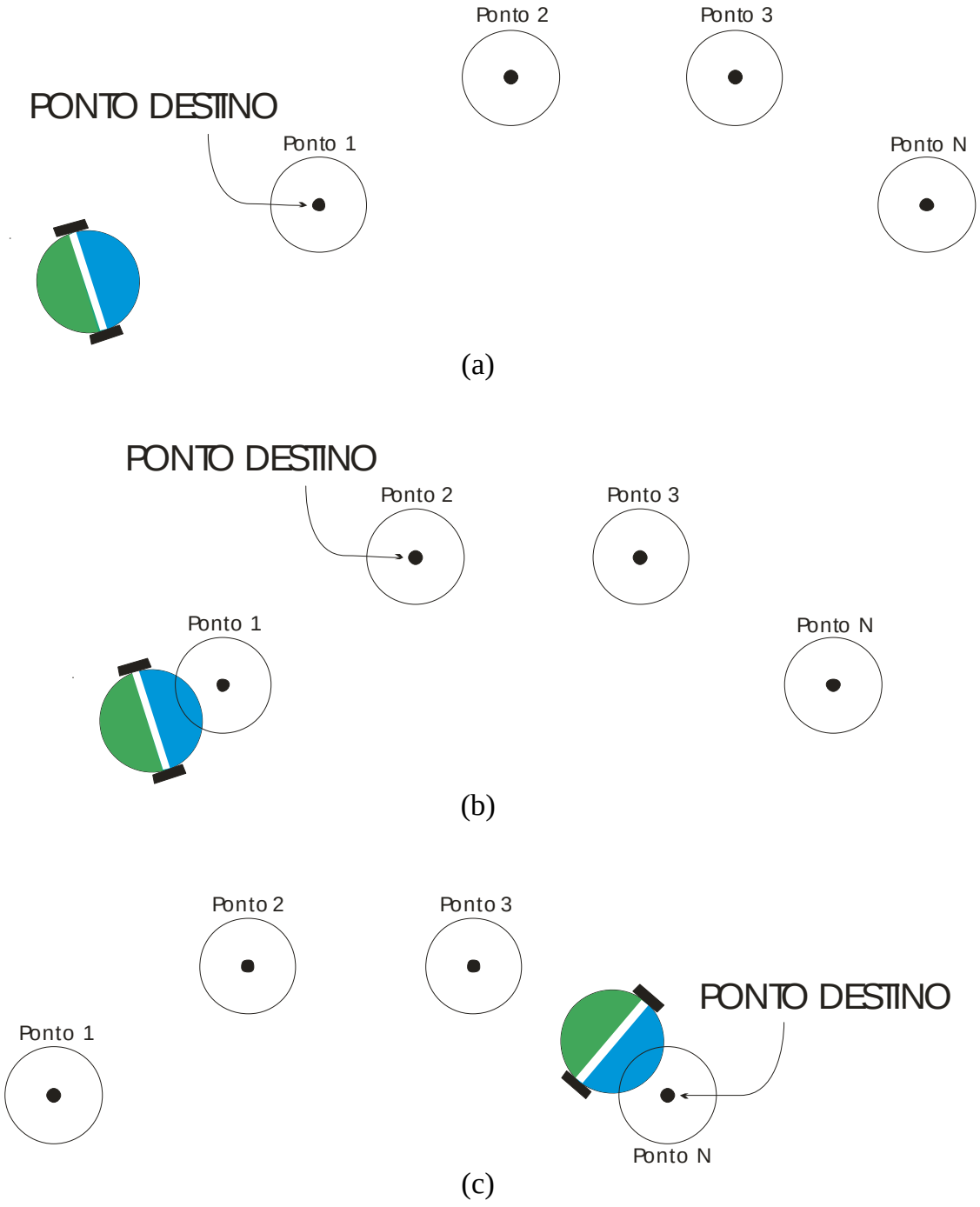


Figura 20 – Tarefa seguir trajetória.

3.5.3 Movimentos em formação

Um outro tipo interessante de estratégia de controle utilizando o controlador de posição está nos movimentos de vários robôs em formação. Este tipo de aplicação consiste em fazer com que os robôs se movimentem obedecendo a uma formação pré-determinada. Neste trabalho, implementamos dois tipos de formação, a formação em fila e a formação lado a lado, utilizando os dois robôs celulares mencionados no capítulo 1. Os movimentos em formação são interessantes por lançarem as bases para trabalhos cooperativos mais elaborados

Tanto no caso da formação em fila quanto na formação lado a lado, é necessário que um robô seja o mestre e o outro seja o escravo. O robô mestre é aquele que executa a trajetória estabelecida pelo usuário, enquanto que o robô escravo tem sua trajetória calculada em função da trajetória do robô mestre. A relação entre a trajetória do robô mestre e a trajetória do robô escravo é uma relação dinâmica, pois a cada instante o robô mestre muda de posição e, conseqüentemente, o robô escravo tem que fazê-lo.

3.5.3.1 Formação em fila

Como dissemos anteriormente, a trajetória do robô escravo é calculado em função da trajetória do robô mestre. A Figura 21 mostra a disposição dos robôs neste tipo de formação.

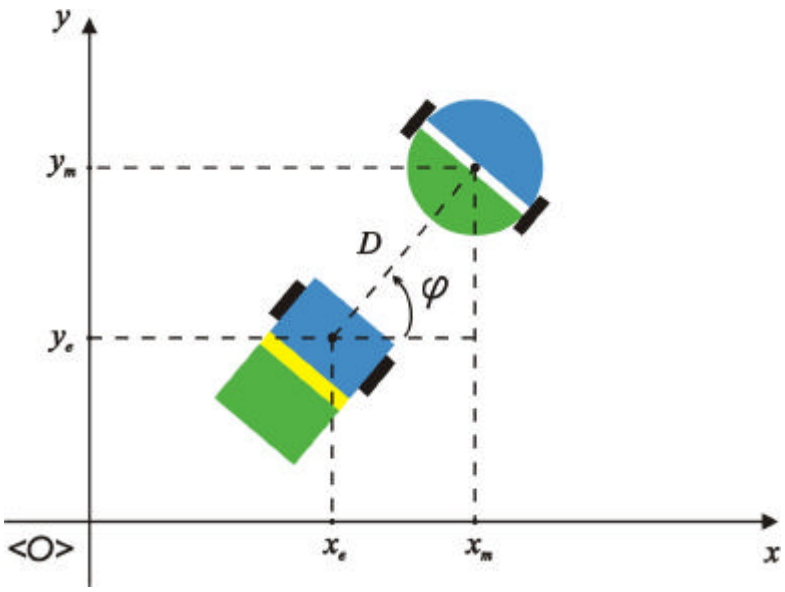


Figura 21 – Formação em fila.

As equações que determinam o ponto que o robô escravo deve seguir são

$$x_e = x_m - D\cos j \tag{3-38}$$

e

$$y_e = y_m - D\sin j \ , \tag{3-39}$$

onde o par (x_e, y_e) corresponde às coordenadas do robô escravo e o par (x_m, y_m) corresponde às coordenadas do robô mestre, D é a distância permitida entre os robôs e j é o ângulo de orientação do robô mestre.

3.5.3.2 Formação lado a lado

Analogamente ao caso de formação em fila, a Figura 22 mostra a disposição dos robôs em formação lado a lado.

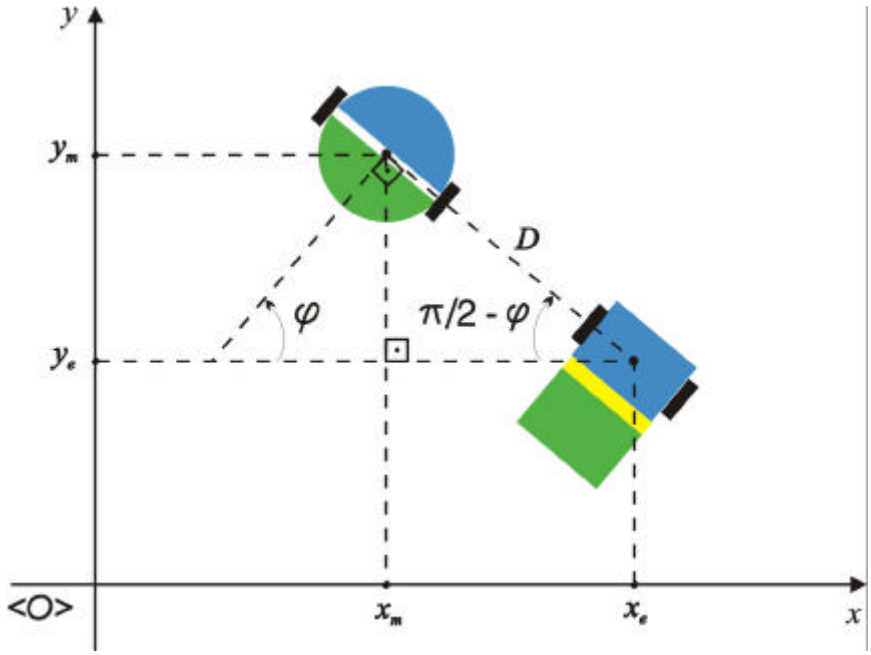


Figura 22 – Formação lado a lado.

As equações que determinam o ponto que o robô escravo deve seguir são

$$x_e = x_m + D\cos(p/2 - j) \tag{3-40}$$

e

$$y_e = y_m - D\,sen(p/2 - j), \tag{3-41}$$

onde o par (x_e, y_e) corresponde às coordenadas do robô escravo e o par (x_m, y_m) corresponde às coordenadas do robô mestre, D é a distância permitida entre os robôs e j é o ângulo de orientação do robô mestre.

3.5.4 Cooperação para manipulação e transporte de uma caixa

Entende-se por cooperação o processo no qual dois ou mais robôs trabalham em conjunto para realizar uma determinada tarefa. Uma das estratégias de controle que foi elaborada neste trabalho, e que faz uso do controlador de posição, é a cooperação entre robôs para manipular e empurrar uma caixa. O objetivo de empurrar a caixa é que esta seja retirada para fora da área de trabalho dos robôs, tarefa esta que poderia ser estendida para mais de uma caixa, no intuito de limpar uma determinada área. A caixa

a ser empurrada é identificada por duas tarjas, uma de cor magenta e outra de cor cyan, como visto na Figura 23. As coordenadas do centro de gravidade das tarjas magenta e cyan são facilmente calculadas, como foi visto no Capítulo 2.

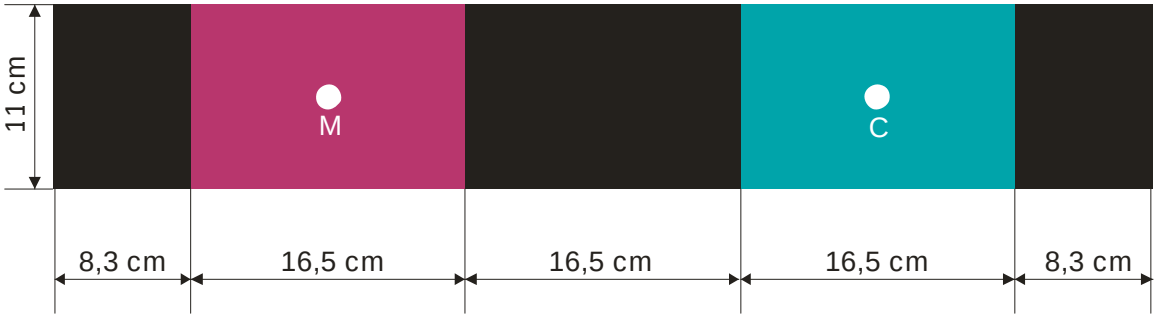


Figura 23 – Caixa a ser empurrada de forma cooperativa pelos robôs.

Depois de calcular o centro de gravidade de cada cor, podemos calcular o ângulo de inclinação da caixa, o qual é medido através de um vetor que sai do centro de gravidade da tarja magenta e chega ao centro de gravidade da tarja cyan, como visto na Figura 24.

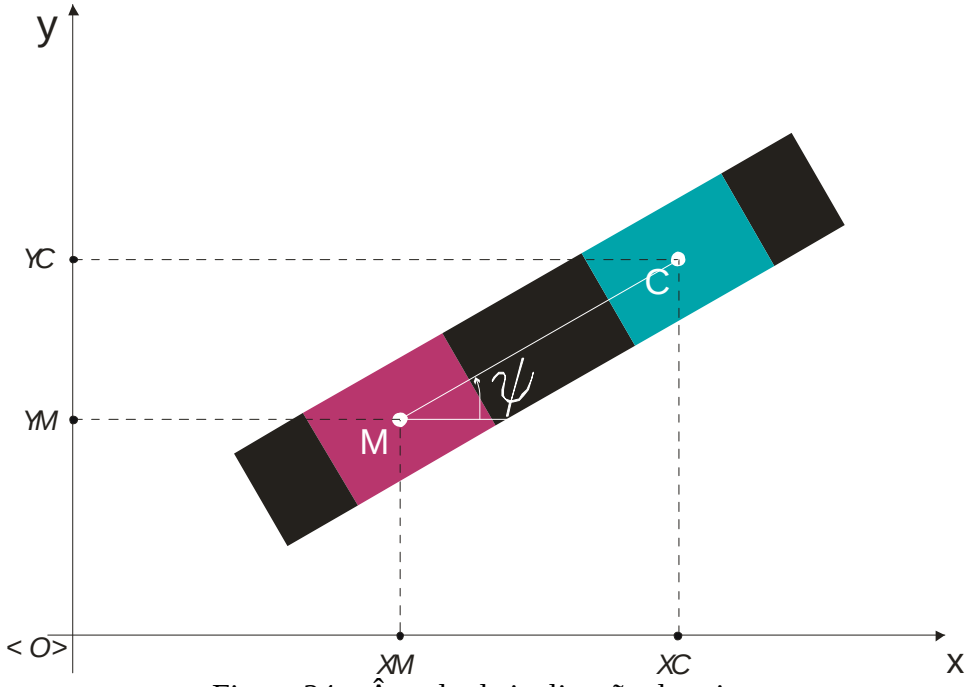


Figura 24 – Ângulo de inclinação da caixa.

A equação que define o ângulo de inclinação da caixa é

$$y = \tan^{-1}\left(\frac{YC - YM}{XC - XM}\right), \qquad (3\text{-}42)$$

onde

?	ângulo de inclinação da caixa a ser empurrada
(*XM*, *YM*)	coordenadas do centro de gravidade da tarja magenta
(*XC*, *YC*)	coordenadas do centro de gravidade da tarja cyan.

De posse dos valores do ângulo de inclinação e das coordenadas dos centros de gravidade das tarjas magenta e cyan da caixa, podemos começar a desenvolver a estratégia de empurrar a caixa, seguindo o fluxograma da Figura 25.

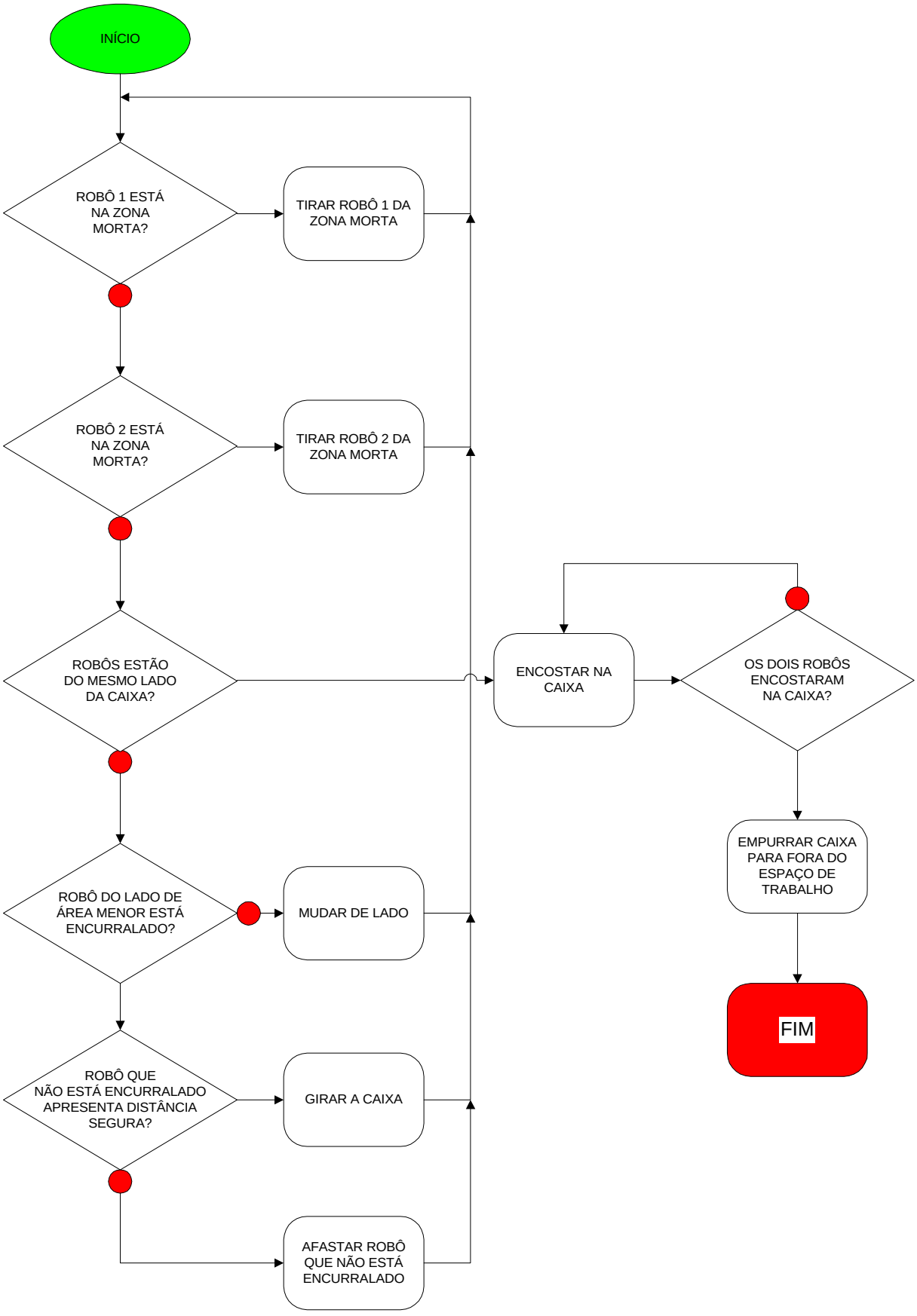


Figura 25 – Fluxograma da estratégia empurrar caixa.

Seguindo o fluxograma da Figura 25, a primeira coisa a ser observada, antes de empurrar a caixa, é se algum robô está na zona morta da mesma. Caso os dois robôs estejam na zona morta, procedemos com a retirada do *Rug Warrior*, para depois retirarmos o *Lego* da zona morta. A Figura 26 ilustra a situação onde o *Rug Warrior* está na zona morta.

Para determinar se um robô está ou não na zona morta, precisamos calcular a distância do centro de gravidade do robô até à reta que passa pelo eixo principal da caixa. A equação da reta *w* que passa pelo eixo principal da caixa, conforme o Apêndice B é

$$w: \quad -\tan(y)\,x + y - (YM - \tan(y)\,XM) = 0\,, \tag{3-43}$$

e a distância *Dzm* do centro de gravidade (*Xc1*, *Yc1*) do *Rug Warrior* até a reta *w*, também conforme o Apêndice B, é

$$Dzm = \left| \frac{-\tan(y)\,Xce + Yce - (YM - \tan(y)\,XM)}{\sqrt{(\tan(y))^2 + (-1)^2}} \right|. \tag{3-44}$$

Se a distância Dzm for menor que a soma da metade da largura do robô (D) com a metade da largura da caixa (2b), significa que o robô está na zona morta, como mostra a Figura 26.

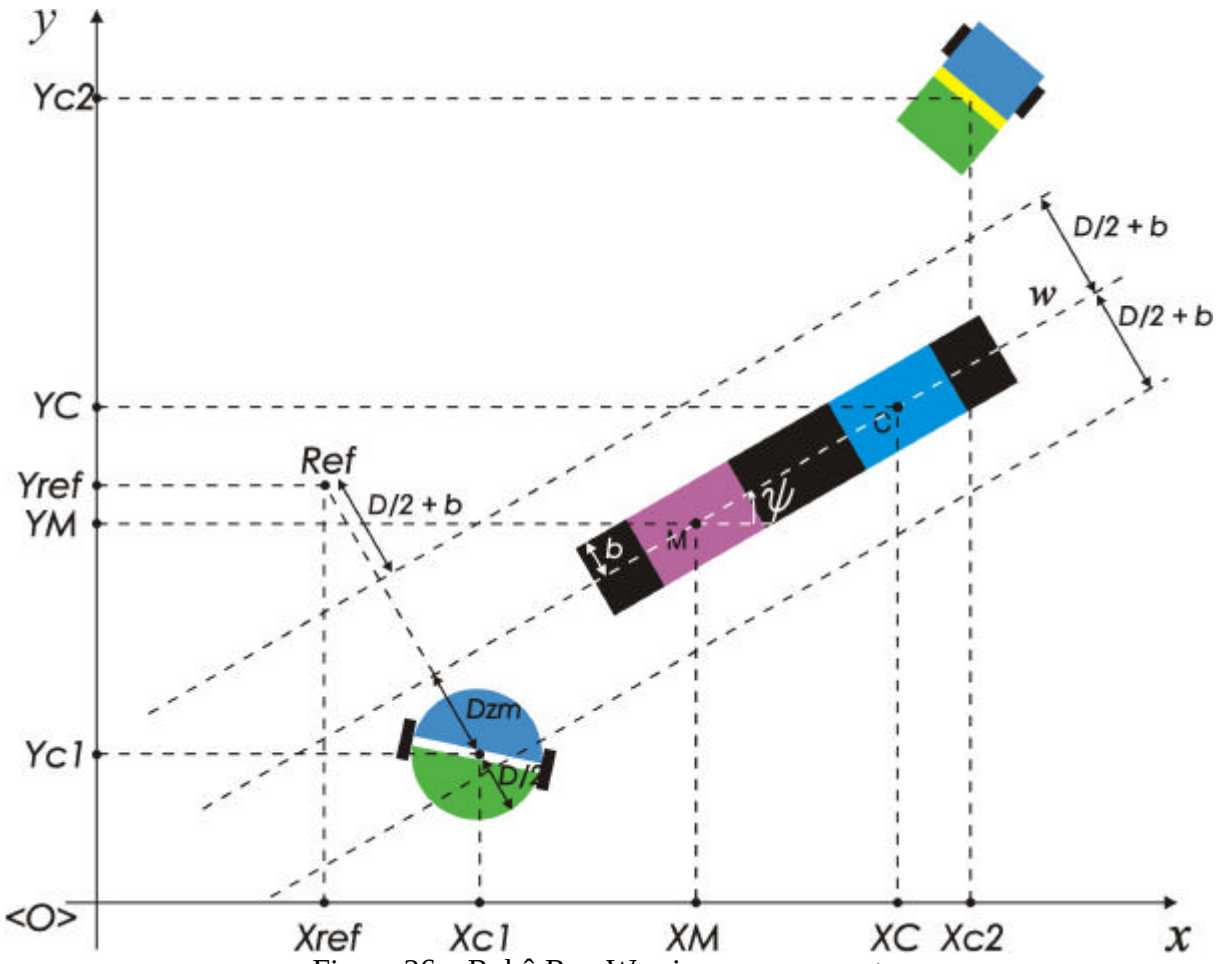


Figura 26 – Robô Rug Warrior na zona morta.

