UNIVERSIDADE FEDERAL DE ITAJUBÁ
PROGRAMA DE PÓS-GRADUAÇÃO EM ENGENHARIA ELÉTRICA

MÉTODOS INTELIGENTES DE NAVEGAÇÃO E DESVIO DE OBSTÁCULOS

HELTON HUGO DE CARVALHO JUNIOR

Itajubá, novembro de 2007

UNIVERSIDADE FEDERAL DE ITAJUBÁ
PROGRAMA DE PÓS-GRADUAÇÃO EM ENGENHARIA ELÉTRICA

HELTON HUGO DE CARVALHO JÚNIOR

MÉTODOS INTELIGENTES DE NAVEGAÇÃO E DESVIO DE OBSTÁCULOS

Dissertação Submetida Ao Programa De Pós Graduação Em Engenharia Elétrica Como Parte Dos Requisitos Para Obtenção Do Titulo De Mestre Em Ciências Em Engenharia Elétrica.

Área De Concentração: Automação E Sistemas Elétricos Industriais.

Orientador: Prof. Dr. Leonardo de Mello Honório

Co-orientador: Prof. Dr. Edison Oliveira de Jesus

NOVEMBRO DE 2007

ITAJUBÁ - MG

À minha Mãe e Pai, Marly Alves de Carvalho e
Helton Hugo de Carvalho.

Agradeço a todos que de alguma forma contribuíram para o desenvolvimento deste trabalho, em especial:

Ao meu querido orientador, Leonardo de Mello Honório, que me ajudou em mais essa etapa da minha vida.

Ao meu querido co-orientador, Edison de Jesus que além de conselhos ainda contribuiu sistemicamente para meu trabalho.

Aos meus amigos do CRTI em especial ao colega Welinton Dias pelo apoio em algumas etapas do trabalho.

Agradecimentos especiais aos meus irmãos e principalmente aos meus pais pelo amor e carinho.

A todos aqueles que direta ou indiretamente colaboraram para que este projeto fosse concluído.

Por fim, quero agradecer a todos os professores e funcionários da Universidade Federal de Itajubá e espero que todos nós possamos continuar a engrandecer ainda mais nossa querida universidade.

RESUMO

Este trabalho tem como objetivo o estudo e a aplicação de métodos inteligentes para navegação e desvio de obstáculos, baseando-se em um simulador previamente construído, o qual é o ponto inicial para a criação de uma prática para aplicação no mundo real. Entretanto para conseguir este objetivo, o trabalho atravessa diversas áreas do conhecimento como computação, visão computacional, programação, lógica matemática, eletrônica, inteligência artificial e automação e controle. A proposta é usar um veículo inteligente, nesse caso um carro de controle remoto, controlado por rádio freqüência, obstáculos, como por exemplo, bolinhas de tênis e um alvo. O local, onde esses objetos estão situados, é denominado “campo de testes”. Através de um software, criado com o propósito de controle e automação, este veículo, por meio de técnicas de inteligência artificial, deve percorrer o campo de testes, desviando dos obstáculos e atingir o alvo. O veiculo é monitorado por uma câmera, localizada acima da área de testes, e recebe os comandos do software de controle, instalado no computador, através de seu controle remoto, na qual a comunicação é feita via radiofreqüência.

ABSTRACT

This work has as objective the implementation and the use of intelligent methods for navigation and obstacles avoidance, being based on a simulator previously constructed, which is the initial point for the creation of a practical application in the real world. However to obtain this objective, it crosses diverse areas of the knowledge as computation, computational vision, programming, mathematical logic, electronic, artificial intelligence and automation and control. The proposal is to use an intelligent vehicle, in this case, a remote-control car, controlled for radio frequency, obstacles, for example, small balls of tennis and a target. The place where these objects are situated is called "field of tests". Software is created with the intention of control and automation, this vehicle, by means of techniques of artificial intelligence, it must cover the field of tests, deviating from the obstacles and reach the target. The vehicle is monitored by a camera, located on the area of tests, and receives the commands from the software of control, installed in the computer, through its remote control, in which the communication is made way radio frequency.

SUMÁRIO

ÍNDICE DE FIGURAS

ÍNDICE DE TABELAS

1. Introdução

Este trabalho visa dotar um veículo inteligente de um sistema de controle, baseado no método de função potencial, a fim de realizar a automação deste veículo de forma que cumpra certos objetivos. O sistema de controle será obtido pela implementação da função potencial em ambiente de tempo real.

Os veículos inteligentes (e.g. carros dirigidos de forma autônoma [1]) têm atraído a atenção de um grande número de pesquisadores da área de Inteligência Artificial, devido ao desafio que este novo domínio de pesquisas nos propõe: dotar sistemas com uma capacidade de raciocínio inteligente (o método da função potencial de Lenard-Jones [2]) e de interação com o meio em que estão inseridos. Os veículos inteligentes devem poder "sentir" o ambiente em que estão inseridos através da leitura de seus sensores (e.g. sensores infra-vermelho, lasers, câmeras de vídeo, etc.), e através desta percepção sensorial eles planejam melhor as suas ações [3]. A percepção do veiculo inteligente é feita através de uma câmera de vídeo e os métodos de tratamento são descritos em [4].

