Universidade Federal do Espírito Santo
Centro Tecnológico
Programa de Pós-Graduação em Engenharia Elétrica

Navegação de Robôs Móveis ou Veículos Multi-Articulados com Realimentação de Informações Visuais

Mariana Rampinelli Fernandes

Vitória

25 Agosto de 2008

MARIANA RAMPINELLI FERNANDES

NAVEGAÇÃO DE ROBÔS MÓVEIS OU VEÍCULOS MULTI-ARTICULADOS COM REALIMENTAÇÃO DE INFORMAÇÕES VISUAIS

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.
Orientador: Prof. Dr. Edson de Paula Ferreira.

VITÓRIA
2008

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

Fernandes, Mariana Rampinelli, 1983-

F363n Navegação de robôs móveis ou veículos multi-articulados com realimentação de informações visuais / Mariana Rampinelli Fernandes. - 2008.

123 f. : il.

Orientador: Edson de Paula Ferreira.

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

1. Navegação de robôs móveis. 2. Robôs móveis. 3. Visão por computador. 4. Processamento de imagens - Técnicas digitais. 5. Sistemas de veículos auto-guiados. 6. Sistemas difusos. I. Ferreira, Edson de Paula. II. Universidade Federal do Espírito Santo. Centro Tecnológico. III. Título.

CDU: 621.3

Mariana Rampinelli Fernandes

Navegação de Robôs Móveis ou Veículos Multi-Articulados com Realimentação de Informações Visuais

Dissertação submetida ao Programa de Pós-Graduação em Engenharia Elétrica da Universidade Federal do Espírito Santo como requisito parcial para a obtenção do grau de Mestre em Engenharia Elétrica.

Aprovada em 25 de Agosto de 2008.

Comissão Examinadora:

Prof. Dr. Edson de Paula Ferreira
Universidade Federal do Espírito Santo, Orientador

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

Prof. Dr. Hans Rolf Kulitz
Centro Federal de Educação Tecnológica do Espírito Santo

Vitória, Agosto de 2008.

Dedicatória

Aos meus pais e irmão, obrigada pelo apoio e carinho.
A Max, meu companheiro de mais uma batalha,
obrigada pela compreensão, pela paciência, pela
dedicação e, principalmente, pelo amor.

Agradecimentos

O que sou hoje é resultado de tudo que vivi e aprendi com aqueles que se fizeram importantes na minha vida.

Aos meus pais, devo o exemplo de amor, carinho e apoio incondicional. São eles também o meu exemplo de honestidade, fé, confiança e perseverança. Aprendi que mesmo nos momentos difíceis, não devemos perder a esperança nem o sorriso. Amo vocês! Ao meu irmão, devo o companheirismo. Crescemos juntos, aprendemos juntos e nos protegemos.

Com meu melhor amigo, namorado e companheiro, Max, compartilhei grandes momentos nesses mais de 10 anos de convivência. Alguns de dificuldade, onde juntos aprendemos muito... lições para uma vida toda. A enorme maioria foram momentos de muita felicidade e vitórias. Com você, meu amor, compartilhei os momentos mais importantes da minha vida.

Com meus amigos de graduação, Douglas, Junim e Wayner, aprendi um pouco mais sobre amizade... e hoje sobre saudades. Vocês fizeram parte de momentos maravilhosos e sinto saudades imensas da época em que nos víamos todos os dias. E foi com nossos projetos de graduação que este trabalho começou.

Ao meu orientador, Edinho, agradeço pela confiança ao me aceitar como sua aluna de projeto de graduação e depois no mestrado e também pelo apoio durante o projeto, com sugestões inteligentes e fundamentais ao andamento dessa dissertação.

Agradeço também ao Tiago, cuja ajuda na montagem e no andamento desse projeto é incalculável. Obrigada pelo apoio nos momentos difíceis, pelo sua amizade e seu otimismo.

Ao Zazá e ao Cléber, que assim como na graduação, também foram amigões no mestrado!

Ao pessoal do LAI, sempre dispostos a ajudar e a rir. A Flávio e Christiano que estão aqui, como eu, nesta aventura longe de casa. Ao Dani Boy, que ajudou na montagem do caminhão! Ao apoio e às conversas de Adilson e David, que começaram o mestrado comigo e agora, também já se preparam para a defesa.

Agradeço ao CNPq que me concedeu a bolsa de mestrado, ao FACITEC pelo financiamento de nosso projeto e a Edson Silva pelo empréstimo do protótipo utilizado nestre trabalho.

Enfim, a Deus, por estar sempre presente em minha vida.

Mariana Rampinelli Fernandes

"DAS UTOPIAS

Se as coisas são inatingíveis... ora!

Não é motivo para não querê-las...

Que tristes os caminhos se não fora

A mágica presença das estrelas!"

Mario Quintana

Lista de Tabelas

Lista de Figuras

Sumário

Resumo

Este trabalho aborda o uso da visão computacional como realimentação do controlador *fuzzy* por propagação de inferências. O estudo é parte do desenvolvimento de um sistema supervisório para a navegação e/ou controle de manobras de robôs móveis ou veículos multi-articulados. Os robôs em questão têm em sua cadeia mecânica o primeiro elemento como trator, com atuação/controle de ângulo de direção nas rodas dianteiras e de velocidade, nas rodas traseiras. Os outros elementos da cadeia são semi-reboques passivos. Tais veículos podem ser utilizados para o transporte de cargas, por exemplo. O problema de controle desses robôs é particularmente complexo nos movimentos à ré, onde o sistema é não holonômico, comportando-se como um pêndulo invertido múltiplo, não linear, instável e de difícil modelagem. A visão computacional foi escolhida para a realimentação do controlador *fuzzy* devido à quantidade de informações a respeito do ambiente de trabalho e do próprio veículo que as imagens capturadas de uma única câmera são capazes de fornecer. Para a extração das características do robô, a segmentação por cor foi a estratégia escolhida. Os experimentos realizados utilizaram um protótipo de veículo multi-articulado dotado de um sistema de controle e sensoreamento embarcado com comunicação RF, que possibilita a troca de informações com um computador remoto. A câmera, base do sistema de visão, estava situada no topo da área de testes e interligada ao computador remoto. A partir dos resultados dos experimentos, foi possível verificar que o controlador com realimentação visual é capaz de seguir referências. Também foi verificado que o sistema de controle funcionou satisfatoriamente com a realimentação visual para o alinhamento do veículo com qualquer configuração inicial, no centro ou na borda do campo de visão da câmera.

Abstract

This work deals with the use of computer vision as the feedback for the fuzzy control based in inference propagation. This approach is a part of a supervisory system development to automate navigation navigation and/or maneuver's control of mobile robots or multi-articulated vehicles. Those robots have a mechanical chain with the first element as a tractor, with control of direction's angle in the front wheels, and speed control in the back wheels. The other elements of the chain are passive traillers. Such vehicles can be used to transport load, for example. The problem of control these robots is particularly complex in the backward movements, where the system is non-holonomic, behaving as a horizontal multiple pendulum, nonlinear, unstable with complex modelling. The computer vision was chosen to feedback the fuzzy controller due to the amount of informations about the work enviroment and about the vehicles itself that the images of a single camera can provide. For the extraction of the robot characteristics, the strategy chosen was the color segmentation. The experiments used a multi-articulated vehicle prototype which one has a control system and a embedded sensing with radio frequency communication that provides a link with a remote computer. The camera, base of the vision system is placed on the top of the tests' area and has a connection with the remote computer. From the experimental results, was possible to verify that the controller with visual feedback is capable to follow the references. It was also noted that the control system performed satisfactorily with the visual feedback to align the vehicle with any initial configuration, either at the center or at the edge of the camera's field of vision.

1 Introdução

"Alguns homens vêem as coisas como são, e dizem 'Por quê?' Eu sonho com as coisas que nunca foram e digo 'Por que não?".

George Bernard Shaw

Nas últimas décadas, o desafio de desenvolver veículos inteligentes tem sido amplamente pesquisado pela robótica móvel. Assim como um robô, um veículo inteligente pode ser definido como aquele capaz de gerenciar informações obtidas de sensores, e planejar e executar ações de controle de maneira semelhante à realizada por um condutor humano [1–4].

As tarefas de condução podem ser total ou parcialmente automatizadas, visando aumentar a dirigibilidade do veículo, melhorar as condições de segurança, otimizar o uso da infra-estrutura de transportes e reduzir o consumo de energia e a emissão de poluentes para preservar o meio ambiente. Essas tarefas incluem a condução de veículos por rodovias e sua manutenção dentro da faixa correta, caso o motorista durma, por exemplo, a manutenção de uma distância segura entre os veículos, o desvio de obstáculos e o deslocamento e o estacionamento do veículo em áreas urbanas [1–6].

Devido à crescente demanda de transporte de passageiros e de cargas, a quantidade de veículos multi-articulados que circulam nas estradas cresce rapidamente. Não só a quantidade, mas o tamanho, o peso e a potência desses veículos crescem. Como resultado, as estradas revelam o elevado número de acidentes graves causados por veículos pesados [7–9].

No entanto, grande parte da tecnologia desenvolvida até agora para aumentar a segurança dos veículos está voltada para carros de passeio, sendo que os veículos multi-articulados poucas vezes são alvos desse tipo de pesquisa. E mesmo quando esses veículos são o foco principal de pesquisas, pouco foi desenvolvido a respeito de seu controle em

manobras a ré, seja como auxílio ou como substituição do motorista. É importante ressaltar que, assim como para veículos de passeio, as ferramentas da robótica móvel também podem ser utilizadas no desenvolvimento de veículos multi-articulados inteligentes, uma vez que estes possuem cadeia cinemática similar à dos robôs móveis multi-articulados tipo "*tractor-trailers*" [10].

Progressos nessa área poderiam contribuir para a redução de acidentes e de custos e para a melhoria da eficiência no setor industrial e na cadeia logística brasileira e mundial.

1.1 Motivação

O transporte rodoviário é responsável por cerca de 60% de toda carga transportada no Brasil [11]. No intuito de diminuir custos, os transportadores têm buscado aumentar a capacidade dos caminhões convencionais. No entanto, o aumento da carga útil transportada por eixo acarreta o aumento do custo de transporte a longo prazo devido, a vários problemas causados pela acentuada deterioração nos pavimentos. Uma solução para esse caso pode ser obtida com o uso de semi-reboques engatados a caminhões convencionais, diminuindo assim a carga transportada por eixo, o prejuízo à pavimentação e, conseqüentemente, o custo global de transporte.

Os veículos multi-articulados são freqüentemente usados em transporte de cargas a granel, combustível, toras, cana e transportes rodoviários em geral. Tais aplicações demandam manobras em pátios de armazenagem ou pátios de carga e descarga. Nesses casos, tais veículo também são chamados de CVC - **C**omposição de **V**eículos de **C**arga [12]. Veículos com a estrutura articulada também são muito utilizados em áreas portuárias e aeroportuárias e também necessitam efetuar manobras complexas. A Figura 1 mostra dois veículos multi-articulados.

Apesar dos benefícios, o acréscimo de semi-reboques prejudica a manobrabilidade da composição uma vez que não possuem nenhum tipo de controle de direção, principalmente em movimentos à ré. Em muitos casos, quando são necessárias manobras para trás, os semi-reboques são desconectados e manobrados individualmente. Esse tipo de operação consome muito tempo e recursos [7], nem sempre disponíveis em quantidade suficientes. Desse modo, mesmo apresentando vantagens, as possibilidades de uso dos sistemas reais multi-articulados são limitadas devido às dificuldades de manobra descritas.

Nesse contexto, este trabalho busca ampliar as ferramentas de auxílio na execução de manobras, aumentando a gama de possibilidades de uso de robôs móveis ou veículos

(a) (b)

Figura 1: Exemplo de aplicações de veículos multi-articulados: (a) transporte de *containeres* em portos e (b) transporte de cana de açúcar [13].

multi-articulados e, conseqüentemente dos CVCs.

1.2 Manobras Assistidas e Manobras Automáticas

Robôs móveis articulados podem ser utilizados para uma ampla classe de aplicações onde, por exemplo, corredores e dutos impõem restrições físicas ao uso de robôs móveis convencionais de maior porte, necessários para o transporte de alguns tipos de cargas e ferramentas. As aplicações desses robôs em armazéns automáticos seriam particularmente atrativas para otimização de área de corredores. Outra grande área de aplicação para pesquisas em robôs articulados são os CVCs, uma vez que ambos possuem estruturas cinemáticas semelhantes.

Veículos ou robôs móveis multi-articulados são sistemas compostos por um elemento trator, o cavalo mecânico, conectado a elementos passivos, os semi-reboques, através de articulações. O cavalo mecânico é responsável por imprimir direção à composição através das rodas dianteiras e velocidade através das rodas traseiras. Os semi-reboques, nesse caso, não possuem nenhum tipo de tração.

O fato de o controle da composição ser todo realizado pelo cavalo mecânico torna o movimento à ré desse tipo de veículo extremamente complexo, não linear e de difícil modelagem [7, 8]. Nessa situação, o movimento do veículo é não holonômico, comportando-se como um pêndulo invertido múltiplo na horizontal. Durante a manobra, o cavalo mecânico deve evitar que a composição entre em uma configuração tal que seja impossível a continuação do movimento para trás, conhecida como "engavetamento" ou "*jacknife*". "Engavetamento" ou "*jacknife*" é uma configuração do veículo na qual para qualquer va-

lor do ângulo de direção, em um movimento à ré, o ângulo entre os últimos semi-reboques aumenta sempre. Mesmo motoristas experientes não conseguem realizar manobras complexas em veículos multi-articulados. Nesse sentido, o desenvolvimento de um sistema supervisório seria uma importante ferramenta para automatizar manobras e tarefas realizadas por tais robôs ou veículos em áreas restritas.

Pátios de estacionamento seriam propícios à implementação real desse tipo sistema. Nesses ambientes poderiam existir obstáculos e apenas um veículo seria manobrado por vez. A cada novo veículo estacionado, o cenário mudaria, ou seja, as restrições de entorno seriam modificadas. Nesse contexto, a implementação do sistema supervisório compreenderia duas estratégias distintas: as manobras assistidas [8] e as manobras automáticas [7]. A Figura 2 ilustra o funcionamento dessas soluções.

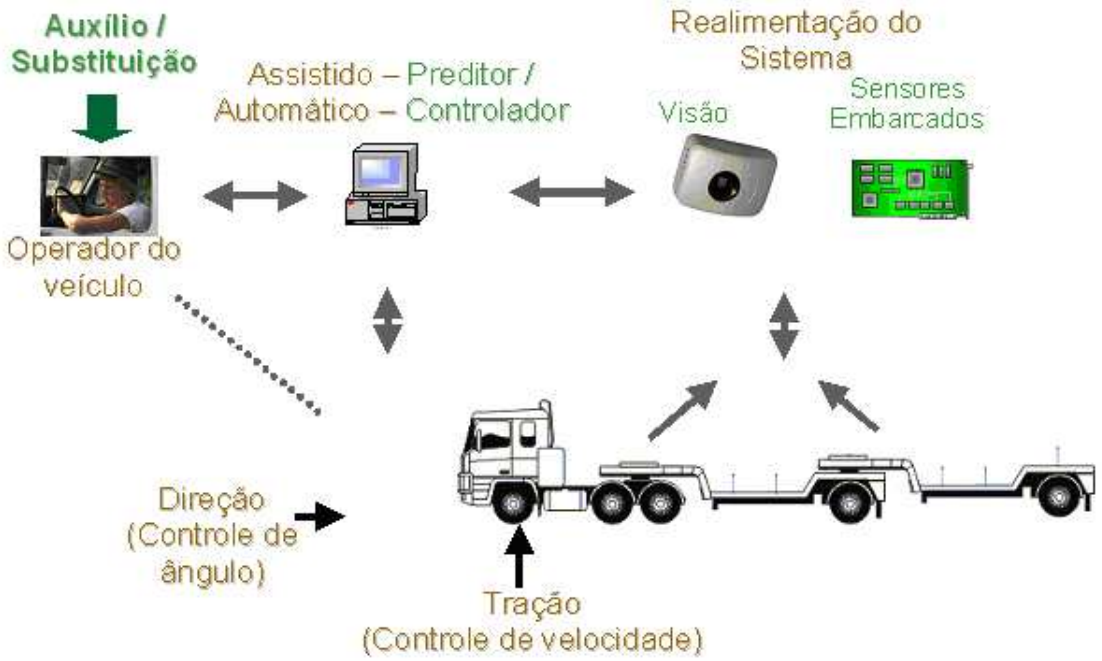


Figura 2: Proposta para implementação de um sistema de auxílio ao motorista em dois modos: assistido ou automático.

As manobras assistidas são manobras realizadas pelo próprio motorista do veículo multi-articulado com o auxílio de um preditor, cujo o desenvolvimento é apresentado em [8] e pode ser sintetizado na Figura 3. O preditor é um dispositivo de auxílio ao motorista constituído por um programa de computador capaz de simular o movimento à ré para um determinado deslocamento. A função deste dispositivo é indicar ao motorista, antes do início do movimento, qual a direção que os semi-reboques tomarão caso a composição se movimente para trás com o ângulo definido pelo volante para as rodas dianteiras e os atuais ângulos entre os semi-reboques e o caminhão. Sendo assim, sua função não é o

controle do movimento para trás, como ocorre na navegação autônoma, mas o auxílio ao motorista, que realiza o controle propriamente dito.

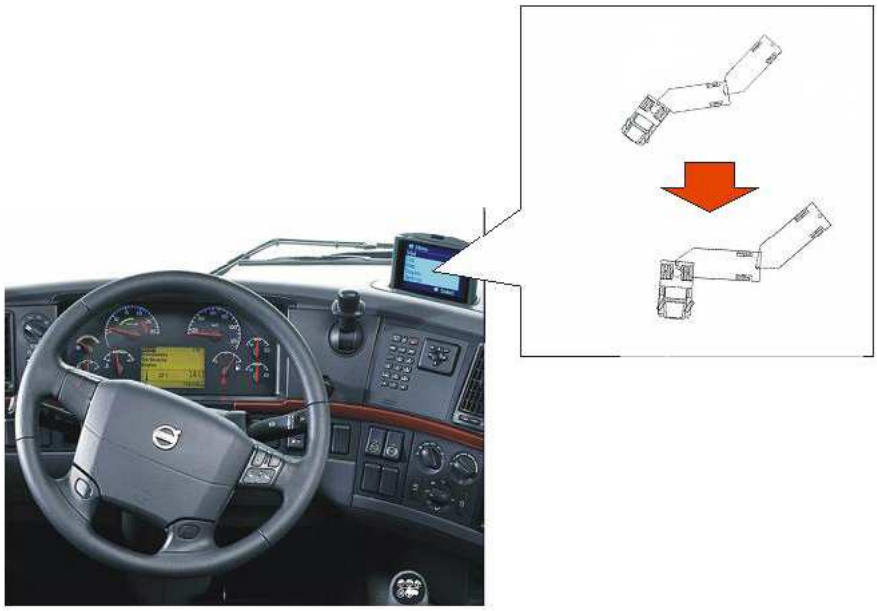


Figura 3: Preditor instalado na cabina de um caminhão.

No modo automático, as manobras à ré são realizadas por um servo sistema acoplado ao volante cuja função é controlar o ângulo de direção das rodas dianteiras de modo que o último semi-reboque siga um caminho de referência. Essa referência pode ser definida diretamente pelo motorista ou um operador externo a partir de um *joystick* e uma tela de visualização do estado do pátio durante a manobra. No entanto, com a implementação do sistema supervisório, a referência é passada ao controlador automaticamente após o planejamento de uma rota livre de obstáculos para o veículo em questão. Um modelo de controlador para esse tipo de manobra é apresentado em [7].

Existem na literatura algumas propostas de controle do movimento à ré de veículos multi-articulados com resultados experimentais utilizando diferentes técnicas de controle. No entanto, em geral são utilizados robôs móveis construídos em laboratório [7, 14–19]. Apenas um número reduzido utiliza um protótipo de veículo real [20], como o utilizado neste trabalho, para realizar experimentos.

Dos trabalhos desenvolvidos sobre controle de veículos ou robôs móveis multi-articulados, poucos trabalham com técnicas clássicas de controle, baseada no modelo cinemático desses veículos [17–20]. Devido à sua complexidade, a maioria das soluções desenvolvidas para o problema de manobras automáticas ou assistidas utilizam técnicas de controle inteligente, baseadas em redes neurais [7, 14, 21, 22] ou lógica *fuzzy* [7, 8, 23–26].

Dando continuidade aos trabalhos desenvolvidos neste laboratório, a estratégia de controle utilizada nesta dissertação foi baseada em lógica *fuzzy* [7]. O controlador implementado será mais bem detalhado no Capítulo 2.

Utilizando a solução de controle clássica ou de controle inteligente, a cada momento é necessário que o controlador conheça a configuração da composição, composta pelos ângulos em suas articulações. Essas informações podem ser obtidas por sensores embarcados, tais como potenciômetros ou codificadores ópticos instalados nos engates do veículo, para informar o ângulo, ou hodômetros, para fornecer a distância percorrida do veículo ao longo de sua manobra [7, 14, 18, 20, 22]. No entanto, a instalação desses tipos de sensores em veículos de dimensões reais seria muito custosa. Além disso, a cada novo veículo a ser controlado, um novo conjunto de sensores deveria ser instalado.

Em ambientes restritos, uma alternativa ao uso de sensores embarcados é o uso de um sistema de visão computacional, uma vez que câmeras externas podem ser instaladas de modo a fornecer imagens com uma grande quantidade de informações a respeito do ambiente de trabalho e do próprio veículo. Portanto, além de medir os ângulos de configuração da composição, o sistema de visão também pode ser utilizado para obter informações mais aprimoradas sobre o entorno do robô, tais como a caracterização de obstáculos e dos aspectos dinâmicos do ambiente. Tais dados são necessários para a implementação do sistema supervisório descrito anteriormente.

1.3 Sistema de Visão Computacional

Ao se trabalhar com visão computacional, é utilizada uma série de novos termos interligados. Apesar de cada um possuir sua própria definição, muita confusão ainda é feita pela afinidade de conceitos. Não existe na literatura um consenso geral sobre a definição precisa dessa terminologia associada. Desse modo, em [27] foram propostas as seguintes definições baseadas em [28–30]:

Processamento de Imagens: refere-se ao processamento digital de imagens através de um computador ou dispositivo eletrônico, onde as entradas e saídas do processamento são imagens. Em geral são operações de pré-processamento, tais como realce, restauração, transformação ou compreensão de imagens. Esse tipo de operação é definido como de nível "baixo" ou até mesmo "intermediário".

Análise de Imagens: processamento no qual somente as entradas são imagens. Em ge-

ral, as saídas apresentam uma descrição ou representação diferente da imagem ou alguma característica sua. A análise de imagens é uma operação de nível "intermediário", podendo chegar a "alto", e está entre o processamento de imagens e a visão computacional.

Visão Computacional: seu objetivo final é a visão do robô, ou seja, utilizar computadores para emular a visão humana, com possibilidades de fazer inferências e tomar decisões baseadas em entradas visuais. Abrange a extração de características importantes que auxiliem na compreensão de imagem ou tomada de decisão inteligente. É uma operação de "alto" nível.

Sistema de Visão Computacional: é um sistema completo, que utiliza imagens capturadas por algum dispositivos eletrônico, tais como *webcams* e câmeras de vídeo, para extrair automaticamente informações úteis contidas nas imagens. Essas informações podem ser utilizadas, por exemplo, na classificação de objetos ou no controle automático de algum dispositivo, como um braço robótico ou uma rede de esteiras rolantes em uma linha de montagem. Nesses casos, as informações desejadas são extraídas em modo *on-line*. É composto tanto por dispositivos de *hardware* como por módulos de *software*, fazendo uso de uma ou até todas as operações previamente citadas. Em geral, etapas como aquisição de imagens, pré-processamento das imagens, extração das características, tomada de decisão inteligente e atuação/realimentação estão contempladas num sistema deste tipo.

A Figura 4 mostra a relação existente entre os termos descritos anteriormente.

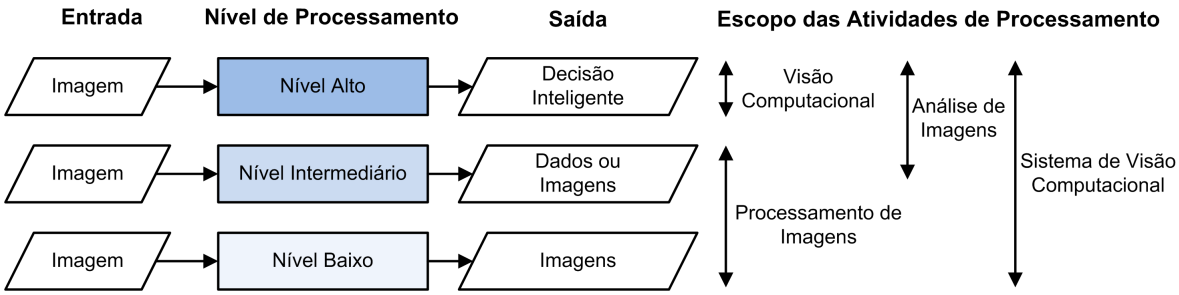


Figura 4: Interdependência entre os termos relacionados à área de visão computacional [27].

1.4 Sistema Implementado x Sistema Real

O sistema de visão computacional necessário para uma implementação capaz de resolver o problema real de manobras de veículos multi-articulados, como o pátio de esta-

cionamento descrito anteriormente, seria constituído por uma ou mais câmeras laterais instaladas em postes ou torres de comando, acima do plano operacional desses veículos. Essa configuração pode ser necessária, pois o campo de visão de uma única câmera é restrito, gerando a necessidade de obter imagens mais completas a partir do mosaico das imagens capturadas por vários sensores. Desse modo, uma estimativa de ângulos entre os elementos da cadeia necessitaria da junção das informações das câmeras e da transformação e correção da perspectiva das imagens.

Além disso, por se tratar de um ambiente externo, deve ser realizado um estudo mais detalhado sobre a influência de mudanças na iluminação do ambiente de trabalho em relação ao desempenho do sistema de visão computacional e de possíveis soluções para minimizá-las. Esse estudo é necessário uma vez que o laboratório é um ambiente influenciado basicamente por iluminação artificial, enquanto que em um ambiente externo existe uma variação constante de luminosidade devido à hora do dia ou às mudanças das condições climáticas.

Neste trabalho, o sistema de visão inicial é constituído por uma única câmera em vista de topo, ou seja, perpendicular ao plano de trabalho do robô, e um computador remoto cuja função é processar e extrair as informações desejadas das imagens capturadas. O estudo desse caso mais simples é importante por ser um passo intermediário para adquirir o domínio das técnicas de processamento de imagem. Além disso, as transformações de perspectivas, necessárias para o caso de câmera lateral do problema geral, conduzem à visão de topo.

Em uma abordagem mais geral do problema e para a própria implementação real do sistema, a estratégia de segmentação deveria extrair diretamente os parâmetros geométricos da composição. No entanto, neste trabalho inicial, foi utilizado um algoritmo mais simples, baseado em segmentação por cor.

Uma visão geral do ambiente montado para a implementação do sistema de visão computacional, pode ser visto na Figura 5.

1.5 Objetivo

O objetivo desta Dissertação de Mestrado é implementar um suporte experimental baseado em visão para o desenvolvimento de estratégias de navegação e controle de veículos ou robôs multi-articulados, principalmente em movimentos à ré. Esses sistemas foram tema de duas dissertações de mestrado [8, 14] e uma tese de doutorado [7], desenvolvidas

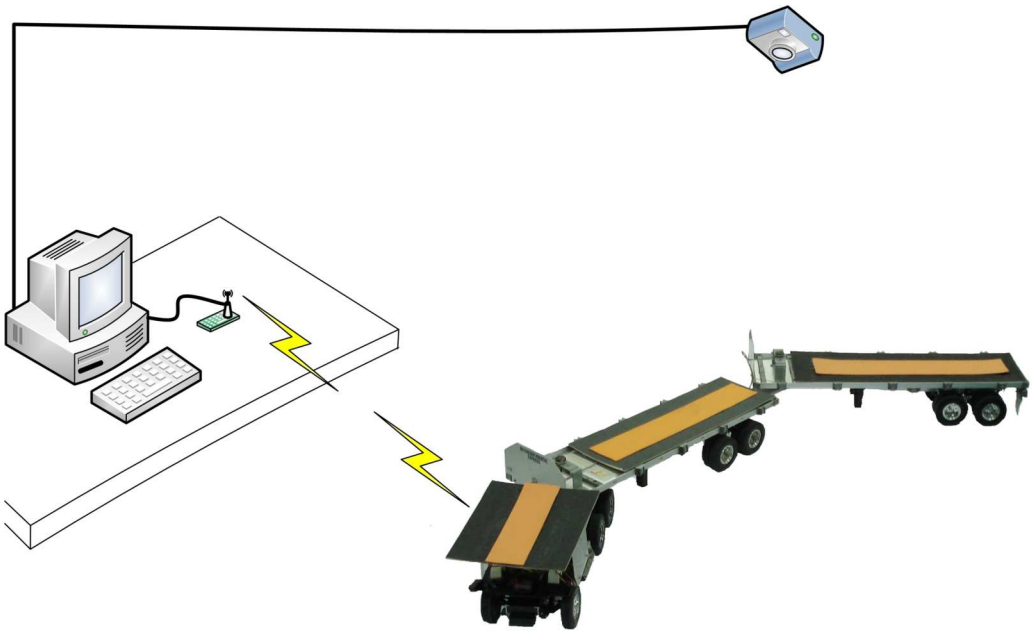


Figura 5: Visão geral do ambiente de experimentos.

neste laboratório.

Silva e Pinheiro [8, 14] implementam modelos diretos para robôs ou veículos multi-articulados. Em [14], Silva propõe uma metodologia não linear, baseada em redes neurais, para o controle de trajetória desses veículos. Pinheiro [8] propõe uma metodologia para o auxílio a manobras de robôs ou veículos multi-articulados, sob a forma de um preditor *fuzzy*. Para isso, obtém um modelo *fuzzy* de uma composição qualquer, utilizando dados do simulador implementado em [7, 14].

Em sua tese de doutorado, Kulitz [7] implementa e descreve o simulador MGSARA e propõe uma técnica geral de controle para veículos multi-articulados chamada de Controle por Propagação de Inferências (CPpI), também baseada em lógica *fuzzy*. No entanto, foram feitos poucos testes práticos para sua validação.

Nesse contexto, esta dissertação também tem por objetivo validar os trabalhos de [7], implementando a técnica de controle proposta, utilizando para isso um protótipo de veículo multi-articulado e a visão computacional como realimentação do controlador.

1.6 Estrutura da Dissertação

De modo a expor toda a proposta e os trabalhos realizados, esta dissertação foi dividida em 5 capítulos. A seguir, cada capítulo é brevemente descrito.

Capítulo 1: Introdução

É o capítulo inicial, cujo objetivo é contextualizar o leitor ao problema exposto, fornecendo uma explanação a respeito das pesquisas consideradas mais relevantes sobre controle de manobras à ré de robôs móveis ou veículos multi-articulados. Além disso, este capítulo também apresenta a motivação e o objetivo do trabalho realizado.

Capítulo 2: Controle *Fuzzy*

Introduz os conceitos envolvidos no processo de modelagem *fuzzy*. Faz uma comparação entre a lógica *crisp* e a *fuzzy* e, em seguida, apresenta os principais aspectos da teoria. Aborda também o funcionamento de um sistema de inferência e de um controlador *fuzzy*. Este capítulo é finalizado com a descrição detalhada do controlador proposto em [7] e implementado neste trabalho.

Capítulo 3: Processamento de Imagens Digitais

Esse capítulo apresenta as técnicas de processamento de imagens que foram empregadas no sistema de visão computacional, explicando e discutindo os algoritmos desenvolvidos. Também aborda técnicas de filtragem utilizadas a fim de refinar as estimativas realizadas pelo sistema. Para finalizar, apresenta experiências realizadas para comprovar a viabilidade do uso de visão computacional.

Capítulo 4: Resultados Experimentais

Neste capítulo são apresentados os equipamentos utilizados para a realização dos experimentos e os resultados obtidos.

Capítulo 5: Conclusões e Trabalhos Futuros

Encerra a dissertação com uma discussão geral do trabalho realizado, as principais contribuições, considerações finais e sugestões de trabalhos futuros.

Apêndice A: Sistema de Visão Computacional e Controle *Fuzzy*

Nesse capítulo está todo o código implementado durante o desenvolvimento deste trabalho.