A técnica utilizada para retirar um robô da zona morta consiste primeiro em identificar se o outro robô, neste caso o *Lego*, está do lado de cima ou do lado de baixo do eixo principal da caixa. Isto é feito substituindo as coordenadas do robô (*Xc2*, *Yc2*) na equação geral da reta *w* que passa pelo eixo da caixa, ou seja, obtendo

$$eq2 = -\tan(y)\,Xc2 + Yc2 - (YM - \tan(y)\,XM), \qquad (3\text{-}45)$$

Se o valor de *eq2* for maior que zero, significa que o outro robô está do lado de cima do eixo da caixa. Se o valor de *eq2* for menor que zero, significa que o outro robô está do lado de baixo do eixo da caixa. No caso da Figura 26, o valor de *eq2* é maior que zero, e, conseqüentemente, o *Lego* está acima do eixo da caixa. Se a situação fosse inversa, ou seja, se o *Lego* estivesse na zona morta, a idéia seria a mesma, apenas invertendo as coordenadas do *Rug Warrior* e do *Lego* nas equações.

Uma vez definida a posição do outro robô em relação à caixa, estabelecemos um ponto de referência *Ref*(*Xref, Yref*) do mesmo lado onde este já se encontra, com distância *2*(D/2 + b)* em relação ao eixo da caixa, como pode ser visto na Figura 26. Este ponto de referência é então passado para o controlador de posição, para que o robô saia da zona morta.

Ainda seguindo o fluxograma da Figura 25, depois que a situação de zona morta estiver resolvida, a próxima etapa consiste em verificar se os robôs estão em lados opostos da caixa. Isto pode ser facilmente determinado substituindo as coordenadas do *Rug Warrior* (*Xc1, Yc1*) e do *Lego* (*Xc2, Yc2*) na equação da reta *w* que passa pelo eixo da caixa, como foi feito no caso de verificação da zona morta, ou seja, obtém-se

$$eq1 = -\tan(y)\,Xc1 + Yc1 - (YM - \tan(y)\,XM), \tag{3-46}$$

e

$$eq2 = -\tan(y)\,Xc2 + Yc2 - (YM - \tan(y)\,XM). \tag{3-47}$$

Se os valores de *eq1* e *eq2* têm sinais diferentes, significa que os robôs estão em lados opostos, e, conseqüentemente, um dos dois precisa trocar de lado, para que então os dois robôs fiquem do mesmo lado da caixa, no intuito de empurrá-la. A escolha de qual robô deverá trocar de lado é feita em função das duas áreas do tablado separadas pelo eixo da caixa, a área acima do eixo da caixa e a área abaixo do eixo da caixa.

Para calcular a área abaixo do eixo da caixa, área hachurada na Figura 27, devemos primeiro encontrar os pontos de interseção das retas

$$w: \quad y = \tan(y)\,x + (YM - \tan(y)\,XM), \tag{3-48}$$

$$y = 0 \text{ (que define } X1\text{)}, \tag{3-49}$$

$$y = 480 \text{ (que define } X2\text{)}, \tag{3-50}$$

$$x = 0 \text{ (que define } Y1\text{)}, \tag{3-51}$$

e

$$x = 640 \text{ (que define } Y2\text{)}. \tag{3-52}$$

Depois de encontrar os pontos de interseção (*X1*, 0), (*X2*, 480), (0, Y*1*) e (640, Y2), fazemos uma análise para calcular a área abaixo do gráfico. A análise feita leva em consideração a inclinação da caixa e os valores dos pontos de interseção. Não discutiremos aqui esta análise, por se tratar de manipulações matemáticas simples. Depois de realizar tal análise, no caso da Figura 27, a área abaixo do gráfico é dada por,

$$Area = (640 - x1) * y2/2 - (640 - x2) * (y2 - 480)/2 - (-x1) * y1/2 . \quad (3\text{-}53)$$

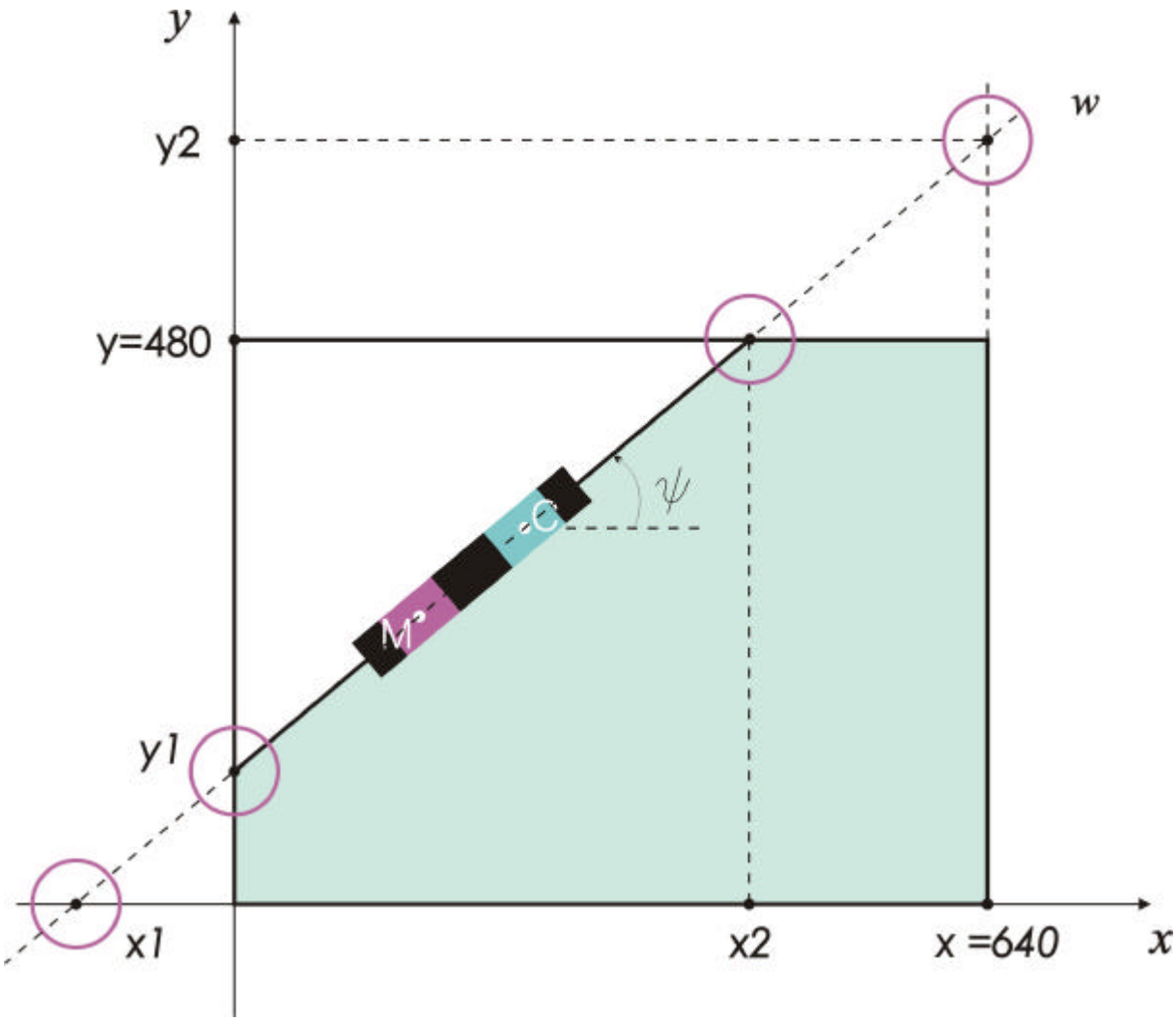


Figura 27 – Cálculo da área abaixo do eixo da caixa.

Como dissemos anteriormente, a escolha de qual robô deverá trocar de lado é feita em função das duas áreas do tablado separadas pelo eixo da caixa. O robô que

estiver do lado de menor área do eixo da caixa deverá trocar de lado, pois se admite que pelo lado de maior área seja mais fácil realizar a cooperação, tendo em vista que a caixa estará mais perto da borda do tablado. Até porque seria incoerente mandar que o robô localizado no lado de maior área mudasse para o lado de menor área, correndo risco de sair do espaço de trabalho.

Porém, existe um problema que pode impedir a troca de lado do robô que está do lado da menor área. Este problema está relacionado com a distância entre os vértices da caixa e os limites do tablado: caso esta distância seja menor que a distância mínima para o robô passar, dizemos que o robô localizado na área menor está encurralado, situação esta ilustrada na Figura 31. Para afirmarmos se o robô está ou não encurralado, precisamos calcular as coordenadas dos vértices V1, V2, V3 e V4, as quais serão úteis também para mudar o robô de lado, caso o mesmo não esteja encurralado. Para calcular as coordenadas dos vértices é necessário conhecer o valor das constantes *a* e *b*, mostradas na Figura 28. O cálculo das coordenadas é feito com base na Figura 29.

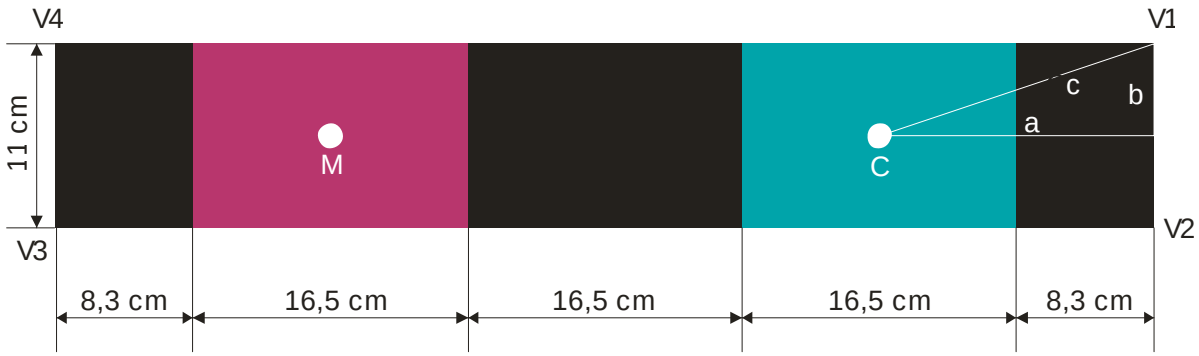


Figura 28 – Vértices da caixa.

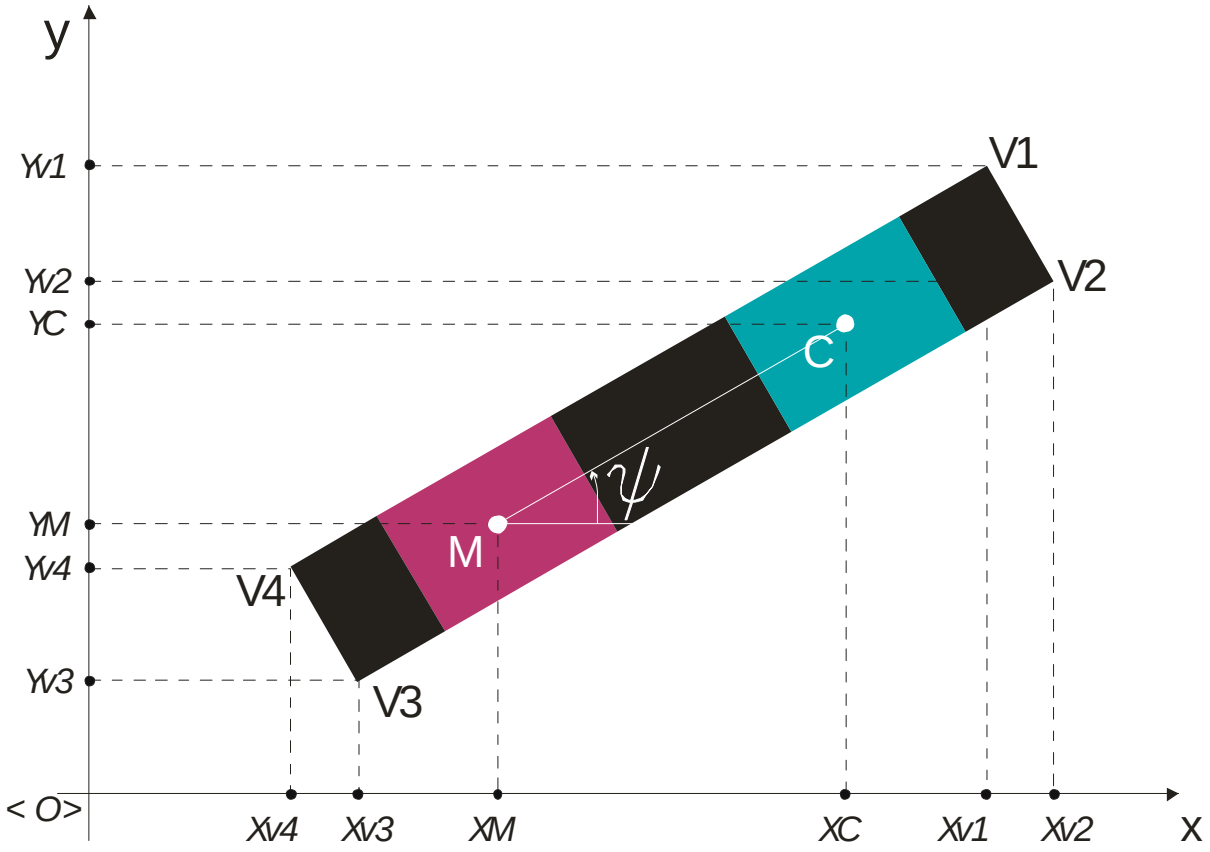


Figura 29 – Coordenadas dos vértices.

As equações dos vértices da caixa a ser empurrada são

$$Xv1 = XC + a\cos(y) - b\cos(p/2 - y), \tag{3-54}$$

$$Yv1 = YC + a\,sen(y) + b\,sen(p/2 - y), \tag{3-55}$$

$$Xv2 = XC + a\cos(y) + b\cos(p/2 - y), \tag{3-56}$$

$$Yv2 = YC + a\,sen(y) - b\,sen(p/2 - y), \tag{3-57}$$

$$Xv3 = XM - a\cos(y) + b\cos(p/2 - y), \tag{3-58}$$

$$Yv3 = YM - a\,sen(y) - b\,sen(p/2 - y), \tag{3-59}$$

$$Xv4 = XM - a\cos(y) - b\cos(p/2 - y) \tag{3-60}$$

e

$$Yv4 = YM - a\,sen(y) + b\,sen(p/2 - y), \tag{3-61}$$

onde

(*Xv1, Yv1*) coordenadas do vértice V1 da caixa
(*Xv2, Yv2*) coordenadas do vértice V2 da caixa
(*Xv3, Yv3*) coordenadas do vértice V3 da caixa
(*Xv4, Yv4*) coordenadas do vértice V4 da caixa.

Uma vez que as coordenadas dos vértices estão determinadas, devemos saber quais serão os dois caminhos possíveis para a passagem do robô. Esta decisão depende da análise do ângulo de inclinação da caixa e do lado para onde se pretende levar o robô da área menor, se do lado superior ao eixo da caixa ou se do lado inferior do eixo da caixa. A Tabela 1 fornece os dois caminhos possíveis, sempre formados por um vértice e uma reta.

Tabela 1 – Caminhos por onde o robô deve passar.

Ângulo de Inclinação (?)	**Robô Deve ir Para Cima**		**Robô Deve ir Para Baixo**	
0 = ? = p/2	V3 e (y = 0)	V2 e (x = 640)	V4 e (x = 0)	V1 e (y = 480)
p/2 < ? = p	V1 e (x = 0)	V4 e (y = 640)	V2 e (y = 480)	V3 e (x = 640)
-p/2 = ? < 0	V2 e (y = 0)	V3 e (x = 0)	V1 e (x = 640)	V4 e (y = 480)
-p < ? < -p/2	V1 e (y = 0)	V4 e (x = 640)	V2 e (x = 0)	V3 e (y = 480)

Observamos que para cada configuração dos robôs no espaço de trabalho, existem dois caminhos possíveis. Devemos então escolher aquele onde o vértice apresenta a maior distância da reta especificada. Posteriormente, verificamos se esta distância calculada é menor que a distância mínima para o robô passar, ou seja, a largura do robô. Se isso acontecer, dizemos que o robô localizado na área menor está encurralado, situação esta ilustrada na Figura 31.

No caso da Figura 30, o *Lego* está do lado do eixo da caixa que apresenta maior área. Logo, o robô que deve trocar de lado é o *Rug Warrior*, e este deve ir para baixo. Observamos ainda que o ângulo de inclinação da caixa *?* obedece à desigualdade 0 = ? = p/2. Levando estas informações na Tabela 1, observamos que existem dois caminhos possíveis, o primeiro caminho é entre o vértice V4 e o eixo *x* = 0, e o segundo caminho é entre o vértice V1 e o eixo *y* = 480. Depois de calcularmos a distância do vértice V4 ao eixo *x* = 0, e a distância do vértice V1 ao eixo *y* = 480, concluímos que o melhor caminho para o robô passar está entre o vértice V4 e o eixo *x* = 0. Calculamos ainda a distância entre o vértice V4 e o eixo *x* = 0. No caso da Figura 30 esta distância é maior que a largura do robô, sendo possível, então, a sua passagem.

Para que o robô possa mudar de lado calculamos três pontos de referência, *Ref1*, *Ref2* e *Ref3*, os quais serão passados para o controlador de posição. As equações para se obter as três referências são

$$Xref1 = Xv4 - L, \tag{3-62}$$

$$Yref1 = Yv4, \tag{3-63}$$

$$Xref2 = \mathrm{MagentaX} - 2{,}5\,\mathrm{L}\cos(y), \tag{3-64}$$

$$Yref2 = MagentaY - 2{,}5\,L\,sen(y), \tag{3-65}$$

$$Xref3 = Xref2 + 1{,}5\,\mathrm{L}\cos(p/2 - y), \tag{3-66}$$

$$Yref3 = Yref2 - 1{,}5\,\mathrm{L}\,sen(p/2 - y), \tag{3-67}$$

onde

L	largura do robô
(*Xref1, Yref1*)	coordenadas do ponto de referência *Ref*1
(*Xref2, Yref2*)	coordenadas do ponto de referência *Ref2*
(*Xref3, Yref3*)	coordenadas do ponto de referência *Ref3*

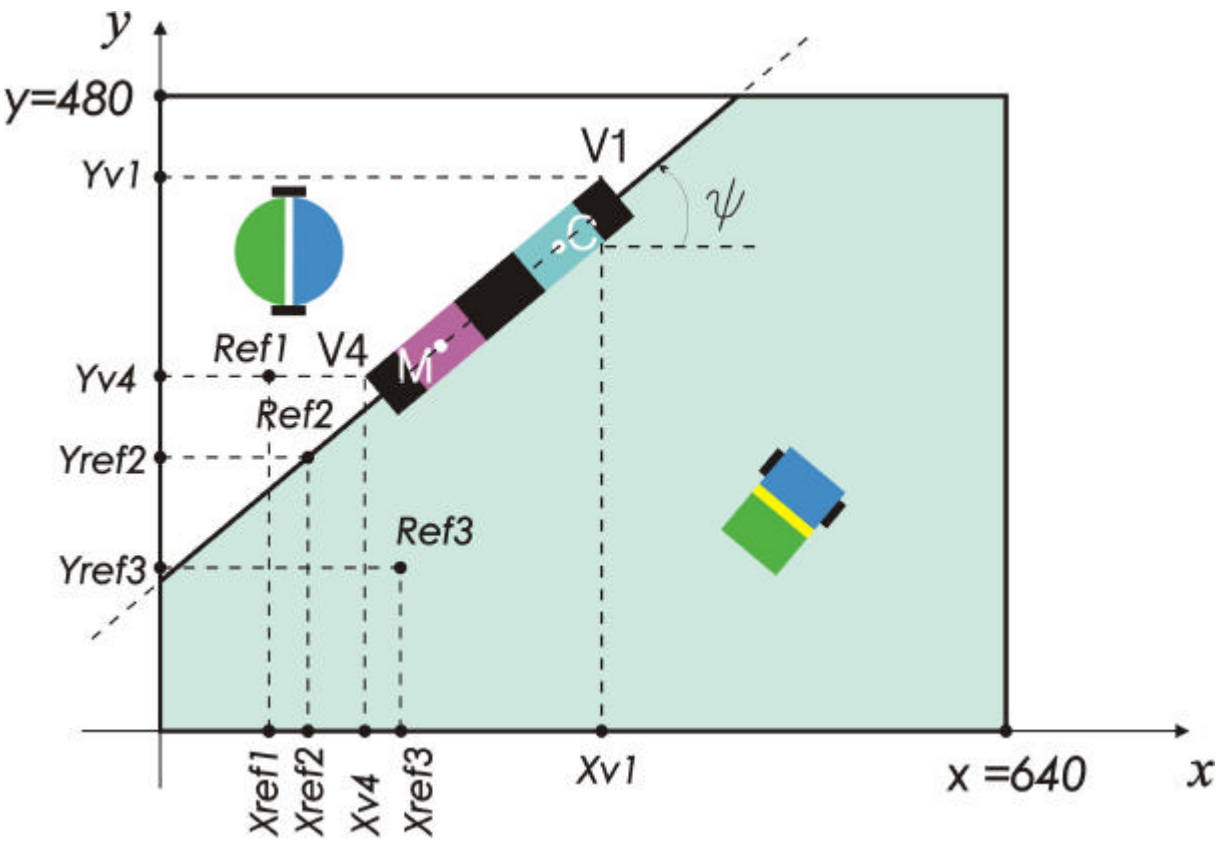


Figura 30 – Estratégia para mudar de lado na caixa.

Caso o robô esteja encurralado, ou seja, a maior das distâncias entre os dois vértices e as duas retas, dados pela Tabela 1, for menor que a largura do robô, será necessário tirar o robô desta situação. Esta situação pode ser vista na Figura 31, onde a distância do vértice V4 ao eixo $x = 0$, e a distância do vértice V1 ao eixo $y = 480$, são menores que a largura do robô. A idéia, para resolver este problema, consiste em fazer com que o robô encurralado dê um giro de 150º na caixa. A partir daí os dois robôs estarão do mesmo lado da caixa, e a mesma poderá ser empurrada até sair totalmente do espaço de trabalho.

O maior problema para girarmos a caixa está na hora de escolher o sentido de giro, pois este depende do ângulo de inclinação da caixa e do lado para onde a mesma será girada. A Tabela 2 fornece o sentido de giro da caixa.

Tabela 2 – Sentido de giro da caixa.