Alguns exemplos de veículos inteligentes que se tornaram bastante conhecidos na atualidade: o robô do tipo *rover* enviado para Marte pela NASA [5]; o robô Dante que explora vulcões [6]; o sistema de controle de um veículo Ligier elétrico desenvolvido pelos pesquisadores do INRIA na França [7]. Todos estes sistemas possuem em comum a capacidade de receber leituras de sensores que lhes dão informações sobre o ambiente em que estão inseridos e de modo semi ou completamente autônomo, geram os comandos que fazem com que eles se desloquem em um ambiente de modo seguro: sem se chocar contra obstáculos ou colocar em risco sua integridade ou a dos diferentes elementos presentes no ambiente.

A partir de um artigo de mestrado desenvolvido pelo Centro de Referencias em Tecnologias da Informação, CRTI, da Universidade Federal de Itajubá, UNIFEI, intitulado Sistemas de Desenvolvimento de Lógica Fuzzy Orientado a Objetos,

relacionado a veículos inteligentes e inteligência artificial, criou-se aplicações (simuladores) na área de robótica autônoma móvel. Para realizar este trabalho, foi implementado um simulador de um veículo (“tipo nave espacial”) e desenvolvido um sistema de controle baseado em lógica f*uzzy.* O sistema de controle, do simulador, “aprendeu” a controlar o veículo a partir de uma série de regras definidas pela sua dinâmica comportamental, através de lógica *fuzzy*. Este simulador *fuzzy* foi o ponto inicial deste trabalho, ou seja, a partir dele criou-se uma aplicação prática, de forma que o veículo autônomo pudesse atingir seu objetivo, que era desviar de obstáculos e atingir um alvo, ambos pré-definidos, como no simulador. Porém as transcrições da simulação para a prática envolvem vários conceitos e novos problemas, como eletrônica, visão computacional, computação, controle e automação, sistemas de tempo real, dentre outros.

A eletrônica usada neste trabalho depende de uma série de itens de hardware que envolve um complexo sistema computacional e de controle. Pode-se citar o sistema computacional em si, usando a definição de microcomputador a qual, neste caso, destacam as operações de processamento e de entrada e saída. A saída, definida neste trabalho como a informação que é enviada ao veículo, foi feita pela porta *LPT1,* ou porta paralela*.* Um circuito baseado em relés foi desenvolvido para receber as informações da porta paralela e retransmiti-las ao controle remoto do carro, o qual era um controlador por freqüência de RF genérico. A entrada é, neste caso, a imagem capturada pelo computador através da câmera de vídeo. Na etapa do processamento é necessário tratar e identificar as imagens recebidas pela câmera, essa tarefa é feito pela área conhecida como processamento de imagens digitais ou PDI.

A área de processamento de imagens vem sendo objeto de crescente interesse por permitir viabilizar grande número de aplicações em duas categorias bem distintas: o aprimoramento de informações pictóricas para interpretação humana; e a análise automática por computador de informações extraídas de uma cena [8]. A segunda categoria que de acordo com [8] pode ser definida como “análise de imagens”, “visão por computador” (ou “visão computacional”) e “reconhecimento de padrões”, o qual é usado neste trabalho. A visão computacional empregada neste trabalho utilizou definições e conceitos largamente difundidos no

campo do processamento digital de imagens como tratamento de imagens, filtros, definição de cores, ruídos dentre outros que serão citados no decorrer do mesmo.

A etapa computacional, que engloba toda a parte de processamento de imagens, inteligência artificial e controle, envolveu a criação de um programa de computador desenvolvido em linguagem de alto nível, o denominado C#, lê-se C-Sharp e é o símbolo que representa o nome da linguagem de programação, da Microsoft. A ferramenta, ou seja, o ambiente de trabalho desta linguagem de alto nível, utilizada para esta tarefa foi o *Visual Studio 2005®,* da *Microsoft®*. Essa ferramenta foi escolhida pela sua grande difusão, facilidade para criação de interfaces, altíssima portabilidade, orientação a objetos, dentre outros. O programa segue a seguinte ordem cronológica: criação de uma interface gráfica; aquisição da imagem; tratamento da imagem; exibição da imagem; sistema de controle; saída de comandos via porta paralela. Também é utilizada a ferramenta *Directx®,* uma ferramenta de software criada para aperfeiçoar o uso dos recursos gráficos que demandam uma grande carga computacional, também é da Microsoft*. O Directx* é muito importante para o processo visual e de tratamento dessas imagens.