2 Controle Fuzzy

"Na medida em que as proposições da matemática se referem à realidade, elas não são certas; na medida em que são certas, elas não se referem à realidade."

Albert Einstein – 1921

2.1 Introdução

Em geral, problemas de engenharia são formulados a partir do conhecimento objetivo humano, utilizando ferramentas como a modelagem matemática para representar o sistema em estudo com a maior fidelidade possível. Um modelo é tanto mais fiel à realidade quanto maior for o número de informações disponíveis sobre o problema. A principal vantagem do uso dessas ferramentas matemáticas consiste na precisão alcançada pelo modelo. No entanto, em muitas situações, processos dinâmicos tornam-se demasiadamente complexos, dificultando a obtenção de informações e identificação de parâmetros que permitam caracterizar precisamente o sistema. Outro aspecto a ser considerado é que um modelo muito complexo pode levar a formulações instáveis. Muitas vezes, a precisão não é sequer um requisito fundamental para a solução desejada. Essa discussão é traduzida pelo princípio da incompatibilidade apresentado em [31]:

"*Conforme a complexidade de um sistema aumenta, nossa habilidade de fazer declarações precisas e significativas sobre o seu comportamento diminui, até alcançar um limite além do qual precisão e relevância tornam-se características mutuamente exclusivas.*"

No mundo real, incluindo grande parte das aplicações de interesse na área de engenharia, os eventos apresentam propriedades que são definidas de forma vaga e imprecisa. Essas características permitem, por exemplo, um operador humano aplicar seu raciocínio

aproximado no intuito de intervir em um sistema e ajustar seus parâmetros de funcionamento. No entanto, as teorias clássicas não são capazes de tratar o aspecto subjetivo dessas informações.

Encorajado pelas limitações das teorias clássicas e com objetivo de fornecer um ferramental que contemplasse os aspectos imprecisos do raciocínio lógico, em 1965, Lotfi Zadeh, professor do Departamento Engenharia Elétrica e Ciências da Computação da Universidade da Califórnia, em Berkeley, publicou seu trabalho sobre a teoria dos conjuntos *fuzzy* ou nebulosos [32]. Essa teoria proporcionou um meio de combinar efetivamente tanto o conhecimento humano objetivo quanto o subjetivo, tornando-se uma ferramenta com capacidades suficiente para lidar com problemas não-lineares e complexos, com formulação imprecisa, vaga e até mesmo ambígua. A lógica *fuzzy*, devido à sua baixa especificidade e por permitir a inclusão de regras sem necessidade de hierarquia, produz soluções mais flexíveis e com maior generalidade, se comparadas às soluções analíticas equivalentes [33]. Desse modo, com essa nova abordagem, a modelagem do problema em um determinado domínio pode passar pela representação do conhecimento acumulado por um especialista nesse domínio.

A modelagem e o controle *fuzzy* podem ser utilizados para manusear informações subjetivas de uma maneira mais rigorosa. Essas técnicas podem manipular o conhecimento humano de maneira conveniente, considerando o modo como a incerteza é descrita. Baseado no relacionamento entre as entradas e as saídas do sistema em estudo, a modelagem e o controle *fuzzy* possibilitam a representação de processos complexos e o desenvolvimento de controladores simples, de fácil manutenção e baixo custo. Desse modo, os sistemas de controle resultantes são capazes de fornecer um resultado mais satisfatório e um desempenho mais robusto. Sendo assim, problemas que anteriormente eram intratáveis podem ter soluções baseadas na teoria *fuzzy* [33, 34].

2.2 Fundamentos dos Conjuntos *Fuzzy*

A teoria dos conjuntos clássicos, também chamados de *crisp*, define uma relação de pertinência dicotômica entre elementos de um universo de discurso U, ou seja, para um dado conjunto A, cada elemento $x \in U$ deve assumir apenas um de dois estados possíveis: $x \in A$ ou $x \notin A$. A teoria dos conjuntos *fuzzy*, por sua vez, foi concebida para permitir uma transição gradual entre a noção de pertencer completamente ou não pertencer completamente a um determinado conjunto. Para isso, foi definida a noção de graus de

pertinência, denotada por $\mu(x)$.

Desse modo, em uma definição mais formal, um conjunto *fuzzy* $\tilde{A}$ no universo de discurso U pode ser visto como um mapeamento dos elementos de U em um espaço de pertinência M através da função de pertinência $\mu_{\tilde{A}}(x)$, ou seja, $\mu_{\tilde{A}} : U \rightarrow M$. Nesse sentido, $\tilde{A}$ é definido como um conjunto de pares ordenados:

$$\tilde{A} = \left\{ \left(x, \mu_{\widetilde{A}(x)} | x \in U \right) \right\}. \tag{2.1}$$

Em geral, o espaço de pertinência M é representado pelo intervalo unitário $[0, 1]$. A função de pertinência $\mu_{\tilde{A}}(x)$ indica com que grau x pertence ao conjunto representado por $\tilde{A}$:

- $\mu_{\tilde{A}(x)} = 1$ indica que x pertence completamente ao conjunto $\tilde{A}$;
- $\mu_{\tilde{A}(x)} = 0$ indica que x não pertence ao conjunto $\tilde{A}$;
- $0 < \mu_{\tilde{A}(x)} < 1$ indica que x pertence parcialmente ao conjunto $\tilde{A}$. Essa parcialidade é medida com grau $\mu_{\tilde{A}(x)}$.

De acordo com essa definição, a teoria de conjuntos *fuzzy* pode ser vista como uma generalização da teoria *crisp*. Um conjunto A definido pela teoria clássica ou *crisp* pode ser entendido como um caso particular da teoria *fuzzy*. Enquanto nesta última o espaço de pertinência é um intervalo fechado entre 0 e 1, nos conjuntos *crisp* o espaço de pertinência é bivalente: "sim" ou "não", "0" ou "1".

2.2.1 Operações em Conjuntos *Fuzzy*

De modo semelhante aos conjuntos *crisp*, uma operação na teoria *fuzzy* representa uma relação entre dois ou mais conjuntos com objetivo de gerar um novo conjunto. A união e a intersecção são duas das operações básicas sobre conjuntos *fuzzy*, também chamadas de operações básicas de Zadeh. Essas operações são definidas em termos de funções de pertinência $\mu(x)$ e ilustradas na Figura 6.

1. **União:** A união entre os conjuntos *fuzzy* A e B, denotada como $A \cup B$, é definida pela Equação 2.2:

$$\mu_{A \cup B}(x) = max[\mu_A(x), \mu_B(x)] \equiv \mu_A(x) \vee \mu_B(x) \qquad \forall x \in U. \tag{2.2}$$

2. Intersecção: A intersecção de conjuntos *fuzzy* A e B, denotada como $A \cap B$, é definida pela Equação 2.3:

$$\mu_{A \cap B}(x) = min[\mu_A(x), \mu_B(x)] \equiv \mu_A(x) \wedge \mu_B(x) \qquad \forall x \in U. \tag{2.3}$$

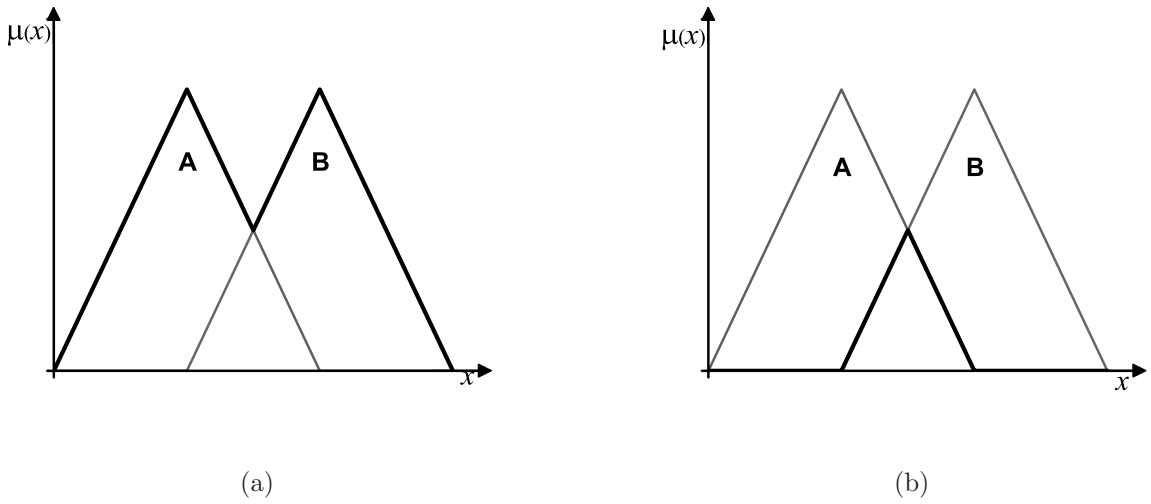


Figura 6: Operação de (a) união e (b) intersecção entre os conjuntos *fuzzy* A e B.

Além das operações básicas de Zadeh, a teoria *fuzzy* apresenta uma série de operações paramétricas e não paramétricas que podem ser adotadas de acordo com as exigências do problema a ser modelado. Essas operações podem ser vistas como generalizações das operações básicas do conjunto *crisp* e proporcionam à teoria *fuzzy* um ferramental para tratar uma grande diversidade de problemas. O principal conjunto de operações são as generalizações da união e da intersecção chamadas de normas triangulares ou t-normas e as conormas triangulares ou t-conormas, respectivamente, que incluem os operadores "*max*" e "*min*" de Zadeh [35].

2.2.2 Variáveis Lingüísticas

Variável lingüística é um importante conceito da lógica *fuzzy* utilizado pela primeira vez por Zadeh [36] para fornecer um meio de aproximar a caracterização de fenômenos que são demasiadamente complexos ou mal definidos para ser satisfatoriamente descrito em termos quantitativos convencionais. Basicamente, uma variável lingüística é uma variável cujos valores são palavras ou expressões em linguagem natural ou artificial. Deste modo, uma variável lingüística pode assumir um valor lingüístico dentre vários outros em um

conjunto de termos lingüísticos.

Uma variável lingüística é formalmente caracterizada pela quíntupla $(x, T(x), U, G, M)$, na qual x é o nome da variável; $T(x)$ é o conjunto de termos de x, ou seja, o conjunto dos nomes de valores lingüísticos que x pode assumir, representados por conjuntos *fuzzy*; U o universo de discurso da variável; G é a gramática para gerar os termos de $T(x)$; e M é uma regra semântica para associar cada valor de x ao seu significado, ou seja, a função de pertinência de cada elemento de $T(x)$. A Figura 7 ilustra uma variável lingüística para a qual $x = \ \hat{a}ngulo$, medida em graus (°), $T(x) = \{Negativo, Zero, Positivo\}$, $U = [-20^o, +20^o]$. Nesse caso, a regra sintática G para geração dos termos do conjunto $T(x)$ é intuitivo e as regras semânticas M podem ser definidas como:

- M(Negativo) = o conjunto *fuzzy* que representa "ângulos aproximadamente menores que -5°", cuja função de pertinência é $\mu_{Negativo}$.
- M(Zero) = o conjunto *fuzzy* que representa "ângulos próximos a 0°", cuja função de pertinência é μ_{Zero}.
- M(Positivo) = o conjunto *fuzzy* que representa "ângulos aproximadamente maiores que $+5^o$", cuja função de pertinência é $\mu_{Positivo}$.

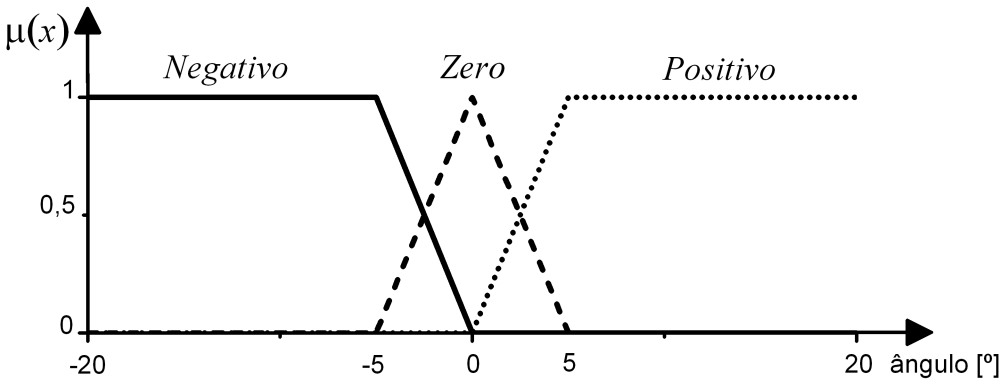


Figura 7: Variável lingüística ângulo.

2.3 Controladores *Fuzzy*

As técnicas de controle *fuzzy* foram desenvolvidas a partir dos trabalhos de Mamdani e Assilian [37], sendo uma das mais ativas e produtivas áreas de pesquisas em teoria de conjuntos e lógica *fuzzy*. A idéia básica dos sistemas de controle *fuzzy* é adicionar

a "experiência de um especialista" humano no projeto do controlador. Para isso, as relações entre as entradas e as saídas do processo são descritas como um conjunto de regras envolvendo variáveis lingüísticas em vez de um modelo dinâmico complexo. Tais regras são do tipo "SE <*premissa*> ENTÃO <*conclusão*>", que definem ações de controle em função das diversas faixas de valores que as variáveis do problema podem assumir [35].

O propósito de um controlador *fuzzy* é calcular o valor das variáveis de controle a partir da medição das variáveis de interesse, que formam, respectivamente, o espaço de entrada e o espaço de saída do sistema. Utilizando a mesma notação adotada para definição de variáveis lingüísticas, o vetor de entrada X, composto pelas variáveis lingüísticas de entrada ou de interesse, e o vetor de saída Y, composto pelas variáveis lingüísticas de saída ou de controle, podem ser definidos respectivamente como:

$$X = \left\{(x_i, U_i, T^1_{x_i}, T^2_{x_i}, ..., T^{k_i}_{x_i}, \mu^1_{x_i}, \mu^2_{x_i}, ..., \mu^{k_i}_{x_i})|_{i=1,...,n}\right\} \tag{2.4}$$

$$Y = \left\{(y_i, V_i, T^1_{y_i}, T^2_{y_i}, ..., T^{k_i}_{y_i}, \mu^1_{y_i}, \mu^2_{y_i}, ..., \mu^{k_i}_{y_i})|_{i=1,...,m}\right\} \tag{2.5}$$

onde as variáveis lingüísticas de entrada x_i formam o espaço de entrada *fuzzy* $U = U_1 \times U_2 \times ... \times U_n$ e as variáveis lingüísticas de saída y_i formam o espaço de saída $V = V_1 \times V_2 \times ... \times V_m$. O tamanho ou a cardinalidade do conjunto de termos, representado por $|T(x_i)| = k_i$, é chamada de "partição *fuzzy*" de x_i. A partição *fuzzy* determina a granularidade do controlador projetado: quanto maior o número de termos das variáveis lingüísticas do espaço de entrada, mais refinada é a inferência realizada pelo controlador, no entanto, maior é a complexidade do sistema.

A literatura apresenta diversos modelos de inferência *fuzzy* que determinam o modo pelo qual é extraído e processado o conhecimento armazenado na forma de declarações condicionais *fuzzy*. Basicamente, os modelos de sistema *fuzzy* podem ser classificados em dois tipos: os de interpolação e os clássicos. Os de interpolação apresentam uma conclusão através de uma função estritamente monotônica, usualmente diferente para cada regra. Os modelos de interpolação mais comuns são o de Takagi-Sugeno e o de Tsukamoto. Nos modelos clássicos, por sua vez, o resultado final da inferência gera um conjunto nebuloso. Nesse caso, é necessário que exista um módulo defuzzificador para geração do resultado final. Os principais modelos clássicos são os de Mamdani e de Larsen [38]. Os modelos diferem quanto à forma de representação dos termos na premissa, quanto à representação das ações de controle e quanto aos operadores utilizados para

implementação do controlador [33]. Neste trabalho, o modelo adotado foi o Mandani, cuja estrutura é composta por quatro módulos principais: o fuzzificador, a base de regras, a máquina de inferências e o defuzzificador, como mostra a Figura 8.

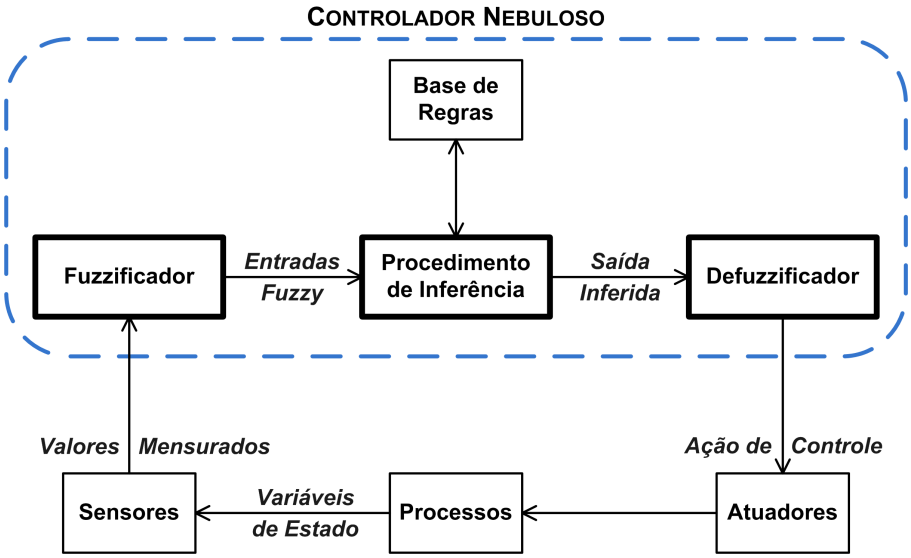


Figura 8: Estrutura de um controlador *fuzzy*.

2.3.1 Fuzzificador

O fuzzificador é o módulo que realiza a interface entre os valores das variáveis de interesse e controlador *fuzzy*. Para cada variável de entrada ou interesse, o fuzzificador mapeia seu valor *crisp* em funções de pertinência dos conjuntos *fuzzy* que compõem a variável lingüística correspondente. A Figura 9 ilustra o modo de funcionamento do fuzzificador, considerando a variável lingüística da Figura 7 e uma entrada *crisp* x^*.

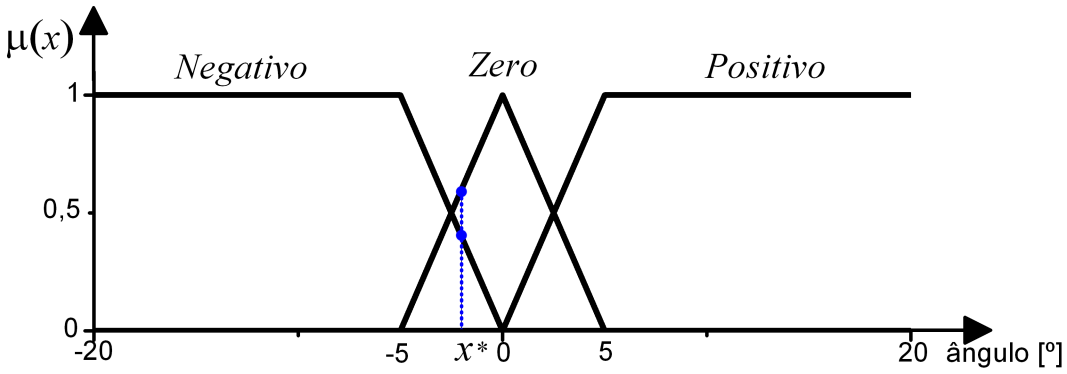


Figura 9: Processo de fuzzificação da variável x^*.

Na Figura 9, o valor da entrada *crisp* é $x^* = -2^o$; seu valor fuzzificado é o grau com que o valor $x^* \in U$ satisfaz cada um dos termos lingüísticos da variável *ângulo*: $\mu_{Negativo}(x^* = -2^o) = 0,4$; $\mu_{Zero}(x^* = -2^o) = 0,4$ e $\mu_{Positivo}(x^* = -2^o) = 0$.

2.3.2 Base de Regras

Na base de regras é possível relacionar o espaço de entrada (que caracteriza o estado atual do sistema) com o espaço de saída, de acordo com o conhecimento de um especialista humano. Esse módulo é constituído por um conjunto regras com estrutura do tipo

$$\textbf{Se} < antecedente > \textbf{então} < conseqüente >, \tag{2.6}$$

cujos antecedente e conseqüente envolvem variáveis lingüísticas. O antecedente de cada regra é formado por um conjunto de premissas que, quando satisfeitas, determinam o processamento do conseqüente da regra a partir de um mecanismo de inferência *fuzzy*, ou seja, dispara a regra. Por sua vez, o conseqüente é composto por um conjunto de ações que são geradas com o disparo da regra. Os conseqüentes das regras disparadas são processados em conjunto e geram conjuntos *fuzzy* para cada uma das variáveis de saída do sistema [39]. As partes que compõem o antecedente e o conseqüente são relacionados pelo conectivo lógico "e", como mostrado na Equação 2.7:

$$\mathbf{R^i} : \textbf{Se } x \text{ é } A_j, ..., \textbf{ E } y \text{ é } B_k \textbf{ então } z \text{ é } C_m, \tag{2.7}$$

onde $x, ..., y$ e z são variáveis lingüísticas representando as variáveis de estado do processo e a variável de controle, respectivamente, e $A_j, ..., B_k$ e C_m são conjuntos *fuzzy* que constituem as variáveis lingüísticas $x, ..., y$ e z no universo de discurso $U, ..., V$ e W, respectivamente.

Em um sistema de controle *fuzzy*, devem existir tantas regras quantas forem necessárias para realizar todas as combinações dos termos da variáveis lingüísticas do espaço de entrada, ou seja, deve ser garantido que, para qualquer entrada, ao menos uma regra seja disparada. Portanto, quanto maior a granularidade de cada variável lingüística do espaço de entrada, maior o número de entradas na base de regras e, portanto, maior a complexidade computacional na inferência das respostas. Desse modo, o número de regras n_r de um controlador *fuzzy* pode ser determinado pela Equação 2.8:

$$n_r = |T(x_1)| \times |T(x_2)| \times ... \times |T(x_n)|, \tag{2.8}$$

onde $x_1, x_2, ..., x_n$ são as variáveis lingüísticas de entrada.

2.3.3 Máquina de Inferência

Esse módulo simula o processo de tomada de decisão de um especialista humano. Para isso, a máquina de inferência recebe os valores de entrada *fuzzy* provenientes do módulo de "fuzzificação", avalia em paralelo as regras existentes na "base de regras" e gera um conjunto *fuzzy* de saída a partir da composição de todas as regras disparadas. A correlação entre o antecedente e o conseqüente de uma regra é feita por meio de uma operação de inferência entre os conjuntos *fuzzy*, que pode ser representada pelo silogismo condicional *modus ponens* generalizado:

$$\begin{array}{rl} Premissa\ 1: & \textbf{Se}\ x\ é\ A,\ \textbf{então}\ y\ é\ B \\ Premissa\ 2: & x\ é\ A' \\ \hline Conclusão: & y\ é\ B' \end{array} \tag{2.9}$$

No *modus ponens* generalizado, o conjunto *fuzzy* A' não é necessariamente o mesmo que A (antecedente da regra), assim como o conjunto *fuzzy* B' não é necessariamente o mesmo que o conseqüente B. Diferente da lógica *crisp*, na lógica *fuzzy*, uma regra é disparada se houver um *grau de similaridade* diferente de zero entre a Premissa 2 e o antecedente da regra. Desse modo, o resultado será um conseqüente com grau de similaridade não nulo em relação ao conseqüente da regra.

O método de implicação utilizado neste trabalho é o de Mamdani, que agrega as regras por meio do operador lógico "ou", modelado pelo operador máximo e, em cada regra, os operadores lógicos "e" e "então" são modelados pelo operador mínimo. Essa estrutura é conhecida como "max-min". A operação que representa todo o processo de inferência de Mamdami pode ser definido pela Equação 2.10:

$$\mu_{B'}(y) = \max_{\forall x \in X} \min[\mu_{A'}(x), \min[\mu_A(x), \mu_B(y)]] \tag{2.10}$$

sendo a regra de implicação definida pela Equação 2.11:

$$A \rightarrow B = min[\mu_A(u), \mu_B(v)] \equiv \mu_A(u) \wedge \mu_B(v), \tag{2.11}$$

A Figura 10 ilustra um exemplo da aplicação do método de inferência de Mamdani.

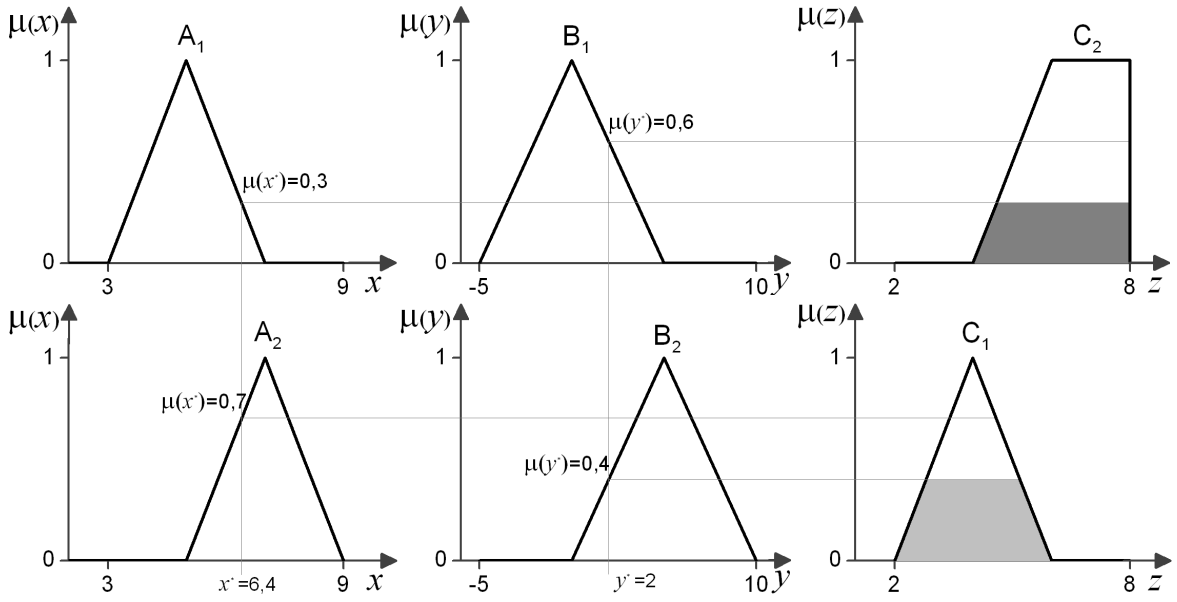


Figura 10: Esquema do processo de inferência.

onde cada linha representa uma regra de inferência:
Regra 1: **Se** x *é* A_1*e* y *é* B_1 **então** z *é* C_2
Regra 2: **Se** x *é* A_2*e* y *é* B_2 **então** z *é* C_1.

Os conjuntos *fuzzy* mostrados na terceira coluna foram obtidos pela Equação 2.10, que pode ser reescrita para o caso específico do seguinte modo:

$$\begin{aligned}(A_i \wedge B_j) \rightarrow C_k &= \min\,[\min\,[\mu_{A_i}(x^*), \mu_{B_j}(y^*)], \mu_{C_k}(z)] \\ &\equiv \mu_{A_i}(x^*) \wedge \mu_{B_j}(y*) \wedge \mu_{C_k}(z),\end{aligned} \tag{2.12}$$

A saída da máquina de inferência é obtida a partir do máximo dos conjuntos *fuzzy* resultantes de cada regra. A Figura 11 mostra o conjunto *fuzzy* C' gerado pela operação de máximo entre os conjuntos C_1 e C_2 inferidos nas Regras 2 e 1, respectivamente.

2.3.4 Defuzzificador

A função do módulo defuzzificador é mapear o espaço *fuzzy* de saída, inferida pelas regras *fuzzy*, em um universo de discurso numérico. O objetivo é obter um único número real para cada variável de controle que melhor represente os valores *fuzzy* inferidos das variáveis lingüísticas de saída. Esse processo é necessário porque, em muitas aplicações

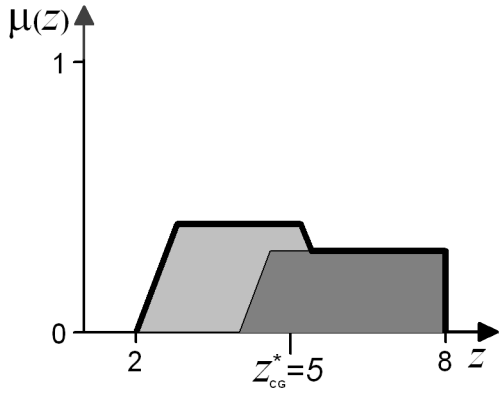


Figura 11: Conjunto *fuzzy* final obtido após agregação dos resultados de inferência das Regras 1 e 2. O valor 5 é o resultado escalar gerado pela "defuzzificação", utilizando o método de "centro de gravidade".

práticas, o sinal de controle deve ser enviado em formato numérico.

Na literatura, existem vários métodos de defuzzificação definidos. Neste trabalho, o método escolhido foi o de "centro de gravidade", que calcula, para um dado conjunto *fuzzy* de saída, a abscissa do ponto de centro de gravidade correspondente e a utiliza como valor escalar de saída. No caso de um universo de discurso discreto, esse método pode ser definido pela Equação 2.13, e para o caso contínuo, definido pela Equação 2.14:

$$z^*_{CG} = \frac{\sum_{z \in U} z \times \mu_C(z)}{\sum_{z \in U} \mu_C(z)} \tag{2.13}$$

$$z^*_{CG} = \frac{\int_{z \in U} z \times \mu_C(z)\, dz}{\int_{z \in U} \mu_C(z)\, dz} \tag{2.14}$$

Utilizando o exemplo anterior, a Figura 11 ilustra o processo de defuzzificação para a saída encontrada. Nesse caso, a saída da defuzzificação é o valor escalar $z^*_{CG} = 5$.

2.4 Controle por Propagação de Inferências - CpPI

Em [7], Kulitz propôs um novo método de obtenção de controlador *fuzzy* para movimentos à ré de robôs móveis ou veículos multi-articulados com dois ou mais semi-reboques sucessivamente acoplados, denominado Controle por Propagação de Inferências (CpPI). Tal controlador foi desenvolvido a partir do modelo cinemático *fuzzy* direto da composição que, por sua vez, foi gerado de dados amostrais de um modelo analítico do sistema obtidos com o simulador MGSARA [7].

A aplicação prática imediata do modelo *fuzzy* direto é a implementação de um sistema de auxílio ao motorista em manobras à ré de robôs ou veículos multi-articulados, também chamado de preditor *fuzzy*. Uma explicação detalhada do modelo *fuzzy* direto e do preditor *fuzzy* é encontrada em [7, 8].

Para a realização do controle automático, o passo seguinte é obter um modelo *fuzzy* para o controlador do sistema. O objetivo do controlador é fazer com que o ângulo na última articulação siga uma dada referência Ref_i. Para isso, deve ser encontrado o ângulo de direção em função da configuração atual do veículo e do valor desejado para a articulação do último semi-reboque. Nas próximas subseções, o controlador proposto em [7] e implementado neste trabalho é melhor detalhado.

2.4.1 Arquitetura CpPI

A metodologia adotada em [7] consiste em gerar um controlador *fuzzy* mais simples para cada articulação do veículo. Desse modo, estabelecida a direção desejada para o último semi-reboque, o sistema avalia a cadeia cinemática e, a partir do último semi-reboque, infere sobre o que deve acontecer em cada articulação para alcançar o objetivo. Só então a ação de controle é definida sobre as rodas de direção. A Figura 12 mostra a arquitetura proposta, dividida em pequenos controladores, cuja implementação é baseada no modelo de Mamdani, descrito na seção 2.3.