Ângulo de Inclinação (?)	**Caixa Deve ser Girada Para Cima**	**Caixa Deve ser Girada Para Baixo**
0 = ? = p/2	Sentido Horário	Sentido Anti-Horário
p/2 < ? = p	Sentido Anti-Horário	Sentido Horário
-p/2 = ? < 0	Sentido Horário	Sentido Anti-Horário
-p < ? < -p/2	Sentido Anti-Horário	Sentido Horário

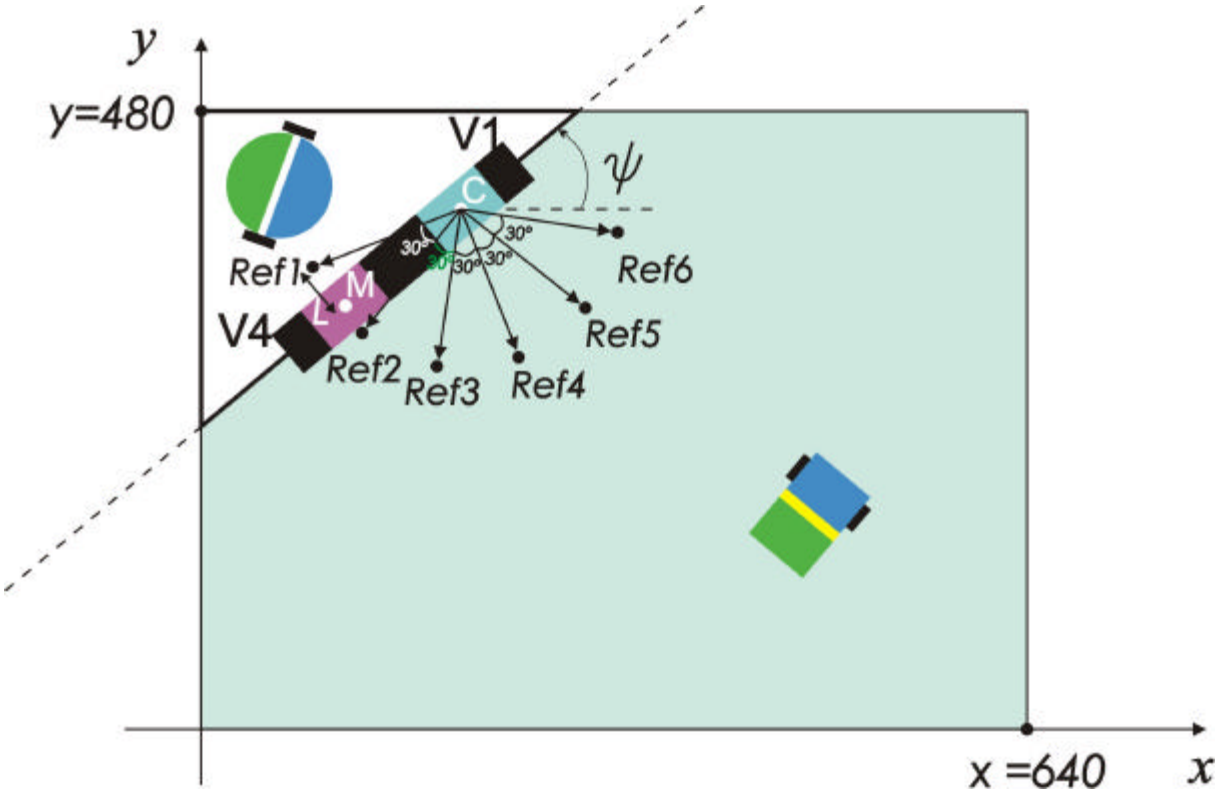


Figura 31 – Situação em que um robô está encurralado.

Antes de definir os pontos para girar a caixa, calculamos um ponto na borda da cor magenta, com distância desta igual a *L*, onde *L* representa a soma da metade da largura da caixa com a metade da largura do robô. Este ponto sempre está do mesmo lado do robô encurralado, e o robô sempre começará a empurrar a caixa por ali. A Tabela 3 determina as coordenadas deste primeiro ponto de referência,

Tabela 3 – Cálculo do primeiro ponto de referência para tirar robô encurralado.

Sentido de Giro da Caixa	***Tan(?)***	***Xref1***	***Yref1***
Para Cima	Tan(?) = 0	$MagentaX + L\,\lvert\cos(p/2 - y)\rvert$	$MagentaY - L\,\lvert sen(p/2 - y)\rvert$
	Tan(?) < 0	$MagentaX - L\,\lvert\cos(p/2 - y)\rvert$	$MagentaY - L\,\lvert sen(p/2 - y)\rvert$
Para Baixo	Tan(?) = 0	$MagentaX - L\,\lvert\cos(p/2 - y)\rvert$	$MagentaY + L\,\lvert sen(p/2 - y)\rvert$
	Tan(?) < 0	$MagentaX + L\,\lvert\cos(p/2 - y)\rvert$	$MagentaY + L\,\lvert sen(p/2 - y)\rvert$

Uma vez definido o sentido de giro, e o primeiro ponto de referência para o robô encurralado, tomamos um vetor imaginário com origem no centro de gravidade da cor cyan e extremidade no primeiro ponto de referência calculado para o robô encurralado. A partir daí aplicamos cinco vezes sucessivas uma matriz de rotação para um ângulo de 30º, como pode ser visto Figura 31. As equações que determinam os demais pontos de referência para o robô que está encurralado são

$$\begin{bmatrix} Xref\,2 \\ Yref\,2 \end{bmatrix} = \begin{bmatrix} Xref\,1 & Yref\,1 \end{bmatrix} \begin{bmatrix} \cos(Sentido * p/6) & sen(Sentido * p/6) \\ -sen(Sentido * p/6) & \cos(Sentido * p/6) \end{bmatrix}, \tag{3-68}$$

$$\begin{bmatrix} Xref\,3 \\ Yref\,3 \end{bmatrix} = \begin{bmatrix} Xref\,2 & Yref\,2 \end{bmatrix} \begin{bmatrix} \cos(Sentido * p/6) & sen(Sentido * p/6) \\ -sen(Sentido * p/6) & \cos(Sentido * p/6) \end{bmatrix}, \tag{3-69}$$

$$\begin{bmatrix} Xref\,4 \\ Yref\,4 \end{bmatrix} = \begin{bmatrix} Xref\,3 & Yref\,3 \end{bmatrix} \begin{bmatrix} \cos(Sentido * p/6) & sen(Sentido * p/6) \\ -sen(Sentido * p/6) & \cos(Sentido * p/6) \end{bmatrix}, \tag{3-70}$$

$$\begin{bmatrix} Xref\,5 \\ Yref\,5 \end{bmatrix} = \begin{bmatrix} Xref\,4 & Yref\,4 \end{bmatrix} \begin{bmatrix} \cos(Sentido * p/6) & sen(Sentido * p/6) \\ -sen(Sentido * p/6) & \cos(Sentido * p/6) \end{bmatrix}, \tag{3-71}$$

$$\begin{bmatrix} Xref\,6 \\ Yref\,6 \end{bmatrix} = \begin{bmatrix} Xref\,5 & Yref\,5 \end{bmatrix} \begin{bmatrix} \cos(Sentido * p/6) & sen(Sentido * p/6) \\ -sen(Sentido * p/6) & \cos(Sentido * p/6) \end{bmatrix}, \tag{3-72}$$

onde *Sentido,* é a representação matemática do sentido de giro da caixa, e tem valor igual a +1 para giros da caixa no sentido anti-horário, e valor igual a -1 para giros da caixa no sentido horário, e

(*Xref1, Yref1*) coordenadas do ponto de referência *Ref*1,
(*Xref2, Yref2*) coordenadas do ponto de referência *Ref2*,
(*Xref3, Yref3*) coordenadas do ponto de referência *Ref3*,
(*Xref4, Yref4*) coordenadas do ponto de referência *Ref4*,
(*Xref5, Yref5*) coordenadas do ponto de referência *Ref5*,
(*Xref6, Yref6*) coordenadas do ponto de referência *Ref6.*

Um detalhe a ser observado, antes de mandar o robô encurralado girar a caixa, é ver se o outro robô, que não está encurralado, apresenta uma distância segura da caixa, para evitar que o mesmo entre em colisão com a caixa que está sendo empurrada. A Figura 32 ilustra a situação onde o robô não encurralado está muito perto da caixa a ser girada. Quando isto acontece, calculamos um ponto de referência com distância *D* da cor magenta, onde *D* representa a largura da caixa. Este ponto sempre está do mesmo lado do robô não encurralado. Depois de calcular o ponto, o mesmo é passado para o controlador de posição, para que o robô não encurralado se afaste da caixa. A Tabela 4 determina as coordenadas deste ponto de referência.

Tabela 4 – Cálculo do ponto de referência para afastar o robô não encurralado da caixa.

Lado Onde Está Robô não Encurralado	***Tan(?)***	***Xref***	***Yref***
Abaixo do Eixo da Caixa	Tan(?) = 0	$MagentaX + D\lvert\cos(p/2 - y)\rvert$	$MagentaY - D\lvert sen(p/2 - y)\rvert$
	Tan(?) < 0	$MagentaX - D\lvert\cos(p/2 - y)\rvert$	$MagentaY - D\lvert sen(p/2 - y)\rvert$
Acima do Eixo da Caixa	Tan(?) = 0	$MagentaX - D\lvert\cos(p/2 - y)\rvert$	$MagentaY + D\lvert sen(p/2 - y)\rvert$
	Tan(?) < 0	$MagentaX + D\lvert\cos(p/2 - y)\rvert$	$MagentaY + D\lvert sen(p/2 - y)\rvert$

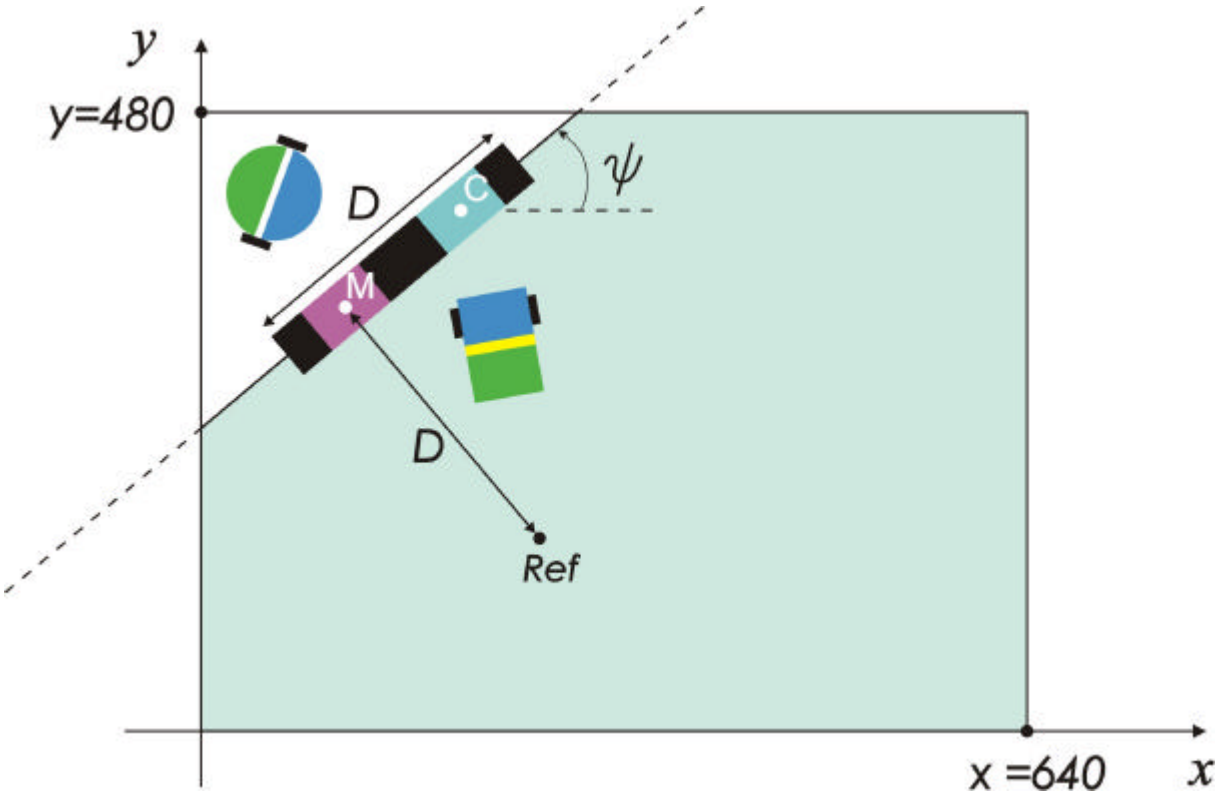


Figura 32 – Situação com um robô encurralado e o outro muito perto.

Finalmente, depois de detectar e resolver todos os possíveis problemas que precedem o transporte da caixa, procedemos com o transporte propriamente dito. Como dissemos anteriormente, a tarefa de empurrar a caixa, implementada neste trabalho, consiste em empurrar a mesma até que esta seja totalmente retirada para fora da área de trabalho dos robôs.

Para empurrar a caixa, precisamos estabelecer dois pontos de referência para cada robô. O primeiro ponto que o robô deve alcançar é a borda de uma das cores da caixa, no intuito de alinhar o mesmo à caixa, já o segundo ponto deve ser a borda do tablado do espaço de trabalho, no intuito de empurrar a caixa até os limites do mesmo.

Para estabelecermos o primeiro ponto de referência dos dois robôs, é necessário verificar qual dos dois robôs está mais perto de uma determinada cor. A verificação do robô mais próximo tem o objetivo de minimizar o tempo de execução da tarefa de empurrar a caixa, evitando que um robô próximo a uma cor vá para outra cor. Esta verificação é feita medindo-se a distância de cada robô ao centro de gravidade de cada uma das duas tarjas, magenta e cyan, como visto na Figura 33.

Depois de verificar a menor das quatro distâncias, fazemos o primeiro ponto de referência do robô mais próximo igual ao ponto tangente àquela tarja que apresentou menor distância, e o primeiro ponto de referência do outro robô fica sendo o ponto tangente à outra tarja. No caso da Figura 33, a menor distância é *D1M*, e, conseqüentemente, o primeiro ponto do *Rug Warrior* fica sendo o ponto tangente à tarja magenta, e o primeiro ponto do *Lego* fica sendo o ponto tangente à tarja cyan.

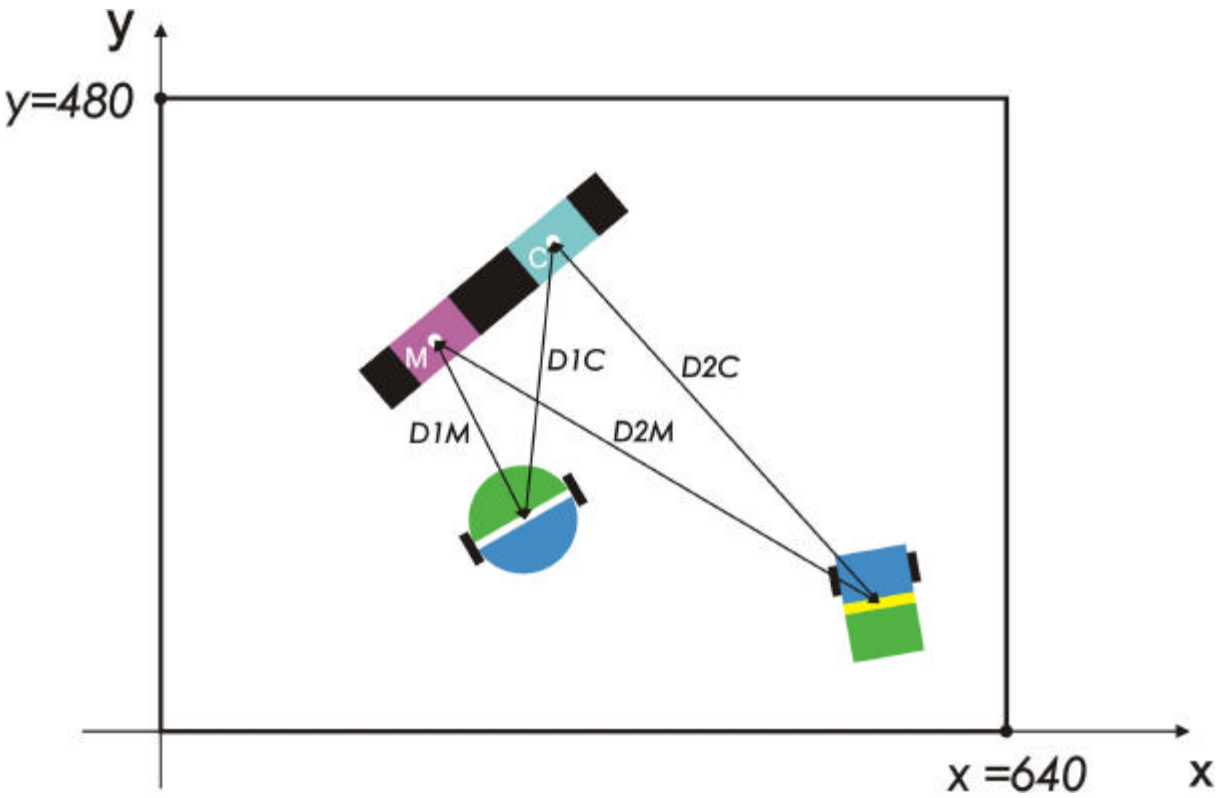


Figura 33 – Distância entre os robôs e as tarjas da caixa.

Um problema que nós encontramos, ao colocar em prática a idéia acima, é que, em algumas situações, as retas que ligam os robôs aos pontos tangentes às tarjas se tocavam, representando assim o choque dos mesmos ao se aproximarem da caixa, situação esta mostrada na Figura 34. Para resolver esta situação devemos calcular o ponto de interseção *Pint*(*Xint, Yint*) da reta *r* que liga o centro de gravidade da cor magenta *M*(*XM, YM*) ao centro de gravidade do robô eleito para a cor magenta *RM*(*XRM, YRM*), com a reta *s* que liga o centro de gravidade da cor cyan *C*(*XC, YC*)

ao centro de gravidade do robô eleito para a cor cyan *RC*(*XRC, YRC*). As equações das retas *r* e *s*, e do ponto de interseção *Pint* entre elas, conforme o Apêndice B, é

$$r: \quad y = \left(\frac{YRM - YM}{XRM - XM} \right) x + \left(\frac{XRM * YM - XM * YRM}{XRM - XM} \right), \tag{3-73}$$

$$s: \quad y = \left(\frac{YRC - YC}{XRC - XC} \right) x + \left(\frac{XRC * YC - XC * YRC}{XRC - XC} \right), \tag{3-74}$$

$$X_{\text{int}} = \frac{\left(\frac{XRC * YC - XC * YRC}{XRC - XC} \right) - \left(\frac{XRM * YM - XM * YRM}{XRM - XM} \right)}{\left(\frac{YRM - YM}{XRM - XM} \right) - \left(\frac{YRC - YC}{XRC - XC} \right)}, \tag{3-75}$$

$$Y_{\text{int}} = \left(\frac{YRM - YM}{XRM - XM} \right) X_{\text{int}} + \left(\frac{XRM * YM - XM * YRM}{XRM - XM} \right), \tag{3-76}$$

onde

(*XM, YM*) representa as coordenadas do centro de gravidade da tarja magenta
(*XC, YC*) representa as coordenadas do centro de gravidade da tarja cyan
(*XRM, YRM*) representa as coordenadas do centro de gravidade do robô magenta
(*XRC, YRC*) representa as coordenadas do centro de gravidade do robô cyan
(*Xint, Yint*) representa as coordenadas do ponto de interseção

Se a distância *Dpint_m* deste ponto de interseção até o centro de gravidade da cor magenta for menor que a distância *Dm_rm* do centro de gravidade da cor magenta ao centro de gravidade do robô eleito para a cor magenta, e se a distância *Dpint_rm* deste ponto de interseção até o centro de gravidade do robô eleito para a cor magenta for menor que a distância *Dm_rm* do centro de gravidade da cor magenta ao centro de gravidade do robô eleito para a cor magenta, então o ponto de interseção *Pint* está entre o robô e a caixa, significando que os dois robôs vão colidir. Neste caso devemos inverter a seqüência dos robôs em relação às cores, mandando o *Lego* para a cor

magenta e o *Rug Warrior* para a cor cyan, no intuito de evitar colisão. Esta situação é mostrada na Figura 34. Se a distância *Dpint_m* ou a distância *Dpint_rm* for maior que *Dm_rm*, significa que o ponto de interseção não está entre a caixa e o robô, de forma que não haverá colisão.

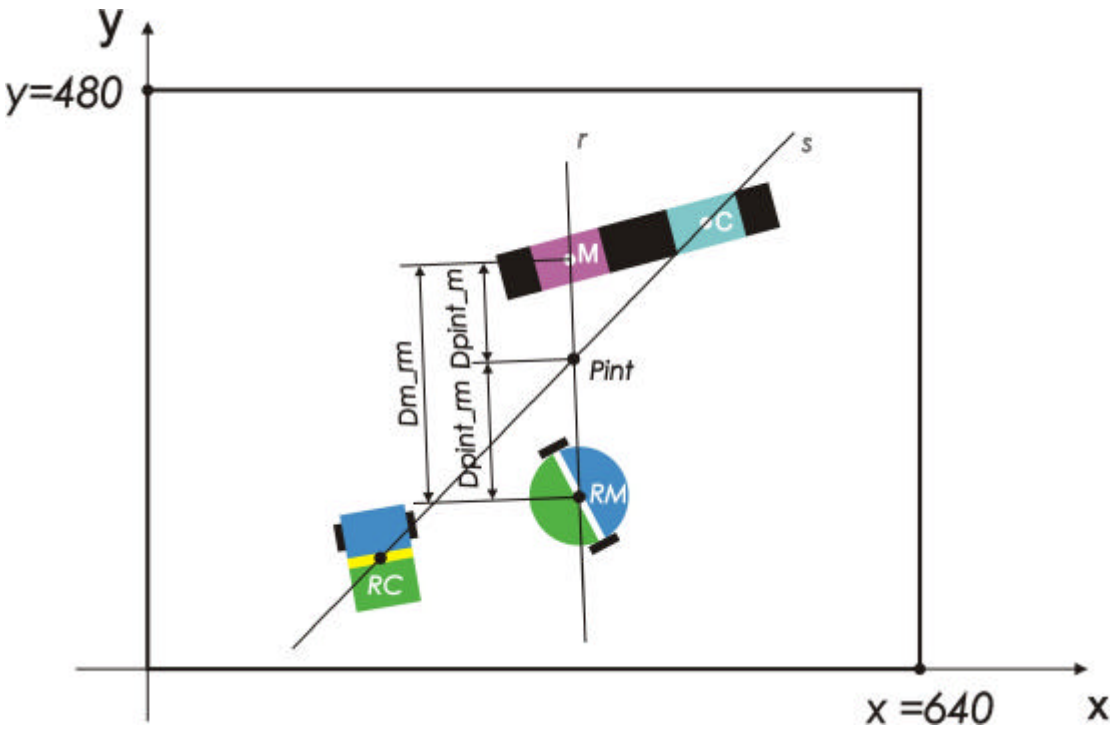


Figura 34 – Ponto de interseção das retas que ligam os robôs à caixa.

Depois de definidas as tarjas em que os robôs devem encostar, precisamos calcular os pontos tangentes a estas tarjas, os quais serão os primeiros pontos de referência de cada robô. As coordenadas destes pontos tangentes dependem da posição dos robôs em relação à caixa, bem como da inclinação da reta que passa pelo eixo principal da caixa. De acordo com a Figura 35, existem dois pontos tangentes à tarja cyan: o primeiro é P1, que está acima do eixo principal, e o segundo é P2, que está abaixo do eixo principal. Analogamente, existem dois pontos tangentes à tarja magenta: o primeiro é P4, que está acima do eixo principal, e o segundo é P3, que está abaixo do eixo principal.

Se os dois robôs estiverem acima do eixo da caixa, definimos a referência do robô eleito para a cor magenta como o ponto P4, e a referência do robô eleito para a cor cyan como o ponto P1. Se os dois robôs estiverem abaixo do eixo da caixa,

definimos a referência do robô eleito para a cor magenta como o ponto P3, e a referência do robô eleito para a cor cyan como o ponto P2.