Cada controlador recebe como única entrada o erro a ser corrigido, ε_i. Esse erro é o resultado da diferença entre o valor desejado, Ref_i, e o valor atual, α_i. Como saída, é fornecida a ação necessária para correção do erro calculado, Ref_{i-1}, que é exatamente o ângulo desejado para a próxima articulação. Essa nova referência é então passada para o controlador da próxima articulação, que realiza o mesmo procedimento, até que seja encontrado o ângulo de referência Ref_1, ou seja, o sinal de controle α_1 enviado para a direção do veículo. O método descrito é denominado CpPI - Controle por Propagação de

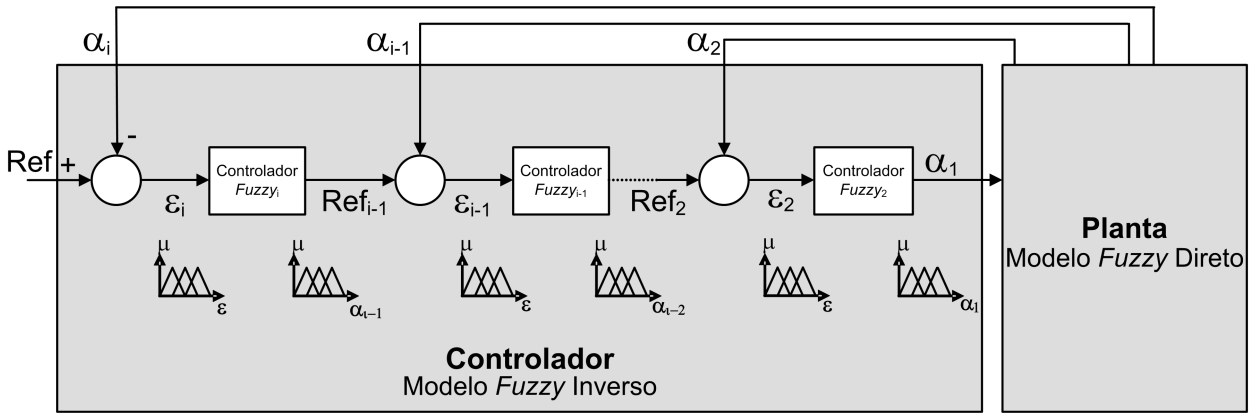


Figura 12: Arquitetura do controlador proposto em [7].
Fonte: Adaptado de [7].

Inferências e seu algoritmo pode ser visto na Figura 13.

2.4.2 Especificidades do Controlador Implementado

O CpPI é um controlador generalizado para um robô móvel ou veículo multi-articulado com N semi-reboques, ou seja, N graus de liberdade. Neste trabalho, para validar o sistema de visão e o controlador descrito, foi utilizado um protótipo de veículo multi-articulado, com um cavalo mecânico e dois semi-reboques. Desse modo, a configuração do veículo é definida pelos dois ângulos de interesse α_2 e α_3, mostrados na Figura 14. Como entradas, o CpPI recebe a configuração do protótipo e a referência para o último semi-reboque, Ref_3. Como saída, o CpPI deve fornecer o ângulo de direção α_1 necessário para que o último semi-reboque siga o ângulo desejado.

Para implementar o controlador, inicialmente foi necessário calcular os ângulos máximos permitidos em cada articulação, também chamados de ângulos críticos. O ângulo crítico define a faixa de valores de cada variável que permite o controle do sistema, ou seja, para qualquer configuração dentro dessas faixas de valores ainda é possível encontrar uma ação de controle que leve o sistema à condição desejada. Na prática, esses limites estabelecem o universo de discurso das variáveis do controlador. É importante ressaltar que a limitação física imposta ao ângulo de direção α_1 influencia nos ângulos máximos de todas as articulações. Desse modo, é possível calcular os ângulos críticos para α_2 e α_3 segundo as Equações 2.15, 2.16 e 2.17 [7].

$$\alpha_{n+1_{max}} = sen^{-1}\left(\frac{LA_{n+1} \times cos(\gamma)}{\beta}\right) + \gamma \tag{2.15}$$

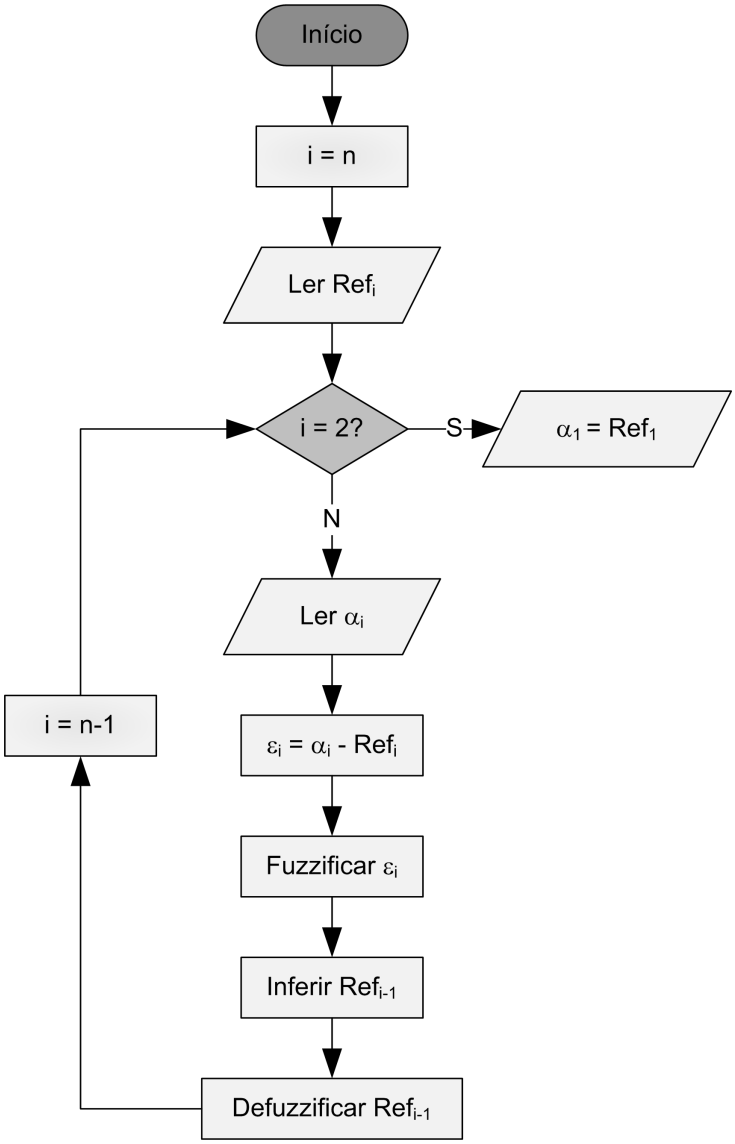


Figura 13: Algoritmo do CpPI.
Fonte: Adaptado de [7].

$$\gamma = tan^{-1}\left(\frac{LB_n}{\beta}\right) \tag{2.16}$$

$$\beta = \left(\frac{LA_n \times cos(\alpha_n) + LB_{n-1}}{sen(\alpha_n)}\right) \tag{2.17}$$

onde $n = 1...N$, sendo N o número de semi-reboques da composição, $LB = 0$ e as dimensões do veículo dadas pela Tabela 1.

A partir das Equações 2.15, 2.16 e 2.17 e das grandezas mostradas na Tabela 1, foi possível calcular os ângulos críticos das articulações como:

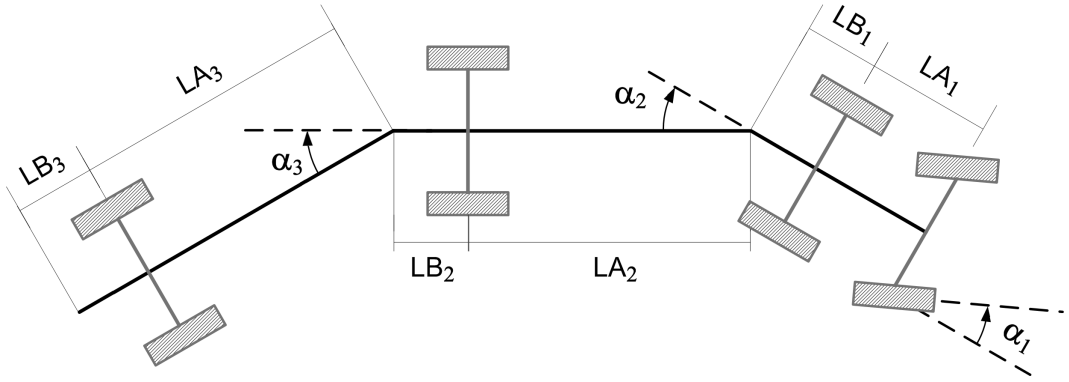


Figura 14: Dimensões do protótipo de veículo multi-articulado.

Tabela 1: Grandezas do protótipo utilizado.

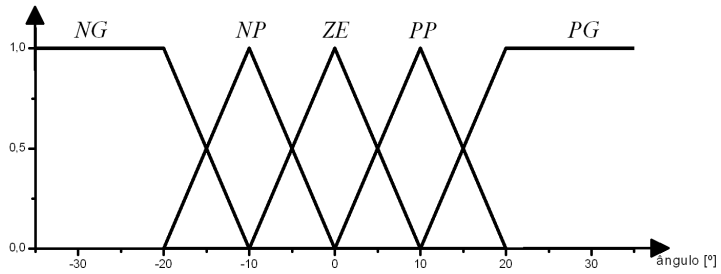
Grandezas do Protótipo	Medidas
α_1	20°
LA1	$30,0\ cm$
LB1	$0,0\ cm$
LA2	$49,5\ cm$
LB2	$26,5\ cm$
LA3	$49,5\ cm$
LB3	$0,0\ cm$

$$\begin{aligned} \alpha_{2_{max}} &= 36^o \\ \alpha_{3_{max}} &= 63^o \end{aligned} \tag{2.18}$$

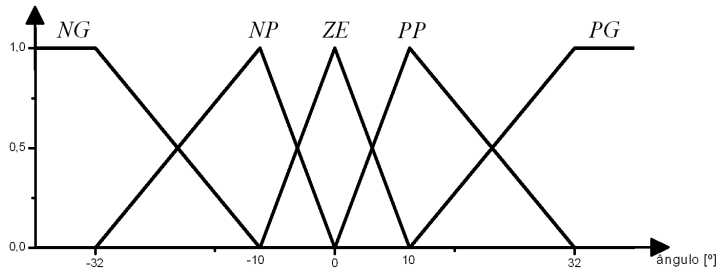
Com o cálculo dos ângulos críticos, o universo de discurso das variáveis α_2 e α_3 foi determinado. Por questões de simplicidade na implementação, foram atribuídos 5 termos lingüísticos para cada uma das três variáveis lingüísticas, representados pelos símbolos NG (negativo grande), NP (negativo pequeno), ZE (zero), PP (positivo pequeno) e PG (positivo grande). As funções de pertinência associados aos termos correspondentes de cada variável podem ser vistas na Figura 15.

A escolha dos termos das variáveis lingüísticas segue a regra de que o particionamento do universo de discurso das variáveis controladas deve observar a posição da variável na cadeia cinemática em relação ao ponto de aplicação do sinal de controle: variáveis que estão mais afastadas do cavalo mecânico devem ter o particionamento concentrado em pontos mais afastados de zero [7].

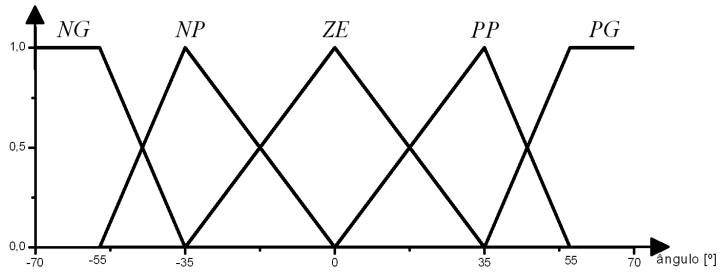
Os controladores da arquitetura mostrada na Figura 12 possuem base de regras semelhantes, que são mostradas na Tabela 2. Como cada controlador possui apenas uma



(a)



(b)



(c)

Figura 15: Variáveis lingüísticas para representação de (a) α_1, (b) α_2 e (c) α_3.

variável de entrada, o número de regras segue o número de partições que a variável possui.

Tabela 2: Base de Regras.

Regra	Entrada $\varepsilon = Ref_i - \alpha_i$	Saída α_{i-1}
1	NG	NG
2	NP	NG
3	ZE	ZE
4	PP	PG
5	PG	PG

3 Processamento de Imagens Digitais

"A arte da vida consiste em fazer da vida uma obra de arte."

Mahatma Gandhi

O interesse da ciência em métodos de processamento de imagens digitais cada vez mais aprimorados decorre principalmente de duas áreas de aplicação: a primeira busca melhorar as informações fornecidas por uma imagem de modo a facilitar interpretação humana e, conseqüentemente, torná-la mais correta, tais como na medicina ou na astronomia; a segunda área de aplicação procura imitar a visão, o sentido humano mais complexo, utilizando câmeras para fornecer informações sobre o ambiente de trabalho para robôs e máquinas, com o objetivo de torná-los mais autônomos e precisos. Essa última aplicação é chamada de visão computacional [30, 40, 41].

A importância do uso da visão computacional deve-se principalmente à quantidade de informações a respeito do ambiente de trabalho que uma imagem é capaz de fornecer. Por isso, o uso de câmeras é capaz de aumentar a flexibilidade e adaptabilidade do sistema, atributos importantes na operação em ambientes dinâmicos e complexos. Uma das vantagens de utilizar a visão computacional é a necessidade de apenas um sensor para a obtenção de informações de diferentes naturezas relativas ao sistema. Desse modo, sua integração a um sistema robótico pode evitar a necessidade do uso de uma combinação de diferentes sensores em situações que requerem informações mais aprimoradas e diversificadas a respeito do ambiente e do próprio robô. A desvantagem é o alto custo computacional inerente ao processamento de imagens [42].

Neste contexto, a proposta deste capítulo é descrever as técnicas de processamento de imagens digitais utilizadas na implementação do sistema de visão computacional, a fim de fornecer a realimentação ao controlador da composição.

3.1 Introdução

No desenvolvimento do sistema de controle para um veículo multi-articulado, descrito no Capítulo 2, é necessário que o controlador receba informações inerentes à configuração da composição. Essa configuração é caracterizada pelos ângulos em cada uma de suas articulações. Para o caso de um veículo composto por dois semi-reboques, estes ângulos são α_3 e α_2, mostrados na Figura 16. Nessa Figura, o ângulo α_1 representa a direção das rodas dianteiras do cavalo mecânico da composição. Neste trabalho, foi adotada a referência dos ângulos como positiva no sentido anti-horário. Desse modo, na Figura 16, α_1 e α_2 são positivos e α_3, negativo.

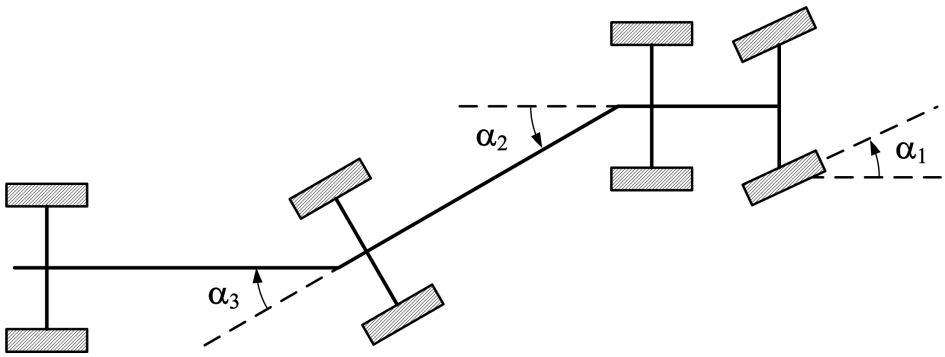


Figura 16: Ângulos de configuração para um veículo multi-articulado composto por um trator e dois trailers.

A visão computacional foi escolhida para obtenção destes dados, pois poderá ser utilizada também na aquisição de informações relativas ao ambiente de trabalho e ao próprio veículo, tais como sua postura em relação a um referencial fixo ou a localização de possíveis obstáculos. Estes dados são necessários para a implementação do sistema supervisório, descrito no Capítulo 1.

Trabalhos anteriores desenvolvidos pelo grupo de pesquisa [43,44] utilizaram processamento de imagens digitais para a estimação dos ângulos de configuração de um protótipo de veículo multi-articulado. Em ambos, os programas trabalhavam de modo *off-line*, ou seja, inicialmente um arquivo de vídeo com o protótipo parado ou executando alguma manobra a ré era gravado e, em seguida, este arquivo era processado pelos programas implementados [43, 44]. Sistemas com essa característica não podem ser utilizados no problema de controle da composição, para o qual seria necessário um sistema de obtenção de ângulos *on-line*. No entanto, a principal contribuição desses trabalhos foi apontar que o uso de visão computacional nesse tipo de problema é viável.

O primeiro trabalho desenvolvido [43] teve como objetivo validar o uso de uma câmera como sensor externo do protótipo, confrontando dados obtidos por processamento de imagem com medidas já conhecidas. O algoritmo foi todo implementado em MatLab®. Os resultados desse trabalho mostraram que o processamento de imagem foi capaz de medir os ângulos das articulações do veículo, parado ou executando manobras em baixa velocidade, de maneira satisfatória, com aproximadamente 1º de incerteza. Deste modo, concluiu-se que a visão computacional pode ser uma solução sensorial viável e confiável para o problema proposto, desde que operasse em modo *on-line*.

O objetivo inicial do segundo trabalho [44] era utilizar os resultados obtidos no primeiro para implementar um sistema que trabalhasse de modo *on-line*, isto é, obtivesse os valores dos ângulos durante a execução de uma manobra do veículo, possibilitando seu uso como realimentação de um controlador. Para isso, o sistema foi implementado em C++, utilizando uma biblioteca própria para o processamento de imagem, chamada OpenCV (*Open Source Computer Vision Library*) [45]. A câmera utilizada no desenvolvimento desse trabalho foi a *PixeLink MegaPixel Firewire Camera*, modelo *PL-A642* com resolução espacial máxima de 1280×1024 [46], mesma utilizada no trabalho anterior [43]. Todavia, o laboratório já não dispunha das ferramentas de desenvolvimento dessa câmera, o que dificultou sua comunicação com os programas implementados e tornou necessário o uso de outra biblioteca, a VideoLab [47], para este fim. Possivelmente por esta dificuldade de comunicação com a câmera, o programa demandou uma alta quantidade de memória virtual durante os testes em modo *on-line*. Esse consumo excessivo de memória inviabilizou o programa implementado para uso *on-line* e, conseqüentemente, para realimentação de um controlador.

Apesar das dificuldades encontradas em [43, 44], existem na literatura diversos relatos a respeito de sistemas de visão computacional implementados para seguimento ou controle de veículos ou robôs móveis [3,4,48–50] e, especificamente, no caso de multi-articulados [15, 19, 51]. Por esse motivo, neste trabalho foi implementado um sistema de processamento de imagens *on-line*, cujos detalhes serão expostos nas seções seguintes.

3.2 Sistema de Processamento de Imagens Digitais

O objetivo do sistema de processamento de imagens digitais desenvolvido é estimar os ângulos de configuração da composição, α_2 e α_3. Sabe-se que em uma abordagem mais geral do problema, a estratégia de segmentação deveria extrair diretamente os parâmetros

geométricos da composição de modo a generalizar o seu uso. No entanto, neste trabalho foi utilizado um algoritmo mais simples baseado em segmentação por cor. Mesmo esse algoritmo simples é computacionalmente complexo e, portanto lento, dependendo da velocidade da CPU.

Para identificar os elementos da composição, foram adaptados cartões coloridos sobre o veículo, como mostra a Figura 17, na qual o cavalo mecânico é identificado com um cartão verde e os semi-reboques com laranja e azul.

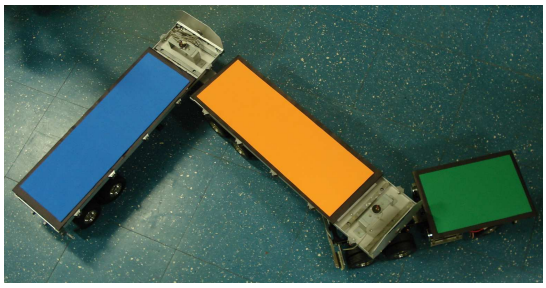


Figura 17: Réplica do veículo com cartões coloridos para facilitar a identificação de cada corpo da composição.

Antes de estimar os ângulos de configuração, o sistema deve conhecer as características que definem cada objeto de interesse. No caso da segmentação de cor, essas características são os limiares de segmentação de cada canal do sistema de cores adotado. Usar o modelo de cor RGB - vermelho, verde e azul ou *Red*, *Green* e *Blue* - no processamento de imagem é um método bastante simples, uma vez que, em geral, as câmeras capturam as imagens utilizam esse modelo diretamente. No entanto, o RGB torna a detecção sensível às condições do ambiente de trabalho, como a variação de iluminação [30], ou seja, um objeto que é reconhecido pelo processamento durante o dia, com o uso de luz solar, pode não ser reconhecido à noite, com uso de luz artificial. Para contornar esse problema, o processamento de imagem deve ser implementado usando o modelo de cor HSV, que emprega os conceitos qualitativos de matiz (*Hue*), saturação (*Saturation*) e brilho (*Value*) e que corresponde à projeção ao longo da diagonal principal, do branco ao preto, no espaço de cor RGB, mostrado na Figura 18. Desse modo, a informação de cor - a matiz ou canal H - é isolada da informação de luminosidade. O resultado dessa projeção é a pirâmide de base hexagonal da Figura 19.

No sistema de cor HSV, a matiz corresponde à cor, propriamente dita. A saturação corresponde à quantidade de cor branca contida em uma cor. Uma saturação igual a 1.0 significa que a cor é pura, enquanto uma saturação igual a 0.0 significa que a cor é totalmente branca e, neste caso, o valor do parâmetro H é irrelevante. O parâmetro V

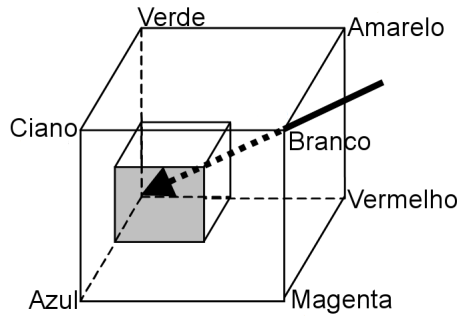


Figura 18: Cubo do espaço de cor RGB.
Fonte: Adaptado de [52].

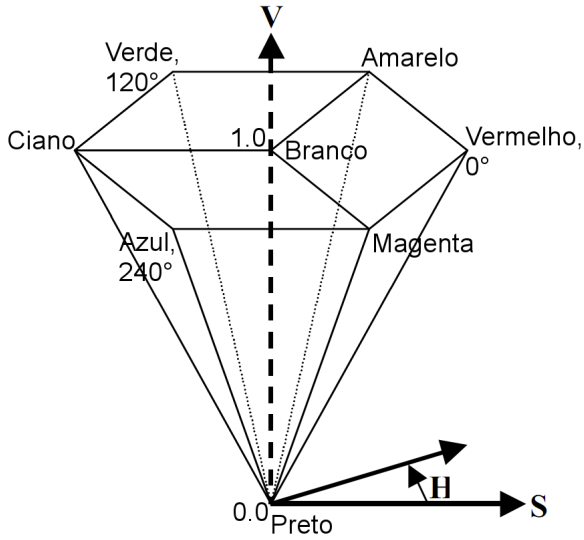


Figura 19: Cone hexagonal do modelo de cor HSV.
Fonte: Adaptado de [52]

corresponde à intensidade da cor e varia entre 0.0 (ausência de luz, ou seja, cor negra, onde os valores de H e S são irrelevantes) e 1.0 (intensidade máxima). Naturalmente, os tons cinzentos encontram-se nos intervalos em que $0 \leq V \leq 1$ e $S = 0$, sendo o valor de H indiferente. Sendo o modelo utilizado neste trabalho, logo após sua aquisição, cada imagem é convertida do modelo RGB, utilizado pela câmera, para o HSV.

De modo geral, o sistema de processamento de imagens implementado pode ser dividido em três etapas:

1. Localização dos elementos da composição.
2. Estimativa de orientação e centro de massa de cada objeto.
3. Cálculo dos ângulos de referência da composição

Como em [44], a implementação de cada uma dessas etapas empregou a biblioteca de processamento de imagens OpenCV [45]. A OpenCV é uma biblioteca gratuita e com código aberto, disponibilizada pela Intel, cujo objetivo é auxiliar o desenvolvimento de aplicações de visão computacional, empregando algoritmos que otimizam o desempenho do processamento de imagens [45].

3.2.1 Localização dos elementos da composição

A fase inicial do sistema de processamento de imagens tem por objetivo localizar cada elemento da composição para que as informações relevantes sobre esses objetos possam ser extraídas das imagens posteriormente. Para isso, como descrito na seção anterior, o sistema deve conhecer informações que caracterizem cada um dos elementos. Nesse caso, essas informações são os valores de limiares de segmentação dos canais H, S e V. Na implementação desse sistema o canal V não foi utilizado, pois é desejável que mudanças na iluminação não influenciem no resultado da segmentação.

O processo de escolha dos limiares de segmentação é realizado de modo iterativo, ou seja, o usuário do sistema é quem define o melhor isolamento possível de cada um dos objetos da composição em relação ao resto da imagem. Ao movimentar as barras com a indicação dos valores de limiar de segmentação, mostradas na Figura 20, o usuário pode visualizar o resultado do processamento das imagens com os valores selecionados. Os limiares de segmentação obtidos nessa etapa serão usados durante todo o processamento.

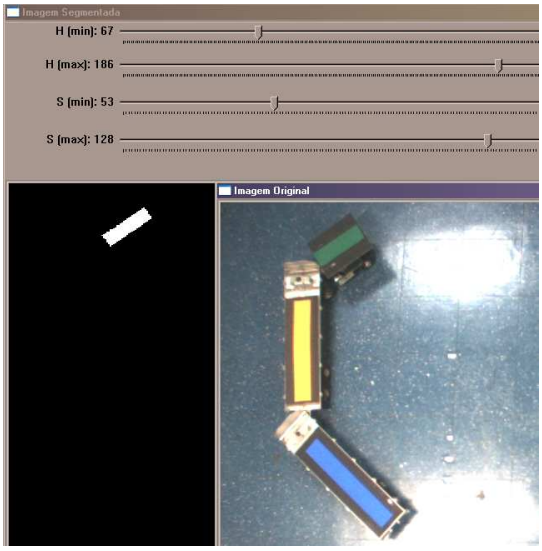


Figura 20: Recorte da visão da imagem original e imagem segmentada com os limiares mostrados nas barras de ajuste.

O processo de escolha dos limiares de segmentação de cada elemento da composição é

detalhado no fluxograma da Figura 21. Após a obtenção de todos os limiares, é iniciada a etapa de extração de características dos elementos já segmentados.

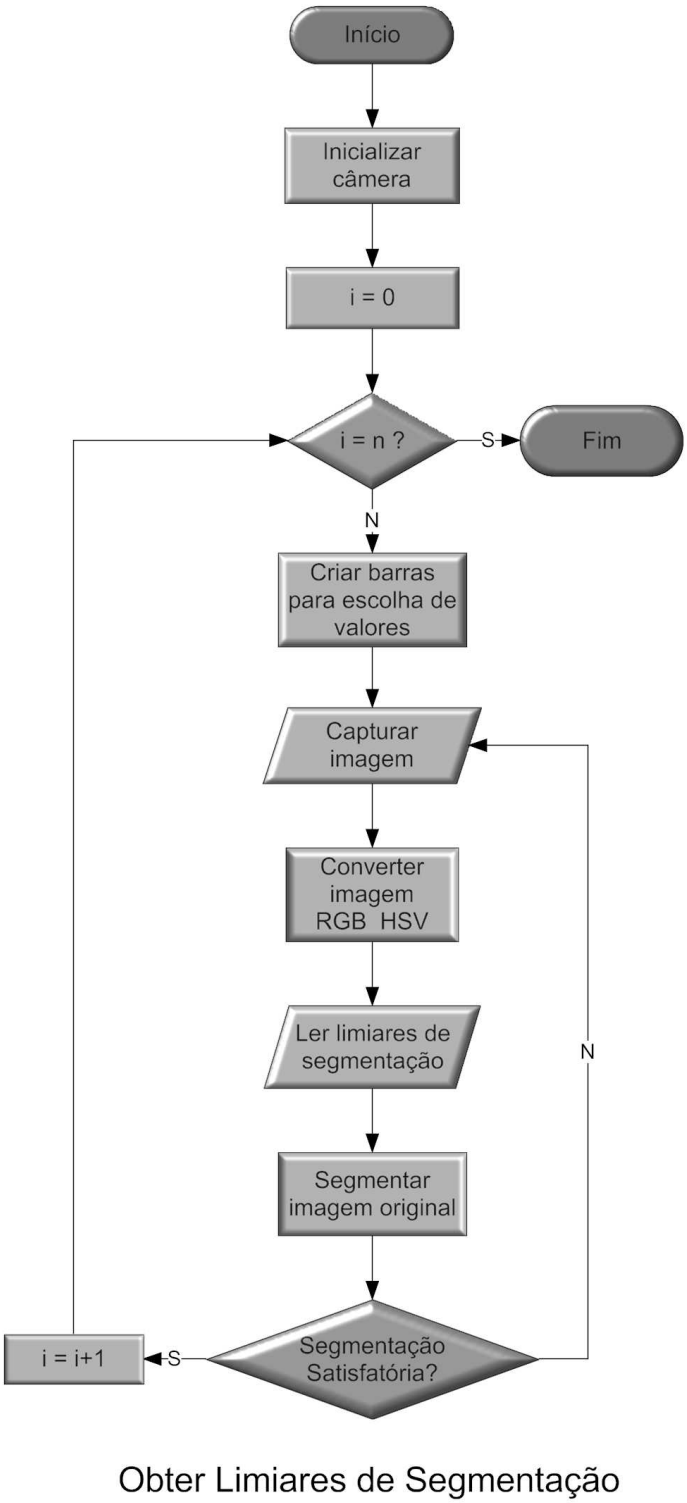


Figura 21: Fluxograma do algoritmo de obtenção dos limiares de segmentação de cada elemento da composição.

3.2.2 Estimativa de orientação e centro de massa de cada objeto

Com os limiares obtidos na primeira etapa do processamento, o sistema passa a segmentar cada imagem capturada pela câmera e a calcular os centro de massa e a orientação de cada elemento da composição. Esses dados são calculados com base em dois tipos importantes de momentos, obtidos diretamente das imagens segmentadas pela biblioteca OpenCV: o momento espacial, m_{pq}, e o momento central, μ_{pq}, cujas fórmulas são dadas respectivamente pelas Equações 3.1 e 3.2. Nessas fórmulas, $f(x, y)$ denota a intensidade do *pixel* na posição (x, y) e n_x e n_y, a quantidade de linhas e colunas da imagem, respectivamente.

$$m_{pq} = \sum_{1}^{n_x} \sum_{1}^{n_y} x^p y^q f(x, y) \tag{3.1}$$

$$\mu_{pq} = \sum_{1}^{n_x} \sum_{1}^{n_y} (x - x_c)^p (y - y_c)^q f(x, y) \tag{3.2}$$

Com os momentos calculados, o centro de massa (x_c, y_c) e a orientação θ de cada elemento da composição são dados por:

$$x_c = \frac{m_{10}}{m_{00}} \qquad y_c = \frac{m_{01}}{m_{00}} \tag{3.3}$$

$$\theta = \tan^{-1} \frac{\mu_{02} - \mu_{20} - 2\mu_{11} + \lambda}{\mu_{02} - \mu_{20} + 2\mu_{11} - \lambda}, \tag{3.4}$$

onde,

$$\lambda = \sqrt{(\mu_{20} - \mu_{02})^2 + 4\mu_{11}^2}. \tag{3.5}$$

Todo o processo de segmentação por cor e de extração das informações de orientação e centro de massa de cada elemento da composição é detalhado no fluxograma da Figura 22. Com esses dados conhecidos, é possível estimar os ângulos de referência da composição.

3.2.3 Cálculo dos ângulos de referência da composição

A última etapa do sistema de processamento de imagens calcula os ângulos de configuração da composição, α_2 e α_3, utilizando as orientações, θ_1, θ_2 e θ_3, de cada elemento

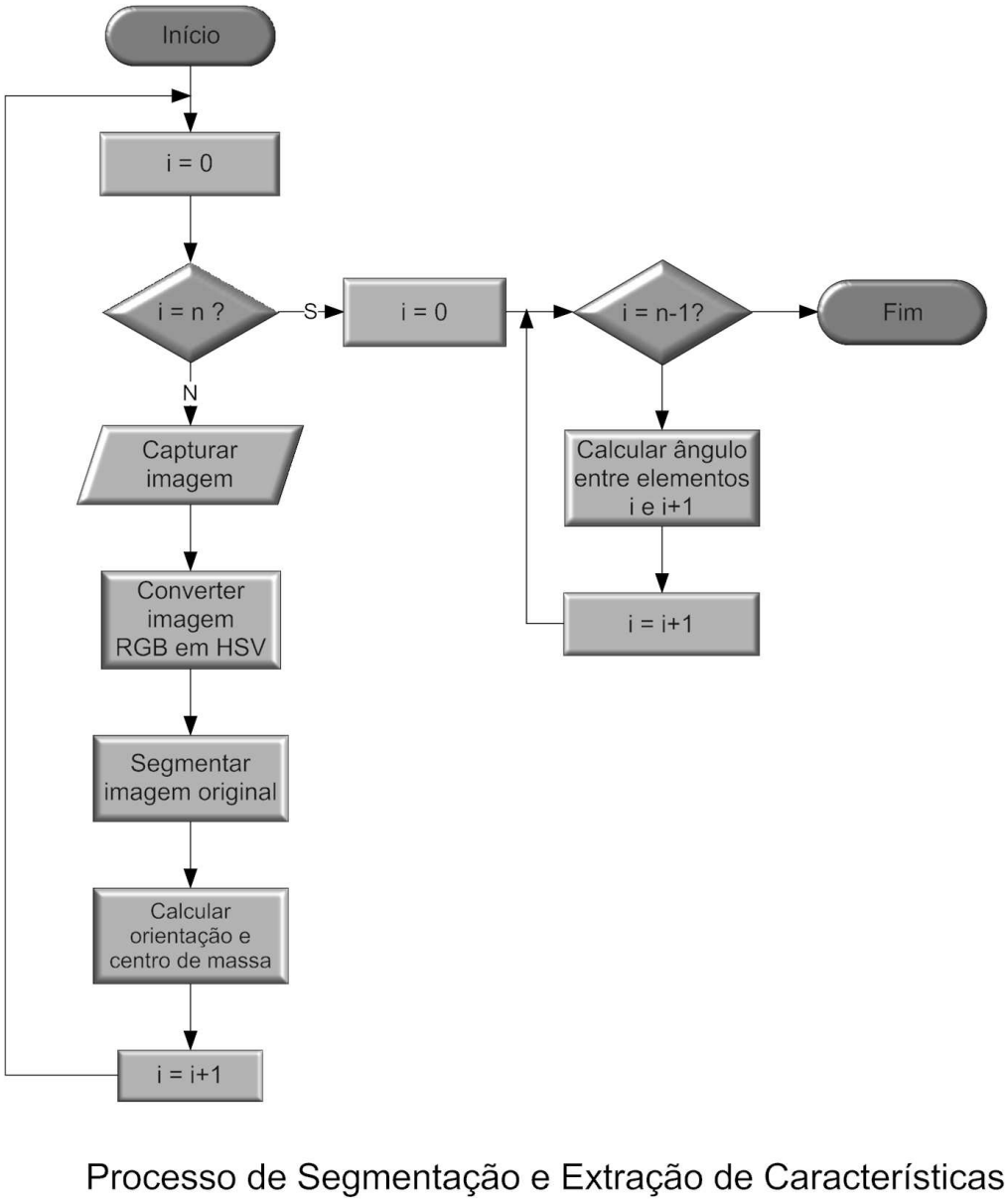


Figura 22: Fluxograma do algoritmo de limiarização e extração de características de cada elemento da composição.

estimadas na etapa anterior. Com essas orientações, é possível encontrar o valor dos ângulos do engate a partir das Equações 3.6 e 3.7. A subtração das orientações é possível porque, devido às limitações físicas do veículo utilizado, o ângulo de configuração máximo é de 90° e o mínimo é de -90°. Desse modo, caso o valor da subtração esteja acima de 90° ou abaixo de -90° é necessário subtrair ou somar 180°, respectivamente, ao resultado. É importante observar a ordem dos termos nas Equações 3.6 e 3.7, pois o sinal do resultado

final representa o sentido do ângulos das articulações, descrito na seção 3.1 deste Capítulo.

$$\alpha_2 = \theta_2 - \theta_1 \tag{3.6}$$

$$\alpha_3 = \theta_3 - \theta_2 \tag{3.7}$$

Apesar da simplicidade desses cálculos, existe uma forma mais geral e mais complexa de calcular os ângulos nas articulações, independente das limitações físicas do veículo. Como mostrado na Figura 23, para encontrar o valor de α_2, são definidos dois vetores: $\vec{\phi_{12}}$, com origem em (x_1, y_1) e destino em (x_{C1}, y_{C1}), e $\vec{\beta_{12}}$, com origem em (x_{C1}, y_{C1}) e destino em (x_2, y_2), sendo (x_1, y_1) e (x_2, y_2) os centro de massa dos elementos identificados com o cartão verde e laranja, respectivamente. O ponto (x_{C1}, y_{C1}), como mostrado na Figura 23, é a intersecção entre as retas que passam pelos centros de massa dos objetos e com orientação obtida no procedimento anterior. O módulo de α_2 pode ser calculado a partir do produto interno desses dois vetores (Equação 3.6). O sinal de α_2 é determinado pelo produto vetorial $\vec{\phi_{12}} \times \vec{\beta_{12}}$. Uma vez que o ângulo tem como referência o cavalo mecânico ou o semi-reboque mais próximo ao cavalo mecânico, se $\vec{\phi_{12}} \times \vec{\beta_{12}} \geq 0$, então α_2 é positivo.

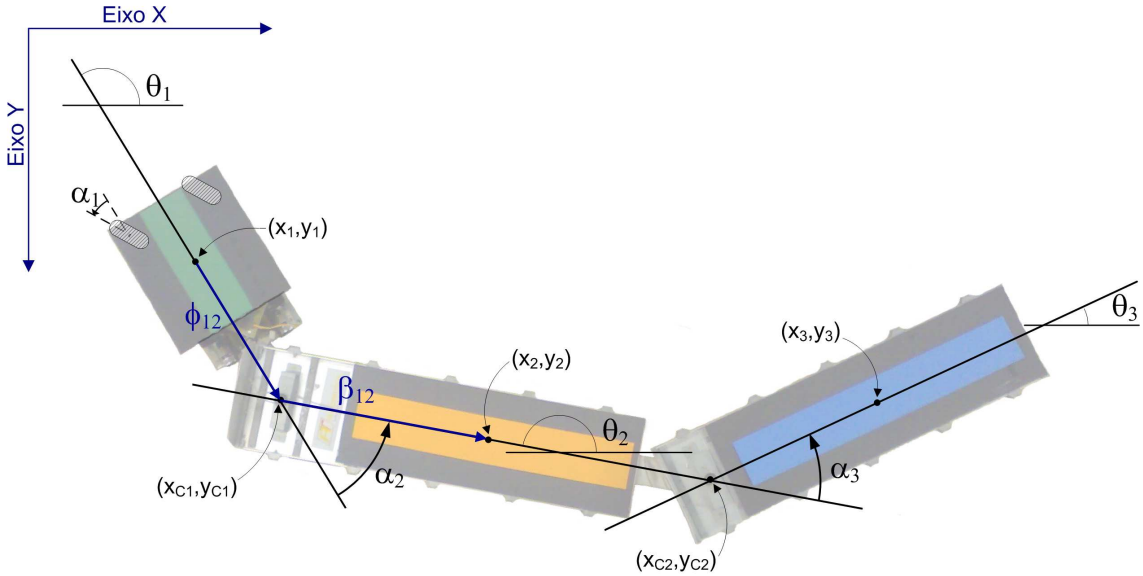


Figura 23: Configuração de um veículo multi-articulado composto de dois semi-reboques.

$$|\alpha_2| = \arccos\left(\frac{\vec{\phi_{12}}.\vec{\beta_{12}}}{\|\vec{\phi_{12}}\|.\|\vec{\beta_{12}}\|}\right) \tag{3.8}$$

O mesmo procedimento é realizado para calcular o ângulo entre os dois últimos semi-

reboques. No entanto, neste cálculo, são utilizadas as informações de centro de massa e orientação relativas aos objetos laranja e azul.

3.2.4 Comparação com potenciômetros de precisão

Os experimentos buscaram analisar a qualidade da medida da configuração por processamento de imagens em diferentes condições, de modo a quantificar possíveis influências de fatores, tais como distorção de borda do campo de visão da câmera e velocidade do veículo. Para isso, procurou-se iniciar todos os testes com o veículo em configuração nula, ou seja $\alpha_2 = \alpha_3 = 0^o$. Em cada experimento, assim que um *frame* da câmera era capturado, o sistema enviava ao microcontrolador da composição um comando solicitando as medidas dos potenciômetros de precisão, que eram armazenados no próprio microcontrolador. O intervalo médio entre cada amostragem foi de 74 ms, tempo aproximado de captura e processamento da imagem. Tal período é um valor esperado, uma vez que a velocidade da câmera limita a captura de um *frame* a cada 66 ms. No fim de cada experimento, os dados armazenados no microcontrolador foram enviados ao computador para análise posterior.

O primeiro teste foi realizado com o veículo parado e alinhado, com o objetivo de verificar o comportamento das estimativas dos ângulos de configuração para esse caso. A Figura 24 mostra o resultado da estimativa e a Tabela 3, os erros quadráticos médios entre as estimativas dos potenciômetros e do processamento de imagem.

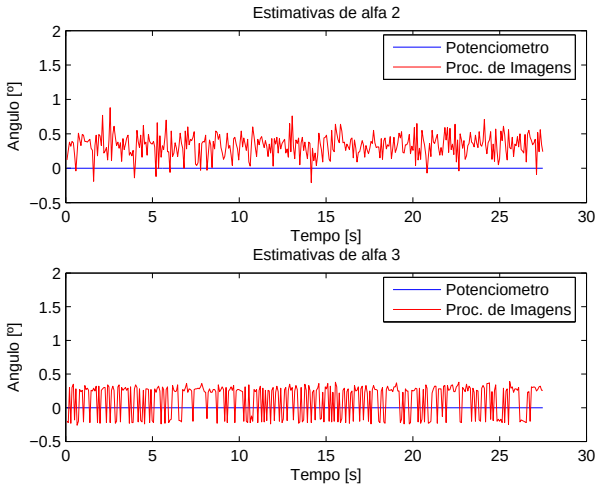


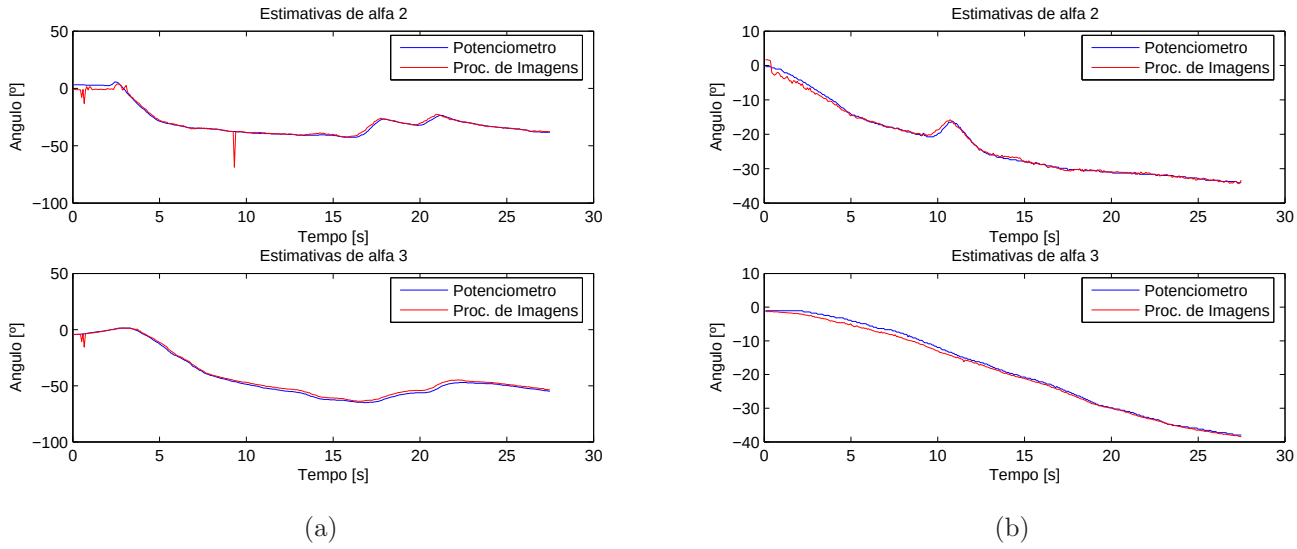
Figura 24: Estimativa dos ângulos de configuração com o veículo parado.

Em seguida, foram observados os efeitos da distorção de bordas, provocados pela lente grande-angular utilizada. Para isto, os dados foram coletados em dois tipos de

Tabela 3: Erro quadrático médio - estimativa de 0°.

Erro Quadrático Médio	
α_2	$0,13$
α_3	$0,069$

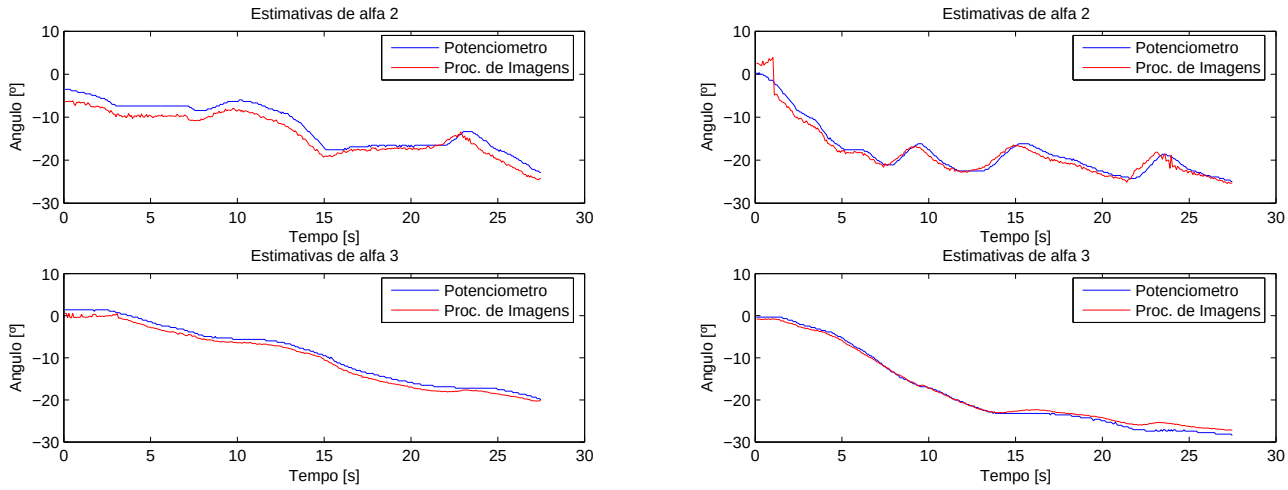
experimentos diferentes: no primeiro, o veículo percorreu uma área restrita ao centro do campo de visão da câmera; no segundo experimento, o veículo executou sua manobra na região de borda. Em ambos, buscou-se manter um percurso semelhante. A Figura 25 mostra os resultados de dois experimentos realizados no centro do campo de visão e a Figura 26, os dois experimentos realizados na borda do campo de visão da câmera, onde a distorção pode ser mais percebida. Os erros quadráticos médios entre as medidas dos potenciômetros e do processamento de imagem são mostrados na Tabela 4. Esses resultados apontam que a distorção da imagem não influencia de modo significativo nas estimativas dos ângulos. É importante ressaltar que o valor alto do erro detectado no resultado do primeiro teste foi, provavelmente, causado pelo ruído perceptível na Figura 25(a).



(a) (b)

Figura 25: Estimativas de α_2 e α_3 realizadas no centro do campo de visão da câmera.

Na terceira fase de testes, o veículo executou manobras com uma velocidade de aproximadamente $0,25\ m/s$ e, em seguida, uma manobra a uma velocidade de aproximadamente $0,50\ m/s$. Em ambos, procurou-se concentrar o percurso percorrido dentro da região não afetada pela distorção da câmera. O objetivo desse teste era analisar a influência da velocidade do veículo na qualidade das estimativas. Os resultados são mostrados nas



(a) (b)

Figura 26: Estimativas de α_2 e α_3 realizadas na borda do campo de visão da câmera.

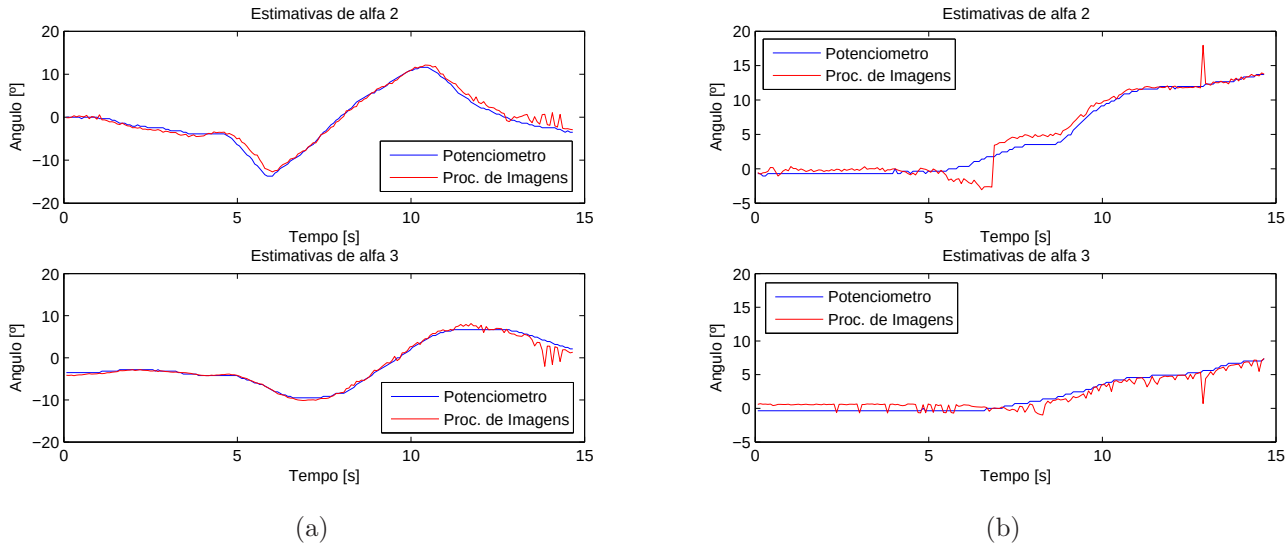
Tabela 4: Erro quadrático médio das medidas de α_2 e α_3 em manobras em áreas com e sem efeito da distorção produzida pela lente.

Erro Quadrático Médio				
	Sem Distorção		*Com Distorção*	
	(a)	(b)	(a)	(b)
α_2	$5,67$	$0,51$	$4,33$	$1,68$
α_3	$3,06$	$0,61$	$1,18$	$0,62$

Figuras 27 e 28. Nos gráficos mostrados na Figura 28, é possível observar um aumento pequeno no erro quadrático médio das estimativas por processamento de imagens, o que é quantificado na Tabela 5. No entanto, vale ressaltar que, em situações reais, as manobras à ré de veículo multi-articulados são realizadas em baixa velocidade.

Para finalizar, foram realizadas algumas manobras lentas à ré. Os resultados são mostrados na Figura 29. Os erros quadráticos médios são descritos na Tabela 6. Como nos outros experimentos, as estimativas da configuração por processamento de imagens seguiram as realizadas pelos potenciômetros.

É possível observar que as estimativas dos ângulos de configuração realizadas com esse sistema de processamento de imagens apresentaram erro quadrático médio baixo. No entanto, nota-se que as estimativas de α_2 apresentam mais ruídos que as estimativas de α_3. Isso ocorre devido à qualidade da segmentação do primeiro cartão, de cor verde, que mostrou-se pior que a segmentação dos demais cartões. O tamanho do cartão também



(a) (b)

Figura 27: Influência da velocidade da composição nas estimativas dos ângulos α_2 e α_3: manobras lentas.

Tabela 5: Erro quadrático médio das medidas de α_2 e α_3 em manobras lentas e rápida.

Erro Quadrático Médio			
	Manobras Lentas		*Manobra Rápida*
	(a)	(b)	
α_2	$0,86$	$1,49$	$3,00$
α_3	$0,68$	$0,70$	$1,12$

pode influenciar a qualidade das estimativas, uma vez que o ângulo é estimado a partir da orientação do maior eixo do cartão. No cartão verde, a proporção entre os seus eixos é menor, sendo mais suscetível a erros.

Tabela 6: Erro quadrático médio das medidas de α_2 e α_3 em manobras lentas à ré.

Erro Quadrático Médio			
	Manobra 1	Manobra 2	Manobra 3
α_2	$0,66$	$15,1$	$2,65$
α_3	$1,03$	$5,98$	$2,97$

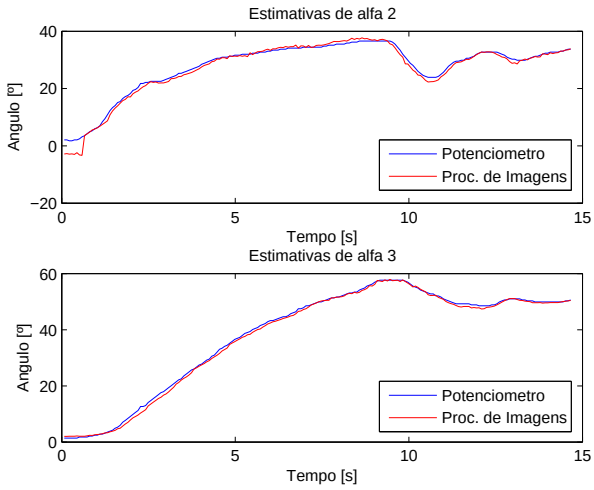


Figura 28: Influência da velocidade da composição nas estimativas dos ângulos α_2 e α_3: manobra rápida.

3.2.5 Adaptações do Sistema de Processamento de Imagens

Apesar dos bons resultados encontrados com o sistema de processamento de imagens implementado, alguns problemas durante sua execução estimularam melhorias no algoritmo. O problema mais grave eram ruídos que surgiam durante o processamento das imagens devido ao aparecimento de algum objeto com características de cor semelhante às características de um dos elementos da composição. Caso ocorresse uma situação desse tipo, o sistema não conseguiria diferenciar o elemento da composição do objeto indesejado e calcularia a orientação final a partir de todos os pixels segmentados. A diferença de orientação causada por esse tipo de problema pode ser observada na Figura 30: em 30(a), a segmentação foi realizada sem ruídos e a orientação calculada foi de $45,09^o$; em 30(b), ocorreu o surgimento de um ruído e, ainda com o objeto na mesma posição da imagem anterior, a orientação calculada foi em $53,46^o$, bem diferente da primeira.

Além do surgimento de possíveis ruídos, outro problema do algoritmo inicialmente utilizado no processamento das imagens é a possibilidade de mudança dos limiares de segmentação de algum dos objetos enquanto o veículo executa uma trajetória. Embora o algoritmo utilize o modelo de cor HSV para realizar a segmentação das imagens, durante execução das manobras o veículo percorre locais com diferenças de iluminação, o que pode causar falha no processamento. A diferença na iluminação também ocorria ao realizar o processamento em diferentes horários do dia.

Para evitar os problemas apresentados, o sistema de processamento de imagens foi atualizado e passou a utilizar um algoritmo de rastreamento e segmentação baseado em

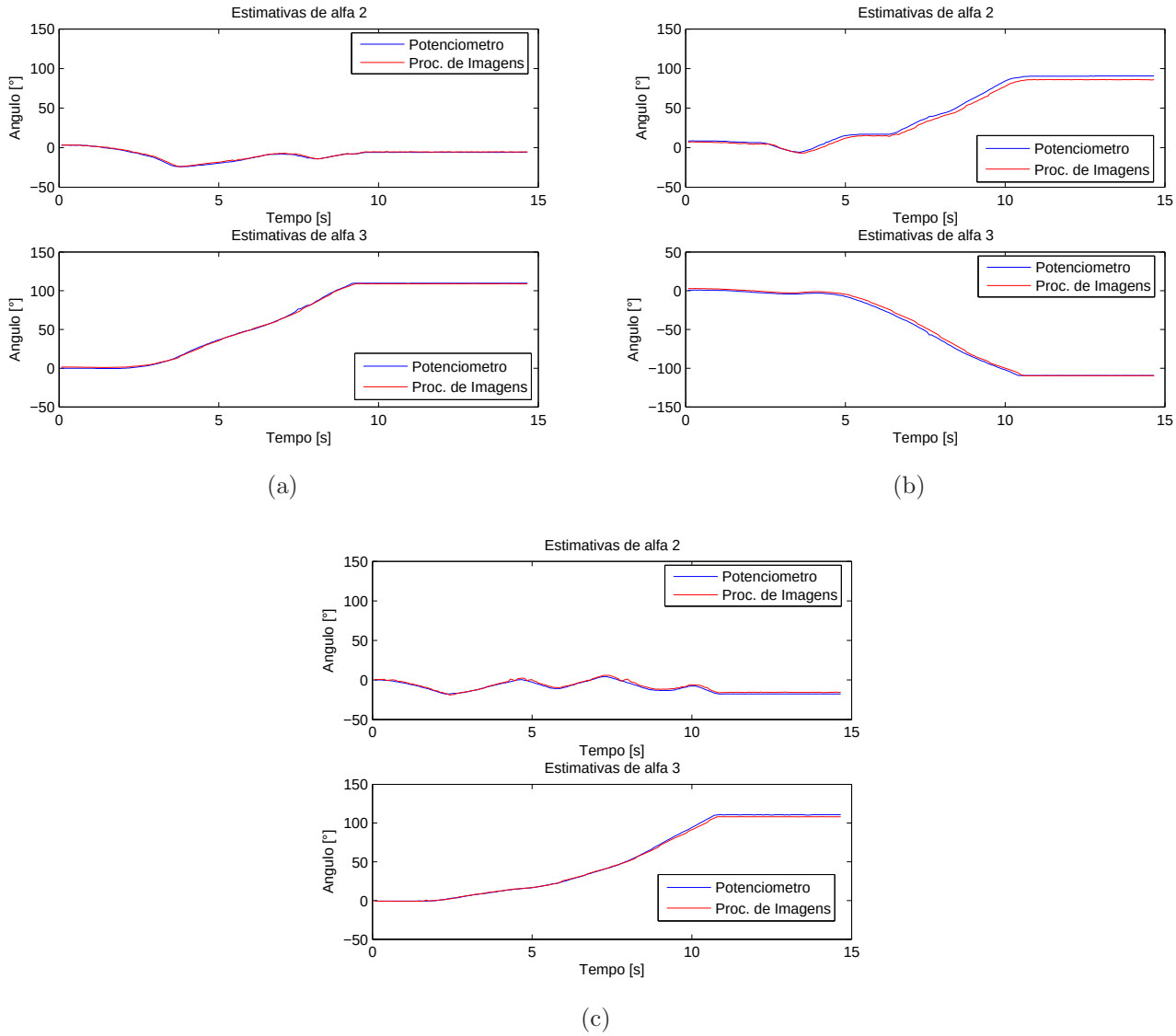
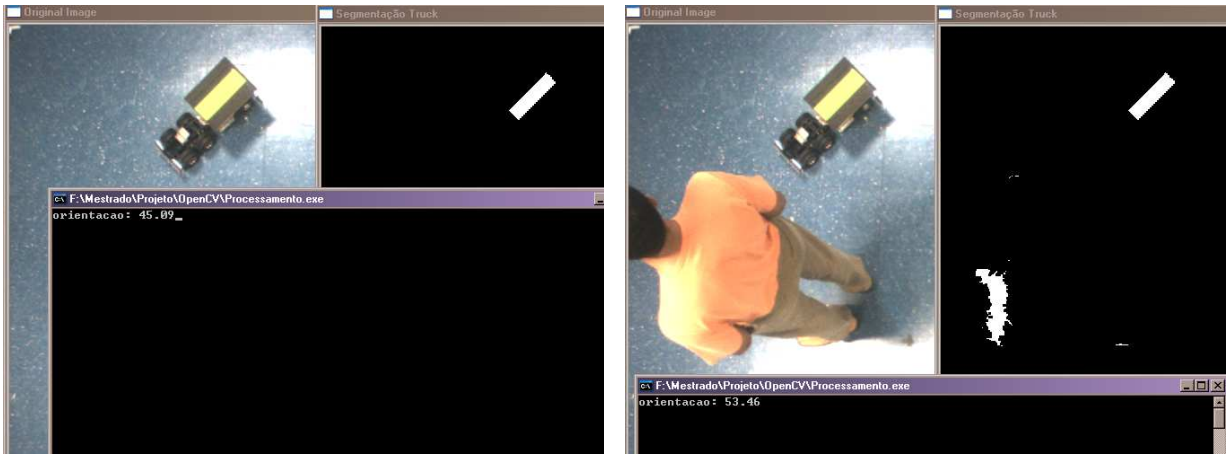


Figura 29: Estimativas dos ângulos α_2 e α_3 em manobras à ré: (a) primeira, (b) segunda e (c) terceira manobra.

cor disponível na biblioteca OpenCV e denominado CAMSHIFT – *Continuously Adaptive Mean-Shift* [52], no lugar do algoritmo de segmentação normal. O algoritmo CAMSHIFT utiliza a imagem original convertida para o modelo de cor HSV e, juntamente com o histograma da cor do objeto de interesse, cria uma imagem de distribuição de probabilidade de cor. Esse processo é chamado de *back-projection*. O histograma deve ser inicializado a partir da amostragem de uma área representativa da imagem, especificada manualmente, isto é, uma janela que possa caracterizar o objeto de interesse na imagem.

Desse modo, ao invés de arrastar as barras com a indicação dos limiares de segmentação, para iniciar o algoritmo CAMSHIFT, é necessário apenas que o usuário do programa clique sobre o elemento da composição, seguindo a ordem pedida pelo pro-



(a) (b)

Figura 30: Influência de ruídos na estimativa da orientação de um objeto: em (a) a estimativa foi realizada sem presença de ruídos no campo de visão da câmera; já em (b), mesmo sem alteração na posição do objeto, a estimativa de orientação muda.

grama implementado: o primeiro passo é clicar sobre a imagem do elemento trator. Com isso, o programa é capaz de criar uma janela de busca com centro no ponto selecionado e com tamanho de 11 *pixels* $\times$ 11 *pixels*. A partir dessa janela, é possível criar a janela de busca inicial do CAMSHIFT e o histograma do canal *Hue* de cada elemento da composição. Os histogramas do canal *Hue* de cada elemento podem ser vistos na Figura 31. É importante observar que, com esse algoritmo, não é necessário que cada elemento da composição seja identificado por uma cor diferente. Assim, é possível utilizar a cor cuja segmentação é menos ruidosa, nesse caso a cor laranja.

Dada uma primeira janela de busca e a imagem de distribuição de probabilidade de cor, o algoritmo detecta os *pixels* internos e vizinhos à janela e que possuem intensidade de cor semelhante à distribuição apresentada no histograma. Essa região é segmentada e o algoritmo guarda o retângulo que contém todos esses *pixels* detectados e a estimativa do tamanho, da orientação e do centróide do objeto formado por esses *pixels*. Os cálculos também utilizam os momentos descritos na seção 3.2.2. O retângulo guardado nesta fase do processamento é chamado de região de interesse ou ROI (*region of interest*). A ROI é então reutilizado pelo algoritmo como janela de busca inicial na próxima imagem a ser processada, não sendo necessário varrer toda a imagem para encontrar o objeto de interesse, como feito no sistema anterior, mas apenas a janela de busca e suas vizinhanças. Essa redução na área de processamento diminui o custo computacional. Durante o processamento, o histograma também é atualizado constantemente [53]. A Figura 31 mostra uma elipse circunscrita nos objetos detectados pelo CAMSHIFT.