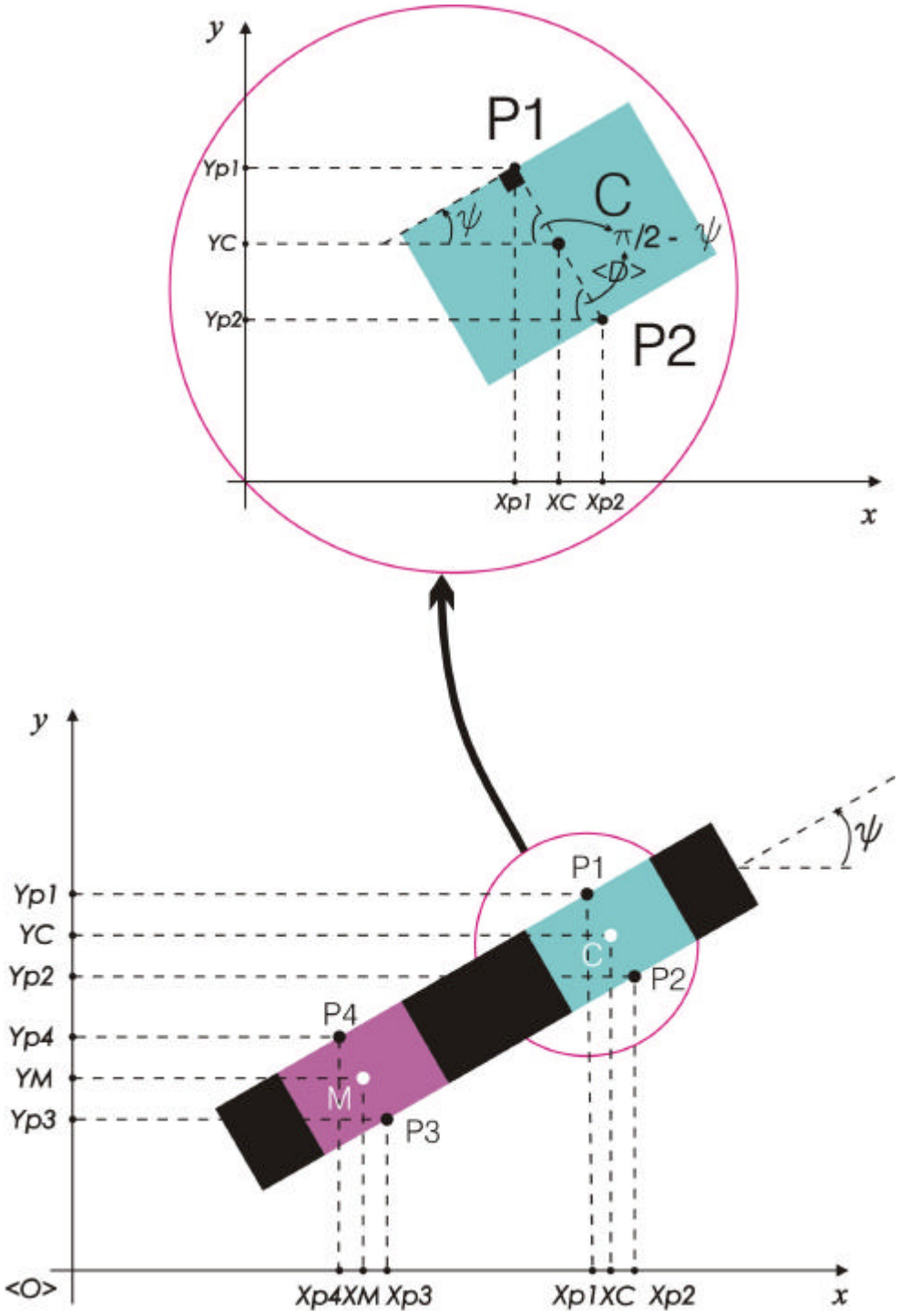


Figura 35 – Coordenadas dos pontos que tangenciam a caixa.

As expressões que calculam as coordenadas destes pontos tangentes são dadas na Tabela 5.

Tabela 5 – Coordenadas dos pontos tangentes às tarjas da caixa.

Ponto	**Tan*(?)***	**Coordenada Eixo *x***	**Coordenada Eixo *y***
P1	Tan(?) = 0	$CyanX - L/2\,\lvert\cos(p/2 - y)\rvert$	$CyanY + L/2\,\lvert sen(p/2 - y)\rvert$
	Tan(?) < 0	$CyanX + L/2\,\lvert\cos(p/2 - y)\rvert$	$CyanY + L/2\,\lvert sen(p/2 - y)\rvert$
P2	Tan(?) = 0	$CyanX + L/2\,\lvert\cos(p/2 - y)\rvert$	$CyanY - L/2\,\lvert sen(p/2 - y)\rvert$
	Tan(?) < 0	$CyanX - L/2\,\lvert\cos(p/2 - y)\rvert$	$CyanY - L/2\,\lvert sen(p/2 - y)\rvert$
P3	Tan(?) = 0	$MagentaX + L/2\,\lvert\cos(p/2 - y)\rvert$	$MagentaY - L/2\,\lvert sen(p/2 - y)\rvert$
	Tan(?) < 0	$MagentaX - L/2\,\lvert\cos(p/2 - y)\rvert$	$MagentaY - L/2\,\lvert sen(p/2 - y)\rvert$
P4	Tan(?) = 0	$MagentaX - L/2\,\lvert\cos(p/2 - y)\rvert$	$MagentaY + L/2\,\lvert sen(p/2 - y)\rvert$
	Tan(?) < 0	$MagentaX + L/2\,\lvert\cos(p/2 - y)\rvert$	$MagentaY + L/2\,\lvert sen(p/2 - y)\rvert$

Depois de definir os pontos tangentes às tarjas, do lado onde se encontram os robôs, e garantir que os dois robôs encostaram na caixa, precisamos calcular as coordenadas dos pontos localizados na borda do tablado, para onde os robôs empurrarão a caixa. Para não haver erro de sincronização entre os robôs na hora de empurrar a caixa, esta só é empurrada quando os dois robôs alcançarem o primeiro ponto de referência, ou seja, os pontos tangentes às tarjas. Se um robô chegar primeiro à caixa, ele deve esperar que o outro também encoste, para que os dois a empurrem sincronizadamente.

Inicialmente definimos duas retas *r* e *s*, perpendiculares ao eixo principal da caixa e passando no centro de gravidade das tarjas magenta e cyan, respectivamente, como podemos observar na Figura 36. Com base no Apêndice B, as equações das retas *r* e *s* são

$$r: \quad y = \tan(y + p/2)\cdot x + (MagentaY - \tan(y + p/2)\cdot MagentaX) \tag{3-77}$$

e

$$s: \quad y = \tan(y + p/2) \cdot x + (CyanY - \tan(y + p/2) \cdot CyanX) \tag{3-78}$$

Para evitar que o robô saia do espaço de trabalho, definimos um espaço de trabalho fictício com 20 *pixels* a menos que o original. Desta forma, quando o robô chegar à borda do espaço de trabalho fictício, a caixa vai estar fora do espaço de trabalho original. Como podemos observar na Figura 36, as quatro equações de reta que delimitam este espaço de trabalho fictício são,

$$t: \quad x = 20, \tag{3-79}$$

$$u: \quad x = 620, \tag{3-80}$$

$$v: \quad y = 20 \tag{3-81}$$

e

$$z: \quad y = 460. \tag{3-82}$$

Como podemos observar na Figura 36, a reta *r* intercepta as retas *t*, *u*, *v* e *z*, determinando quatro pontos. São eles: (*X1M*, 20), (*X2M*, 460), (20, *Y1M*) e (620, *Y2M*). Analogamente, a reta s intercepta as retas *t*, *u*, *v* e *z*, nos pontos (*X1C*, 20), (*X2C*, 460), (20, *Y1C*) e (620, *Y2C*). Dos quatro pontos que cada uma das retas *r* e *s* interceptam, dois deles não fazem sentido, pelo fato da abcissa do ponto ser menor que 20 ou maior que 620, ou a ordenada do ponto ser menor que 20 ou maior que 460. No caso da Figura 36, os pontos de interseção da reta *r* que não fazem sentido são (20, *Y1M*) e (620, *Y2M*) restando os pontos (*X1M*, 20) e (*X2M*, 460) como pontos válidos. Já para a reta *s*, os pontos que não fazem sentido são (20, *Y1C*) e (620, *Y2C*), restando os pontos (*X1C*, 20) e (*X2C*, 460) como pontos válidos.

Depois de calcular os pontos de interseção válidos das retas r e s com as bordas do espaço de trabalho fictício, temos que definir quais deles estão acima do eixo principal da caixa e quais deles estão abaixo do eixo principal da caixa, no intuito de estabelecer os pontos para onde os robôs empurrarão a caixa. A Tabela 6 fornece as posições dos pontos de borda, calculados a partir da reta *r*, em relação ao eixo principal da caixa. As posições dos pontos de borda, calculados à partir da reta *s*, em relação ao eixo principal da caixa, podem ser obtidas de forma análoga à reta *r*, apenas

substituindo na Tabela 6 o índice *M*, referente à cor magenta, pelo índice *C*, referente à cor cyan.

Tabela 6 – Posição dos pontos de borda em relação ao eixo da caixa.

Ponto	**Tan*(?)***	**Posição do Ponto em Relação ao Eixo Principal da Caixa**
(*X1M*, 20)	Tan(?) = 0	Abaixo do Eixo
	Tan(?) < 0	Abaixo do Eixo
(*X2M*, 460)	Tan(?) = 0	Acima do Eixo
	Tan(?) < 0	Acima do Eixo
(20, *Y1M*)	Tan(?) = 0	Acima do Eixo
	Tan(?) < 0	Abaixo do Eixo
(620, *Y2M*)	Tan(?) = 0	Abaixo do Eixo
	Tan(?) < 0	Acima do Eixo

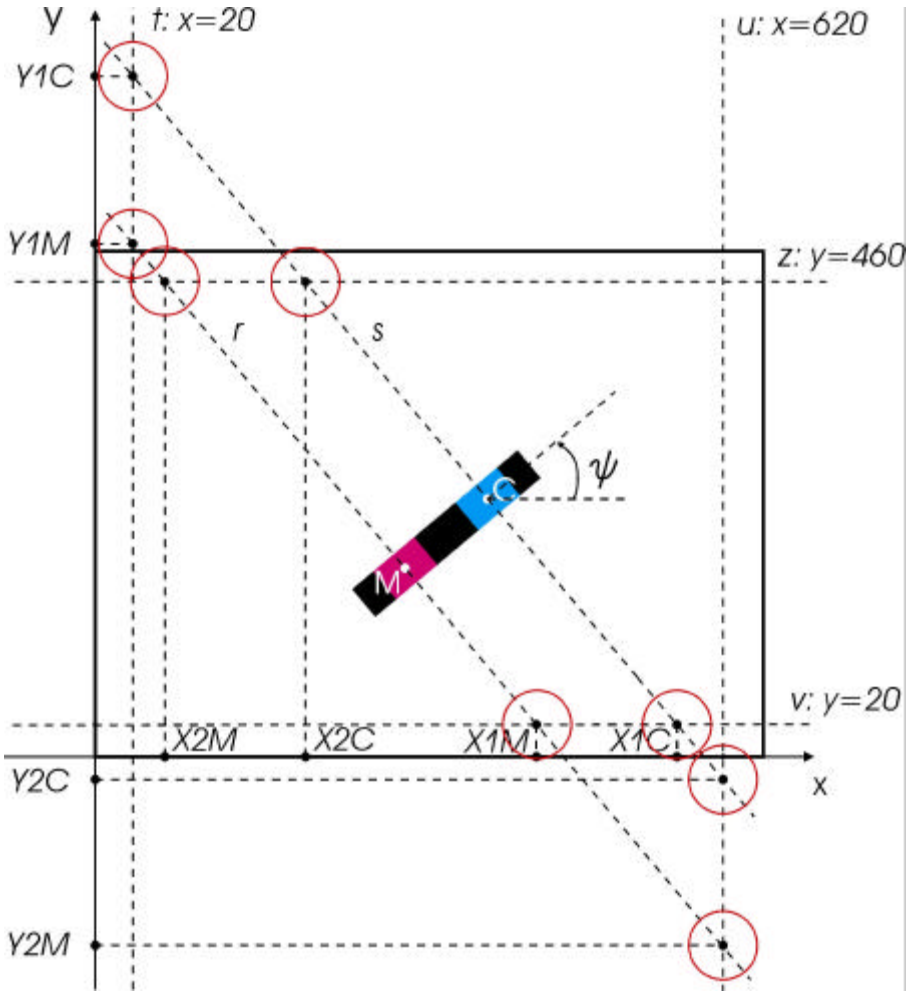


Figura 36 – Pontos de interseção com os eixos.

Uma vez definidos os pontos de interseção acima e abaixo do eixo principal da caixa para cada reta, basta sabermos agora a posição dos robôs em relação ao eixo principal da caixa. Se os robôs estiverem abaixo do eixo principal da caixa, devemos definir o segundo ponto de referência do robô que está encostado na tarja magenta como sendo o ponto de interseção da reta *r* que está acima do eixo da caixa, e o segundo ponto de referência do robô que está encostado na tarja cyan como sendo o ponto de interseção da reta *s* que está acima do eixo da caixa. Se os robôs estiverem acima do eixo principal da caixa, devemos definir o segundo ponto de referência do robô que está encostado na tarja magenta como sendo o ponto de interseção da reta *r* que está abaixo do eixo da caixa, e o segundo ponto de referência do robô que está encostado na tarja cyan como sendo o ponto de interseção da reta *s* que está abaixo do eixo da caixa.

Na Figura 37 temos uma situação típica para realizar a tarefa de empurrar a caixa. Inicialmente calculamos as distâncias dos robôs às tarjas e verificamos que a menor distância é entre o *Rug Warrior* e a tarja magenta. Logo, elegemos tal robô para empurrar a tarja magenta e o *Lego* para empurrar a tarja cyan, isto depois de verificar se a rota dos dois robôs não se interceptava entre a caixa e eles próprios. Como os dois robôs estão localizados abaixo do eixo principal da caixa, os dois primeiros pontos de referência para o *Rug Warrior* e o *Lego* são P3 e P2, respectivamente. A utilização destes dois pontos por parte do controlador de posição nos garante que os robôs vão encostar na caixa. Se um robô chegar primeiro à caixa, ele não pode começar a empurrá-la sem que o outro também encoste.

Observamos que as retas *r* e *s*, perpendiculares à caixa, tocam os limites do espaço de trabalho fictício em dois pontos válidos cada uma. Como os dois robôs estão localizados abaixo do eixo principal da caixa, vamos escolher o segundo ponto de referência de cada robô como sendo os dois pontos que estão acima do eixo principal da caixa, (*X2M*, 460) para o *Rug Warrior* e (*X2C*, 460) para o *Lego*, o que nos garante que a caixa será empurrada até a borda superior do espaço de trabalho.

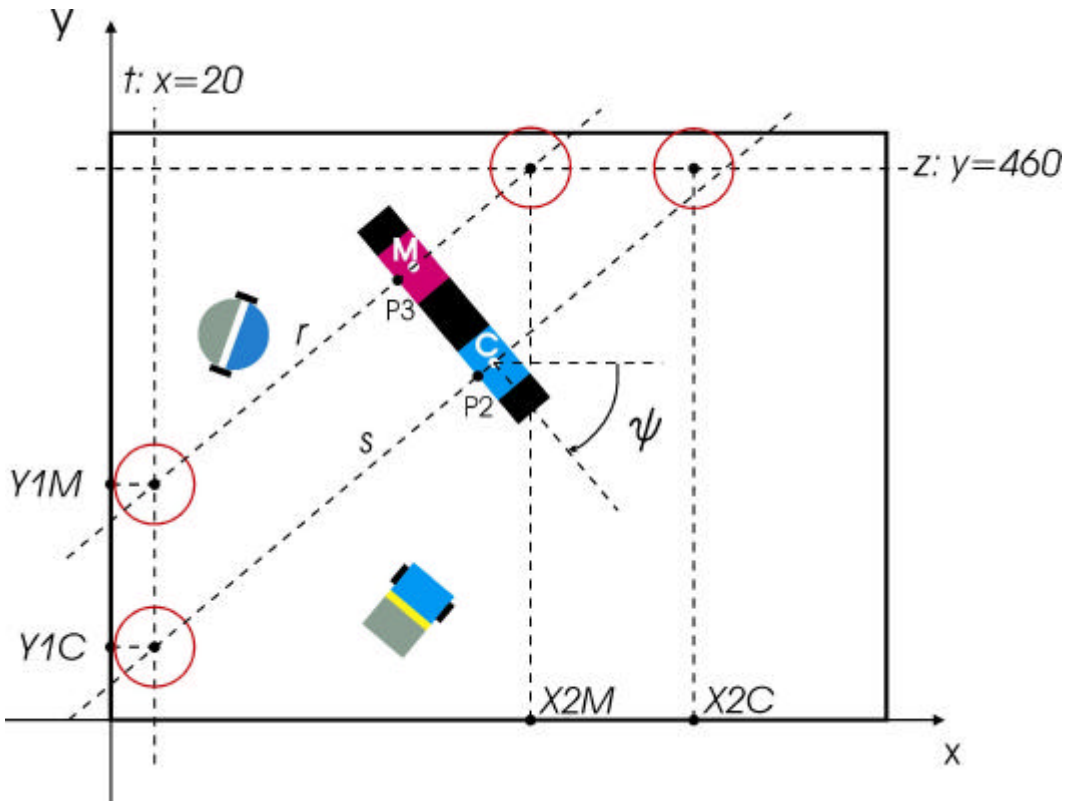


Figura 37 - Primeiro e segundo pontos destino para os robôs empurrarem a caixa.

4 EXPERIMENTOS REALIZADOS E ANÁLISE DOS RESULTADOS

4.1 Introdução

Neste capítulo discutiremos os resultados dos experimentos, cuja teoria foi desenvolvida no Capítulo 3. Em todos os casos que serão expostos neste capítulo foi utilizado o controlador desenvolvido em [2] e também discutido no Capítulo 3.

Como vimos no Capítulo 3, as equações do controlador não linear são

$$U_{dif} = \frac{tD}{KR}\left(k_a a - k_b w\right), \tag{4-1}$$

$$U_{com} = \frac{2t}{KR}\left(k_c(e)\, e\cos a - k_d v\right) \tag{4-2}$$

e

$$k_c(e) = \frac{k_c}{a+e} \quad , \quad k_d > 0\,. \tag{4-3}$$

A partir deste ponto faremos algumas modificações, no intuito de facilitar a escrita do texto. São elas:

$$Ka = \frac{k_a t\ D}{KR}, \tag{4-4}$$

$$Kb = \frac{k_b t\ D}{KR}, \tag{4-5}$$

$$Kc = \frac{k_c(e)\,2t}{KR}, \tag{4-6}$$

e

$$Kd = \frac{k_d\ 2t}{KR}\,. \tag{4-7}$$

Após alguns testes, chegamos aos valores das constantes do controlador utilizadas nos experimentos. Estes valores para o *Rug Warrior* e para o *Lego* são dados na Tabela 7.

Tabela 7 – Valores das constantes do controlador para os robôs utilizados.

Constante	***Rug Warrior***	***Lego***
Ka	50	25
Kb	1	1
Kc	70	40
Kd	0	0
a	10	10
?	0,1	0,1

Como vimos no Capítulo 3, todos os experimentos realizados neste trabalho aplicam o mesmo tipo de controlador de posição, só mudando a quantidade de pontos que o robô deve alcançar. Vimos, também, que quando o robô deve percorrer uma determinada trajetória (mais de um ponto), podemos fazer com que ele alcance o objetivo final através do cumprimento de objetivos parciais. Desta forma, podemos entender que todos os experimentos, ao serem particionados, se resumem ao experimento de alcançar um ponto destino, exceto pelo fato de ser repetido quantas vezes forem o número de pontos da trajetória. Por este motivo só faremos a análise gráfica das variáveis, e discutiremos estes gráficos, no experimento de alcançar o ponto destino.

4.2 Experimento 1: Alcançar Um Ponto Destino na Imagem

Como vimos no Capítulo 3, existem duas situações diferentes para o robô no que diz respeito ao seu erro de orientação em relação ao ponto destino. Estas duas situações são vistas a seguir.

4.2.1 Erro de orientação inicial pequeno

Neste ensaio foi possível observar o bom comportamento do controlador de posição utilizado neste trabalho. O robô utilizado no ensaio foi o *Rug Warrior*. Na Figura 38 podemos observar a posição inicial do robô, o ponto destino, o caminho seguido, plotado pelo *software* desenvolvido ao final do experimento, e ainda observar o pequeno erro de orientação.

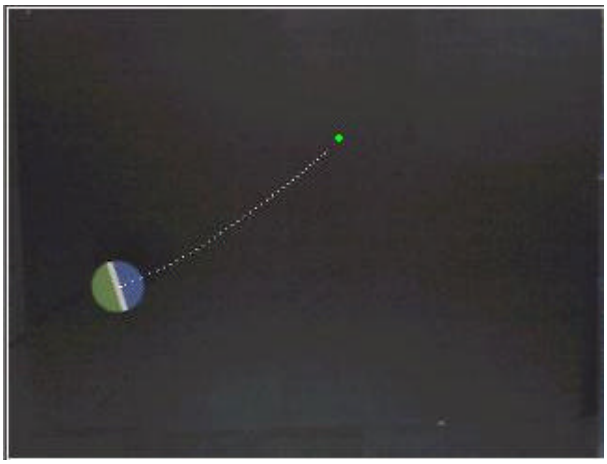


Figura 38 – Alcançar ponto destino com erro inicial de orientação pequeno.

Como podemos ver na Figura 39, a velocidade linear diminui à medida que o erro *e* diminui, ou seja, à medida que o robô se aproxima do ponto destino. Isto ocorre devido à relação da tensão comum U_{com} com o erro *e* na equação do controlador.

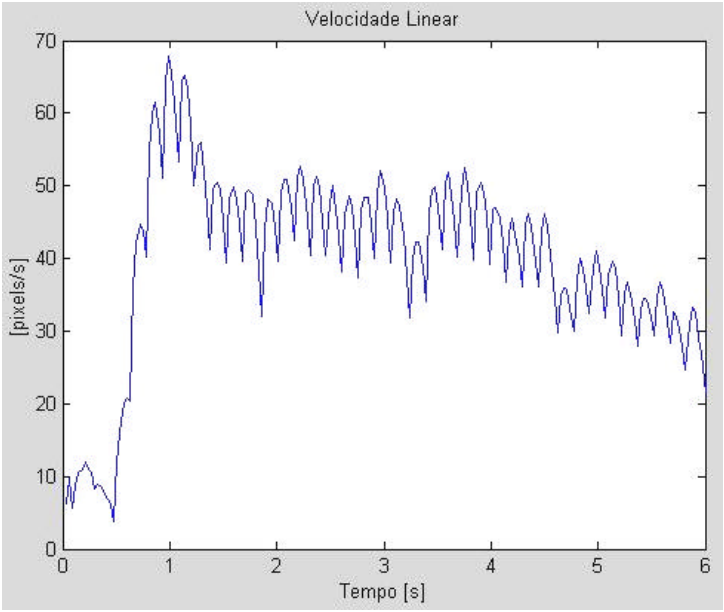


Figura 39 – Velocidade linear.