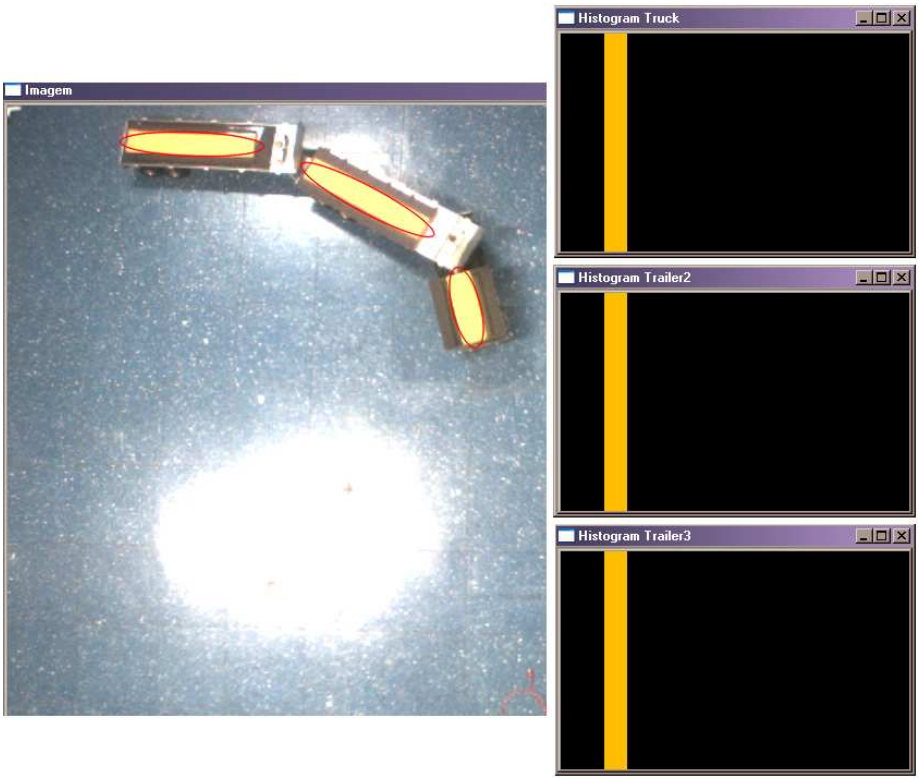


Figura 31: Elipses circunscritas sobre os objetos identificados pelo algoritmo CAMSHIFT e seu histograma.

Além da substituição do algoritmo de segmentação comum pelo algoritmo CAMSHIFT, com o objetivo de diminuir o efeito dos ruídos na estimativa dos ângulos de configuração, optou-se por utilizar o Filtro de Kalman pelo seu bom desempenho e pela simplicidade de sua implementação, sendo o seu custo computacional relativamente baixo. Tal característica é importante, uma vez que a filtragem é realizada a cada *frame* capturado.

3.2.6 Comparação com potenciômetros de precisão

Para validar o desempenho do último sistema implementado, foi realizada uma série de testes semelhante à realizada para a validação do primeiro sistema. Como na seção 3.2.4, o objetivo destes testes era analisar a qualidade das estimativas da configuração do veículo multi-articulado em diferentes situações, de modo a comparar os dois sistemas. A primeira disparidade percebida entre eles foi relativa ao tempo de processamento de cada imagem capturada. Nesse novo sistema, o intervalo aproximado entre cada amostragem era de 77 ms, cerca de 4 ms mais lento que o anterior.

O primeiro teste foi realizado com o veículo parado e alinhado. Os resultados das estimativas dos ângulos de configuração podem ser vistos na Figura 32 e o erro quadrático

médio, na Tabela 7. A partir da comparação desses dados com os obtidos pelo primeiro sistema, é possível verificar que o erro diminuiu. Além disso, aplicação do Filtro de Kalman nos valores estimados ajudou a reduzir o erro quadrático médio, no entanto, não expressivamente.

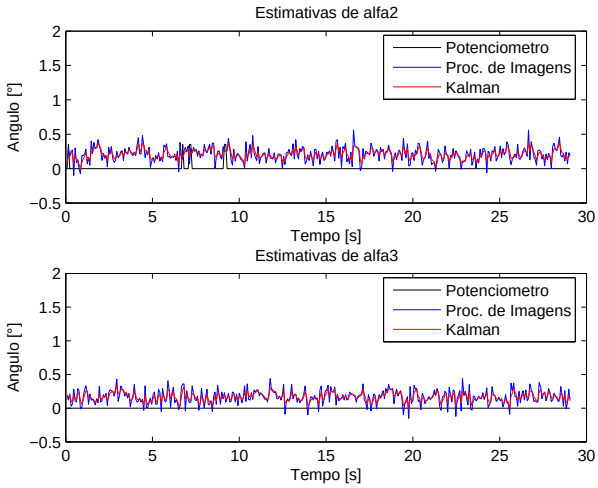


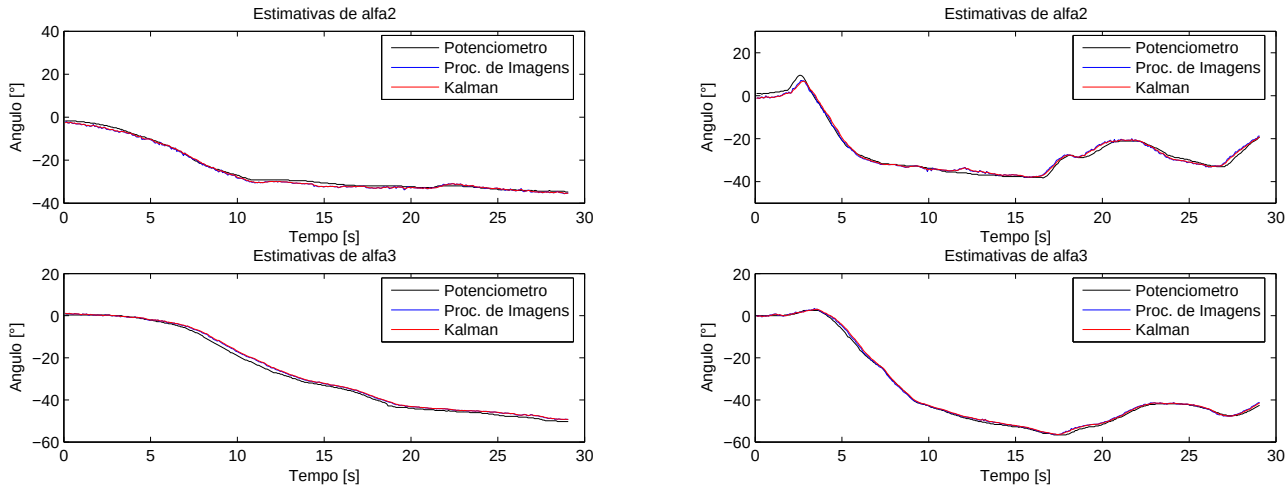
Figura 32: Estimativa dos ângulos de configuração com o veículo parado.

Tabela 7: Erro quadrático médio - estimativa de 0°.

Erro Quadrático Médio	
α_2	$0,054$
$\alpha_{2_{Kalman}}$	$0,047$
α_3	$0,038$
$\alpha_{3_{Kalman}}$	$0,031$

A segunda fase de testes foi relativa à influência do efeito de distorção de borda do campo visual da câmera na estimativa dos ângulos. Os testes foram realizados de modo semelhante aos do primeiro sistema: inicialmente foram percorridos dois caminhos parecidos e restritos ao centro de visão da câmera e, em seguida, percorrido um percurso na região de borda. Os resultados desses testes são mostrados na Figura 33 e 34, respectivamente. A Tabela 8 contém os erros quadráticos médios desse experimento. Assim como para o sistema anterior, a distorção de borda não interferiu significativamente no resultado final das estimativas. Além disso, os resultados obtidos a partir da aplicação do Filtro de Kalman não gerou melhoras em alguns casos.

O teste seguinte tinha por objetivo analisar o desempenho do sistema de visão em relação à velocidade da composição. Nessa fase, foram realizados quatro experimentos diferentes, com movimentos concentrados no centro da área de visão da câmera. Nos primeiros dois testes, cujos resultados são mostrados na Figura 35, a velocidade do veículo



(a) (b)

Figura 33: Estimativas de α_2 e α_3 realizadas no centro do campo de visão da câmera.

Tabela 8: Erro quadrático médio das medidas de α_2 e α_3 em manobras em áreas com e sem efeito da distorção produzida pela lente.

Erro Quadrático Médio			
	Sem Distorção		*Com Distorção*
	(a)	(b)	
α_2	$0,75$	$1,58$	$1,43$
$\alpha_{2_{Kalman}}$	$0,62$	$1,45$	$1,16$
α_3	$1,24$	$0,51$	$1,59$
$\alpha_{3_{Kalman}}$	$1,60$	$0,61$	$1,65$

era de aproximadamente $0,25$ m/s. No segundo teste, a velocidade do primeiro experimento, Figura 36(a), era de aproximadamente $0,50$ m/s, e a velocidade do segundo, Figura 36(b), de aproximadamente $0,75$ m/s. A Tabela 9 contém os erros quadráticos médio desses experimentos. Os dados indicam que, para uma velocidade alta, o erro quadrático médio aumenta. Contudo, é importante relembrar novamente que manobras à ré com veículos multi-articulados reais, em geral, são realizadas a baixa velocidade. Outro ponto importante é o desempenho do Filtro de Kalman sobre as estimativas, que em geral melhoraram após a aplicação do filtro.

Como testes finais, o veículo executou algumas manobras lentas à ré. Os resultados são mostrados na Figura 37 e os erros quadráticos médio, discriminados na Tabela 10. Nesses últimos testes, também foi possível observar que o Filtro de Kalman, em geral, melhora a estimativa, diminuindo o erro quadrático médio.

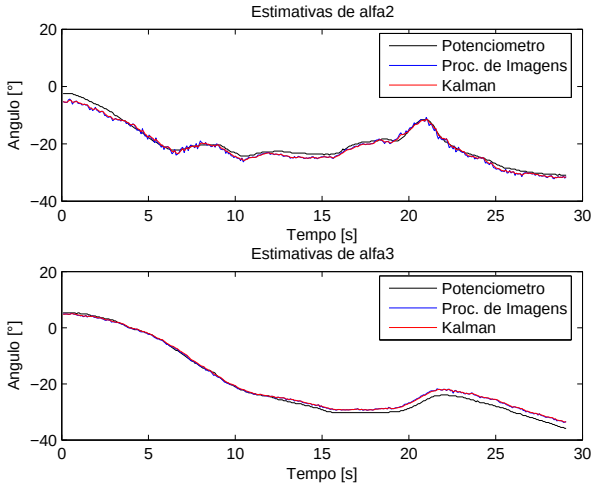
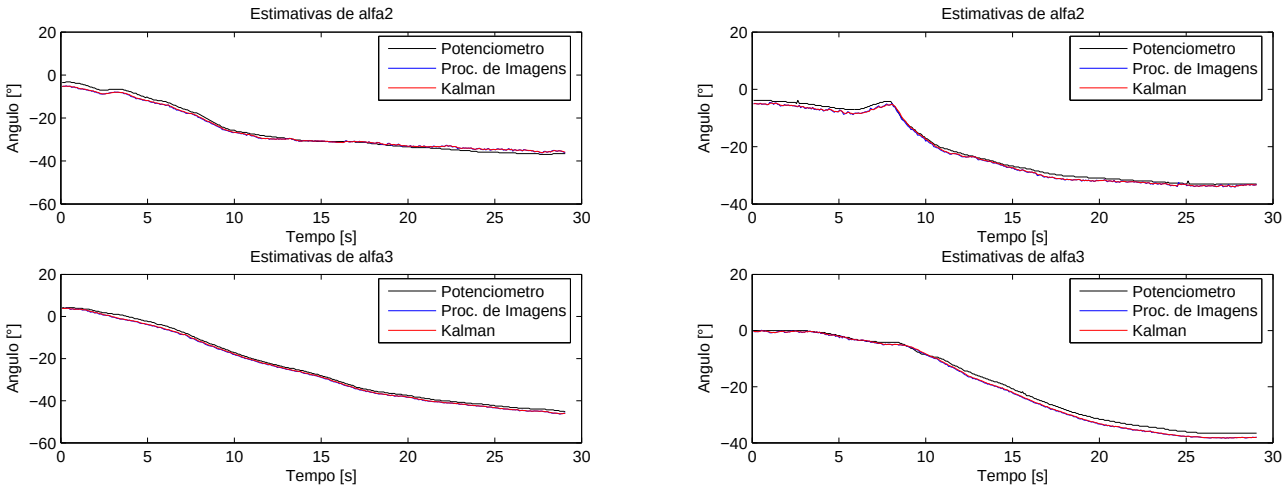


Figura 34: Estimativas de α_2 e α_3 realizadas na borda do campo de visão da câmera.

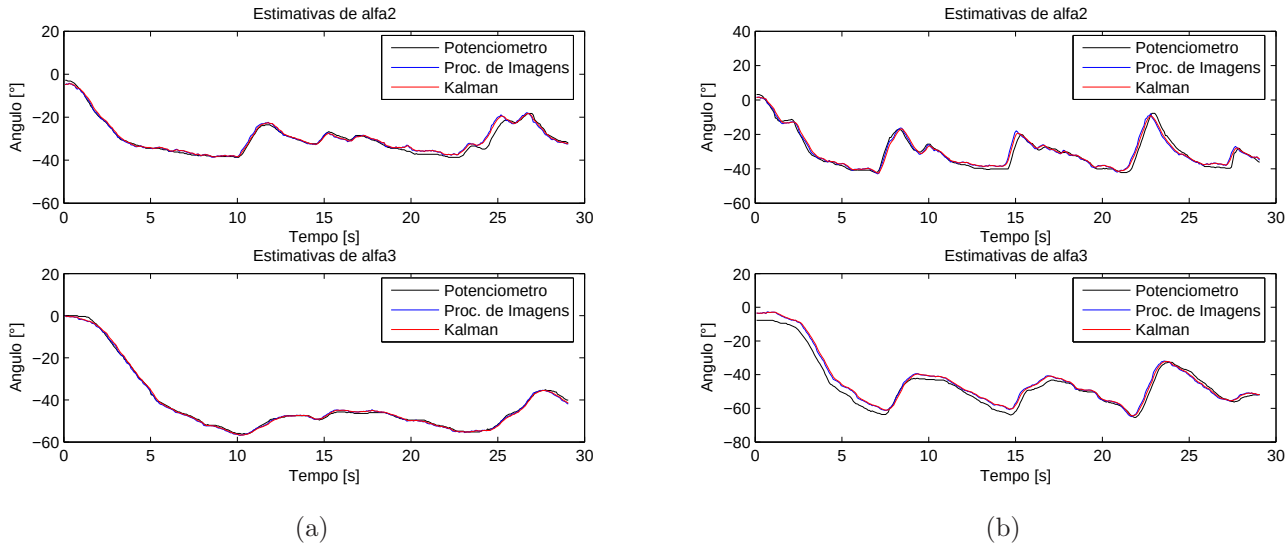


(a) (b)

Figura 35: Influência da velocidade da composição nas estimativas dos ângulos α_2 e α_3: manobra lenta.

Tabela 9: Erro quadrático médio das medidas de α_2 e α_3 em manobras lentas e rápida.

Erro Quadrático Médio				
	Manobras Lentas		*Manobra Rápida*	
	(a)	(b)	(c)	(d)
α_2	$1,48$	$0,95$	$1,95$	$5,17$
$\alpha_{2_{Kalman}}$	$1,30$	$0,85$	$1,46$	$3,85$
α_3	$1,16$	$1,69$	$0,46$	$9,13$
$\alpha_{3_{Kalman}}$	$0,89$	$1,46$	$0,42$	$9,74$



(a) (b)

Figura 36: Influência da velocidade da composição nas estimativas dos ângulos α_2 e α_3: (a) velocidade alta e (b) velocidade muito alta.

Tabela 10: Erro quadrático médio das medidas de α_2 e α_3 em manobras lentas à ré.

Erro Quadrático Médio			
	Manobra 1	Manobra 2	Manobra 3
α_2	$0,95$	$1,50$	$1,10$
$\alpha_{2_{Kalman}}$	$0,85$	$0,67$	$0,88$
α_3	$0,53$	$0,62$	$0,09$
$\alpha_{3_{Kalman}}$	$0,53$	$0,66$	$0,16$

Assim como nos experimentos anteriores, também foi possível observar que as estimativas dos ângulos de configuração obtidas a partir do sistema de processamento de imagens seguem os valores medidos pelos potenciômetros de precisão, apresentando, em geral, erro quadrático médio baixo. Contudo, pode-se notar que os ruídos presentes nas estimativas diminuíram consideravelmente em comparação ao sistema anterior. Além disso, observou-se que as estimativas de α_2 não apresentaram tantos ruídos como percebidos na seção 3.2.4. Essa informação indica que a cor do cartão influencia mais na qualidade da estimativa que o seu tamanho.

3.3 Conclusão

Os resultados obtidos nesses experimentos mostraram que as estimativas dos ângulos de configuração utilizando o processamento de imagens apresentaram erro quadrático

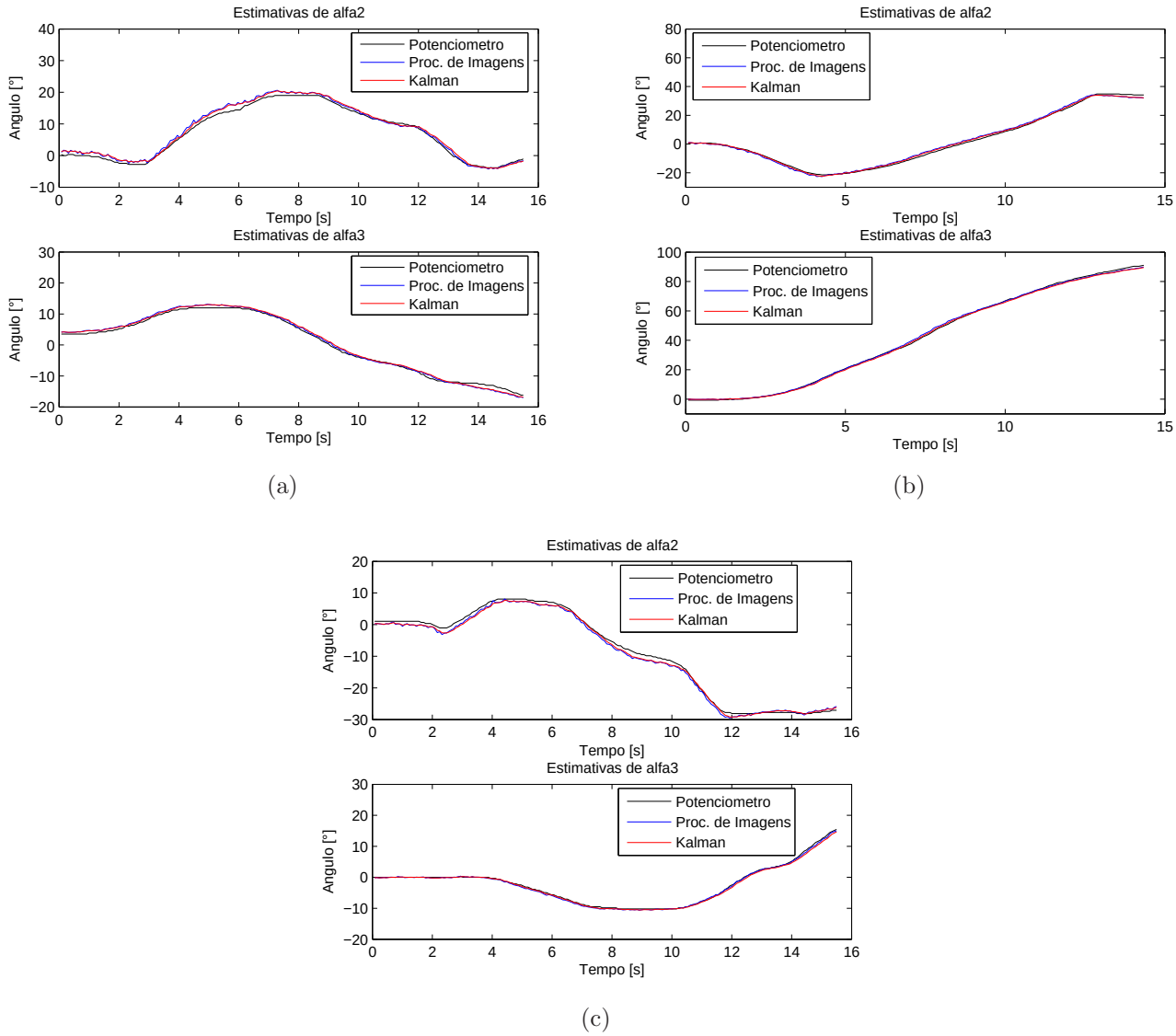


Figura 37: Estimativas dos ângulos α_2 e α_3 em manobras à ré: (a) primeira e (b) segunda manobra.

médio baixo, principalmente em movimentos lentos, quando comparados com os valores obtidos com potenciômetros de precisão. O uso do CAMSHIFT como algoritmo de segmentação melhorou consideravelmente os resultados obtidos.

Manobras com controle em malha fechada foram bem sucedidas em trabalhos anteriores, mesmo utilizando potenciômetros comuns, cujas medidas são mais incertas que as dos potenciômetros utilizados neste experimento [22]. Além disso, a implementação de filtros para correção de erros ocasionados por possíveis ruídos na captura da imagem aumentou a qualidade das medidas de configuração por processamento de imagens.

Também foi observado que a distorção provocada pelo uso de lentes grande-angulares não afeta significativamente a estimativa dos ângulos de configuração, sendo que o erro

ocasionado por esse efeito é baixo. Tal erro não compensaria o custo computacional inerente a uma possível correção nas imagens antes de seu processamento. Outro aspecto importante, observado durante os experimentos, é o aumento significativo do erro em manobras rápidas. No entanto, em aplicações reais, manobras à ré são realizadas em baixa velocidade, possibilitando o uso do processamento de imagens. Sendo assim, o processamento de imagens mostra-se uma ferramenta viável para leitura da configuração da composição.

4 Resultados Experimentais

"A única possibilidade de descobrir os limites do possível é aventurar-se um pouco mais além deles, até o impossível".

Arthur C. Clarke

4.1 Introdução

Para verificar o uso do sistema de visão computacional como realimentação do controlador *fuzzy* proposto, foram realizados experimentos com o protótipo de veículo multi-articulado dotado de recursos embarcados. O protótipo foi implementado sobre a estrutura mecânica e eletro-mecânica da réplica do caminhão *Globe Liner* em escala 1/14, produzido pela empresa Tamiya® [54]. As dimensões do veículo são mostradas na Tabela 1 e o seu comprimento é de $1,73\ m$, quando montado e alinhado. A réplica pode ser vista em mais detalhes na Figura 38.

A Figura 38 também mostra o sistema embarcado do veículo contendo um microcontrolador Atmega32 da Atmel Products [55], com 32 KB de memória *flash*, 2 KB de memória RAM, 40 pinos e com até 16 MHz de freqüência. Por sua velocidade e memória maiores que de outros microcontroladores similares, baixo consumo de energia e programador gratuito, o Atmega32 é uma ferramenta adequada para aplicações de robótica móvel. A finalidade do microcontrolador é receber os comandos de velocidade linear e de direção do computador remoto e aplicar sinais de PWM ao motor de passo e ao ESC - *Eletronic Speed Control*, fabricados pela Futaba® e responsáveis pela direção e velocidade, respectivamente. O controle mais refinado da velocidade também conta com o auxílio de um condificador óptico conectado a uma das portas A/D do Atmega32. Além disso, se solicitado, o microcontrolador é capaz de ler os valores dos potenciômetros acoplados aos engates do protótipo. A Figura 39 exibe o sistema embarcado juntamente com seus principais componentes.

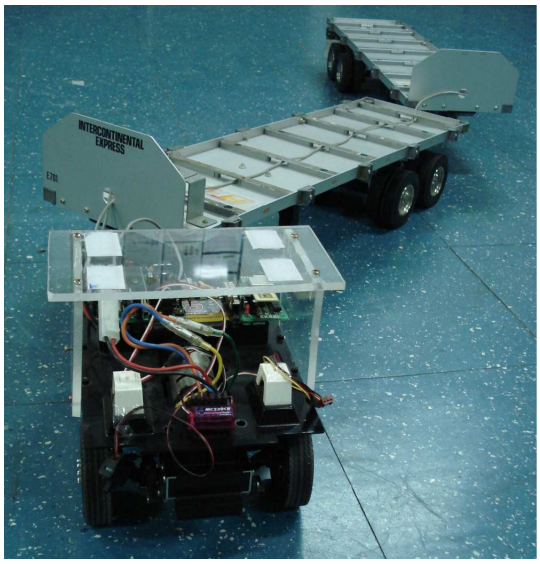


Figura 38: Réplica de caminhão multi-articulado *Globe Liner* com dois semi-reboques.

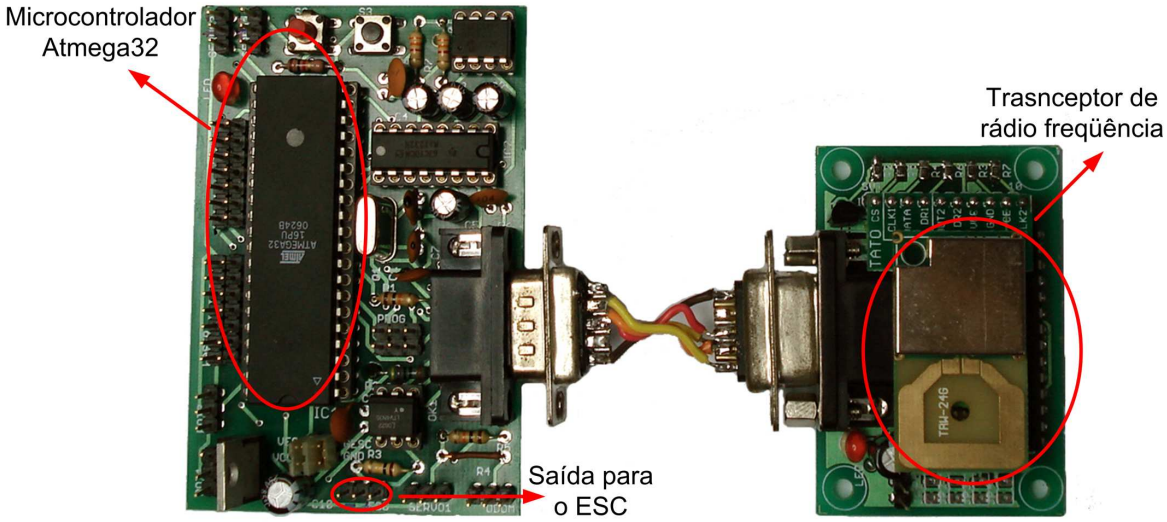


Figura 39: Sistema embarcado no veículo multi-articulado para leitura dos potenciômetros e controle da velocidade e da direção.

O computador remoto utiliza um sistema de comunicação em rádio freqüência (RF) para transmitir e receber dados do microcontrolador. O sistema de comunicação foi implementado com transceptores, modelo TRW-2.4 fabricado pela LaiPac Tech Inc., operando na freqüência de $2,4\ GHz$ e permitindo transferência de dados a uma taxa de até $1\ Mbps$. Foram desenvolvidas duas placas semelhantes: a primeira, mostrada na Figura 39, para ser conectada ao *hardware* embarcado e a segunda, mostrada na Figura 40, conectada ao computador remoto.

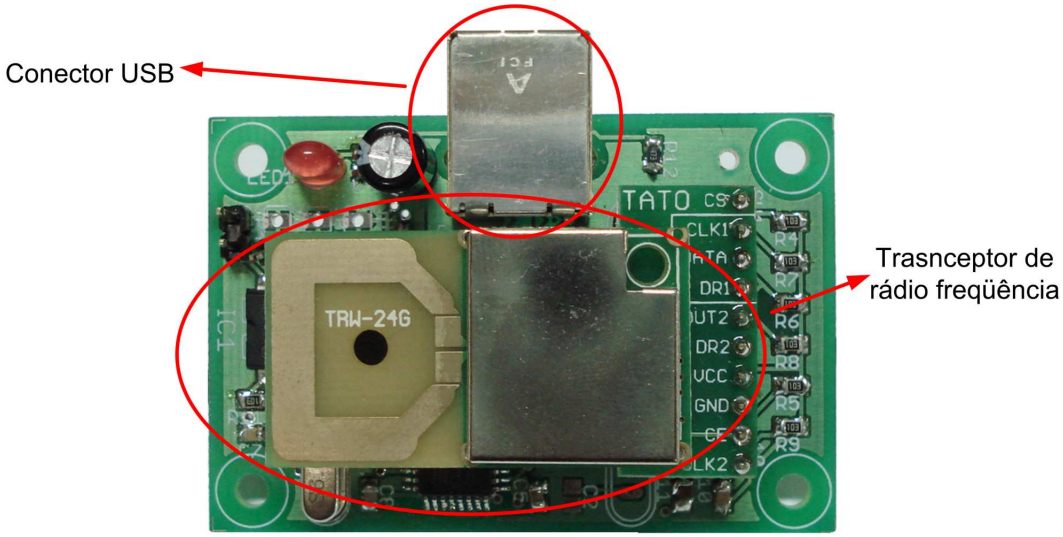


Figura 40: Sistema de comunicação RF com entrada USB para conexão com computador.

O sistema de visão computacional utilizado neste trabalho é composto por uma câmera CCD PointGrey, modelo DragonFly®, com comunicação *firewire* e velocidade de 15 *frames*/s, instalada sobre o plano de trabalho do robô, ou seja, em vista de topo ou *bird's eyes view*, a uma altura de aproximadamente $3,5\ m$. A câmera foi conectada a um computador com processador Intel® Core^{TM} 2 Duo 2,66 GHz e 2 GB de memória RAM, que realiza todo o processamento de imagem, executa o controlador e envia o comando de controle ao protótipo. A área de trabalho do sistema, neste caso, é o campo visual da câmera utilizada, aproximadamente $3,30\ m \times 4,40\ m$. Uma visão geral da configuração do ambiente de experimentos já foi mostrada na Figura 5, mas é mostrada novamente na Figura 41 para melhor ilustrar a execução dos experimentos. O código do sistema implementado para o computador remoto é descrito no Apêndice A.

4.2 Experimentos

Dentre os vários experimentos realizados durante este estudo, nesta seção serão mostrados aqueles que apresentaram resultados mais significativos. Em todas as etapas, foi utilizado o mesmo controlador descrito na seção 2.4, comparando o desempenho deste quando realimentado com visão computacional e quando realimentado pelos valores dos potenciômetros de precisão. Para isso, buscou-se realizar os testes sob as mesmas condições iniciais, incluindo um atraso no sistema utilizando potenciômetro para que se aproximasse aos 77 *ms* de amostragem do sistema por visão. No entanto, é importante

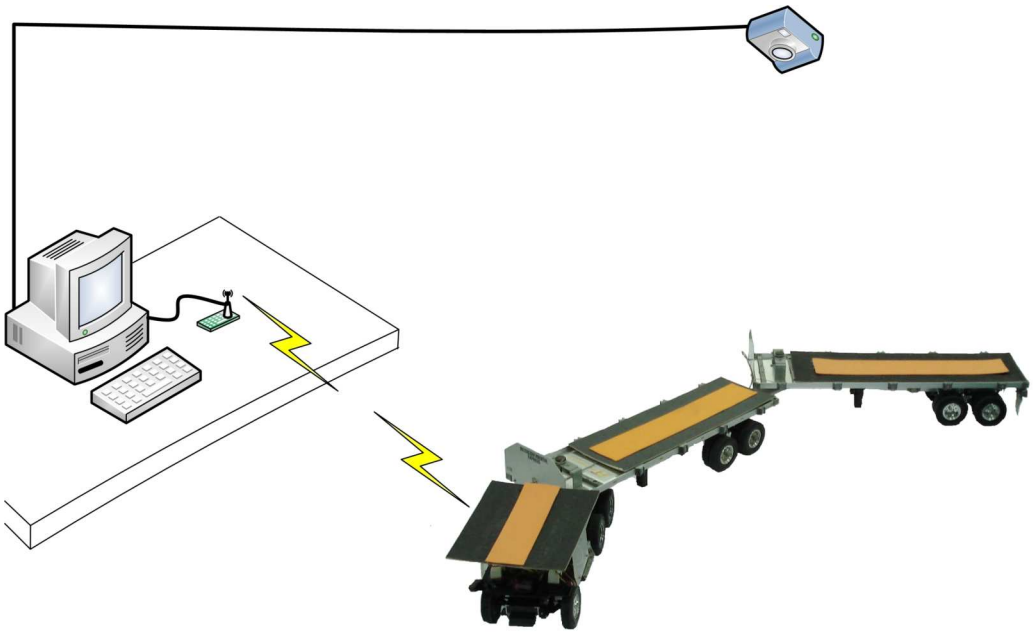


Figura 41: Visão geral do ambiente de experimentos.

ressaltar que o espaço físico do laboratório e a área de visão da câmera são fatores limitantes para os testes, considerando as dimensões do protótipo. Por esse motivo, em alguns testes tiveram duração menor. Todos os testes foram realizados para manobras à ré.

Para todos os experimentos, são mostrados os valores de α_2 e α_3 lidos pelo processamento de imagens ou pelos potenciômetros e a referência dada como entrada ao controlador, Ref_3. A referência para o ângulo α_2 é a saída calculada pelo primeiro controlador do CpPI. Além desse gráfico, também é exibido o erro em α_3 ao longo dos experimentos.

4.2.1 Experimento I

No primeiro teste, foi utilizado apenas um dos semi-reboques do protótipo para averiguar o comportamento do controlador implementado. No experimento realizado com visão, o ângulo inicial da única articulação deste caso foi estimado em $\alpha_2 \approx 16^o$. O ângulo desejado para essa articulação era de 0°, ou seja, o controlador deveria alinhar o veículo durante a manobra. O experimentou durou cerca de 42 s, devido ao limite imposto pelo espaço de visão da câmera. A configuração inicial do veículo é mostrada na Figura 42.

A partir do resultados mostrados na Figura 43, observa-se que o ângulo α_2 aumenta a até aproximadamente 20° antes de diminuir até próximo a 0°.

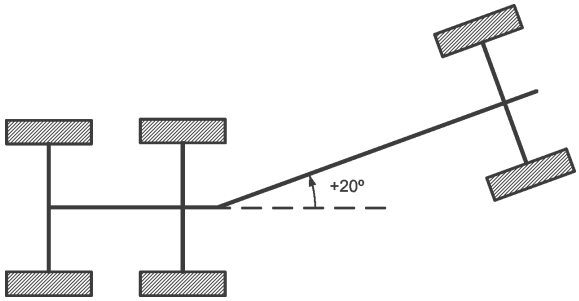
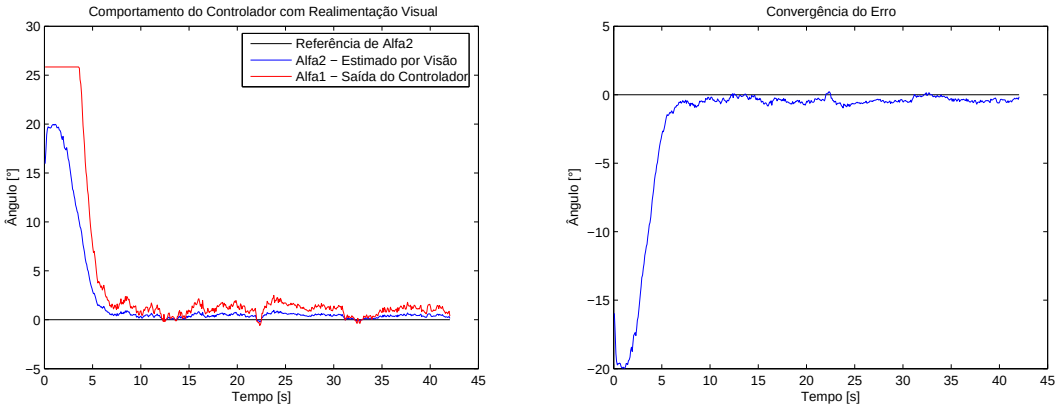


Figura 42: Experimento I - configuração inicial da composição: $\alpha_2 \approx +20^o$.



(a) (b)

Figura 43: Experimento I - visão computacional: (a) comportamento do sistema e (b) erro de α_2.

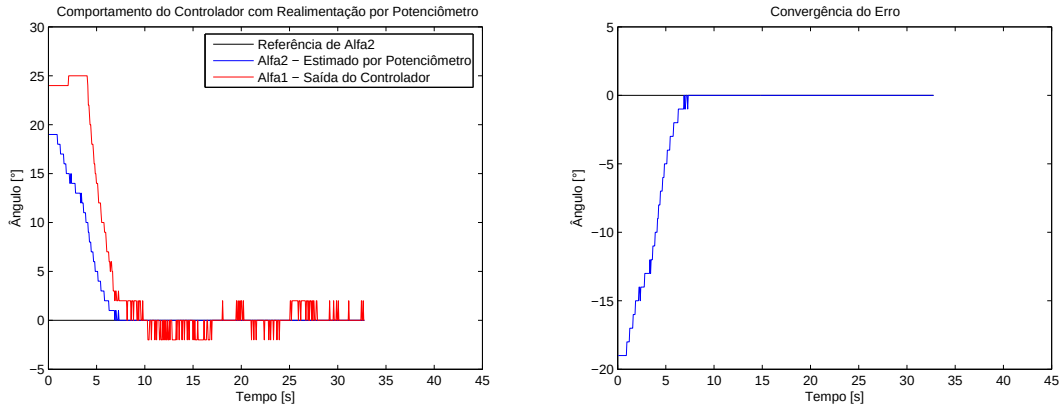
No controle utilizando as leituras do potenciômetro de precisão, o valor inicial de α_2 era de aproximadamente 20°, como mostrado na Figura 44. Ao contrário do experimento anterior, durante a manobra, o valor de α_2 apenas diminui até um limite igual ou próximo a 0°.

É importante observar que o erro, Figura 43(b) e Figura 44(b), converge para 0°, confirmando que o controlador é capaz alinhar o veículo com um semi-reboque, independente do tipo de realimentação, seja visual ou por potenciômetro.

A partir do segundo experimento, foram utilizados os dois semi-reboques do protótipo.

4.2.2 Experimentos II

A segunda fase de experimentos também focou no alinhamento do protótipo a partir de uma condição inicial de $\alpha_3 \neq 0^o$, no entanto, utilizando dois semi-reboques. No primeiro experimento, cujos resultados são mostrados na Figura 46, inicialmente, $\alpha_3 \approx -10^o$ e



(a) (b)

Figura 44: Experimento I - potenciômetro: (a) Comportamento do sistema e (b) erro de α_2.

$\alpha_2 \approx 0^o$, como visto na Figura 45. O experimentou durou cerca de 22 s, tempo também imposto pelo espaço de visão da câmera. É possível observar que α_3 finaliza muito próximo a 0°, oscilando por alguns instantes até alcançar o objetivo.

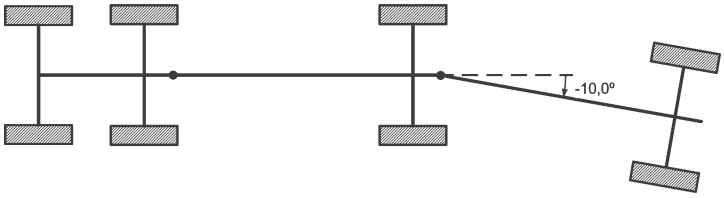
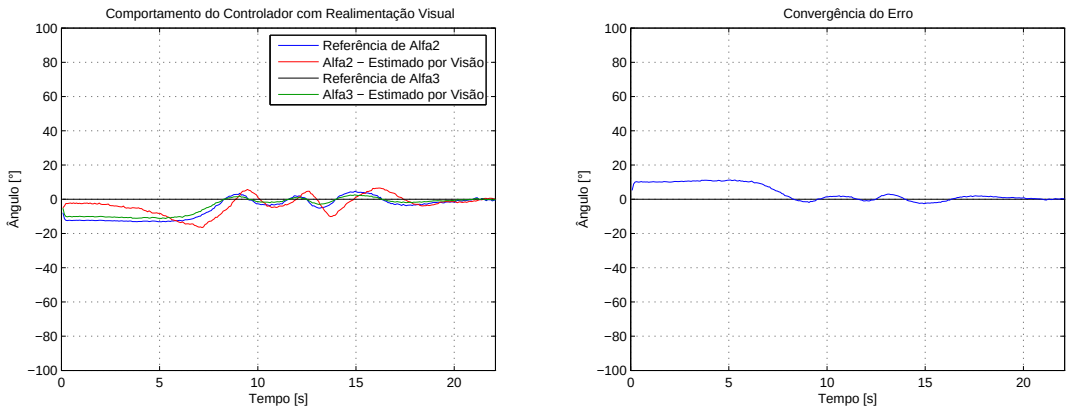


Figura 45: Experimento II - configuração inicial da composição com $\alpha_3 \approx -10^o$.



(a) (b)

Figura 46: Experimento II - visão computacional: (a) α_3 iniciando em -10° e referência nula e (b) erro α_3.

No segundo experimento, também utilizando visão computacional, o valor inicial de

α_3 era de aproximadamente -20°, mostrado na Figura 47. O tempo deste teste foi um pouco menor, cerca de 15 s, devido ao módulo do ângulo inicial maior, o que causou a saída mais rápida do veículo da área de trabalho da câmera. Os resultados deste teste, mostrados na Figura 48, mostram que o controlador foi capaz de alinhar a composição, oscilando um pouco menos que no primeiro caso, mesmo com o ângulo inicial um pouco maior.

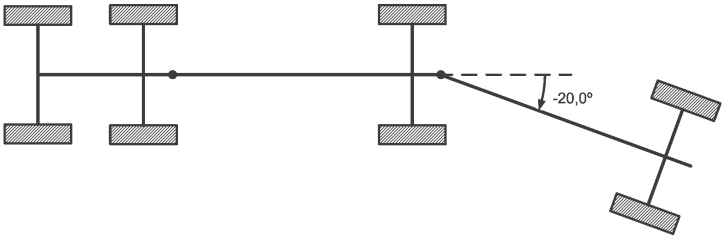
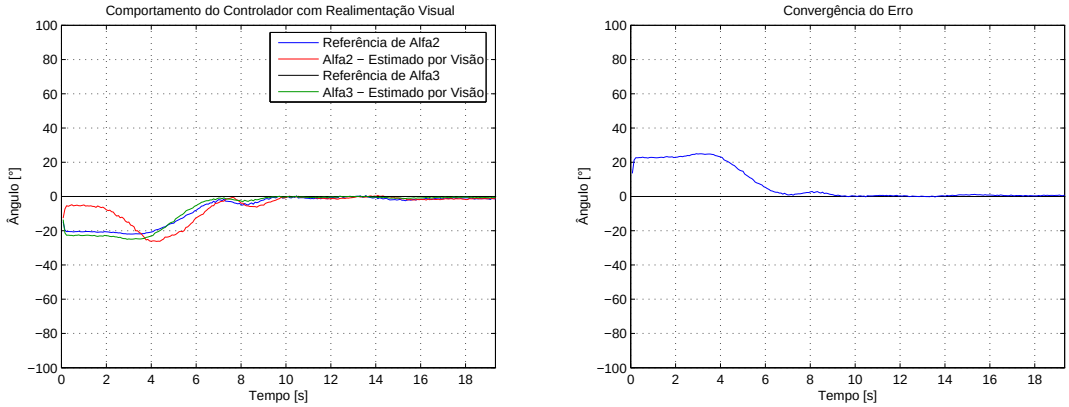


Figura 47: Experimento II - configuração inicial da composição com $\alpha_3 \approx -20^o$.



(a) (b)

Figura 48: Experimento II - visão computacional: (a) α_3 iniciando em -20° e referência nula e (b) erro α_3.

Como base de comparação, esses mesmos testes foram realizados com potenciômetros de precisão. Nesses novos experimentos, o valor inicial de α_3 é de aproximadamente -20° e -30°, como visto nas Figuras 47 e 49. Os resultados são mostrados nas Figura 50 e 51, respectivamente.

Em ambos, α_3 converge para 0°. Na Figura 51, é possível observar que, com um valor inicial de α_3 maior, a amplitude dos ângulos de configuração durante a manobra aumenta consideravelmente antes de estabilizar em 0°, indicando que o ângulo está aproximando-se do limite a partir do qual a manobra não poderia ser realizada.

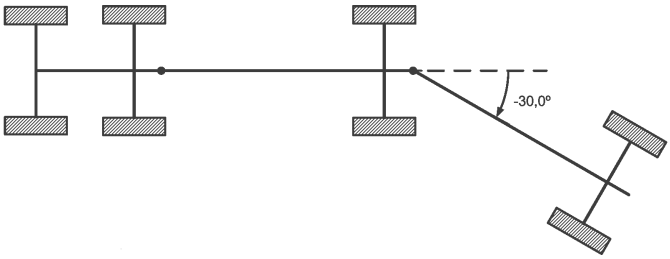
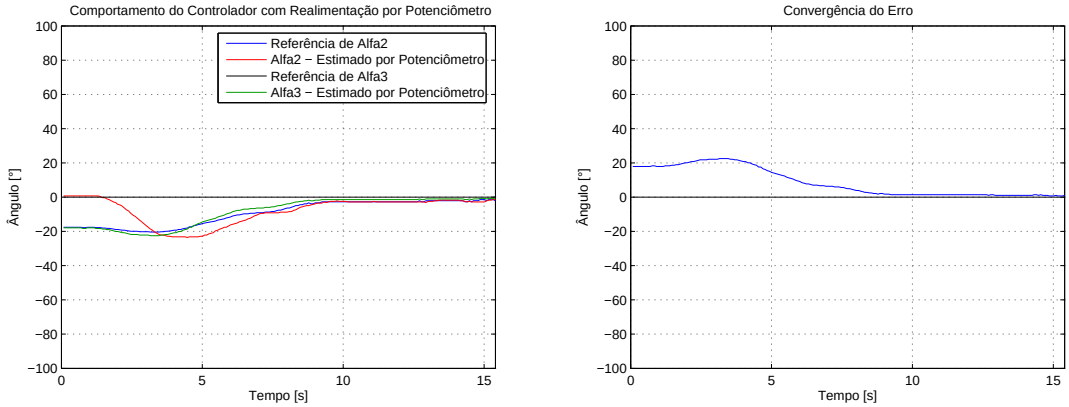
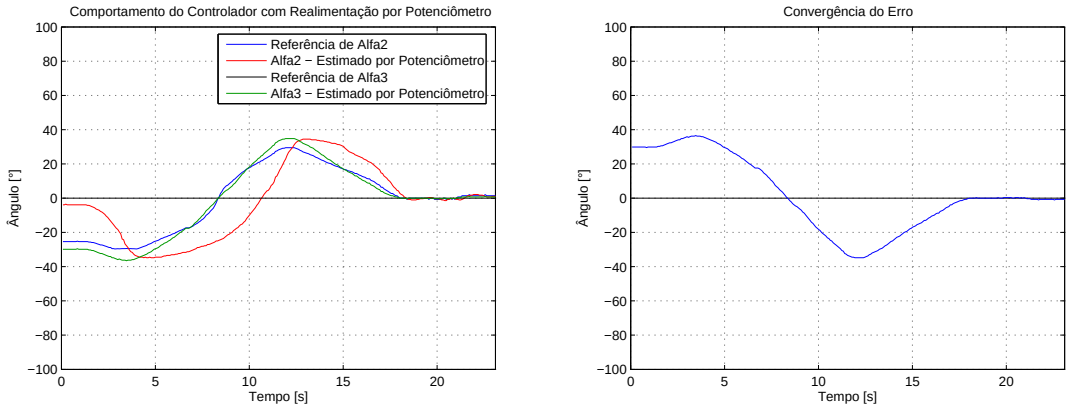


Figura 49: Experimento II - configuração inicial da composição com $\alpha_3 \approx -30^o$.



(a) (b)

Figura 50: Experimento II - potenciômetros: (a) α_3 iniciando em -20° e referência nula e (b) erro α_3.



(a) (b)

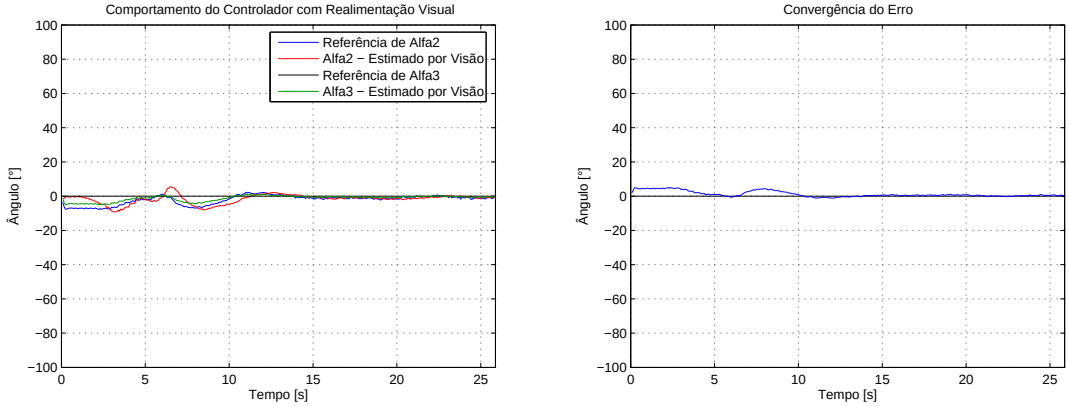
Figura 51: Experimento II - potenciômetros: (a) α_3 iniciando em -30° e referência nula e (b) erro α_3.

4.2.3 Experimento III

O objetivo do terceiro experimento era verificar o comportamento do sistema no seguimento de um caminho retilíneo. Desse modo, foram realizados dois testes, um para

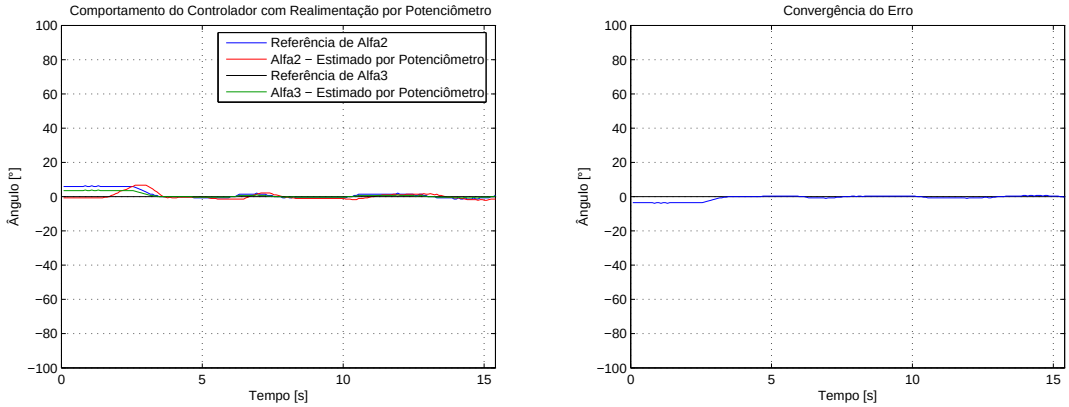
realimentação visual e outro para realimentação com potenciômetros. Em ambos, a configuração inicial do veículo era $\alpha_2 \approx 0^o$ e $\alpha_3 \approx 0^o$. Como referência, foi dado como entrada ao controlador $Ref_3 = 0^o$. O resultado para alimentação visual e com potenciômetros podem ser vistas nas Figuras 52 e 53.

Observa-se nesses testes que o sistema é capaz de manter o veículo alinhado durante a trajetória, com pequenas oscilações iniciais, uma vez que α_2 e α_3 não eram exatamente nulos.



(a) (b)

Figura 52: Experimento III - visão computacional: caminho retilíneo.



(a) (b)

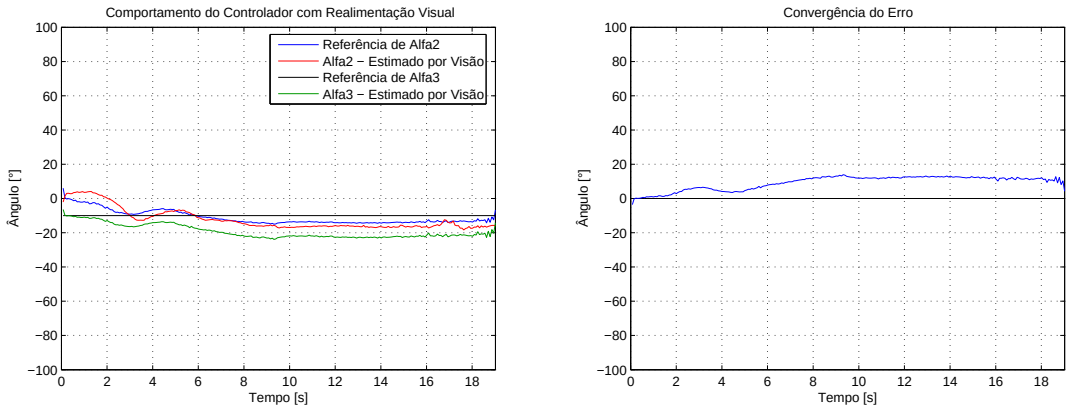
Figura 53: Experimento III - potenciômetros: caminho retilíneo.

4.2.4 Experimento IV

Na quarta etapa de testes, foram realizados experimentos com o veículo inicialmente alinhado e Ref_3 não nulo. Nos dois primeiros testes a referência dada era de -10°. A

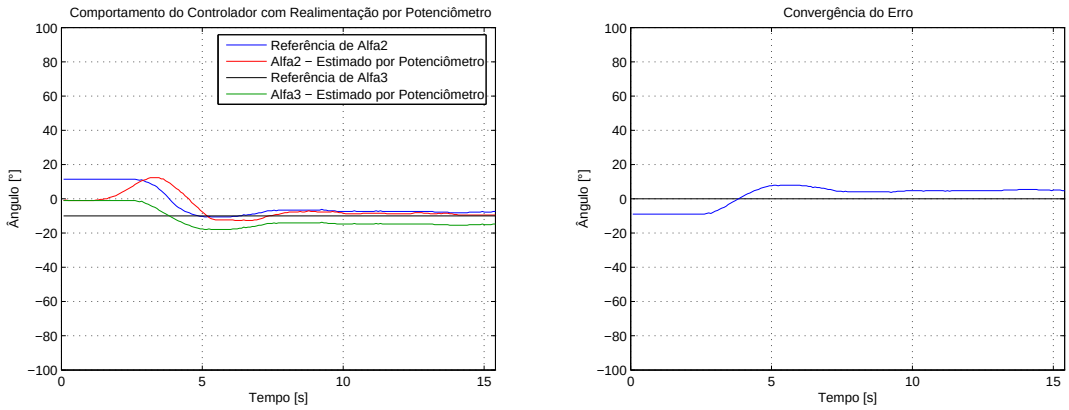
Figura 54 mostra o resultado para o teste com realimentação visual e a Figura 55, com realimentação por potenciômetros de precisão.

Analisando as Figuras 54 e 55, é possível verificar, que ainda que os experimentos tenham sido realizados para um mesmo particionamento *fuzzy* e com a mesma restrição de tempo de amostragem, a realimentação com potenciômetros resultou em duas vezes mais precisão que a realimentação visual. Aparentemente, isso é devido à melhor qualidade de medida da dispositivo. De todo modo, o controle do sistema para valores de referência de α_3 não nulos, tais como com o sistema em giro, demandam mais pesquisa para sua solução.



(a) (b)

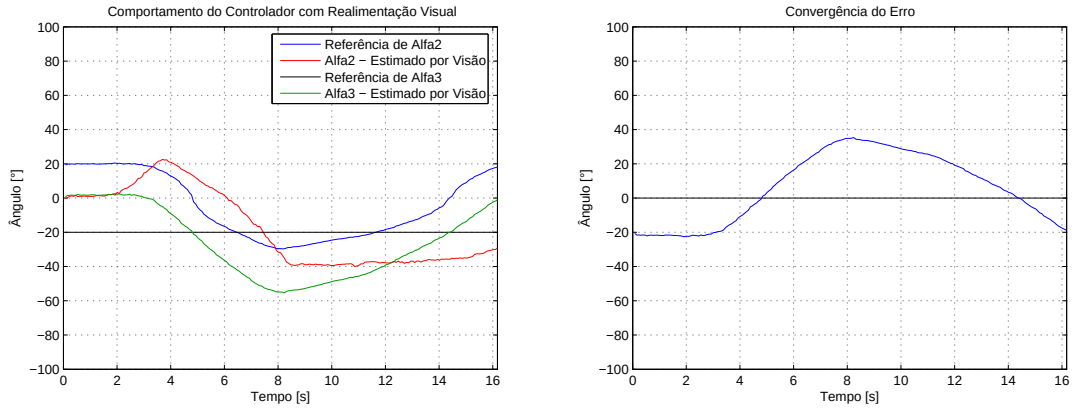
Figura 54: Experimento IV - visão computacional: (a) veículo inicialmente alinhado e referência de α_3 igual a -10° e (b) erro α_3.



(a) (b)

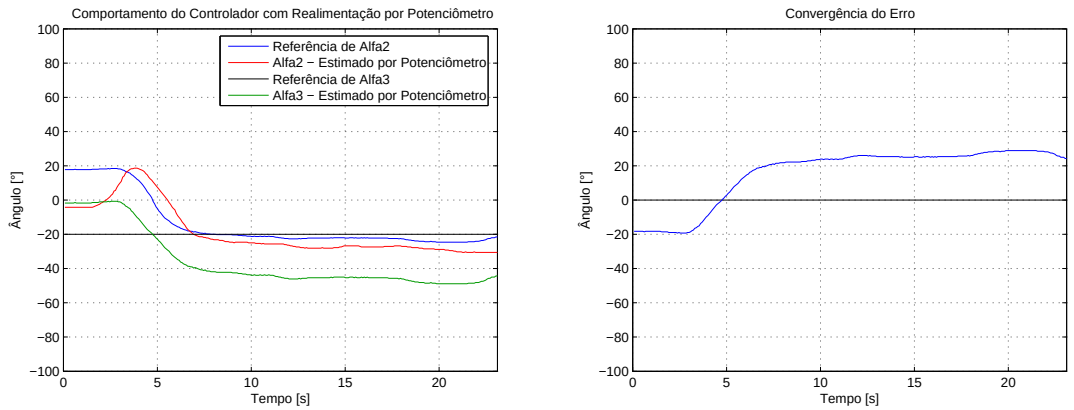
Figura 55: Experimento IV - potenciômetros: (a) veículo inicialmente alinhado e referência de α_3 igual a -10° e (b) erro α_3.

Em seguida, foram realizados dois experimentos semelhantes aos anteriores, no entanto, com $Ref_3 = -20^o$. Os resultados são mostrados nas Figuras 56 e 57.



(a) (b)

Figura 56: Experimento IV - visão computacional: (a) veículo inicialmente alinhado e referência de α_3 igual a -20° e (b) erro α_3



(a) (b)

Figura 57: Experimento IV - potenciômetros: (a) veículo inicialmente alinhado e referência de α_3 igual a -20° e (b) erro α_3.

Para um pedido de variação angular grande, o controlador realimentado com potenciômetro atingiu precisão muito ruim, enquanto que o com realimentação visão entrou em *jacknife*, ou seja, o sistema ficou instável. Uma análise conclusiva sobre esse resultado deverá ser objeto de estudo de trabalhos futuros para controle de ângulos não nulos, ou seja, com o veículo em giro.

4.2.5 Experimento V

Nos últimos testes, foi solicitado ao veículo que realizasse manobras de giro, com ângulo de referência fixo, como mostrado nas figuras abaixo.

Os dois primeiros testes foram realizados para referência de α_3 igual a -20° e com

realimentação visual. O veículo iniciou o movimento com $\alpha_3 \approx -30^o$, como visto na Figura 58. Como pode ser observado nas Figuras 59 e 60, foi verificado que para uma inicialização favorável, ou seja, ângulos de configuração não nulos e com mesmo sinal, o controlador foi capaz de seguir a tranjetória circular, ainda que com erro, diferente da situação comentada no experimento anterior (Figura 56).

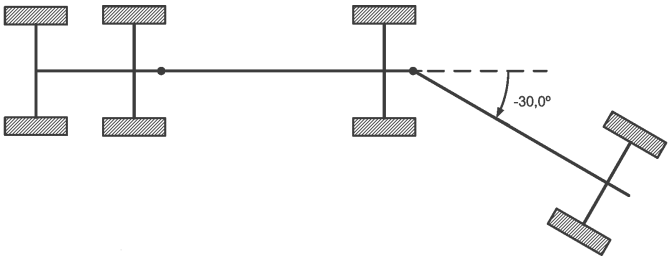
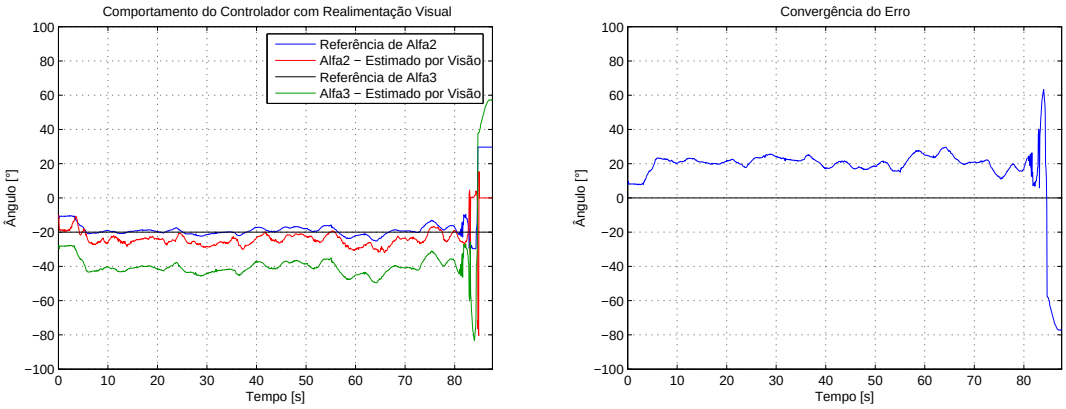


Figura 58: Experimento V - configuração inicial da composição com $\alpha_3 \approx -30^o$.

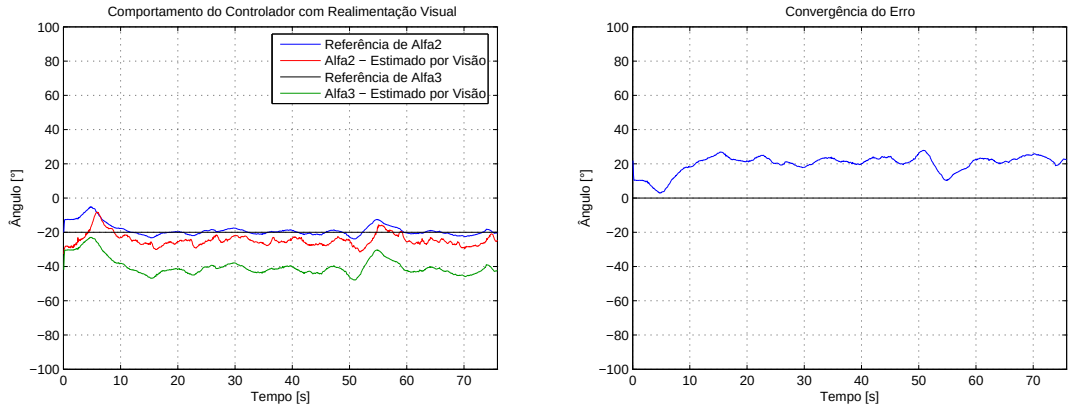


(a) (b)

Figura 59: Experimento V - visão computacional: Trajetória circular com referência de α_3 igual a -20°.

A alteração mostrada no fim do gráfico da Figura 59 foi causada pela saída do veículo da área de visão da câmera. Nesse momento, sem valores de realimentação adequados, o controlador parou de responder satisfatoriamente e o veículo entrou em *jacknife*.

A Figura 62 mostra os resultados do terceiro percurso circular com realimentação visual. Nesse teste, o valor de referência foi de $+10^o$, sendo esse também o valor aproximado da configuração inicial vista na Figura 61. Esse valor de referência gerou um caminho mais aberto e, conseqüentemente, mais propenso à saída do campo de visão da câmera, como ocorreu com o cavalo mecânico em alguns momentos, gerando os picos observados na estimativa de α_2. Apesar dessas saídas, não totais, o controlador conseguiu manter a



(a) (b)

Figura 60: Experimento V - visão computacional: Trajetória circular com referência de α_3 igual a -20°.

trajetória, ainda que com erro, até que o cavalo mecânico saísse completamente do campo de visão da câmera, no fim do experimento.