O erro *e* de posição é mostrado na Figura 40.

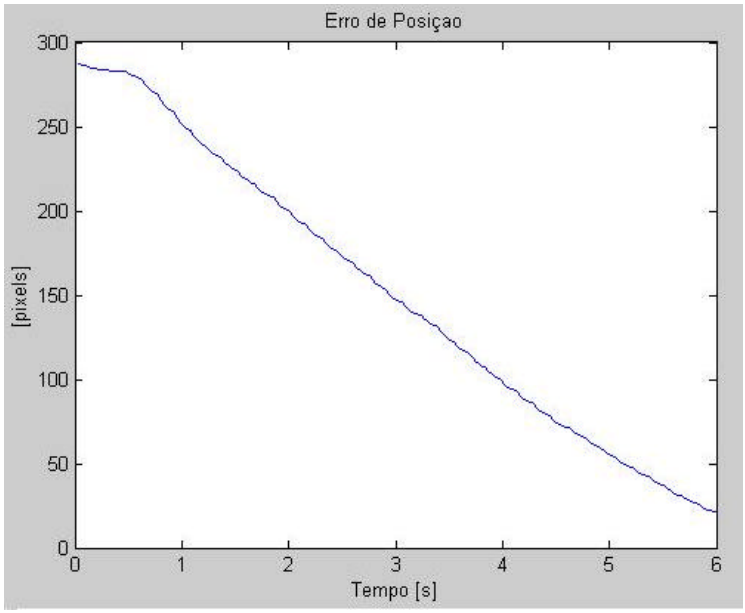


Figura 40 – Erro de posição.

Na Figura 41 é possível observar que a velocidade angular é muito pequena, visto que não há necessidade do controlador aplicar grandes correções na orientação do robô, já que o erro de orientação inicial já é pequeno. Podemos observar ainda, na Figura 42, que o valor do erro angular começa a aumentar quando o robô se aproxima

do ponto destino. Isto acontece porque próximo ao ponto destino os valores do erro de posição são pequenos, e uma pequena variação na medida deste valor implica em grandes variações angulares, devido à singularidade que aparece nas equações cinemáticas ao mudar as coordenadas para o sistema polar, como descrito na Seção 3.1.1.

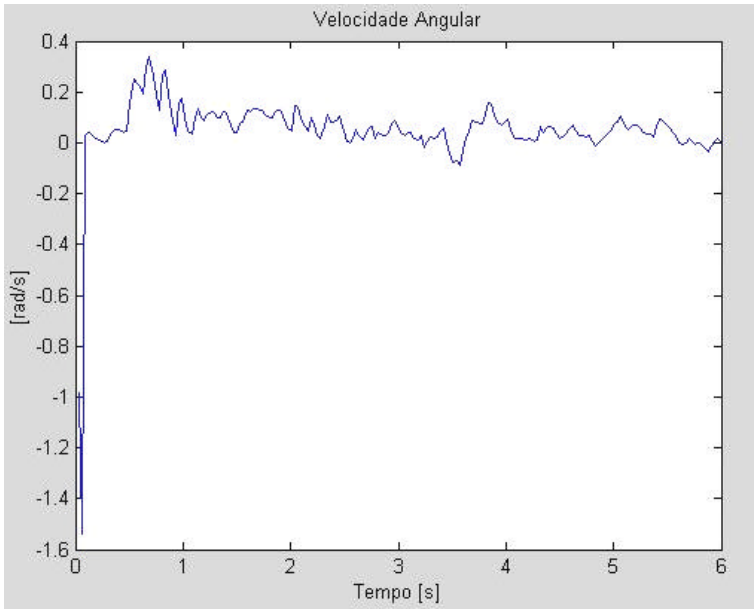


Figura 41 – Velocidade angular.

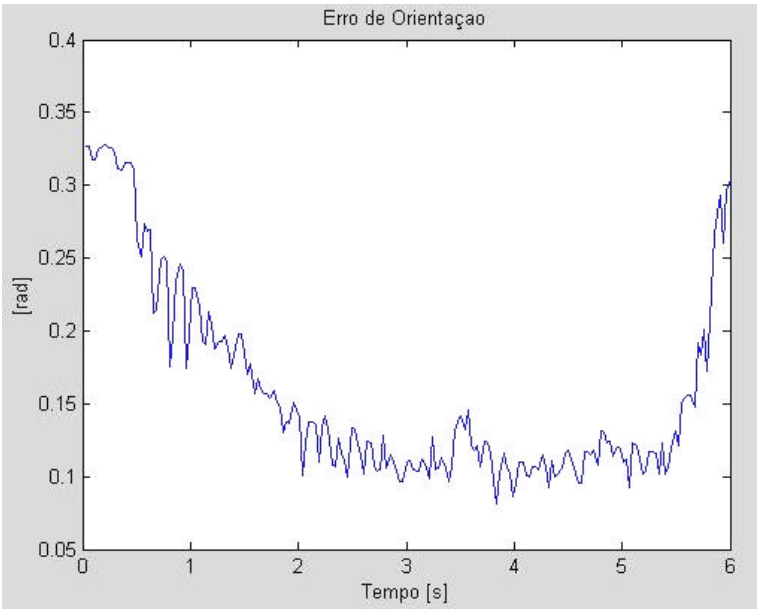


Figura 42 - Erro de orientação angular.

Na Figura 43 a tensão comum tem comportamento semelhante ao comportamento da velocidade linear, ou seja, vai diminuindo à medida que o robô se aproxima do ponto destino. Já a tensão diferencial, mostrada na Figura 44, só apresenta variações no início e no fim do experimento, quando o erro de orientação é um pouco maior.

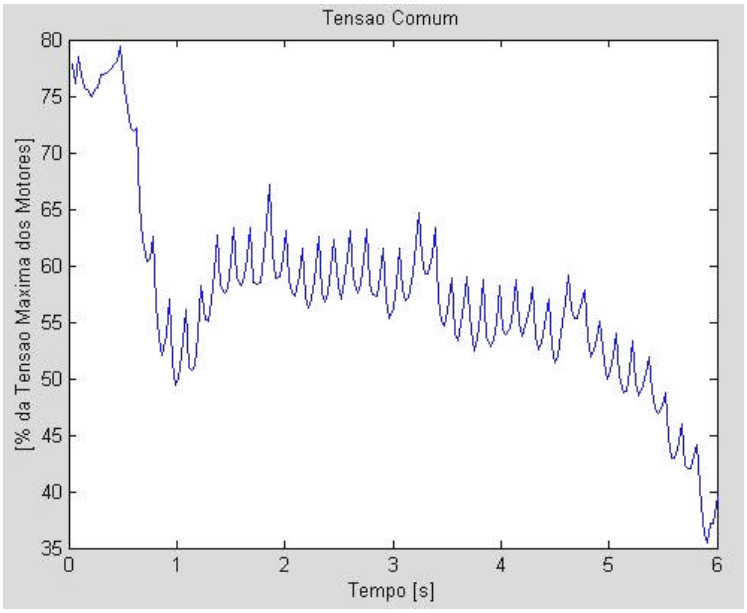


Figura 43 – Tensão comum.

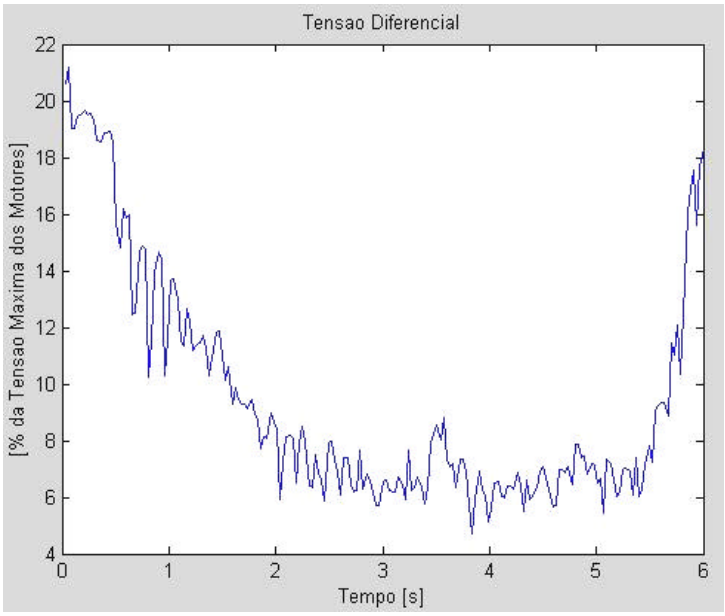


Figura 44 - Tensão diferencial.

4.2.2 Erro de orientação inicial grande

O objetivo deste experimento é analisar o comportamento do controlador quando o robô está em uma situação adversa. A Figura 45 mostra a posição inicial do robô, o ponto destino, o caminho do robô, plotado pelo *software* ao final do experimento, e ainda o erro de orientação elevado. A prioridade do controlador é orientar o robô em relação ao ponto destino, para depois fazer a aproximação.

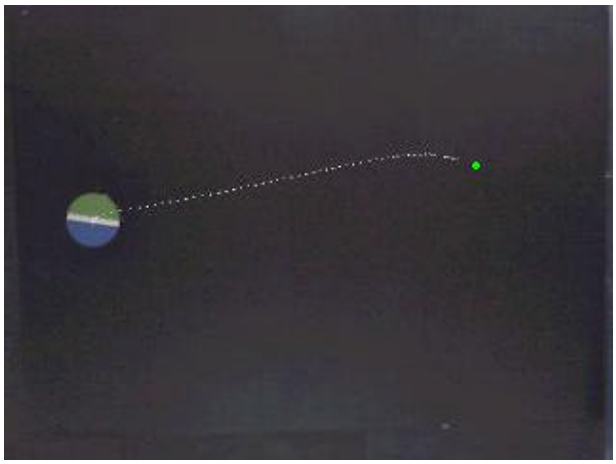


Figura 45 – Alcançar ponto destino com erro inicial de orientação grande.

Como podemos ver na Figura 46, a velocidade linear é pequena no início, até que o robô se oriente em relação ao ponto destino. A partir deste ponto, o experimento se torna semelhante ao experimento de erro de orientação inicial pequeno, onde a velocidade linear diminui à medida que o robô se aproxima do ponto destino.

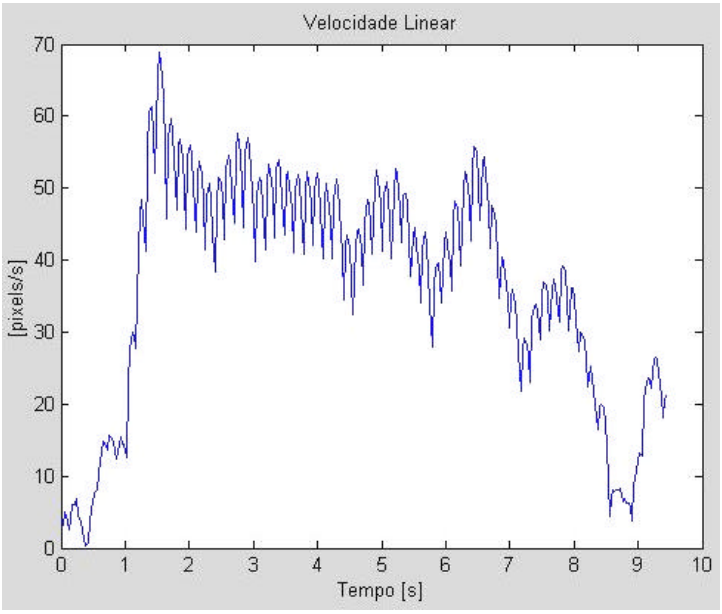


Figura 46 – Velocidade linear.

O erro *e* de posição é mostrado na Figura 47.

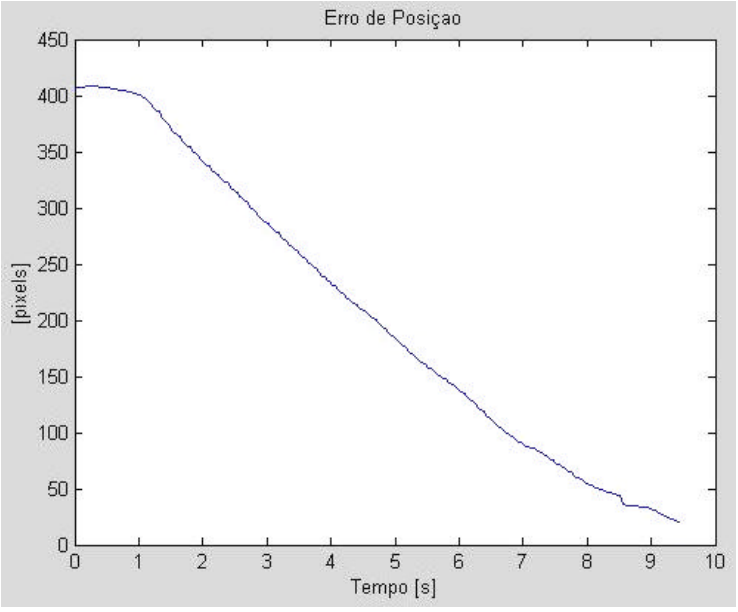


Figura 47 – Erro de posição.

Na Figura 48 e na Figura 49, é possível observar que a velocidade angular e o erro angular sofrem variações consideráveis no início, devido à característica do controlador de primeiro orientar o robô. Uma vez que o robô esteja orientado, o experimento se torna semelhante ao experimento de erro de orientação inicial pequeno, onde a velocidade e o erro angular praticamente não sofrem variações.

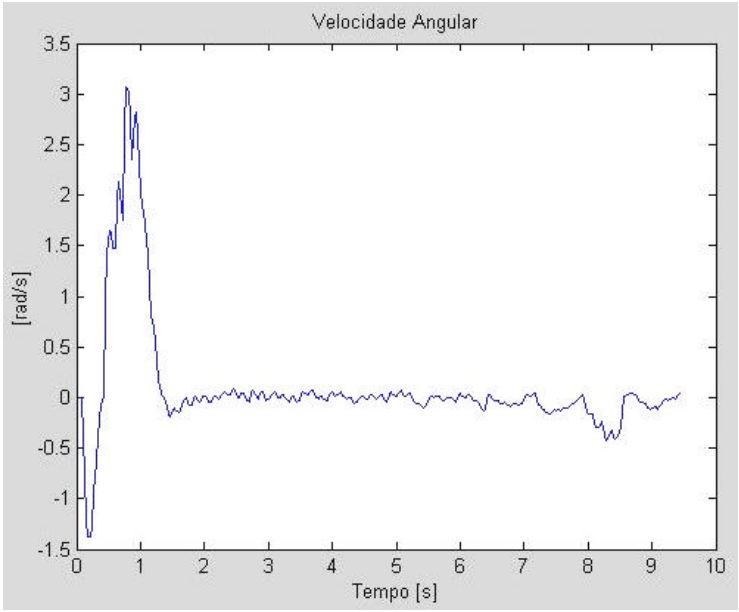


Figura 48 – Velocidade angular.

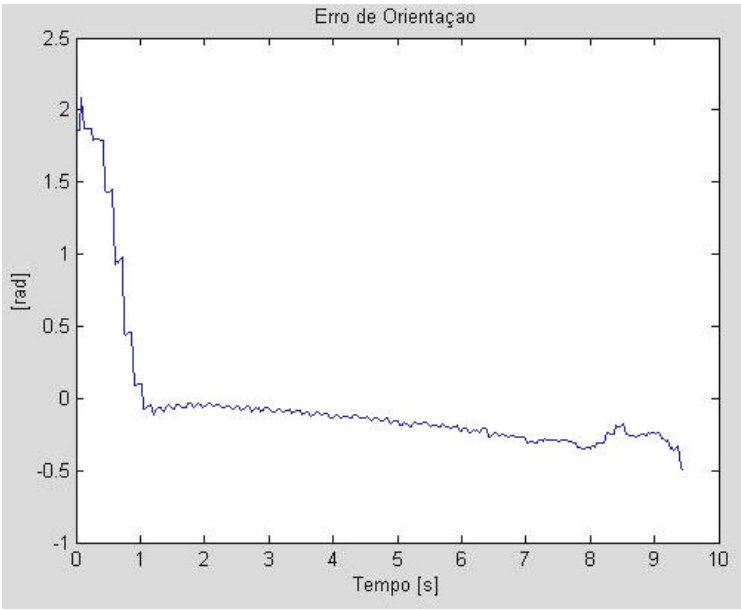


Figura 49 – Erro de orientação angular.

Na Figura 50 a tensão comum se mostra coerente com o experimento, de forma que no início seu valor é baixo e posteriormente ele é elevado. Já a tensão diferencial, mostrada na Figura 51, é elevada no começo, devido à reorientação do robô, e no restante do experimento sofre pequenas variações devido às pequenas oscilações do erro de orientação.

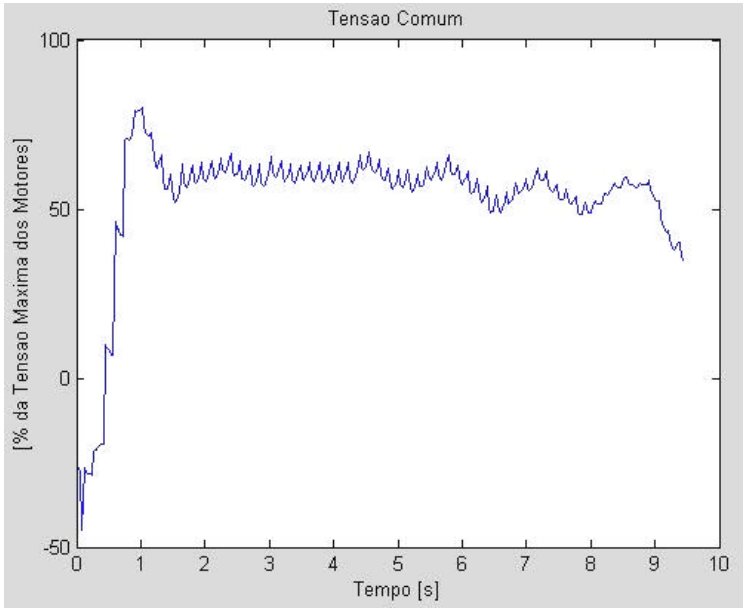


Figura 50 – Tensão comum.

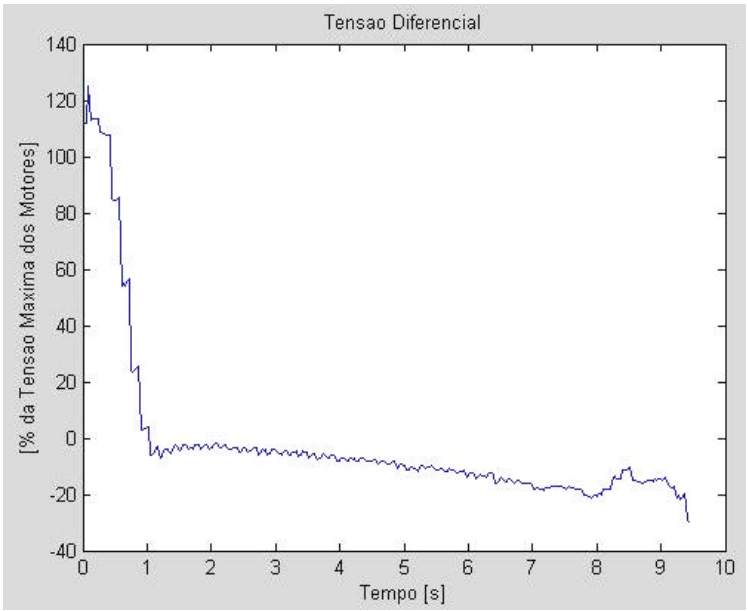


Figura 51 - Tensão diferencial.

4.3 Experimento 2: Percorrer Uma Trajetória na Imagem

Como vimos no Capítulo 3, este experimento pode ser particionado e resolvido como feito no caso do experimento de alcançar um ponto destino na imagem. A Figura 52 mostra a posição inicial do robô, os pontos por onde ele deve passar e o caminho do robô, plotado pelo *software* ao final do experimento.

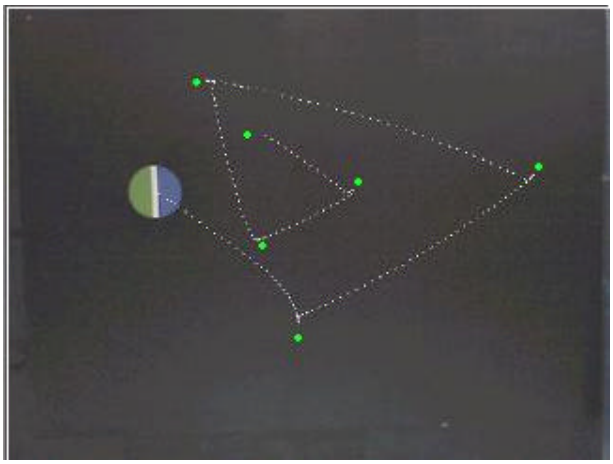


Figura 52 – Percorrer trajetória na imagem.

4.4 Experimento 3: Movimentos em Formação

Os tipos de movimento em formação são os mesmos descritos nos experimentos 2 e 3, que é, respectivamente, alcançar um ponto destino e percorrer uma trajetória na imagem. A única diferença é que, agora, o experimento é feito com dois robôs, onde o líder é que executa a trajetória passando pelos pontos determinados pelo usuário, ficando o robô escravo com a tarefa de seguí-lo, de forma que suas coordenadas são calculadas em função do robô líder. Como vimos no Capítulo 3, neste trabalho foram implementados dois tipos de formação, a formação em fila e a formação lado a lado.

4.4.1 Formação em fila

Para este caso a referência do robô escravo é sempre atrás do robô líder. A Figura 53 mostra a posição inicial dos robôs, os pontos por onde o líder deve passar e os caminhos percorridos por eles, plotados pelo *software* ao final do experimento.

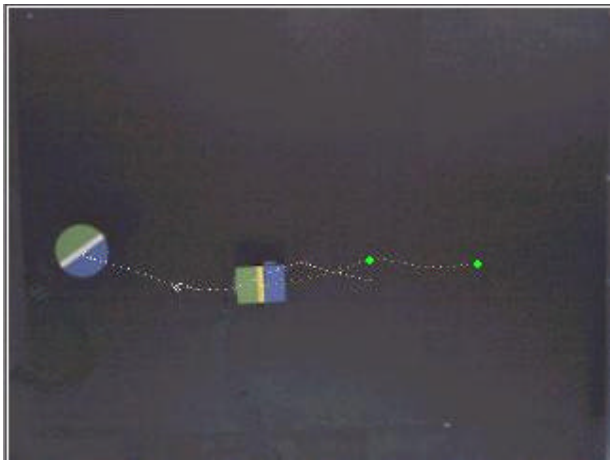


Figura 53 – Formação em fila.

4.4.2 Formação lado a lado

Para este caso a referência do robô escravo é sempre ao lado do robô líder. A Figura 54 mostra a posição inicial dos robôs, os pontos por onde o líder deve passar e os caminhos percorridos por eles, plotados pelo *software* ao final do experimento.

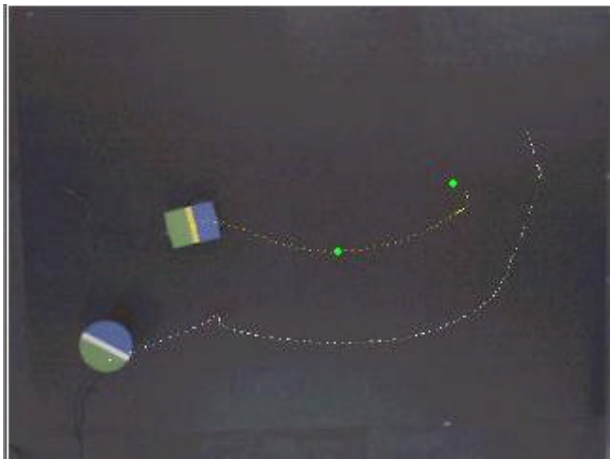


Figura 54 – Formação lado a lado.

4.5 Experimento 4: Empurrar Caixa

Como vimos no Capítulo 3, todos os experimentos de empurrar a caixa usam a idéia do experimento seguir trajetória, ou seja, fazer com que o robô passe por pontos pré-estabelecidos, tendo diferença apenas na forma de calcular os pontos de referência dos robôs, o que é feito dependendo da particularidade de cada problema. Nas

próximas sub-seções mostraremos a forma como os robôs se comportam para cada tipo de situação detectada na hora de empurrar a caixa, entre aquelas discutidas no Capítulo 3.

4.5.1 Robô na zona morta

Antes de começar a empurrar a caixa, é necessário observar se algum robô está na zona morta da caixa, como visto no Capítulo 3. Se isto acontecer é necessário tirar o robô (ou os robôs) desta zona morta, para só depois começar a empurrar a caixa. A Figura 55 mostra a posição inicial dos robôs, os pontos por onde eles devem passar e os caminhos percorridos por eles, plotados pelo *software* ao final do experimento.

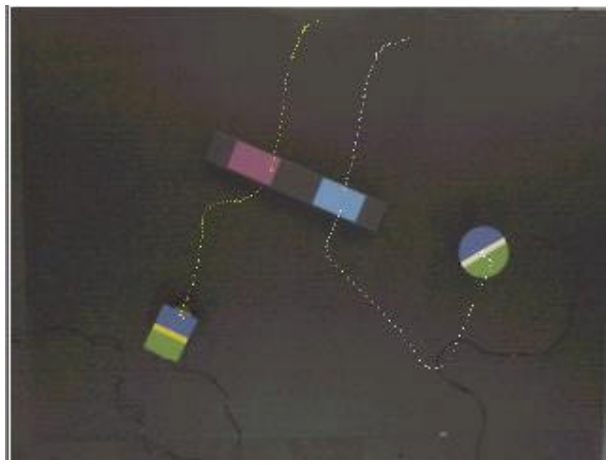


Figura 55 – Robô na zona morta.

4.5.2 Robôs do mesmo lado em relação ao eixo da caixa

Neste caso os robôs estão do mesmo lado em relação ao eixo da caixa e só precisam encostar nela para começar a empurrá-la até a borda do tablado. A Figura 56 mostra a posição inicial dos robôs, os pontos por onde eles devem passar e os caminhos percorridos por eles, plotados pelo *software* ao final do experimento.

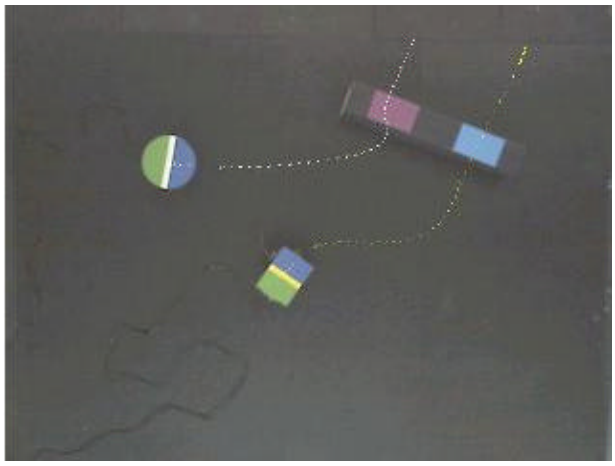


Figura 56 – Robôs do mesmo lado em relação ao eixo da caixa.

4.5.3 Robôs em lados opostos da caixa e nenhum deles encurralado

Neste caso os robôs estão em lados opostos em relação ao eixo da caixa, e nenhum deles está encurralado. Desta forma, é necessário que o robô que está na menor área do tablado percorra os pontos calculados para que ele mude para o mesmo lado do outro robô, para só depois começar a empurrar a caixa. A Figura 57 mostra a posição inicial dos robôs, os pontos por onde eles devem passar e os caminhos percorridos por eles, plotados pelo *software* ao final do experimento.

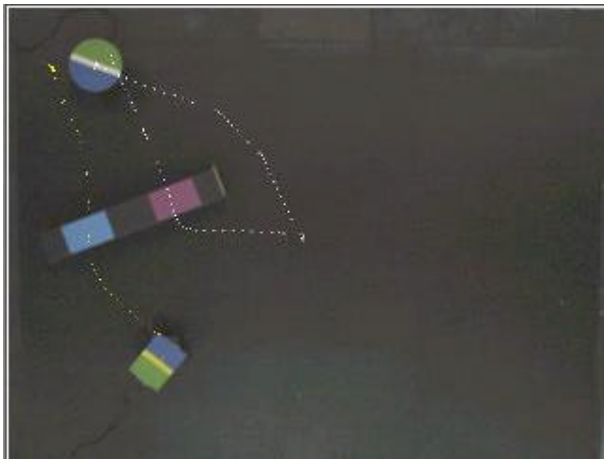


Figura 57 – Robôs em lados opostos da caixa e nenhum deles encurralado.

4.5.4 Robôs em lados opostos da caixa com um encurralado e outro distante

Neste caso os robôs estão em lados opostos em relação ao eixo da caixa, um deles está encurralado pela caixa e o outro apresenta uma distância segura em relação à caixa para que o robô encurralado possa girá-la. Desta forma, é necessário que o robô encurralado gire a caixa para que os dois fiquem do mesmo lado, para só depois começar a empurrá-la. A Figura 58 mostra a posição inicial dos robôs, os pontos por onde eles devem passar e os caminhos percorridos por eles, plotados pelo *software* ao final do experimento.

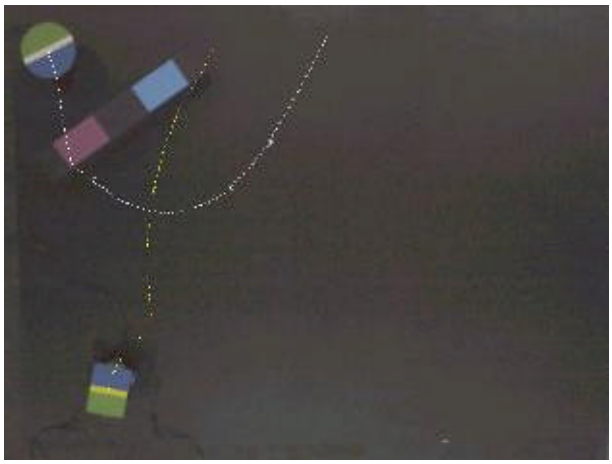


Figura 58 – Robôs em lados opostos da caixa, com um encurralado e outro distante.

4.5.5 Robôs em lados opostos da caixa com um encurralado e outro perto

Neste caso os robôs estão em lados opostos em relação ao eixo da caixa, um deles está encurralado pela caixa e o outro está muito perto desta, podendo colidir com ela quando o robô encurralado começar a girá-la. Desta forma, é necessário que o robô que não está encurralado se afaste da caixa, para que o robô encurralado gire-a para que os dois fiquem do mesmo lado, para só depois começar a empurrá-la. A Figura 59 mostra a posição inicial dos robôs, os pontos por onde eles devem passar e os caminhos percorridos por eles, plotados pelo *software* ao final do experimento.

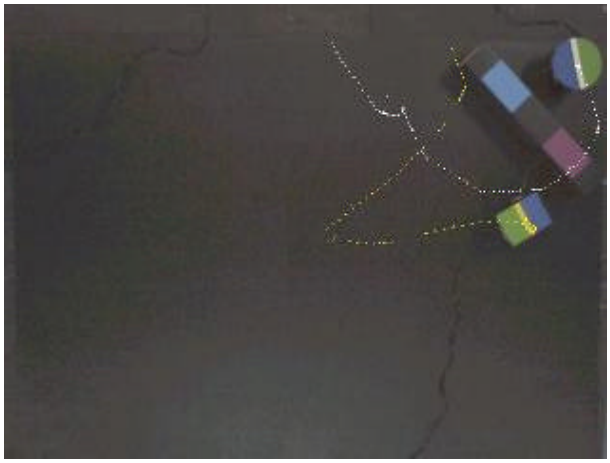


Figura 59 – Robôs em lados opostos da caixa, com um encurralado e outro perto.

5 CONCLUSÕES E TRABALHOS FUTUROS

5.1 Conclusões

Neste trabalho foram relatados os passos necessários à realização do controle de um robô através de realimentação visual, bem como a implementação de um controlador não linear de posição, a partir do qual foi possível controlar um ou mais robôs celulares para alcançar um ponto destino na imagem, e, posteriormente, estender esta idéia para mais de um ponto, no intuito de criar estratégias de controle. As estratégias de controle, implementadas a partir do controlador de posição, consistiam em tarefas simples como seguir uma trajetória na imagem ou permitir que os robôs se movimentassem obedecendo a uma determinada formação, e só depois se tornaram tarefas mais complexas, como verificar eventuais problemas antes de empurrar uma caixa para só depois tirá-la para fora do espaço de trabalho.

Uma característica interessante deste trabalho foi sua implementação utilizando C++, uma linguagem de programação de baixo nível, o que nos permitiu algumas particularidades como uma interface gráfica amigável com o usuário, e a aplicação do conceito de *multi-threads*, a partir do qual foi possível dividir o tempo de processamento para os dois robôs. O conceito de *multi-threads* deixa em aberto a possibilidade da inclusão de mais robôs, como no caso das pesquisas feitas pelo grupo de pesquisa em robótica do INAUT - Argentina, os quais utilizam a mesma estratégia de controle e conseguiram bons resultados com três robôs se movimentando com diferentes tipos de formação. Vale lembrar que bons resultados do sistema de controle dependem diretamente do hardware utilizado, uma solução para reduzir o tempo de aquisição das imagens seria a utilização de um hardware dedicado chamado *framegrabber* [9].

Nossa principal colaboração, neste trabalho, foi a criação das estratégias de controle que têm por objetivo tirar a caixa para fora do espaço de trabalho, bem como a verificação e solução de eventuais problemas que possam prejudicar este objetivo.

Como primeiro trabalho na linha de pesquisa de robôs móveis cooperativos no Programa de Pós Graduação em Engenharia Elétrica da UFES, concluímos que os resultados alcançados foram satisfatórios, levando em consideração o que foi desenvolvido, desde a construção dos robôs e do sistema físico até as estratégias de controle, passando pelo processamento de imagens.

5.2 Trabalhos Futuros

Como trabalhos futuros podemos pensar primeiramente na migração do sistema operacional Windows, utilizado neste trabalho, para um sistema que permita o processamento em tempo real, como por exemplo um sistema operacional Linux com o *kernel* modificado.

Outra proposta de trabalho futuro é a aplicação de controle de impedância para evitar obstáculo, onde um campo de repulsão seria criado em volta do obstáculo para evitá-lo, ou então criar este campo de repulsão em volta da área de trabalho, para evitar que o robô saia fora da mesma, e, conseqüentemente, do campo de captura da câmara.

Uma limitação encontrada neste trabalho é o espaço de trabalho limitado pelo uso da webcam. Para resolver este problema poderíamos utilizar mais de uma câmara, posicionadas estrategicamente sobre o espaço de trabalho, e posteriormente colar estas imagens para formar uma espécie de mosaico do mesmo, o que aumentaria consideravelmente o espaço para atuação dos robôs.

Os experimentos realizados neste trabalho ficaram limitados ao uso do controlador de posição. Uma proposta para desenvolvimentos futuros seria a implementação de um controle de orientação, que junto com o controle de posição permitiria desenvolver estratégias de controle mais complexas, principalmente no momento de manipular objetos para conduzi-los até uma posição específica, como a caixa utilizada neste trabalho.

Para evitar alguns problemas de processamento das imagens, causados por deficiências na iluminação e variações de luminosidade, poderíamos mudar o sistema de cor utilizado: ao invés de usar o modelo RGB poderíamos utilizar o modelo HSV. A principal vantagem do modelo HSV é o fato de poder separar a informação de cor (matiz no canal H) da informação de luminosidade, trazendo, assim, melhores resultados no momento de segmentar a imagem.

REFERÊNCIAS BIBLIOGRÁFICAS

[1] José Santos-Victor. “Vision Based Remote Control of Cellular Robots”. Robotics and Autonomous Systems, (23) pp. 221-234, 1998.

[2] José Santos-Victor, Ricardo Carelli, Sjoerd Van der Zwaan. “Nonlinear Visual Control of Remote Cellular Robots”, Mediterranean Control Conference, Lisbon, Portugal, 2002.

[3] Aicardi M., G. Casalino, A. Bicchi and A. Balestrino. “Closed Loop Steering of Unicycle-Like Vehicles via Lyapunov Techniques”. IEEE Robotics & Automation Magazine, Vol.2, No.1, pp.27-35, March 1995.

[4] Ogata, K. “Modern Control Engineering”. Prentice Hall, 1998.

[5] Benjamin C. Kuo, “Automatic Control Systems”, Prentice Hall, 1995.

[6] B.T. Rhatigan, P.R. Kalata, T.A. Chmielewski, “An a – ß Target Tracking Approach to the Benchmark Tracking Problem”, Proceedings of the American Control Conference, pp. 2076-2080, 1994.

[7] P.R. Kalata, Daka, S., Rawicz, P.L., Chmielewski, T.A., “On H_8 a – ß Target Tracking”, Proceedings of the Thirty-Sixth Southeastern Symposium on System Theory, pp. 26 – 30, 2004.

[8] B. Madden. “Kalman Filtering”, Internal Report, Department of Computer Science, University of Rochester, USA, 1996.

[9] Renzo Furlani, Andrés Gatica, Juan Videla, “Control de Robots Celulares Mediante Realimentación Visual”, INAUT, Trabajo Final de Graduación, Universidad Nacional de San Juan, San Juan, Argentina, 2002.

[10] Arkin, R. C. “Behavior Based Robotics”. MIT Press, 1998.

[11] Gonzales, Rafael C., Woods, Richard E. “Digital Image Processing”. Addison-Wesley, 1993.

[12] Foley, van Dam, Feiner, Hughes, “Computer Graphics”. Addison-Wesley, 1997.

[13] Ashitey Trebi-Ollennu, Hari Das, Hrand Aghazarian, Anthony Ganino, Paolo Pirjanian, Terry Huntsberger, and Paul Schenker, "Mars Rover Pair Cooperatively Transporting a Long Payload”, Proceedings of the IEEE International Conference on Robotics and Automation, Washington, D.C., May 11 - 15, 2002.

[14] Intel Image Processing Library v2.5, Reference Manual.

[15] Joseph L. Jones, Bruce A. Seiger, Anita M. Flynn, “Mobile Robots: Inspiration to Implementation”, Editorial A K Peters, 1999.

[16] Torres Müller, S. M., “Robô teleoperado por RF”, UFES , Projeto de Graduação, Universidade Federal do Espírito Santo, Vitória-ES, Brasil, 2004.

[17] Radiometrix Ltd., TX2 & RX2 Data Sheet, 2000. http://www.radiometrix.co.uk, Página acessada em 10 de Maio de 2004.

[18] Microchip Tecnology Inc., PIC16F87X Data Sheet, 1999. http://www.microchip.com, Página acessada em 10 de Maio de 2004.

[19] David J. Kruglinski, George Shepherd, Scot Wingo, “Programación avanzada con Microsoft Visual C++ 6.0”, Editorial McGraw-Hill, 1999.

APÊNDICE A – ASPECTOS CONSTRUTIVOS DOS ROBÔS UTILIZADOS

A.1 – Lego

Para construir a estrutura mecânica do robô *Lego*, mostrado na Figura 60, foram utilizadas peças do kit lego (*MindStorm*), além de dois motores CC também pertencentes ao kit.

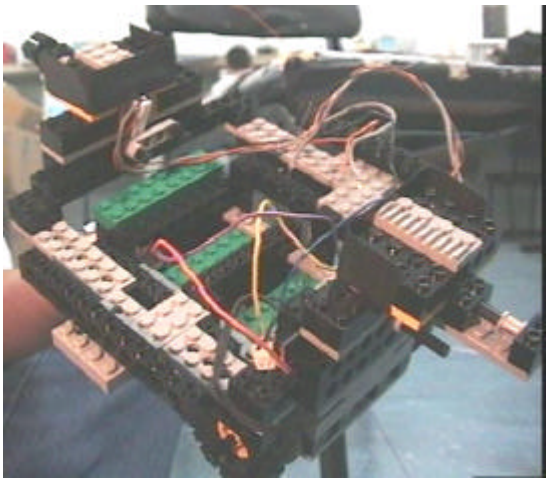


Figura 60 – Estrutura mecânica do robô *Lego*.

As especificações do motor, mostrado na Figura 61, são

- Tensão Máxima 9 V
- Rotação 350 rpm
- Consumo 350 mA (sem carga).
- Redução 10 para 1

Figura 61 – Um dos motores utilizados.

O responsável pelo controle do robô é o microcontrolador PIC 16F876A [18], mostrado na Figura 62. Além de entradas e saídas digitais, foram utilizadas do PIC as duas saídas PWM e a entrada de comunicação serial.

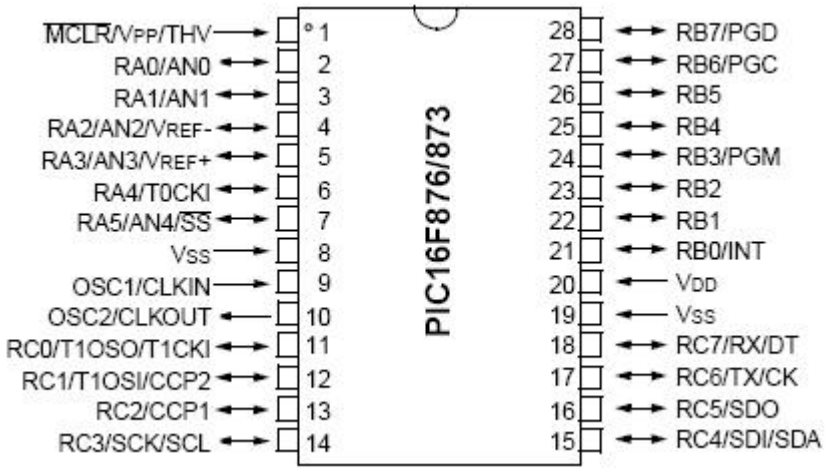


Figura 62 – PIC 16F876A utilizado no robô *Lego*

A Tabela 8 fornece as funções dos pinos do microcontrolador utilizados no *Lego*.

Tabela 8 – Função dos pinos utilizados do PIC.

Pinos	**Função**
2 (RA0)	Controle da ponte-H
3 (RA1)	Controle da ponte-H
5 (RA3)	Sinalização circuito ON
12 (CCP2)	Saída PWM 2
13 (CCP1)	Saída PWM 1
18 (RX)	Recebimento dos dados seriais
21 (INT)	Proteção externa
23 (RB2)	Sinalização de proteção ON

A.2 – Rug Warrior

O *Rug Warrior*, mostrado na Figura 63, foi desenvolvido por pesquisadores do Laboratório de Inteligência Artificial do Instituto Tecnológico de Massachusetts (Massachusetts Institute of Technology). Ele é um dos robôs utilizados neste projeto de cooperação com realimentação visual. Ele tem dimensões 18,5 cm de diâmetro por 12 cm de altura, e pode ser utilizado para diversas funções, dentre elas a cooperação. Está disponibilizado à venda sob a forma de um kit de hardware, detalhado em [15].

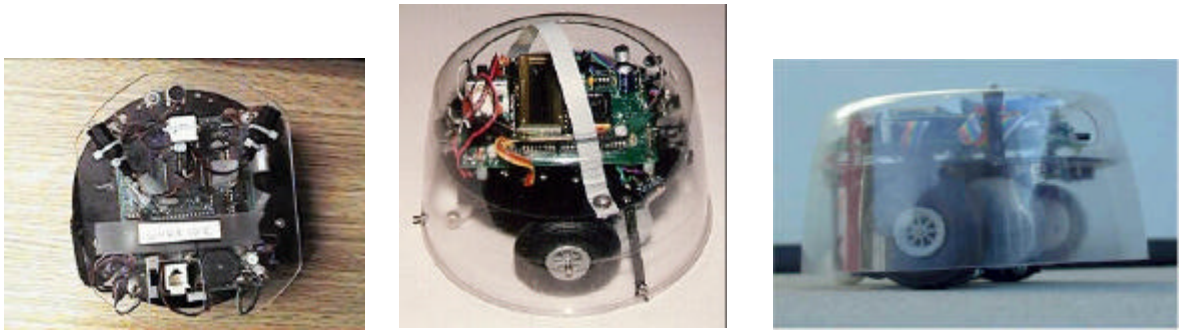


Figura 63 – Robô *Rug Warrior* utilizado.

O kit contém todos os itens necessários à montagem, como sensores, hardware mecânico e eletrônico, microprocessador, memória, motores, display de cristal líquido e circuitos de interface. Na Figura 64 é mostrada a disposição dos componentes na placa do *Rug Warrior*.

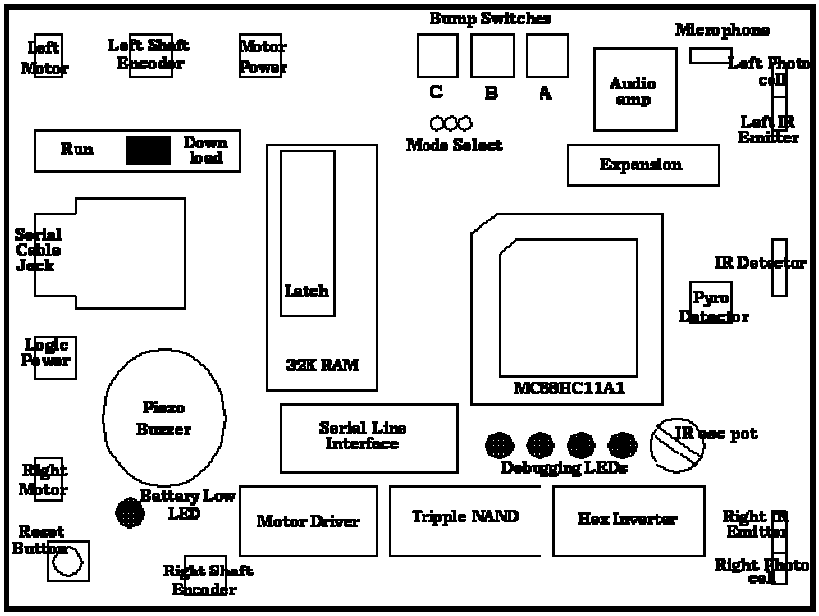


Figura 64 – Disposição dos componentes na placa do *Rug Warrior*.

A Figura 65 mostra o diagrama de blocos do sistema do robô.