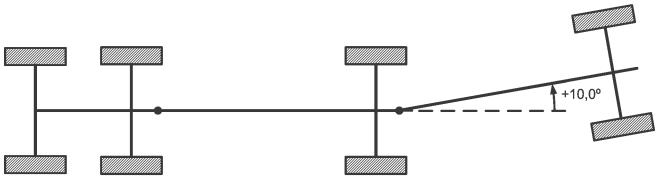
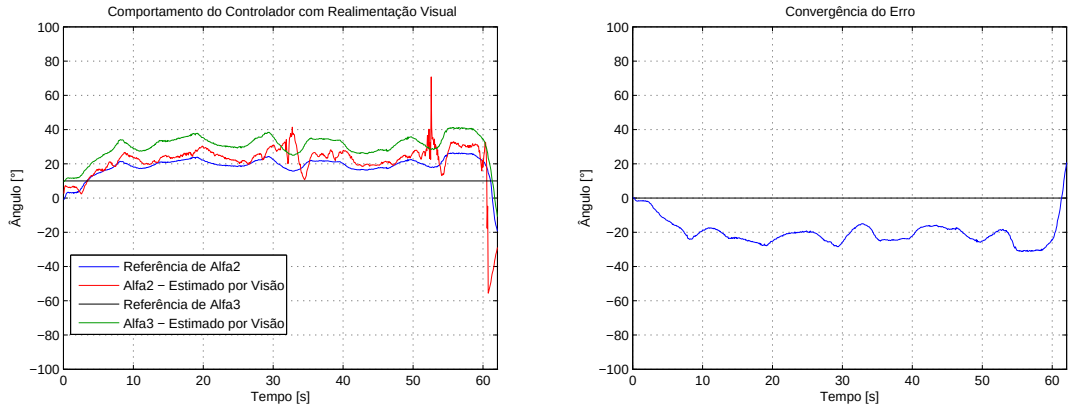


Figura 61: Experimento V - configuração inicial da composição com $\alpha_3 \approx +10^o$.

Os testes seguintes foram realizados com o controlador *fuzzy* realimentado pelos potenciômetros, para dois percursos de referência não nula e constante: a primeira em -20° e a segunda em $+10^o$. A configuração inicial da composição em cada teste pode ser vista respectivamente nas Figuras 63 e 65. Para o primeiro caminho, mostrado na Figura 64, a precisão obtida foi muito parecida com a obtida pela realimentação visual, Figuras 59 e 60. No segundo percurso, por sua vez, a precisão foi um pouco melhor, mas ainda com erro significativo. O resultado desse último teste é mostrado na Figura 66. A alteração percebida ao fim desse experimento deve-se a colisão em um obstáculo, pela falta de mais espaço no laboratório.

4.3 Conclusão

A partir dos resultados dos experimentos, é possível verificar que o controlador com realimentação visual é capaz de seguir referências dadas com precisão pouco inferior ao



(a) (b)

Figura 62: Experimento V - visão computacional: Trajetória circular com referência de α_3 igual a $+10^o$.

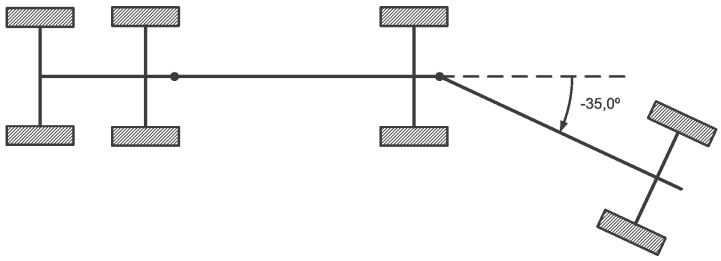
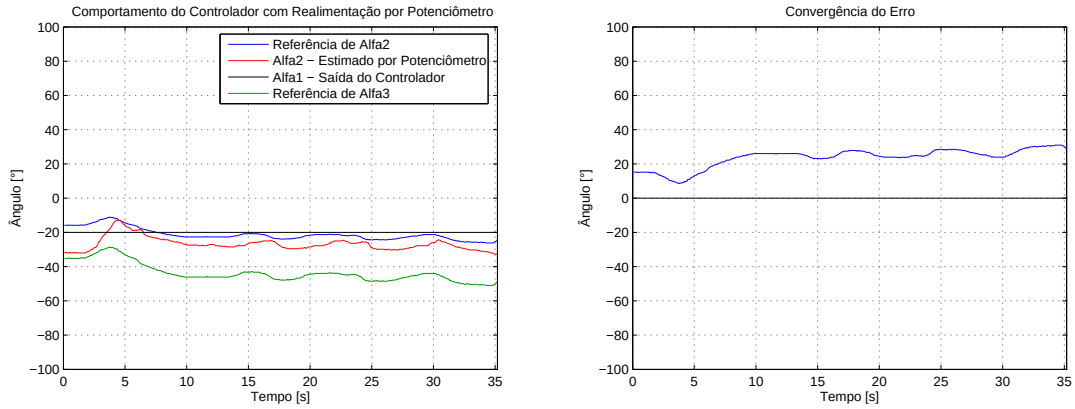


Figura 63: Experimento V - configuração inicial da composição com $\alpha_3 \approx -35^o$.



(a) (b)

Figura 64: Experimento V - potenciômetros: Trajetória circular com referência de α_3 igual a -20°.

sistema com realimentação por potenciômetros, dependendo da configuração de partida da composição. Para o caso de alinhamento, o controlador é capaz de executar manobra satisfatoriamente quaisquer que sejam as configurações iniciais, independente de sua loca-

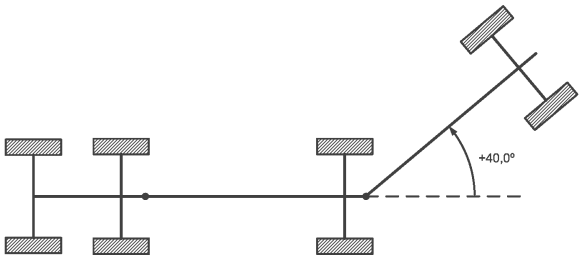
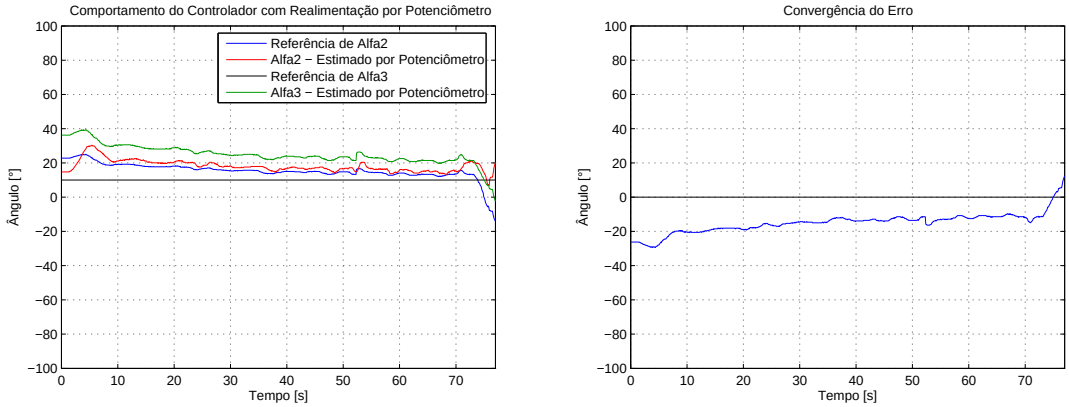


Figura 65: Experimento V - configuração inicial da composição com $\alpha_3 \approx +40^o$.



(a) (b)

Figura 66: Experimento V - potenciômetros: Trajetória circular com referência de α_3 igual a $+10^o$.

lização no campo de visual da câmera. Para o caso de giro ou referências fixas não nulas, a partir de configuração desfavorável, como todos os ângulos de configuração em 0°, seu comportamento não foi apropriado. No entanto, é importante ressaltar que nesses casos, o controlador com realimentação por potenciômetros também não apresentou um desempenho adequado, o que parece indicar a necessidade de ajustes no próprio controlador.

5 Conclusões

Valeu a pena? Tudo vale a pena
Se a alma não é pequena.
Quem quere passar além do Bojador
Tem que passar além da dor.

Fernando Pessoa

O objetivo proposto neste trabalho, de desenvolver um sistema de controle *fuzzy* baseado em visão computacional para robôs móveis ou veículos multi-articulados, constituía um problema em aberto na literatura. Desse modo, foi abordado um problema sem garantia inicial de obter êxito em sua realização. Sendo assim, os resultados obtidos podem ser considerados como relevantes mesmo em face às condições de contorno, restrições do sistema e simplificações adotadas, tais como uso de um protótipo e não uma aplicação real, segmentação por cor e visão de topo.

Os resultados dos experimentos de comparação entre estimativas de ângulos por visão computacional e potenciômetros mostraram que as estimativas do primeiro caso apresentaram erro quadrático médio baixo, principalmente em movimentos lentos, quando comparados com os valores obtidos com pela segunda ferramenta. O uso do CAMSHIFT como algoritmo de segmentação melhorou consideravelmente os resultados obtidos.

Também foi observado que a distorção provocada pelo uso de lentes grande-angulares não afetou significativamente a estimativa dos ângulos de configuração, sendo que o erro ocasionado por esse efeito foi baixo, o que compensaria o custo computacional inerente a uma possível correção nas imagens antes de seu processamento. Outro aspecto importante observado foi o aumento significativo do erro em manobras rápidas. No entanto, em aplicações reais, manobras à ré são realizadas em baixa velocidade, possibilitando o uso do processamento de imagens. Sendo assim, o processamento de imagens mostra-se uma ferramenta viável para leitura da configuração da composição.

Quanto ao seguimento de trajetórias ou giro, a partir de uma condição inicial desfavorável, como a alinhada, pôde ser observado que o controlador *fuzzy* não funcionou como esperado. No entanto, nesse caso, o controlador com realimentação por potenciômetros também não apresentou desempenho adequado. Tal fato indica a necessidade de pesquisas adicionais em relação ao controlador *fuzzy* utilizado.

Finalmente, foi verificado que o sistema de controle funcionou satisfatoriamente com a realimentação visual para o alinhamento do veículo com qualquer configuração inicial, no centro ou na borda do campo de visão da câmera.

Como resultado desta dissertação, foram apresentados dois artigos em congressos. O primeiro, "Análise da Estimação de Configuração de Robôs Móveis ou Veículos Multi-Articulados por Meio de Visão Computacional de Topo: Efeito da Velocidade e da Distorção Óptica" [56], apresentado na VIII Conferência Internacional de Aplicações Internacionais e o outro, "Análise da Robustez da Visão Computacional de Topo no Controle de Movimentos de Robôs Móveis ou Veículos Multi-Articulados" [57], apresentado no XVII Congresso Brasileiro de Automática.

5.1 Trabalhos Futuros

De acordo com os resultados encontrados ao longo desta dissertação e das conclusões parciais, podem ser sugeridos os seguintes trabalhos futuros para melhoria e continuidade desta pesquisa:

- Detecção dos ângulos por um tipo de segmentação mais sofisticado, tal como a geométrica, tendo em vista seu uso em aplicações reais.
- Ampliação da análise das soluções baseadas em segmentação de cor, como a utilizada neste trabalho, para viabilizar seu uso em aplicações reais a partir de finas faixas coloridas e laterais, substituindo cartões coloridos sobre o veículo.
- Mudança de perspectiva da câmera para configurações laterais, que seriam mais adequadas para uso com caminhões em escala real.
- Implementação de um sistema de visão que trabalhe com mais de uma câmera, utilizando técnicas de mosáico de imagens, de modo a aumentar a área de manobra do protótipo, ou seja, o campo visual do sistema.

- Comparação do desempenho de outros controladores *fuzzy* com granularidade diversas, com realimentação visual e por potenciômetros, em situações de giro ou no seguimento de trajetórias de ordem superior. Também seria interessante a análise desses controladores com diferentes alternativas de particionamento, inclusive dinâmicas.

- Desenvolvimento de um controlador *fuzzy* global para substituição do CpPI, utilizado neste trabalho. Além disso, também seria necessário uma análise comparativa entre o controlador global e o CpPI para as diversas situações nas quais o último apresentou fraco desempenho.

- Comparação entre o controlador *fuzzy* e outros controladores não lineares, tais como os neurais e os por desacoplamento via modelo, principalmente em situações de giro ou seguimento de trajetórias.

- Inclusão das análises de desempenho de diversos controladores com realimentação por potenciômetro, sem a restrição de tempo de 77 ms na amostragem.

Referências

[1] BROWN, N. H.; WOOD, H. C.; WILSON, J. N. Image analysis for vision-based agricultural vehicle guidance. In: SPIE. *Proceedings of SPIE Optics in Agriculture*. Boston, MA, USA: SPIE, 1991. (Presented at the Society of Photo-Optical Instrumentation Engineers (SPIE) Conference, v. 1379), p. 54–68.

[2] DAXWANGER, W. A.; SCHMIDT, G. K. Skill-based visual parking control using neural and fuzzy networks. In: *Proceedings of IEEE International Conference on Systems, Man and Cybernetics*. Vancouver, BC, Canada: IEEE, 1995. v. 2, p. 1659–1664.

[3] XU, J.; CHEN, G.; XIE, M. Vision-guided automatic parking for smart car. *IEEE Intelligent Vehicles Symposium*, IV, p. 725–730, Octuber 2000.

[4] LI, T.-H. S.; CHANG, S.-J. Autonomous fuzzy parking control of car-like mobile robot. *IEEE Transactions on System, Man and Cybernetics - Part A: Systems and Humans*, v. 33, n. 4, p. 451–465, Julho 2003.

[5] NGUYEN, T. M.; SINKO, J. W.; GALIJAN, R. C. Using differential carrier phase gps to control automated vehicles. *IEEE Transactions on System, Man and Cybernetics - Part A: Systems and Humans*, Sacramento, CA, USA, v. 1, n. 4, p. 493–496, Agosto 1997.

[6] JUNG, C. R. et al. Computação embarcada: Projeto e implementação de veículos autônomos inteligentes. In: *Anais do XXIV Congresso da Sociedade Brasileira de Computação. Mini-Curso: Jornada de Atualização em Informática*. São Leopoldo, RS: SBC, 2005. v. 1, p. 1358–1406.

[7] KULITZ, H. R. *Modelagem e Controle Fuzzy de Robôs e Veículos Multi-Articulados.* Tese (Doutorado) — Programa de Pós-Graduação em Engenharia Elétrica, Universidade Federal de Espírito Santo, Vitória, 2003.

[8] PINHEIRO, M. P. L. *Simulação do Movimento de Robôs Móveis Multi-Articulados e Obtenção de um Preditor Fuzzy. 130f.* Dissertação (Mestrado) — Programa de Pós-Graduação em Engenharia Mecânica, Universidade Federal do Espírito Santo, Vitória, 2004.

[9] BOUTELDJA, M. et al. Prediction and detection of jackknifing problems for tractor semi-trailer. In: *Proceedings of IEEE Conference on Vehicle Power and Propulsion Conference*. Windsor, UK: IEEE, 2006. v. 1, p. 1–6.

[10] WIDROW, B. et al. Adaptive inverse control based on nonlinear adaptive filtering. In: *Proceedings of 5th IFAC Workshop on Algorithms and Architectures for Real-Time Control (AARTC'98)*. Cancun, Mexico: Elsevier Science Ltd, Oxford, UK, 1998.

[11] KATO, J. M. *Cenários Estratégicos para a Indústria de Transportes Rodoviários de Cargas no Brasil. 167f.* Tese (Doutorado) — Programa de Pós-Graduação em Engenharia de Produção, Universidade Federal de Santa Catarina, Florianópolis, 2005.

[12] WIDMER, J. A. Compatibilidade de tráfego de bitrens de 25m com a infra-estrutura viária brasileira. *2º Colloquium Internacional de Suspensões*, 2002.

[13] MERCEDES-BENZ. *Mercedes-Benz apresenta na Fenatran tecnologias e soluções para várias aplicações de transporte.* Disponível em: <http://www.daimlerchrysler.com.br/noticias/Outubro/Fenatran07_Resumo/popexpande.htm>. Acesso em: 15 Mar 2008.

[14] SILVA, E. B. *Modelagem e Controle Neural Inverso dos Movimentos de um Veículo Articulado Complexo.* Dissertação (Mestrado) — Programa de Pós-Graduação em Engenharia Elétrica, Universidade Federal do Espírito Santo, Vitória, 2003.

[15] KIM, D.-H.; OH, J.-H. Experiments of backward tracking control for trailer system. In: *Proceedings of IEEE International Conference on Robotics and Automation.* Detroit, MI: IEEE, 1999. v. 1, p. 19–22.

[16] NAKAMURA, Y. et al. Design of steering mechanism and control of nonholonomic trailer systems. In: *Proceedings of IEEE International Conference on Robotics and Automation.* San Francisco, CA, USA: IEEE, 2000. p. 247–254.

[17] DIVELBISS, A.; WEN, J. Nonholonomic path planning with inequality constraints. In: *Proceedings of IEEE International Conference on Robotics and Automation.* San Diego, CA, USA: IEEE, 1994. v. 1, p. 52–57.

[18] LAMIRAUX, F.; LAUMOND, J. P. A practical approach to feedback control for a mobile robot with trailer. In: *Proceedings of IEEE International Conference on Robotics and Automation.* Leuven, Belgium: IEEE, 1998. v. 4, p. 3291–3296.

[19] SAMPEI, M. et al. Arbitrary path tracking control of articulated vehicles usingnonlinear control theory. *IEEE Transactions on Control Systems Technology*, IEEE, v. 3, n. 1, p. 125–131, Mar 1995.

[20] ALTAFINI, C.; SPERANZON, A.; WAHLBERH, B. A feedback control scheme for reversing a truck and trailer vehicle. *IEEE Transactions on Robotics and Automation*, v. 17, n. 6, p. 915–922, Dec 2001.

[21] NGUYEN, N.; WIDROW, B. The truck backer-upper: An example of self learning in neural networks. In: *Proceedings of the International Joint Conference on Neural Networks.* Piscataway, NU: IEEE Press, 1989. v. 2, p. 357–363.

[22] FERREIRA, E. P.; LAMEGO, M. M.; WIDROW, B. Neurointerfaces for semi-autonomous object moving systems. In: *Proceedings of the 14th World IFAC Congress.* Beijing, China: Elsevier Science Ltd, Oxford, UK, 1999. K, p. 155–160.

[23] PARRA-LOERA, R.; CORELIS, D. J. Expert system controller for backing-up a truck-trailer system in a constrained space. In: *Proceedings of the 37th Midwest Symposium on Circuits and Systems.* Lafayette, LA, USA: IEEE Press, 1994. v. 2, p. 1357–1361.

[24] YI, J.; YUBAZAKI, N.; HIROTA, K. Backing up control of truck-trailer system. In: *Proceedings of the 10th IEEE International Conference on Fuzzy Systems*. Melbourne, Australia: IEEE Press, 2001. v. 1, p. 489–492.

[25] HALGAMUGE, S. K.; RUNKLER, T. A.; GLESNER, M. A hierarchical hybrid fuzzy controller for real-time reverse driving support of vehicles with long trailers. In: *Proceedings of the Third IEEE Conference on Fuzzy Systems, 1994. IEEE World Congress on Computational Intelligence*. Orlando, FL, USA: IEEE, 1994. v. 2, p. 1207–1210.

[26] KONG, S.-G.; KOSKO, B. Adaptive fuzzy systems for backing up a truck-and-trailer. *IEEE Transactions on Neural Networks*, v. 3, n. 2, p. 211–223, Mar 1992.

[27] PAVIM, A. X.; ROLOFF, M. L. *Curso de Processamento e Análise de Imagens*. Florianópolis, SC, BR: [s.n.], Out 2005. Disponível em: <s2i.das.ufsc.br/cursos/eneca05/curso-proc-imagem-eneca.pdf>. Acesso em: 11 Jun 2008.

[28] JAIN, R.; KASTURI, R.; SCHUNCK, B. G. *Machine Vision*. 1st. ed. New York, NY, USA: McGraw Hill, 1995.

[29] ERHARDT-FERRON, A. *Theory and Applications of Digital Image Processing*. 1st. ed. New York, NY, USA: University of Applied Sciences Offenburg, 2000.

[30] GONZALEZ, R. C.; WOODS, R. E. *Digital Image Processing*. 2nd. ed. Upper Saddle River, NJ, USA: Prentice Hall, 2002.

[31] ZADEH, L. A. Outline of a new approach to the analysis of complex system and decision processes. *IEEE Transactions on Systems, Man, and Cybernetics*, v. 3, n. 1, p. 28–44, 1973.

[32] ZADEH, L. A. Fuzzy sets. *Information and Control*, v. 8, n. 1, p. 338–353, 1965.

[33] SANDRI, S.; CORREA, C. Lógica nebulosa. In: *Anais da V Escola de Redes Neurais*. São José dos Campos, SP: ITA, 1999. p. c073–c090. Disponível em: <http://www.ele.ita.br/cnrn/minicursos-5ern/log-neb.pdf>. Acesso em: 23 Ago 2007.

[34] GOMIDE, F. A. C.; GUDWIN, R. R. Modelagem, controle, sistemas e lógica *Fuzzy*. *Controle & Automação, Revista da Sociedade Brasileira de Automática*, v. 4, n. 3, p. 97–115, Set/Out 1994.

[35] LIN C. T.; LEE, G. *Neural Fuzzy Systems*. Upper Saddle River, NJ: Prentice Hall, 1995.

[36] ZADEH, L. A. The concept of a linguistic variable and its applications to approximate reasoning - part i. *Information Sciences*, v. 8, n. 1, p. 199–249, 1975.

[37] MAMDANI, E. H.; ASSILIAN, S. An experiment in linguistic synthesis with a fuzzy logic controller. *International Journal of Man-Machine Studies*, Academic Press, Inc., Duluth, MN, USA, v. 7, n. 1, p. 135–147, Jan 75.

[38] JÚNIOR, F. G. F. et al. Implementação de controladores PID utilizando lógica *fuzzy* e instrumentação industrial. In: *VII Simpósio Brasileiro de Automação Inteligente*. São Luís, MA: [s.n.], 2005.

[39] ALMEIDA, P. E. M. de; EVSUKOFF, A. G. Sistemas fuzzy. In: SOLANGE OLIVEIRA REZENDE (ORG.). *Sistemas Inteligentes: Fundamentos e Aplicações*. 1. ed. SP: Editora Manole, 2003. p. 169–202.

[40] SHAPIRO, L.; STOCKMAN, G. *Computer Vision*. Upper Saddle River, NJ: Prentice-Hall, 2001.

[41] PRATT, W. K. *Digital Image Processing: PIKS Inside*. 3th. ed. Los Altos, California: PixelSoft, Inc., 2002.

[42] FERREIRA, E. de P. *Robótica Básica*. Preliminar. Rio de Janeiro, RJ: V Escola Brasileiro-Argentina de Informática, 1991.

[43] LOUREIRO, R. Z. *Validação de Modelos de Movimento de Robôs Móveis Multi-Articulados via Processamento de Imagens Digitais*. 2006. Monografia (Engenharia de Computação) - Departamento de Informática, Universidade Federal do Espírito Santo.

[44] JESUS, J. D. F. de. *Aplicação da Visão Artificial ao Estudo do Movimento de Robôs e Veículos Multi-Articulados*. 2006. Monografia (Engenharia de Computação) - Departamento de Informática, Universidade Federal do Espírito Santo.

[45] INTEL. *Open Source Computer Vision Library*. Disponível em: <http://www.intel.com/technology/computing/opencv/index.htm>. Acesso em: 26 Fev 2008.

[46] PIXELINK. *Megapixel FireWire Camera User's Manual*. 2004.

[47] MITOV. *Video Capture and Processing component library for .NET, C++ and Delphi*. Disponível em: <http://www.mitov.com/html/videolab.html>. Acesso em: 26 Fev 2008.

[48] SIU, N. *An Analysis of Object Tracking Techniques in the Global Vision System*. Ago 2002. Disponível em: <http://www.ece.ubc.ca/~nsiu>. Acesso em: 20 Ago 2007.

[49] BRAGANÇA, J. O. *Estratégia para Deslocamento de Cargas Através de Cooperação entre Robôs Móveis a Roda. 116f.* Dissertação (Mestrado) — Programa de Pós-Graduação em Engenharia Elétrica, Universidade Federal do Espírito Santo, Vitória, 2004.

[50] GAVA, C. C. *Controle de Formação de Robôs Móveis Baseado em Visão Omnidirecional*. Dissertação (Mestrado) — Programa de Pós-Graduação em Engenharia Elétrica, Universidade Federal do Espírito Santo, Vitória, 2007.

[51] TANAKA, K.; TANIGUCHI, T.; WANG, H. O. Trajectory control of an articulated vehicle with triple trailers. In: *Proceedings of 1999 IEEE International Conference on Control Applications*. Kohala Coast, HI, USA: IEEE Press, 1999. p. 1673–1678.

[52] BRADSKI, G. R. Computer vision face tracking for use in a perceptual user interface. *Intel Technology Journal*, Intel Corp., n. Q2, p. 1–15, 1998. Disponível em: <ftp://download.intel.com/technology/itj/q21998/pdf/camshift.pdf>.

[53] FRANÇOIS, A. R. J. CAMSHIFT experiments. 2004. Disponível em: <http://mfsm.sourceforge.net/MFSM_0.7/tutorials/Camshift.html>.

[54] TAMIYA Home Page. Disponível em: <http://www.tamiya.com>. Acesso em: 20 Ago 2007.

[55] CORPORATION, A. *ATMega32 Datasheet*. [S.l.]. Disponível em: <http://www.atmel.com>.

[56] RAMPINELLI, M.; OLIVEIRA, T.; FERREIRA, E. de P. Análise da estimação de configuração de robôs móveis ou veículos multi-articulados por meio de visão computacional de topo: Efeito da velocidade e da distorção Óptica. In: *Anais da VIII Conferência Internacional de Aplicações Internacionais*. Poços de Caldas, MG: [s.n.], 2008.

[57] RAMPINELLI, M.; OLIVEIRA, T.; FERREIRA, E. de P. Análise da robustez da visão computacional de topo no controle de movimentos de robôs móveis ou veículos multi-articulados. In: *Anais do XVII Congresso Brasileiro de Automática*. Juiz de Fora, MG: [s.n.], 2008.

APÊNDICE A – Sistema de Visão Computacional e Controle Fuzzy

A.1 main.cpp

```
/*********************************************************************/
/* Arquivo: main.cpp                                                 */
/* Autor: Mariana Rampinelli Fernandes                               */
/* Em: 30/04/2008                                                    */
/* Testes com a câmera                                               */
/*********************************************************************/

/************************* Includes ********************************/
// System Includes
#include <assert.h>
#include <ctype.h>
#include <math.h>
#include <stdio.h>
#include <stdlib.h>
#include <string>
#include <time.h>
#include <vector>
#include <windows.h>

// Includes OpenCV and PGR
#include <cv.h>
#include <highgui.h>
#include <pgrflycapture.h>
#include <pgrcameragui.h>

// I/O communication Include
#include <cstdlib>
```

```
#include <ios>
#include <iostream>
#include <istream>
#include <fstream>
#include <ostream>

// Personal libraries
#include "ProImagem.h"
#include "trucktrailer.h"
#include "fuzzycontroller.h"

double getAnglesBetween( double theta1, CvPoint center1,
                         double theta2, CvPoint center2);

/*********************************************************************/

/****************************** Defines ******************************/
#define escala (255.0*255.0)/(360.0*360.0)
#define PI 3.14159
//#define CALCTIME
/*********************************************************************/

/****************************** Using ********************************/
using namespace std;
using namespace System::IO::Ports;
/*********************************************************************/

/****************** Camera's Variables *******************************/
FlyCaptureContext context;
FlyCaptureError error;
/*********************************************************************/

/****************************** Main *********************************/

int backproject_mode = 0; // classe vehicle
int show_hist = 1; // classe vehicle

int main()
{
   /* A fuzzy controller for each angle*/
   FuzzyController CpPI1("variaveis1tr.txt");
   FuzzyController CpPI2("variaveis2tr.txt");
```

```
TruckTrailer Truck;
int error;
double alpha[2];
double alpha2;
double alpha3;
double Ref1[1];
double Ref2[1];

/* It is to save the execution's results. */
FILE *file;
file = fopen("result.txt", "w");

/* It is to communicate with the truck. */
SerialPort Serial("COM4",38400,System::IO::Ports::Parity::None);
Serial.StopBits = System::IO::Ports::StopBits::One;
Serial.Handshake = System::IO::Ports::Handshake::None;
Serial.Open();
Serial.Write("n");
Serial.Write("c");

#ifdef CALCTIME
clock_t tInicio, tFim;
#endif

CpPI1.LoadRules("rules.txt");
CpPI2.LoadRules("rules.txt");

/* System with three objects: a truck and two trailers. */
Truck.InitSystem( 3);

cvNamedWindow( "Imagem", 1 );
ImageProcess imageProcess;

if(!imageProcess.IplOriginalImagem)
        return 1;

Sleep(1000);
Serial.Write("o+7");

int i = 0;
int c = 0;
/* Reference to the ultimate trailer. */
double Ref3 = -10;
```

```
for(;;)
{
    #ifdef CALCTIME
            tInicio = clock();
    #endif

    error = imageProcess.imageCapture();
    if( !imageProcess.IplOriginalImagem )
        break;

    Truck.GetAngles(imageProcess.IplOriginalImagem, alpha);
    cvShowImage( "Imagem", imageProcess.IplOriginalImagem );

    alpha3 = Ref3 - alpha[1]; // First controller reference
    CpPI2.Controller(&alpha3, Ref2, 7);
    // Second controller reference = first controller output.
    alpha2 = -(*Ref2) - alpha[0];
    CpPI1.Controller(&alpha2, Ref1, 7);

    int tmp = fabs(*Ref1)/10;
    int rst = int(fabs(*Ref1))%10;;

    if ((*Ref1) >= 0)
        Serial.Write("t+");
    else
        Serial.Write("t-");

    Serial.Write(tmp.ToString());
    Serial.Write(rst.ToString());

    i++;
    fprintf(file, "%d %f %f %f %f %f\n",
                i, alpha[0], alpha[1], (*Ref1), (*Ref2), Ref3);
    if (i == 13)
        Serial.Write("o+4");

    #ifdef CALCTIME
        tFim = clock();
        cout << "Tempo inicial :" << tInicio << endl;
        cout << "Tempo final :" << tFim << endl;
        cout << "Tempo decorrido: " <<  ((float)(tFim - tInicio)/
                                (CLOCKS_PER_SEC / 1000)) << endl;
    #endif
```

```
        c = cvWaitKey(1);
        if( (char) c == 27 ){ // exit
            Serial.Write("n");
            Serial.Close();
            fclose(file);
            cout << "Saindo..." << endl;
            cvDestroyWindow("Imagem");
            system("pause");
            break;
        }
    }
    return 0;
}
```

A.2 fuzzycontroller.h

```
/*********************************************************************/
/* Arquivo: fuzzycontroller.h                                        */
/* Autor: Mariana Rampinelli Fernandes                               */
/* Em: 25/02/2008                                                    */
/* Adapted  from David Martinek                                      */
/*********************************************************************/

/************************* Includes ********************************/
#ifndef FUZZYCONTROLLER_H
#define FUZZYCONTROLLER_H

/* System includes*/
#include <cstdlib>
#include <iostream>
#include <vector>

/* Our library*/
#include "fuzzyvr.h"

/*********************************************************/
/* FuzzySet - definition of all membership functions     */
/*********************************************************/

/* Implementation of a VariableSet - Input and Outputs. */
class VariableSet
{
```

```
protected:
  unsigned n;          /* actual number of elements */
  enum { MAX=10 };     /* implementation limit */
  FuzzyVariable *array[MAX]; /* is owner of objects on pointers */

public:

  // Creates variable.
  VariableSet( ) : n(0) { };

  /* Creates variable set. */
  VariableSet( FuzzyVariable &in1);

  // Creates variable set.
  VariableSet( FuzzyVariable &in1,
       FuzzyVariable &in2);

  // Creates variable set.
  VariableSet( FuzzyVariable &in1,
               FuzzyVariable &in2,
               FuzzyVariable &in3);

  // Creates variable set.
  VariableSet( FuzzyVariable &in1,
               FuzzyVariable &in2,
               FuzzyVariable &in3,
               FuzzyVariable &in4);

  /** Destructor removes all input variables.*/
  virtual ~VariableSet( );

  /* It adds next input variable into controller. */
  void add( FuzzyVariable &x);

  /** Number of variable input.*/
  int count() const { return n; }

  /** It selects i-th member function.*/
  virtual const FuzzyVariable *operator[] (int i) const;

  /** It selects i-th member function.*/
  virtual FuzzyVariable &operator[] (int i);
```

```
    /** It selects i-th input variable.*/
    virtual const double operator() (int i, int j) const;

    /** It sets i-th input variable.*/
    virtual double &operator() (int i, int j);

    /** It computes i-th variable input */
    void Fuzzify(int i, double x);

    /** It computes i-th value */
    double Defuzzify(unsigned i, unsigned tipo);

    /** Minimal value of universum for i-th variable.*/
    double minclass( int i) const { return array[i] -> minclass(); }

    /** Maximal value of universum i-th variable.*/
    double maxclass(int i) const { return array[i] -> maxclass(); }

    /** Name of this fuzzy set.*/
    const char * name(int i, int j) const { return array[i] -> wordValue(j); }

    /** Word value of i-th membership function. */
    const char * wordValue(int i, int j) const { return array[i]->wordValue(j); }

    /** Minimum of maxims. */
    double min1(); // min of 1

    /** Maximum of maxims.*/
    double max1(); // max of 1
}; // VariableSet

class FuzzyInference
{
     int nRules;
     int nVariables;
     int *data;

  public:
     /* Creates a Base Rules to Controler */
     FuzzyInference ( ) : nRules(0), nVariables (0) { } ;

     /* Destructor .*/
     virtual ~FuzzyInference( );
```

```
    /* Allocates memory to Base Rules */
    void Allocate (int nVar, int nRul);

    /* Set the variable value of a rule */
    int &operator() (unsigned rule, unsigned variable);

    /* Return the variable value of a rule */
    int  operator() (unsigned rule, unsigned variable) const;

    void PrintInference();

  protected:

    void SetVariablesNumber(int n) { nVariables = n;  }

    void SetRulesNumber(int n) { nRules = n; }
};

class FuzzyController
{
    VariableSet Inputs;    /* Inputs vector */
    VariableSet Outputs;   /* Outputs vector */
    unsigned nRules;       /* Rules quantity */
    unsigned nInputs;      /* Input quantity */
    unsigned nOutputs;     /* Output quantity */
    unsigned nVariables;
    FuzzyInference Rules; /* Base Rules */

   public:
    /* Constructor */
    FuzzyController (const char *VarFileName);

    /* Destructor */
    virtual ~FuzzyController( ) { TRACE(printf("~FuzzyController()\n")); }

    /* Loads the Inference Rules */
    void LoadRules ( const char *RulesFileName);

    /* Prints de Base Rules contents */
    void PrintRules( );

    /* Controler: main function*/
```

```
    void Controller (double *inputs, double outputs[], unsigned deffuzifyType);

    /* Make inference*/
    void Inference();

    /* Gets nInputs*/
    unsigned GetnInputs() {return nInputs;}

    /* Gets nOutputs*/
    unsigned GetnOutputs() {return nOutputs;}

    void InitOutputs();
};
#endif
```

A.3 fuzzycontroller.cpp

```
/*********************************************************************/
/* Arquivo: fuzzycontroller.cpp                                      */
/* Autor: Mariana Rampinelli Fernandes                               */
/* Em: 25/02/2008                                                    */
/* Adapted  from David Martinek                                      */
/*********************************************************************/

/************************* Includes ******************************/
/* Our library*/
#include "fuzzycontroller.h"

using namespace std;
/****************** VariableSetInput Implementations **************/

//Creates variable set.
VariableSet :: VariableSet( FuzzyVariable &in1)
 : n ( 0 )
{
   add ( in1 );
}

// Creates variable set.
VariableSet :: VariableSet( FuzzyVariable &in1,
          FuzzyVariable &in2)
  : n ( 0 )
```

```
{
  add(in1);
  add(in2);
}

// Creates variable set.
VariableSet :: VariableSet(  FuzzyVariable &in1,
                  FuzzyVariable &in2,
                  FuzzyVariable &in3)
  : n ( 0 )
{
  add(in1);
  add(in2);
  add(in3);
}

// Creates variable set.
VariableSet :: VariableSet(  FuzzyVariable &in1,
                  FuzzyVariable &in2,
                  FuzzyVariable &in3,
                  FuzzyVariable &in4)
  : n ( 0 )
{
  add(in1);
  add(in2);
  add(in3);
  add(in4);
}

/** Destructor removes all input variables.*/
VariableSet:: ~VariableSet()
{
  TRACE(printf("~VariableSet()\n"));
  for (unsigned i = 0; i < n; i++)
          delete array[i]; // clean memory
}

// It adds next variable input into cotroller
void VariableSet::add( FuzzyVariable &x)
{
  TRACE(printf("VariableSet::add(%d)\n", n+1));
  if (n >= MAX){
          cout << "FuzzySet limit exceeded" << endl;
```

```
            system("PAUSE");
    }
    array[n++] = x.clone();
}

/** It selects i-th input variable.*/
const FuzzyVariable *VariableSet :: operator[] (int i) const
{
    if (unsigned(i) >= n){
            cout << "VariableSet[]: index out of range";
            system("pause");
        exit(-1);
    }
    return array[i];
}

/** It selects i-th input variable.*/
FuzzyVariable &VariableSet :: operator[] (int i)
{
    if (unsigned(i) >= n){
            cout << "VariableSet[]: index out of range";
            system("pause");
        exit(-1);
    }
    return (*array[i]);
}


/** It selects i-th input variable.*/
const double VariableSet :: operator() (int i, int j) const
{
        return (*array[i])[j];
}

double &VariableSet :: operator() (int i, int j)
{
        return (*array[i])[j];
}


void VariableSet :: Fuzzify(int i, double x)
{
    if (unsigned(i) >= n){
```

```
    cout << "FuzzySet::Membership: index out of range\n";
    system("pause");
    exit(-1);
  }
  array[i]->Fuzzify(x);
}

/** It computes i-th value */
double VariableSet :: Defuzzify(unsigned i, unsigned tipo)
{
  if (i >= n){
    cout << "FuzzySet::Membership: index out of range\n";
    system("pause");
    exit(-1);
  }
  return array[i]->Defuzzify(tipo);
}

/****************** Fuzzy Inference Implementations **************/

/** Destructor removes all input variables.*/
FuzzyInference:: ~FuzzyInference()
{
  TRACE(printf("~FuzzyInference()\n"));
  delete[] data;
}

/* Allocates memory to Base Rules */
void FuzzyInference :: Allocate (int nRul, int nVar)
{
  SetVariablesNumber(nVar);
  SetRulesNumber (nRul);

  data  = new int[nRul * nVar];

  if (nRul == 0 || nVar == 0)
  {
    cout << "Matrix constructor has 0 size.\n" ;
    system("pause");
    exit(-1);
  }
}
```

```
/* Set the variable value of a rule */
int &FuzzyInference :: operator() (unsigned rule, unsigned variable)
{
  if (rule >= nRules || variable >= nVariables)
  {
    printf ("Index (%d, %d) exceeds matrix dimension (%d, %d).",
            rule, variable, nRules, nVariables); ;
    system("pause");
    exit(-1);
  }

  return data[nVariables*rule + variable];
}

/* Return the variable value of a rule */
int  FuzzyInference :: operator() (unsigned rule, unsigned variable) const
{
  if (rule >= nRules || variable >= nVariables)
  {
    printf ("Index (%d, %d) exceeds matrix dimension (%d, %d).",
            rule, variable, nRules, nVariables);
    system("pause");
    exit(-1);
  }
  return data[nVariables*rule + variable];
}

void FuzzyInference ::PrintInference()
{ cout << nRules << " " << nVariables << endl;}

/****************** Fuzzy Controler Implementations **************/

/* Creates Fuzzy Controler */
FuzzyController :: FuzzyController(const char *VarFileName)
  : nRules(0)
{
    FILE *file;

    int nVar, nmf, type;
    float x0, x1, x2, x3;
    float max, min;
    string name;
    file = fopen (VarFileName, "r");
```

```
if ( !file )
{
      cout << "\n Can't open \"" << VarFileName << "\" file" << endl;
      system("PAUSE");
      exit(1);
}

// Read inptus variable quantities
fscanf ( file , "%d", &nVar );
nInputs = nVar;
cout << "Lendo inputs... " << nVar << endl;
    while (nVar)
{
       nVar--;
              fscanf(file , "%s %f %f %d", name.data(), &min, &max, &nmf);
       FuzzySet SetMf(name.data(), min, max);

      while (nmf){
             nmf --;
             fscanf(file , "%d", &type);
             switch ( type ){
                    case 1:{
                          // Função Triangular
                          fscanf(file , "%s %f %f %f",
                                 name.data(), &x0, &x1, &x2);
                          FuzzyTriangle mf(name.data(), x0, x1, x2);
                          SetMf.add(mf);
                              break;
                    }
                    case 2:{
                          // Função Trapezoidal
                          fscanf(file , "%s %f %f %f %f",
                                 name.data(), &x0, &x1, &x2, &x3);
                          FuzzyTrapez mf(name.data(), x0, x1, x2, x3);
                          SetMf.add(mf);
                          break;
                    }
                    default:
                          cout << "\n Opção não encontrada\n";
             }
      }
      FuzzyVariable Variable(SetMf);
      Inputs.add(Variable);
```

```
    }
    // Le quantidade de variáveis de Saída
    fscanf ( file , "%d", &nVar );
    nOutputs = nVar;
    cout << "Lendo outputs... " << nVar << endl;
    while (nVar)
    {
        nVar--;
        fscanf (file , "%s %f %f %d", name.data(), &max, &min, &nmf);
        FuzzySet SetMf(name.data(), max, min);
        while (nmf){
            nmf --;
            fscanf (file , "%d", &type);
            switch ( type ){
                case 1:{
                    // Função Triangular
                    fscanf (file , "%s %f %f %f",
                            name.data(), &x0, &x1, &x2);
                    FuzzyTriangle mf(name.data(), x0, x1, x2);
                    SetMf.add(mf);
                    break;
                }
                case 2:{
                    // Função Trapezoidal
                    fscanf (file , "%s %f %f %f %f",
                            name.data(), &x0, &x1, &x2, &x3);
                    FuzzyTrapez mf(name.data(), x0, x1, x2, x3);
                    SetMf.add(mf);
                    break;
                }
                default:
                    cout << "\n Opção não encontrada\n";
            }
        }
        FuzzyVariable Variable(SetMf);
        Outputs.add(Variable);
    }
    nVariables = nInputs + nOutputs;
    fclose(file);
}

/* Loads the Inference Rules */
void FuzzyController :: LoadRules ( const char *RulesFileName)
```

```
{
    FILE *file;
    int value;
    file = fopen (RulesFileName, "r");
    if ( !file ) {
        cout << "\n Can't open \"" << RulesFileName << "\" file" << endl;
        system("PAUSE");
        exit(1);
    }
    /* Read rules quantity */
    fscanf (file, "%d", &value);
    nRules = value;

    /* Allocate memory*/
    Rules.Allocate(nRules, nInputs+nOutputs);

    for (unsigned i = 0; i < nRules; i++){
      for (unsigned j = 0; j < (nInputs + nOutputs); j++){
        fscanf (file, "%d", &value);
        Rules(i,j) = value;
      }
    }
}

void FuzzyController :: PrintRules( )
{
   cout << "N Inputs  : " << nInput s << endl ;
   cout << "N Outputs : " << nOutputs << endl ;
   for (unsigned i = 0; i < nRules; i++)
      for (unsigned j = 0; j < (nInputs + nOutputs); j++)
        cout << "Rules[" << i << "][" << j << "]=" << Rules(i,j) << endl;
}

void FuzzyController :: Inference( )
{
    /* For all rules*/
    for (unsigned i = 0; i < nRules; i++)
    {
        unsigned j = 0;
        double mi = 0.0;

        /* For all input variables*/
        double alpha = Inputs( j, Rules(i, j));
```

```
        for(j = 1 ; j < nInputs; j++){ // all input variable
            mi = Inputs( j, Rules(i, j));
            alpha = min(alpha,mi);
        }

        /* For all output variables*/
        for ( ; j < nVariables ; j++){
            mi = Outputs((j - nInputs), Rules(i, j));
            Outputs((j - nInputs), Rules(i, j)) = max(mi, alpha);
        }
    }
}

///* Controller: reset outpus variables */
void FuzzyController :: InitOutputs()
{
    for (unsigned i = 0; i < nOutputs; i++)
                Outputs[i].Init();
}

/* Controler: main function*/
void FuzzyController :: Controller (double *inputs, double outputs[],
                                    unsigned deffuzifyType)
{
    InitOutputs();

        for (unsigned i = 0; i < nInputs; i++)
        Inputs.Fuzzify(i, inputs[i]);

    Inference();

        for (unsigned i = 0; i < nOutputs; i++)
        outputs[i] = Outputs.Defuzzify(i, deffuzifyType);
}
// end
```

A.4 ProImagem.h

```
/*************************************************************/
/* Arquivo: ProImagem.cpp                                    */
/* Autor: Mariana Rampinelli Fernandes                       */
/* Em: 09/04/2008                                            */
/* Funções de Processamento de Imagem                        */
```

```
/*************************************************************/

#ifndef PROIMAGE_H
#define PROIMAGE_H

/************************* Includes ************************/
// Includes OpenCV and PGR
#include <cv.h>
#include <highgui.h>
#include <pgrflycapture.h>

// System Includes
#include <assert.h>
#include <stdio.h>
#include <stdlib.h>
#include <windows.h>
#include "math.h"
#include <vector>

// I/O communication Include
#include <iostream>

/*************************************************************/
/*                          Class ImageProcess                 */
/*************************************************************/
class ImageProcess
{
   public:
         IplImage *IplOriginalImagem; // OpenCV image

   private:
      FlyCaptureContext context;        // Camera Context
      FlyCaptureError error;            // Camera Error
      FlyCaptureImage flyOriginalImage;// PRG Images
      FlyCaptureImage flyColorImage;

   public:
      // constructor
      ImageProcess( );

      // destructor
      ~ImageProcess( );
```

```
        // capture image from camera
        int imageCapture( );

        // Get OpenCV image
        IplImage *getIplImage( );

        // Shows the Original Image
        void showOriginalIplImage( char *windowName );

    protected:
        // Starts camera
        int startCamera( );
};

#endif
```

A.5 ProImagem.cpp

```
/*************************************************************/
/* Arquivo: ProImagem.cpp                                    */
/* Autor: Mariana Rampinelli Fernandes                       */
/* Em: 09/04/2008                                            */
/* Funções de Processamento de Imagem                        */
/*************************************************************/

/************************* Includes *************************/
#include "ProImagem.h"
#include <pgrcameragui.h>
/*************************************************************/

/************************** Using ***************************/
using namespace std;
/*************************************************************/

/******************** ImageProcess **************************/
// Class Constructor
ImageProcess :: ImageProcess( )
{
    int error = 0;
    // Starts the camera
    error = startCamera( );

    if (error) {
```

```
        cout << "Can not starts the selected camera." << endl;
        system("pause");
        exit (-1);
    }
    // Inicialize OpenCV Image
    IplOriginalImagem = cvCreateImage( cvSize( flyOriginalImage.iCols,
                                flyOriginalImage.iRows ), IPL_DEPTH_8U,3);
    IplOriginalImagem->origin = IPL_ORIGIN_TL;
    IplOriginalImagem->dataOrder = IPL_DATA_ORDER_PIXEL;

    flyColorImage = flyOriginalImage;
    flyColorImage.pixelFormat = FLYCAPTURE_BGR;
    flyColorImage.pData = reinterpret_cast<unsigned char*>
                                (IplOriginalImagem->imageData);

}

// Class Constructor
ImageProcess :: ~ImageProcess( )
{
    cvReleaseImage( &IplOriginalImagem );
    flycaptureStop( context );
    flycaptureDestroyContext( context );
}

// Capture Image from Camera
int ImageProcess :: imageCapture( )
{
   error = flycaptureGrabImage2( context, &flyOriginalImage);
   if ( error != FLYCAPTURE_OK ){
        cout << "Can not capture 1." << endl;
        system("pause");
        exit(-1);
    }

    error = flycaptureConvertImage( context, &flyOriginalImage,
                                                &flyColorImage);
    if ( error != FLYCAPTURE_OK ){
        cout << "Can not capture 2." << endl;
        system("pause");
        exit(-1);
    }
    return 0;
```

```
}

// Get OpenCV image
IplImage *ImageProcess :: getIplImage( )
{
   IplImage *tmp = cvCreateImage(cvGetSize(IplOriginalImagem),
                                                IPL_DEPTH_8U, 3 );
   tmp = cvCloneImage(IplOriginalImagem);
   return tmp;
}

// Starts camera to capture
int ImageProcess :: startCamera( )
{
    unsigned long serial = 0;
    INT_PTR dialogStatus;
    CameraGUIContext guicontext;
    CameraGUIError guierror;

    error = flycaptureCreateContext( &context);
    guierror = pgrcamguiCreateContext( &guicontext);

    if ( error != FLYCAPTURE_OK){
            cout <<  flycaptureErrorToString( error ) << endl;
            return 1;
    }

    // Starts window camera select
    guierror = pgrcamguiShowCameraSelectionModal( guicontext,
                                context, &serial, &dialogStatus);
    if ( guierror != PGRCAMGUI_OK)
        return 1;

    if ( serial == 0)
        return 1;

    error = flycaptureInitializeFromSerialNumber( context, serial);
    if ( error != FLYCAPTURE_OK){
        cout << "Falha na iniciacao pelo serial...\n";
        return 1;
    }
    error = flycaptureStart( context, FLYCAPTURE_VIDEOMODE_ANY,
                                            FLYCAPTURE_FRAMERATE_ANY);
```

```
    if ( error != FLYCAPTURE_OK){
        cout << "Falha na inicição de captura...\n";
        return 1;
    }

    // Inicia aquisição das imagens
    error = flycaptureGrabImage2( context, &flyOriginalImage);
    if ( error != FLYCAPTURE_OK){
    cout << "Falha na aquisição das imagens...\n";
    return 1;
    }
    return 0;
}

void ImageProcess :: showOriginalIplImage( char *windowName )
{
    cvNamedWindow( "Original Image", 1 );
    // Show the current image
    cvShowImage( "Original Image", IplOriginalImagem );
}
```

A.6 trucktrailer.h

```
/*********************************************************/
/* Arquivo: ProImagem.h                                  */
/* Autor: Mariana Rampinelli Fernandes                   */
/* Em: 19/12/2007                                        */
/* Cabeçalho de Funções de Processamento de Imagem       */
/*********************************************************/

#ifndef TRUCKTRAILER_H
#define TRUCKTRAILER_H

// Includes OpenCV and PGR
#include <cv.h>

// System Includes
#include "math.h"

// I/O communication Include
#include <iostream>
#include "ProImagem.h"
```

```
/*********************************************************/
/*                    Class ObjectVehicle                */
/*********************************************************/
class ObjectVehicle
{
   public:
      /* Kalman Parameters*/
      double kalmanAlpha;
      double K;
      double P;

      /* OpenCV images */
      IplImage *backproject;
      IplImage *histimg;

      /* OpenCV structs */
      CvHistogram *hist;
      CvRect selection;
      CvRect track_window;
      CvBox2D track_box;
      CvConnectedComp track_comp;

      /* Histogram values */
      int smin;
      int hdims;
      float* hranges;

      /*Object Name*/
      char *ObjectName;
      char HistimgName[50];

      /* Flag status*/
      int status;

   public:
      /* Constructor */
      ObjectVehicle( char *name, int satmin, CvPoint point,
                     int ImageOrigin, int ImageWidth, int ImageHeight);

      /* Destructor */
      ~ObjectVehicle();

      /* Inicializa object */
```

```
        void InitVehicle ( IplImage *OriginalImage);

        void InitTracking( IplImage *hue, IplImage *mask );

        /* Makes a object clone */
        ObjectVehicle *ObjectVehicle::clone() const;

        void TrackingObject (IplImage *image, IplImage *hue,
                                        IplImage *mask, IplImage *hsv);

        int getOrientation(IplImage *image);

        void CreateHistWindows();

        void ShowHistogram();

    protected:
        CvScalar hsv2rgb( float hue );
};

class TruckTrailer
{
    public:
        /* General Images */
        IplImage *image;
        IplImage *hsv;
        IplImage *hue;
        IplImage *mask;

        /* Tractor Truck System */
        enum {MAX = 10};
        ObjectVehicle *array[MAX]; // 1 truck + n trailers
        int Ntrailers;

        /* Kalman Parameters*/
        double alphas[MAX-1];
        double kalmanAlphas[MAX-1];
        double P[MAX-1];
        double K[MAX-1];

        /* Flags */
        int center_ok;
        int firstFrame; // Uses in kalman filter
```

```
    public:
        /* Constructor Class */
        TruckTrailer( );

        /* Destructor Class */
        ~TruckTrailer( );

        /* Inicialize the system truck trailer*/
        int InitSystem(int trailerNum );

        /** It selects i-th trailer .*/
        const ObjectVehicle *TruckTrailer :: operator[] (int i) const;

        void add( ObjectVehicle &x );

        void Tracking(IplImage *image);

        void GetAngles(IplImage *OriginalImage, double *alpha);

        // estimation by kalman
        void KalmanEstimation( int firstFrame, int i);
};
#endif
```

A.7 trucktrailer.cpp

```
/*************************************************************/
/* Arquivo: trucktrailer.cpp                                 */
/* Autor: Mariana Rampinelli Fernandes                       */
/* Em: 18/03/2007                                            */
/* Objeto para caracterização de cada elemento do veículo    */
/*************************************************************/

/************************* Includes ************************/
#include "trucktrailer.h"

// Includes OpenCV
#include <cv.h>
#include <highgui.h>
/*************************************************************/

/********************** Using ****************************/
```

```
using namespace std;
/*************************************************************/

/********************* Defines ******************************/
#define escala (255.0*255.0)/(360.0*360.0)
#define PI 3.14159
//#define KALMANTEST
/*************************************************************/

/****************** Global Variables ************************/
CvPoint center = cvPoint(0,0);
int select_object;

/* Selects a object point */
void on_mouse( int event, int x, int y, int flags, void* param );
/* Calculates angles between two elements */
double getAnglesBetween( double theta1, CvPoint center1,
                         double theta2, CvPoint center2);

/******************** ObjectVehicle Class *******************/
/* Constructor */
ObjectVehicle :: ObjectVehicle( char *name, int satmin, CvPoint point,
            int ImageOrigin, int ImageWidth, int ImageHeight )
  : smin(satmin), status(0), kalmanAlpha(0), K(0), P(0)
{
        int wdt = 11, hgt = 11;
        /* OpenCV images */
        backproject = 0;
        histimg = 0;

        /* OpenCV structs */
        hist = 0;
        hdims = 16;
        hranges  = (float*) malloc(2*sizeof(float));
        hranges[0] = 1; hranges[1] = 180;
        ObjectName = strdup(name);

        /* Inicialize selection */
        point.x = MAX(point.x - 5, ImageOrigin);
        point.y = MAX(point.y - 5, ImageOrigin);
       if ( point.x+wdt > ImageWidth )
         wdt = ImageWidth - point.x;
       if ( point.y+hgt > ImageHeight )
```

```
        hgt = ImageHeight - point.y;
      /* Create a seleted region with 10x10 pixels*/
      selection = cvRect(point.x, point.y, wdt, hgt)
      if (selection.x && selection.y && selection.width)
      status = -1;
}

/* Destructor */
ObjectVehicle :: ~ObjectVehicle( )
{
      /* OpenCV images */
      cvReleaseImage(&backproject);
      cvReleaseImage(&histimg);
      cvReleaseHist(&hist);

      /* OpenCV images */
      backproject = 0;
      histimg = 0;

      /* OpenCV structs */
      hist = 0;
      hdims = 16;

      /* Inicialize selection */
      selection = cvRect(0, 0, 0, 0);
      status = 0;
}

void ObjectVehicle :: InitTracking( IplImage *hue, IplImage *mask)
{
    float max_val = 0.f;
    float min_val = 0.f;
    int bin_w;

    /* Inicializes histogram and images members */
    backproject = cvCreateImage( cvGetSize(hue), 8, 1 );
    histimg = cvCreateImage( cvSize(320,200), 8, 3 );
    cvZero( histimg );
    hist = cvCreateHist( 1, &hdims, CV_HIST_ARRAY, &hranges, 1 );

    cvSetImageROI( hue, selection );
    cvSetImageROI( mask, selection );
```

```
    cvCalcHist( &hue, hist, 0, mask );
    cvGetMinMaxHistValue( hist, &min_val, &max_val, 0, 0 );
    cvConvertScale( hist->bins, hist->bins, max_val ? 255. / max_val : 0., 0 );

    cvResetImageROI( hue );
    cvResetImageROI( mask );

    track_window = selection;
    status = 1;

    cvZero( histimg );
    bin_w = histimg->width / hdims;

    for( int i = 0; i < hdims; i++ ){
        int val = cvRound( cvGetReal1D( hist->bins,i) * histimg->height/255 );
        CvScalar color = hsv2rgb(i*180.f/hdims);
        cvRectangle( histimg, cvPoint(i*bin_w, histimg->height),
        cvPoint((i+1)*bin_w, histimg->height - val),
        color, -1, 8, 0 );
    }
}

/** It duplicates object. */
ObjectVehicle *ObjectVehicle::clone() const
{
  ObjectVehicle *result = new ObjectVehicle(*this);
  return result;
}

void ObjectVehicle :: TrackingObject (IplImage *image, IplImage *hue,
                                      IplImage *mask, IplImage *hsv)
{
    /* Segments the image with selected parameters.*/
    cvInRangeS( hsv, cvScalar( 0, smin, 0, 0),
                            cvScalar( 180, 256, 256, 0), mask );
    /* Splits image */
    cvSplit( hsv, hue, 0, 0, 0 );

    if( status < 0 ) // Na primeira vez apenas
        InitTracking(hue, mask);

    cvCalcBackProject( &hue, backproject, hist );
    cvAnd( backproject, mask, backproject, 0 );
```

```
    cvCamShift( backproject, track_window,
    cvTermCriteria( CV_TERMCRIT_EPS | CV_TERMCRIT_ITER, 20, 1 ),
                        &track_comp, &track_box );
    cvErode(backproject, backproject, 0, 1);
    track_window = track_comp.rect;
    track_box.angle = -track_box.angle;

    if (track_box.angle < 0)
        track_box.angle = track_box.angle + 180;
}

CvScalar ObjectVehicle :: hsv2rgb( float hue )
{
    int rgb[3], p, sector;
    static const int sector_data[][3]=
        {{0,2,1}, {1,2,0}, {1,0,2}, {2,0,1}, {2,1,0}, {0,1,2}};
    hue *= 0.033333333333333333333333333333333f;
    sector = cvFloor(hue);
    p = cvRound(255*(hue - sector));
    p ^= sector & 1 ? 255 : 0;
    rgb[sector_data[sector][0]] = 255;
    rgb[sector_data[sector][1]] = 0;
    rgb[sector_data[sector][2]] = p;
    return cvScalar(rgb[2], rgb[1], rgb[0],0);
}

void ObjectVehicle :: CreateHistWindows()
{
        sprintf(HistimgName, "Histogram_%s", ObjectName);
        cvNamedWindow(HistimgName, 1);
}

void ObjectVehicle :: ShowHistogram()
{
        cvShowImage(HistimgName, histimg);
}

/***************** ObjectVehicle Class ****************************/
/* Constructor */
TruckTrailer :: TruckTrailer( )
   : center_ok(0), firstFrame(1), Ntrailers(0)
{
        image = 0;
```

```
        hsv = 0;
        hue = 0;
        mask = 0;
}

/* Destructor */
TruckTrailer :: ~TruckTrailer( )
{
        center_ok = 0;
        select_object = 0;
        firstFrame = 0;
        cvReleaseImage(&image);
        cvReleaseImage(&hsv);
        cvReleaseImage(&hue);
        cvReleaseImage(&mask);
        for(int i = 0; i < Ntrailers; i++)
                delete array[i]; // cleam memory
        Ntrailers = 0;
}

/* Inicializes the system truck trailer*/
int TruckTrailer :: InitSystem( int trailerNum )
{
        ImageProcess process;
        int error;
        int i = 0;
        int smin = 50;

        /* Inicializes images */
        error = process.imageCapture();
        if (!process.IplOriginalImagem)
           return 1;
        while (i != trailerNum){
        i++;
        select_object  = 0;
        char trailerName[50]; // Object name
        char segmentedImage[50] = "Segmented Image";
        if (i == 1)
          sprintf(trailerName, "Truck");
        else
          sprintf(trailerName, "Trailer%d", i);
        cvNamedWindow( trailerName, 1 );
        cvNamedWindow( segmentedImage, 1 );
```

```
        cvSetMouseCallback( trailerName, on_mouse, 0 );
        cvCreateTrackbar( "Smin", trailerName, &smin, 256, 0 );
        for( ; ; ){
          if (!image){
            image = cvCreateImage(cvGetSize(process.IplOriginalImagem),8,3);
            image->origin = process.IplOriginalImagem ->origin;
            hsv = cvCreateImage( cvGetSize(process.IplOriginalImagem),8,3);
            hue = cvCreateImage( cvGetSize(process.IplOriginalImagem),8,1);
            mask = cvCreateImage( cvGetSize(process.IplOriginalImagem),8,1);
          }
          cvCopy( process.IplOriginalImagem, image, 0);
          cvShowImage( trailerName, process.IplOriginalImagem );
          cvCvtColor( image, hsv, CV_BGR2HSV );
          error = process.imageCapture();
          if (image && select_object > 0 && center.x > 0 && center.y > 0){
             ObjectVehicle vehicle(trailerName, smin, center,
                                   process.IplOriginalImagem->origin,
                                   process.IplOriginalImagem->width,
                                   process.IplOriginalImagem->width);
             vehicle.TrackingObject(process.IplOriginalImagem,hue,mask,hsv);
             cvShowImage(segmentedImage, vehicle.backproject);
          }
          if ( ( cvWaitKey( 1 ) & 255 ) == 13 ){
          // Creates a object with found properties
          ObjectVehicle vehicle(trailerName, smin, center,
                                process.IplOriginalImagem->origin,
          process.IplOriginalImagem->width, process.IplOriginalImagem->width);
          add(vehicle);
          center = cvPoint(0,0);
          select_object = -1;
          cvDestroyWindow(trailerName);
          cvDestroyWindow(segmentedImage);
          break;
        }
      }
    }
    return 0;
}

/** It selects i-th trailer .*/
const ObjectVehicle *TruckTrailer :: operator[] (int i) const
{
  if (unsigned(i) >= Ntrailers){
```

```
        cout << "TruckTrailer[]: index out of range";
        system("pause");
        exit(-1);
  }
  return array[i];
}

// Adds next object vehicle in trucktrailer system
void TruckTrailer :: add( ObjectVehicle &x )
{
     printf("TruckTrailer::add(%d)\n", Ntrailers+1);
     if (Ntrailers >= MAX) cout << "System limit exceeded" << endl;
     array[Ntrailers++] = x.clone();
}

void TruckTrailer :: Tracking(IplImage *OriginalImage)
{
        cvCopy( OriginalImage, image, 0 );
        cvCvtColor( image, hsv, CV_BGR2HSV );
        for (int i = 0; i < Ntrailers; i++)
                array[i]->TrackingObject (OriginalImage, hue, mask, hsv);
}

void TruckTrailer :: GetAngles(IplImage *OriginalImage, double *alpha)
{
        Tracking(OriginalImage);
        for (int i = 0; i < (Ntrailers - 1); i++){
           alphas[i] = array[i+1]->track_box.angle-array[i]->track_box.angle;
           alphas[i] = alphas[i] >  90 ? alphas[i] - 180: alphas[i];
           alphas[i] = alphas[i] < -90 ? alphas[i] + 180 : alphas[i];
           KalmanEstimation(firstFrame, i);
           alpha[i] = alphas[i];
           alpha[i+2] = kalmanAlphas[i];
        }
        firstFrame = 0;
}

void TruckTrailer :: KalmanEstimation( int firstFrame, int i)
{
   double  Q = 1, H = 1, A = 1, R = 2;
   if (firstFrame)
           kalmanAlphas[i] = alphas[i];
   K[i] = H * P[i] * (1/(H * P[i] * H + R));
```

```
    kalmanAlphas[i] = kalmanAlphas[i]+K[i]*(alphas[i]-H*kalmanAlphas[i]);
    P[i] = (1 - K[i] * H) * P[i]; // Estimativa
    kalmanAlphas[i] = A * kalmanAlphas[i];
    P[i] = A * A * P[i] + Q;
}

/* Selects a object point */
void on_mouse( int event, int x, int y, int flags, void* param )
{
    if (select_object < 0)
                return;
        if (event == CV_EVENT_LBUTTONDOWN){
        center.x = x;
                center.y = y;
        select_object = 1;
                printf("\nx: %d\ty: %d", center.x, center.y);
        }
}

// Encontra o ângulo entre dois corpos
// O primeiro corpo é o trator ou o mais próximo ao trator
double getAnglesBetween( double theta1, CvPoint center1,
                         double theta2, CvPoint center2)
{
    double alpha = 0;

    alpha = theta2 - theta1;
    alpha = alpha >  90 ? alpha - 180 : alpha;
    alpha = alpha < -90 ? 180 + alpha : alpha;
    return alpha;
}
```