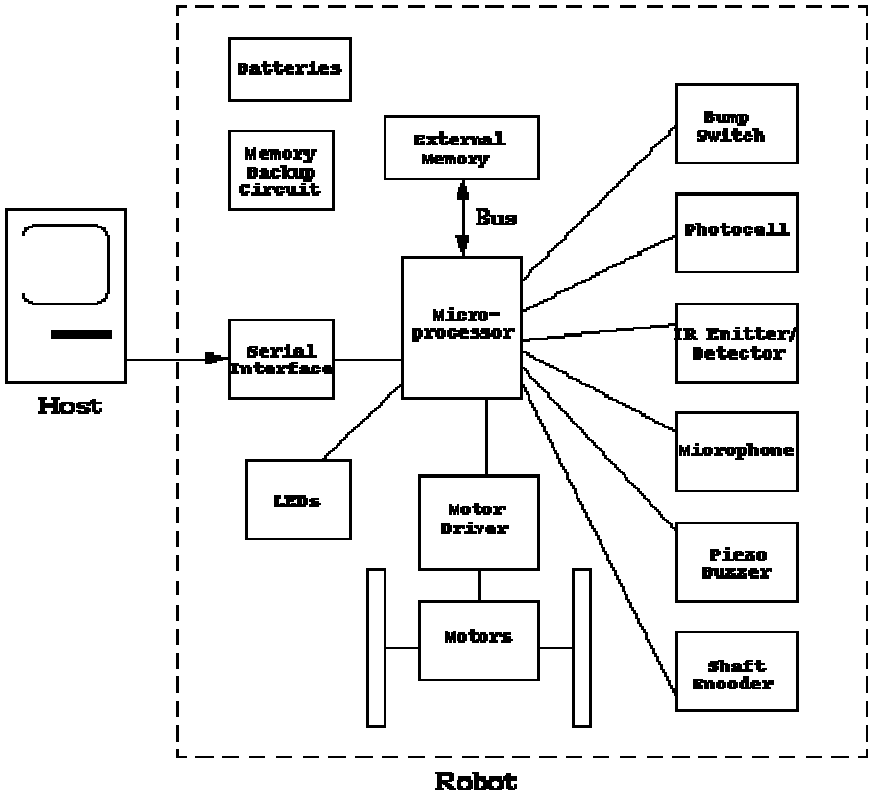


Figura 65 – Diagrama de blocos da organização do sistema do robô

O *Rug Warrior* é programado diretamente pelo computador, através da porta serial. O ambiente para programação é o Interactive C. Um ambiente de programação fácil de se trabalhar e que permite ao usuário testar e "debugar" partes do programa que deve ser carregado no robô. O microprocessador responsável pelo controle é o da Motorola MC68HC11A0. A Figura 66 mostra sua estrutura interna.

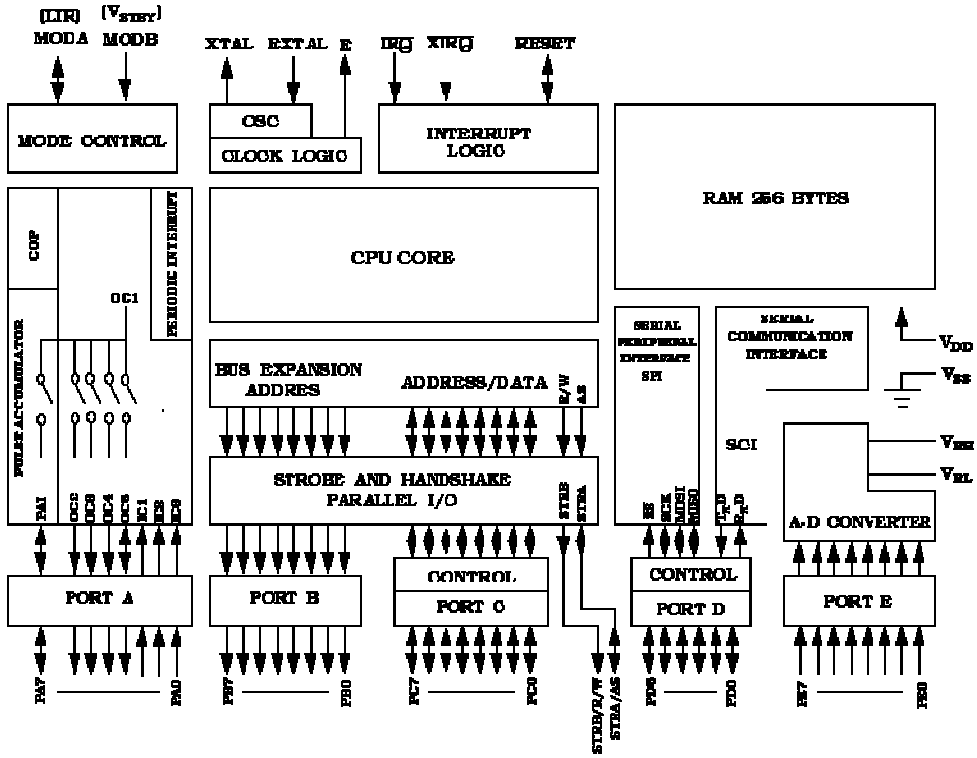


Figura 66 – Estrutura interna do microcontroador MC68HC11A0.

São vários os sensores que compõem o *Rug Warrior*, como sensores de infravermelho (emissor e receptor), posicionados nas laterais direita e esquerda, sensores de contato posicionados nas laterais direita e esquerda do robô e na sua traseira, encoders nas rodas para que o microprocessador possa calcular distâncias percorridas, sensor de ultra-som e microfone. Porém, o *Rug Warrior* foi configurado para mover um objeto até um ponto determinado pelo usuário em cooperação com outro robô. Houve a necessidade, portanto, de desabilitar os sensores antichoque.

O *Rug Warrior* apresenta três portas de entrada analógicas e três linhas digitais que não foram utilizadas na montagem padrão. Essas linhas de entrada podem ser

utilizadas para implementação de outros sensores. As especificações da plataforma robótica utilizada neste projeto podem ser observadas na Tabela 9.

Tabela 9 – Especificações técnicas do *Rug Warrior*.

Componente	***Típico***	***Mínimo***	***Máximo***	***Unidade***
Tensão de alimentação (circuito lógico)	5,0	4,6	7,0	V
Tensão de alimentação (motor)	5,0	4,0	9,0	V
Freqüência de clock do microprocessador	2.0	-	-	MHz
Velocidade de transmissão serial	9600	75	312500	bps
Freqüência de oscilação do infravermelho	40	38	42	kHz
Dist. de detecção de objetos pelo infravermelho	30	-	-	cm
Velocidade do robô	0.20	-	-	m/s
Cliques do encoder por rotação	16	-	-	unidades
Peso do robô (sem baterias)	765	-	-	g
Corrente do motor (cada)	1,0	-	-	A
Diâmetro da roda	6,3	-	-	Cm
Diâmetro do robô	18,5	-	-	cm
Altura	12,0	-	-	cm

A.3 – Sistema de Comunicação Implementado

A cooperação entre os robôs é controlada pelo computador, através de realimentação visual. Após a chegada dos dados provenientes da webcam, o computador precisa processar as imagens e enviar informações sobre a velocidade e

direção dos motores aos robôs. Utiliza-se a interface RS-232 para enviar os dados ao transmissor.

Nessa interface o nível lógico “alto” é associado a sinais com tensão entre –3 volts e –25 volts e nível lógico “baixo” a sinais com tensão entre +3 volts e +25 volts. A faixa de tensões entre –3 volts e +3 volts é considerada uma região instável, como visto na Figura 67.

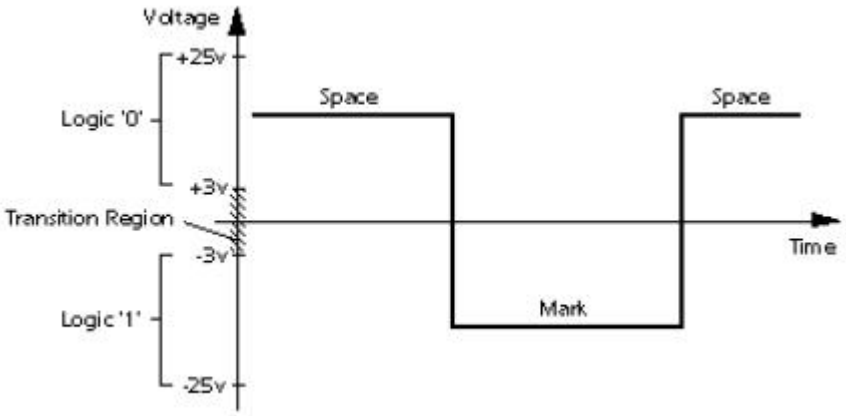


Figura 67 – Sinais RS-232.

Na Figura 68 é mostrada a pinagem do conector DB9 utilizado no projeto.

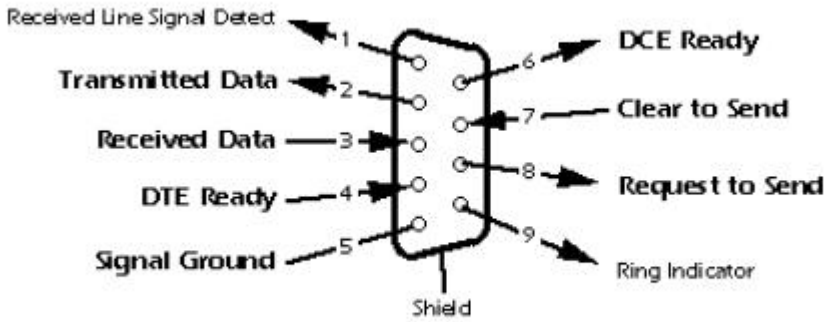


Figura 68 – Pinagem do conector DB9 utilizado.

Tanto o transmissor quanto o receptor FM utilizam sinais TTL (0 a 5V). Portanto é necessária a conversão de sinais do padrão RS-232 para o TTL. Isto é feito por conversores de nível, como por exemplo o CI MAX232 da *Maxim*, mostrado na

Figura 69. Ele inclui um circuito conversor CC capaz de gerar tensões de +10 volts e -10 volts a partir de uma fonte de alimentação simples de +5 volts, bastando, para isso, alguns capacitores externos.

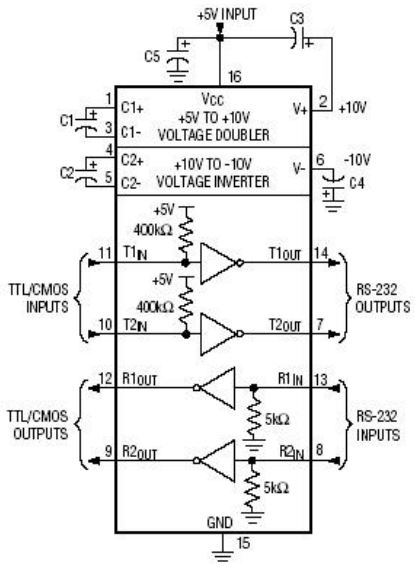


Figura 69 - Circuito interno do MAX232.

Na Figura 70 são representados os circuitos eletrônicos de transmissão e de recepção, em diagrama de blocos. Os circuitos de alimentação são omitidos, mas vale ressaltar que são alimentados com 5 V, a partir de uma bateria de 9 V em série com um regulador de tensão de 5 V, o LM7805.

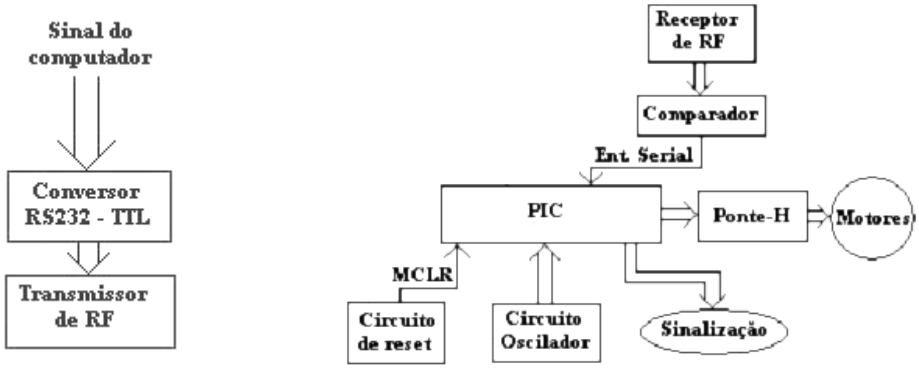


Figura 70 - Diagrama de blocos dos circuitos de transmissão e de recepção.

Foram utilizados dois pares transmissor-receptor de rádio UHF, da *Radiometrix* [17], mostrado na Figura 71, um par operando a 433 e outro par operando a 418 MHz. Sua

modulação é em FM, e ele é projetado para funcionar em um raio de até 75 m, para ambientes fechados, e 300 m, para ambientes abertos. A Figura 72 e a

Tabela 10 mostram as características do módulo transmissor e a Figura 73 e a Tabela 11 mostram as características do módulo receptor.

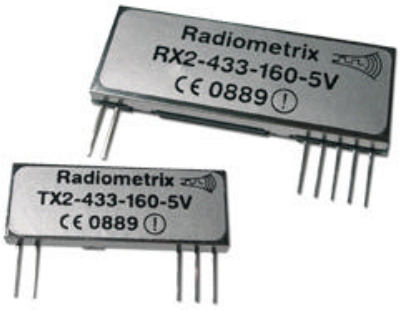


Figura 71 – Par transmissor-receptor de 433 MHz da *Radiometrix*.

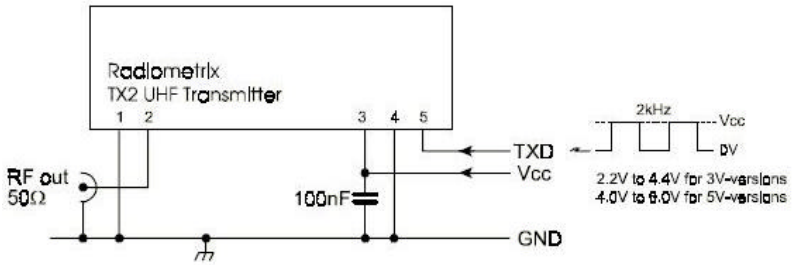


Figura 72 – Pinagem do módulo transmissor.

Tabela 10 – Descrição da pinagem do módulo transmissor.

Pino 1 (RF GND)	Pino terra, conectado internamente ao pino 4.
Pino 2 (RF out)	Saída RF de 50 O para a antena, que é nada mais que um fio de15,5 cm, calculado conforme manual.
Pino 3 (Vcc)	Alimentação de 5 V, podendo excursionar de 4 V a 6 V. O módulo irá gerar RF quando o mesmo for alimentado. Um capacitor de 10 µF e resistência de 10 O sobre a alimentação é sugerido pelo manual.
Pino 4 (0V)	Conectar ao terra do circuito.
Pino 5 (TXD)	Entrada do sinal a ser modulado (níveis de 0 V a Vcc)

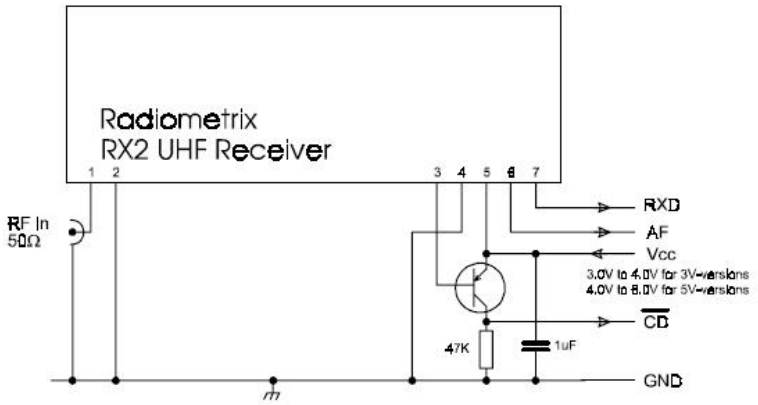


Figura 73 – Pinagem do módulo receptor.

Tabela 11 – Descrição da pinagem do módulo receptor.

Pino 1 (RF in)	Entrada RF de 50 O para a antena, que é nada mais que um fio de15,5cm, calculado conforme manual.
Pino 2 (RF GND)	Pino terra, conectado internamente ao pino 4.
Pino 3 (Carrier Detect)	O detector de portadora pode ser usado para acionar um transistor PNP que indica a existência ou não da portadora no sinal recebido. Neste trabalho este sinal não foi utilizado, sendo então conectado ao Vcc.
Pino 4 (0V)	Conectar ao terra do circuito.
Pino 5 (Vcc)	Alimentação de 5 V, podendo excursionar de 4 V a 6 V. O módulo irá gerar RF quando o mesmo for alimentado. Um capacitor de 10 µF e resistência de 10 O sobre a alimentação é sugerido pelo manual.
Pino 6 (AF)	Saída analógica do demodulador FM.
Pino 7 (RXD)	Saída digital do sinal modulado.

A concepção inicial era usar um transmissor e dois receptores (um para cada robô). Uma palavra seria enviada junto com os dados enviados, diferenciando para que robô se destinaria o dado. Mas por questão de disponibilidade de material (apenas dois receptores com freqüências diferentes), utilizou-se dois transmissores (418 e 433 MHz) e dois receptores (418 e 433 MHz).

O circuito de transmissão, mostrado na Figura 74, funciona da seguinte forma: o computador envia o sinal RS232 para a placa, o CI MAX232 converte o sinal para TTL. Como este sinal está conectado ao transmissor de RF, ele é enviado via rádio freqüência até o receptor.

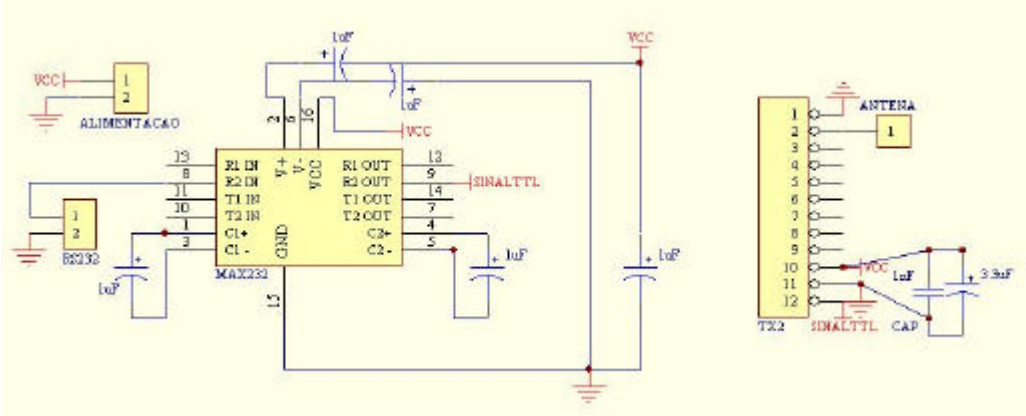


Figura 74 - Esquemático do circuito de transmissão.

Em relação ao circuito de recepção do *Lego*, mostrado na Figura 75, o receptor de RF converte a informação do receptor em um sinal TTL. Esse sinal é enviado a um comparador (INA118) e a sua saída é enviada para o microcontrolador. O motivo do uso do comparador é que o sinal TTL na saída do receptor é muito ruidoso [16], como visto na Figura 76, mesmo com o uso do resistor de “pull-up”. Assim, o sinal é comparado com um nível de tensão próximo de zero, resultando em um sinal mais “limpo”. A escolha do INA118 se justifica porque este possui níveis de saturação, com uma alimentação unipolar de 5 V, de 0,2 V (nível baixo) e 4,2 V (nível alto). Isto é compatível com os níveis de tensão da entrada RX do PIC.

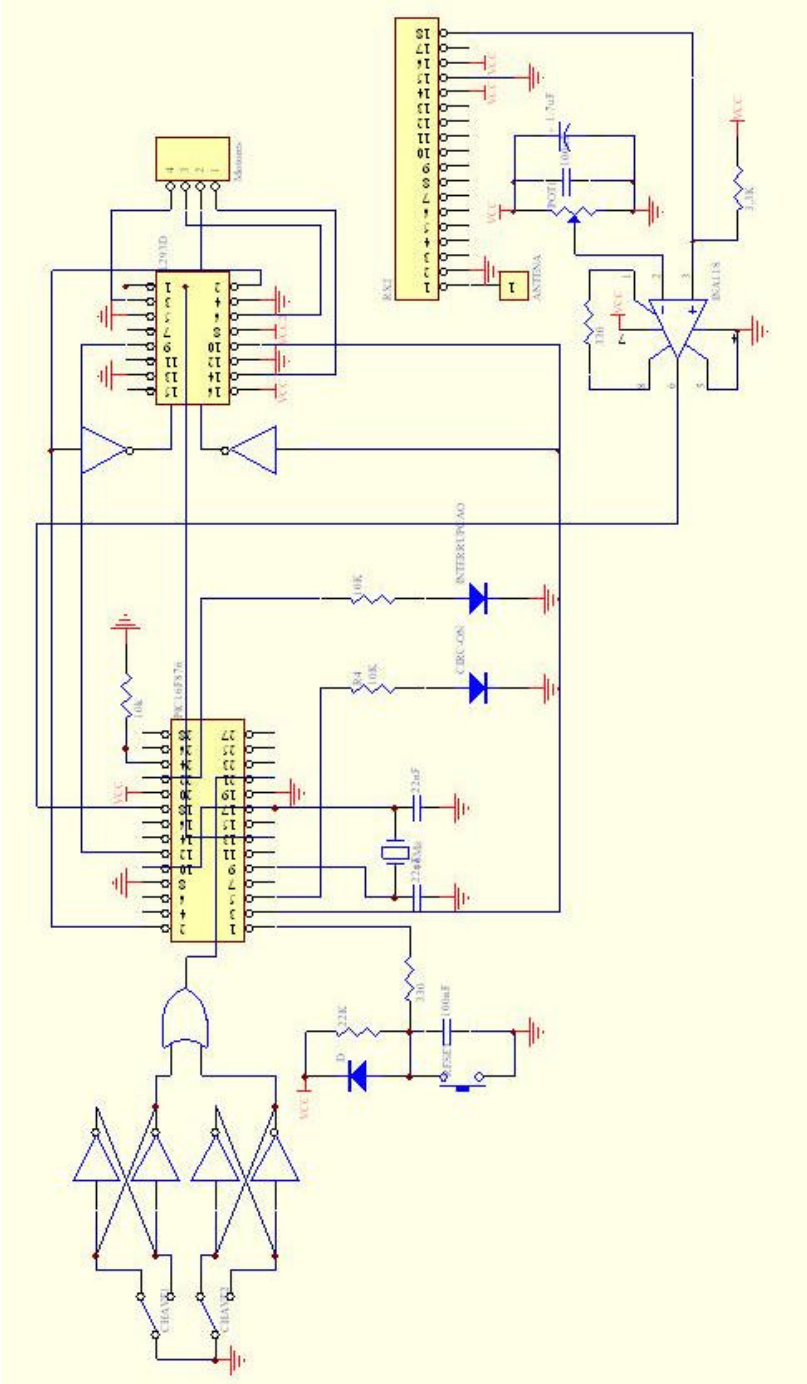


Figura 75 - Esquemático do circuito de recepção

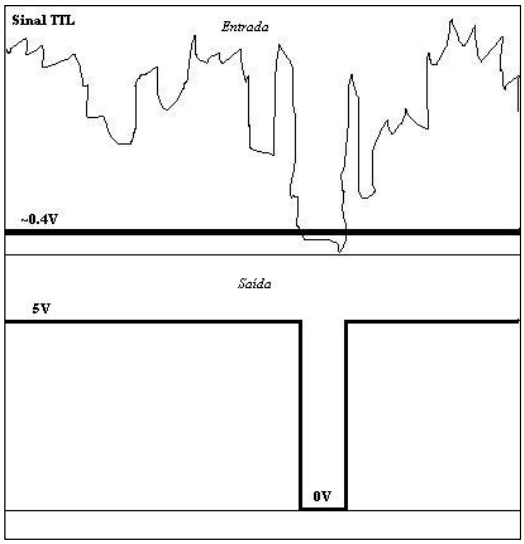


Figura 76 – Simulação do ruído da saída do receptor.

De acordo com o programa gravado na memória do PIC, um pacote de dados recepcionados gera duas saídas PWM nos seus pinos 12 e 13. Este sinal vai para a ponte-H (L293D), assim como dois sinais digitais (pinos 2 e 3), indicando o sentido de velocidade para cada motor. Toda a placa é alimentada com uma tensão de 5 V proveniente de um regulador de tensão (LM7805) que é alimentado por uma bateria de 9 V. A ponte-H necessita ainda ser alimentada com outro valor de tensão diferente de 5 V. Esta tensão alimentará os dois motores, e, neste trabalho, foi utilizada uma segunda bateria de 9 V.

O circuito de recepção do *Rug Warrior* se resume a uma placa que acomoda o receptor e um CI MAX232, usado para converter o sinal TTL do receptor para RS-232, visto que a porta de comunicação serial do *Rug Warrior* se comunica neste formato.

A.4 – Programas Embarcados nos Robôs

A programação dos robôs foi executada com facilidade, visto que toda a estratégia de controle desenvolvida é executada no computador. Desta forma, o único objetivo do programa embarcado no robô é acionar os motores de forma a imprimir a velocidade estabelecida no controlador de posição. No robô *Rug Warrior* foi utilizado

o Interactive C, o qual possui funções próprias de comandos do robô. No caso do robô *Lego*, foi utilizado o PIC, como vimos anteriormente, juntamente com o software de programação PICC da C Compiler IDE e o software Mplab, responsável por carregar o programa no PIC.

Nos dois robôs os programas são bem simples e consiste em receber um pacote de mensagens composto por seis bytes. São eles:

- Identificador: indica para qual é a mensagem;
- Velocidade do motor esquerdo: valor que varia de -100 a +100;
- Velocidade do motor direito: valor que varia de -100 a +100;
- Byte “checksum”: usado para detecção de erro de transmissão.

Ao receber o pacote de informação, o receptor envia estes dados ao microcontrolador, o qual interpreta os dados e executa comandos nos motores de acordo com o pacote enviado. Os programas embarcados nos robôs serão mostrados nas próximas seções.

A.4.1 – Programa embarcado no *Lego*

```
#include <16F876A.h>
#include <stdlib.h>
#use delay(clock=16000000)
#fuses HS,NOWDT,PUT,NOPROTECT,NOBROWNOUT,NOLVP,NOCPD,NOWRT
#use rs232(baud=9600,parity=N,xmit=PIN_C6,rcv=PIN_C7,errors)

// Variaveis do programa principal
signed int dados[3];
int vazio[20];
int1 dados_ok = 0;
int1 check_ok = 0;
/* dados[3] representa:
 [0]-velocidade do motor esquerdo
 [1]-velocidade do motor direito
 [2]-checksum */

// Funcao que comanda o motor do robo
void comandamotor(){
        long int ciclo1,ciclo2;

        //determina o sentido de cada motor: PIN_A0 = esquerdo e PIN_A1 = direito
        if (dados[0]>0) output_high(PIN_A0);
```

```
		else output_low(PIN_A0);
		if (dados[1]>0) output_high(PIN_A1);
		else{
			output_low(PIN_A1);
			if (dados[1] != 0){		//Quando a velocidade da Roda esquerda (2ª)
				dados[1] = -100 - dados[1];   //é negativa o módulo da velocidade fica complementar a 100
			}
		}

		//calcula o PWM e comanda o motor
		dados[0]=abs(dados[0]);
		dados[1]=abs(dados[1]);
		if ((dados[0] < 101) && (dados[1] < 101)){

			ciclo1 = (signed long int)(((signed int32)(dados[0])*1023)/100);
			ciclo2 = (signed long int)(((signed int32)(dados[1])*1023)/100);

			set_pwm1_duty (ciclo1); //ativa o pwm
			set_pwm2_duty (ciclo2);
		}
}
// Funcao que coleta os dados
#int_RDA
RDA_isr() {
		signed int dado;
		static int i=0;
		dado = getc();
		if (kbhit())
			output_high(PIN_A3);

		if (check_ok){
			dados[i] = dado; //armazena o dado
			i++;
		}
		else
		if (dado == 102){ //checa o start byte
		i = 0;
		check_ok = 1;
		}

		if (i >= 3){ //finaliza a coleta
		check_ok = 0;
		if (dados[2] == (dados[0]^dados[1])) dados_ok = 1;
		}
}

//Programa Principal
void main() {

		// configuracao do PIC
		setup_adc_ports(NO_ANALOGS);
```

```
        setup_adc(ADC_CLOCK_DIV_2);
        setup_spi(FALSE);
        setup_counters(RTCC_INTERNAL,RTCC_DIV_2);
        setup_timer_1(T1_DISABLED);
        setup_timer_2(T2_DISABLED,0,1);
        setup_ccp1(CCP_OFF);
        setup_ccp2(CCP_OFF);
        //output_high(PIN_A3);
        enable_interrupts(global);
        enable_interrupts(INT_RDA);
        ext_int_edge (0, L_TO_H); //especifica borda de subida para interrupcao
        setup_timer_2 (T2_div_by_16,255,1);
        setup_ccp1 (ccp_pwm);   //configura ccp1 para modo pwm
        setup_ccp2 (ccp_pwm);   //configura ccp2 para modo pwm
        set_pwm1_duty(0);   //configura o ciclo ativo em 0 (desligado)
        set_pwm2_duty(0);   //configura o ciclo ativo em 0 (desligado)
        dados_ok = 0;
        check_ok = 0;

          //Inicio - ciclo infinito
          while(true){
                if (dados_ok)
                {
                        disable_interrupts(INT_RDA);
                        comandamotor();
                        dados_ok = 0;
                        enable_interrupts(INT_RDA);
                }//if
          }//while
}//main
```

A.4.2 – Programa embarcado no *Rug Warrior*

```
/* Programa de Comunicação para o Rug Warrior */
   /* Com detecção de erro com Check Sum */

int Inicio, Motor_0, Motor_1, Ok;

void main()
{
        init_velocity();

        poke(0x3C,1);           /* Desabilita o recebimento de programas pela porta serial */

        poke(0x102B,0b00110000); /* Seta Baud Rate p/ 9600 */

        while(bumper() != 4){

                while(!(peek(0x102E) & 0x20))
```

```
                {}
                Inicio = peek(0x102F);

                if (Inicio == 101){

                        Inicio = 0x00;

                        while(!(peek(0x102E) & 0x20))
                        {}
                        Motor_1 = peek(0x102F);

                        while(!(peek(0x102E) & 0x20))
                        {}
                        Motor_0 = peek(0x102F);

                        while(!(peek(0x102E) & 0x20))
                        {}
                        Ok = peek(0x102F);

                        if ((Motor_0 ^ Motor_1) == Ok){

                                if (Motor_0 >= 156){/*Valores negativos vinham em complemento de 2*/
                                        Motor_0 = Motor_0 - 256;
                                }

                                if (Motor_1 >= 156){
                                        Motor_1 = Motor_1 - 256;
                                }

                                if((Motor_0 >=-100) && (Motor_0 <=100) && (Motor_1 >=-100) && (Motor_1 <=100)){
                                        motor(0,Motor_0);
                                        motor(1,Motor_1);
                                        printf("m1= %d, m2= %d\n", Motor_0,Motor_1);
                                }
                        }
                }
        }

poke(0x3C,0);       /* Habilita o recebimento de programas pela porta serial */
printf("Saiu\n");
}
```

APÊNDICE B – FORMULAÇÃO GEOMÉTRICA UTILIZADA

B.1 – Equações da Reta:

B.1.1 – Dado um ponto P(x_c, y_c) e o coeficiente angular tan(a):

$$\frac{y - y_c}{x - x_c} = \tan(a) \qquad (B1)$$

Equação Geral:

$$-\tan(a)\,x + y - (y_c - \tan(a) \cdot x_c) = 0 \qquad (B2)$$

Equação reduzida:

$$y = \tan(a) \cdot x + (y_c - \tan(a) \cdot x_c) \qquad (B3)$$

B.1.2 – Dados dois pontos P1(x_1, y_1) e P2(x_2, y_2):

$$\begin{vmatrix} x_1 & y_1 & 1 \\ x_2 & y_2 & 1 \\ x & y & 1 \end{vmatrix} = 0 \qquad (B4)$$

Equação Geral:

$$\left(\frac{y_2 - y_1}{x_2 - x_1}\right)x - y + \left(\frac{x_2 y_1 - x_1 y_2}{x_2 - x_1}\right) = 0 \qquad (B5)$$

Equação reduzida:

$$y = \left(\frac{y_2 - y_1}{x_2 - x_1}\right)x + \left(\frac{x_2 y_1 - x_1 y_2}{x_2 - x_1}\right) \qquad (B6)$$

B.2 – Distância de um Ponto P(x_c, y_c) a uma Reta *r*:

$$r: \quad ax + by + c = 0$$

$$d_{pr} = \left| \frac{ax_c + by_c + c}{\sqrt{a^2 + b^2}} \right| \qquad (B7)$$

B.3 – Ponto de Interseção de Duas Retas *r* e *s*:

$$r: \quad y = a_1 x + b_1$$

$$s: \quad y = a_2 x + b_2$$

$$X_{\text{int}} = \frac{b_2 - b_1}{a_1 - a_2} \qquad (B8)$$

$$Y_{\text{int}} = a_2 X_{\text{int}} + b_2 \qquad (B9)$$

onde

(X_{int}, Y_{int}) representa as coordenadas do ponto de interseção das duas retas.

APÊNDICE C – SOFTWARE PARA O SISTEMA DE CONTROLE

Como vimos anteriormente, toda a parte de processamento é feita no computador, desde a aquisição e processamento das imagens até o envio dos dados aos robôs através da porta serial, incluindo a aplicação do controlador e a coleta dos dados relativos a cada experimento para posterior análise, bem como a interface gráfica com o usuário. Como mencionamos na introdução, devido a um convênio promovido pela CAPES entre a Universidade Federal do Espírito Santo - Brasil (UFES) e a Universidad Nacional de San Juan – Argentina (UNSJ), foi possível aproveitar a experiência já adquirida por eles, nesta linha de pesquisa de robôs móveis cooperativos. Com isso, foi facilmente implementada toda a parte de interface com os dispositivos externos ao computador, como a webcam e os robôs, sobrando mais tempo para implementar estratégias de controle.

O programa implementado mescla duas filosofias distintas de programação. A primeira filosofia é a programação orientada a objetos, a partir da qual foi possível criar a interface gráfica com o usuário através da biblioteca MFC (*Microsoft Foundation Class*) [19]. A segunda filosofia implementada foi a construção de funções globais que se encarregaram de todo o processamento necessário à implementação do sistema de controle [9].

Em linhas gerais podemos dizer que para cada robô que compõe o sistema é necessário um timer, de modo que cada timer “roda” dentro de uma *thread*. Vale a pena lembrar que programação *multi-thread* não é a mesma coisa que programação multi-tarefa, visto que o Windows não é um sistema de tempo real.

O fluxograma do programa principal é mostrado na Figura 77. A rotina empurrar caixa, neste fluxograma, era muito grande, de forma que foi feito um fluxograma só para o algoritmo desta rotina, o seu fluxograma pode ser visto na Figura 78. Finalmente, na Figura 79, podemos ver o fluxograma do timer que é criado para cada robô.

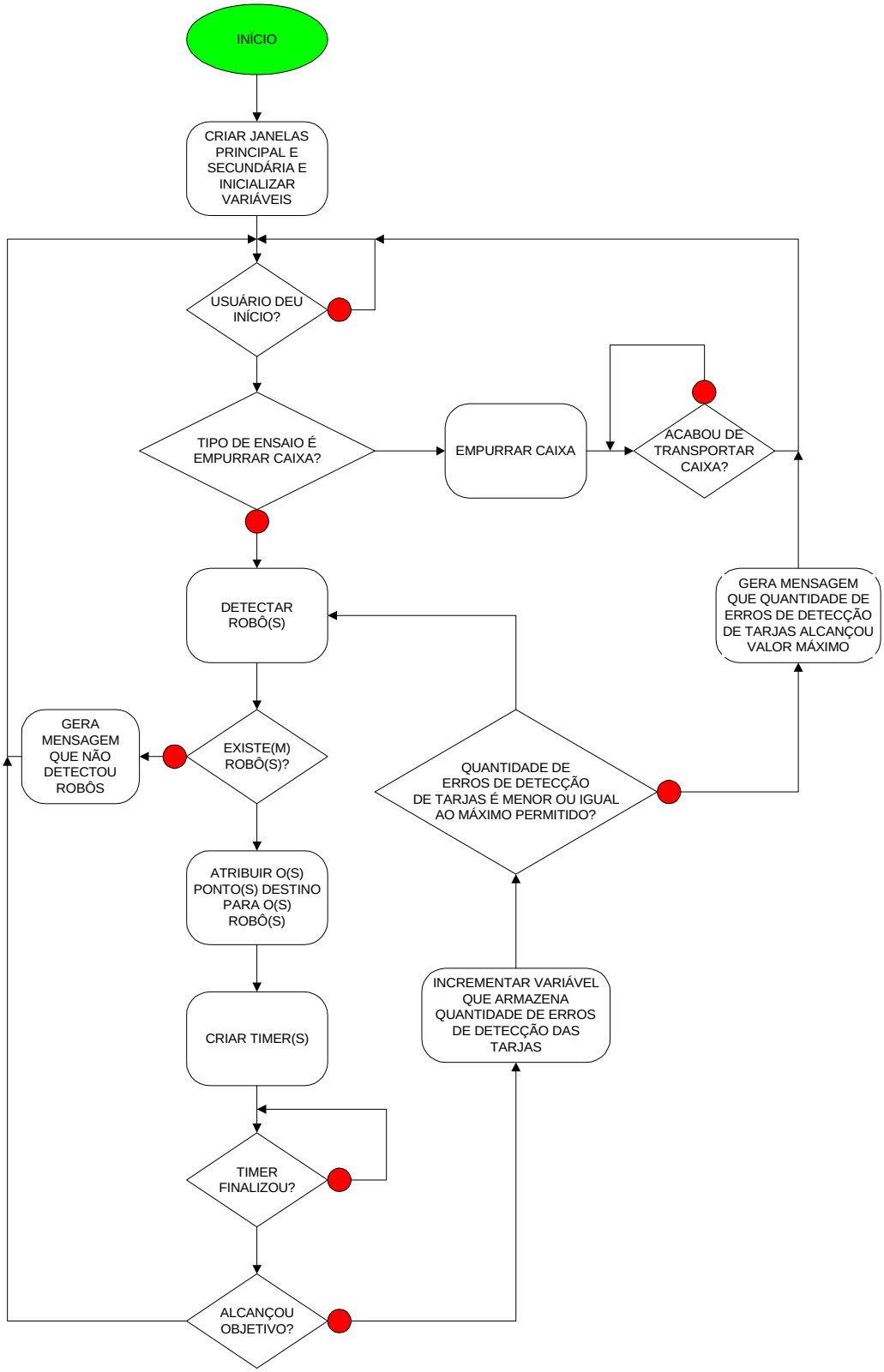


Figura 77 – Fluxograma do programa principal.

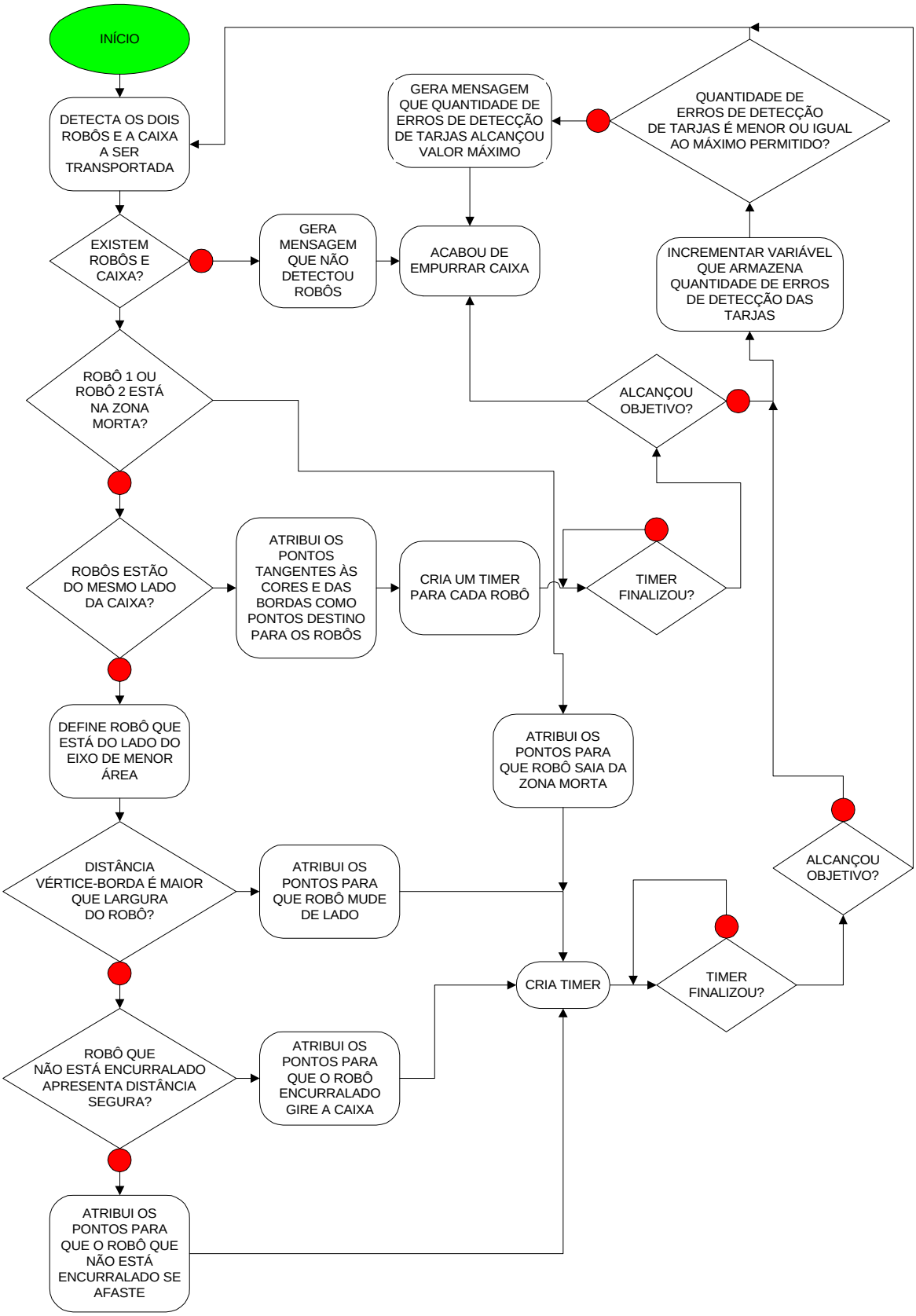


Figura 78 – Fluxograma da rotina responsável por empurrar caixa.

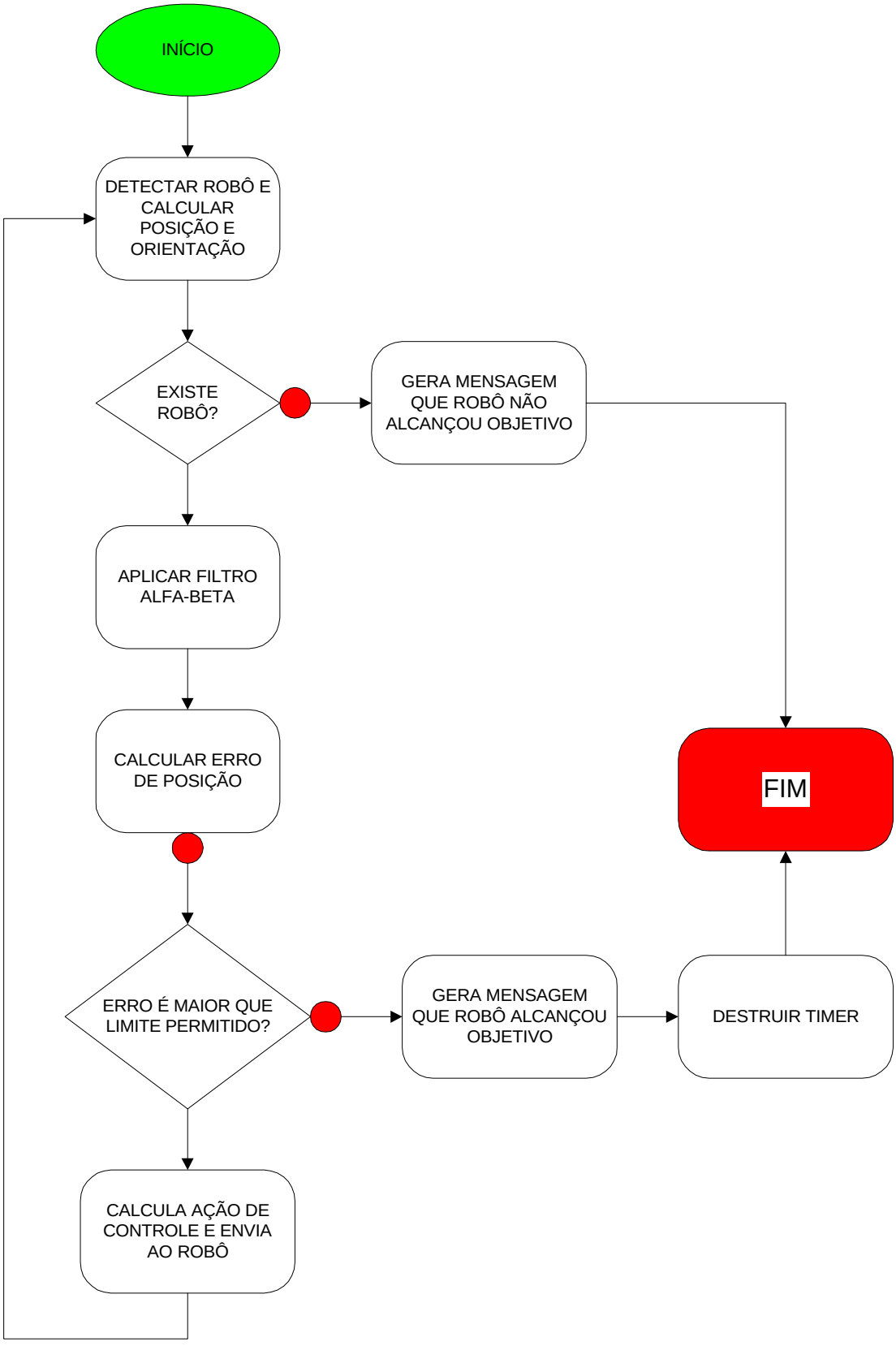


Figura 79 – Fluxograma do timer de cada robô.