Para a automação do veículo inteligente foi utilizado o método da função potencial [2]. Essa função potencial tem relações físicas bem definidas, como por exemplo a energia magnética [2]. Essa função se define como resultado da forca de atração ou repulsão gerada, ou definida, por dois ou mais objetos. Essa força é inversamente proporcional à distância entre esses objetos, ou seja, quanto mais perto maior a força de atração, ou repulsão, exercida entre eles. Na área da física essa “energia” é conhecida como potencial Lenard-Jones [2]. Este método foi implementado de maneira a ser executado em tempo real, ou seja, ele usa iterações por determinado período de tempo. Portanto a automação, do veículo inteligente, é feita baseada nos dados da saída deste método, ou seja, o método retornará, resumidamente, a direção que o veículo deve seguir. Para esta tarefa o sistema computacional precisa utilizar conceitos de *thread (*resumidamente *thread* é a execução de tarefas, podendo por exemplo ser simultâneas, conseguintes, etc.), que é uma ferramenta para processamento em tempo real.

Sistemas de Tempo Real são caracterizados por atender, não só requisitos de correta execução lógica de tarefas, como também a limites de tempo de execução

[9]. Logo, para que uma tarefa em tempo real seja considerada bem sucedida, ela deve ser concluída com sucesso e dentro de um tempo pré-determinado. Um Sistema de Tempo Real pode ser classificado em função das conseqüências oriundas de uma falha no cumprimento dos limites de tempo especificados. Tal classificação está diretamente relacionada com a natureza do elemento (objeto ou sistema) a ser controlado. Segundo este ponto de vista, tem-se a seguinte classificação: *Soft Real Time e Hard Real Time* (ou Sistema de Tempo Real Crítico) [9]. *Soft Real Time* são os sistemas cuja falha no cumprimento dos limites de tempo não acarreta danos e/ou prejuízos significativos, tais como: sistemas que envolvem compartilhamento de voz e de imagem, transações bancárias on-line, dentre outros. Por outro lado, os sistemas ditos *Hard Real Time* (Tempo Real Crítico) são aqueles cujas conseqüências de uma falha no cumprimento dos limites de tempo podem ser catastróficas, ou melhor, o custo de tais falhas é de uma ordem de grandeza que é superior à da própria utilidade do sistema, tais como: controle de processos industriais, controladores de vôo, dentre outros. A ordem de grandeza do tempo de resposta necessário para o sistema está diretamente relacionada com o tipo de equipamento, processo ou sistema a ser controlado. Neste trabalho o sistema pode ser classificado como *Hard Real Time,* dada as características da proposta, onde qualquer atraso na execução de qualquer tarefa pode comprometer a função do veículo autônomo, pois, por exemplo caso o processamento não seja rápido o suficiente ele pode bater em um obstáculo.

Este trabalho será organizado de forma a seguir a ordem da ocorrência dos fatos. Ou seja:

- Inicialmente tem-se uma explanação sobre o simulador;
- É detalhada a teoria empregada sobre processamento de imagens digitais;
- Veículos inteligentes;
- O sistema de controle utilizado;
- Técnicas de inteligência artificial, com destaque para o método da função potencial;
- Aplicação e os resultados;

- A conclusão do trabalho.
- Trabalhos Futuros

2. O Simulador

2.1. Introdução

Este trabalho baseou-se no simulador e no artigo redigido por [25]. Este simulador é, basicamente, um ambiente para implementação de algoritmos de inteligência artificial e, com isso, o teste dos algoritmos de I.A (inteligência artificial) em um "jogo". O jogo possui uma nave espacial (que tem como função desviar de obstáculos), os meteoros (obstáculos), e atingir seu alvo, um planeta (objetivo). Originalmente o simulador utilizava regras de lógica *fuzzy* para essa tarefa, entretanto neste trabalho o algoritmo *fuzzy* foi substituído pelo método da função potencial.

2.2. Funcionalidades do Simulador Original

O desenvolvimento do simulador foi feito em uma plataforma para sistemas distribuídos, abrangendo seu uso. A linguagem utilizada é compatível com o Framework. NET. O sistema funciona como uma dll (Dynamic Link Library) que, assim como uma função, pode ser utilizada por qualquer aplicativo, desde que incluídas no código do aplicativo as chamadas para as mesmas.

Como vantagem a ferramenta desenvolvida apresenta recursos gráficos de configuração da Lógica *fuzzy*, simuladores de resultados, saída e entrada de dados via planilha Excel e o mais importante, métodos abertos para o usuário. Com esta última funcionalidade é possível a interrupção do sistema em qualquer etapa de processamento podendo-se alterar ou implementar novas soluções. Desta forma o sistema pode ser utilizado para auxiliar quaisquer aplicações de Lógica *Fuzzy* tornando-o uma poderosa ferramenta de auxilio e desenvolvimento de soluções

computacionais para sua utilização em situações de incertezas e para controle de processos.

Para demonstrar o uso da ferramenta, um jogo foi implementado. A simulação consta de um sistema responsável por definir a direção que uma nave deve seguir com o objetivo de chegar à Terra. Entretanto existem alguns obstáculos - os meteoros. A nave se orienta segundo regras inseridas no algoritmo de inteligência artificial, a lógica *fuzzy*. O desenvolvimento da aplicação será feito com uma plataforma para sistemas distribuídos, alcançando uma maior confiabilidade. A linguagem utilizada foi o Visual Basic .NET. Para a simulação a linguagem utilizada foi o C #, ambas proprietárias Microsoft.

2.3. A arquitetura para a Construção do Jogo

A aplicação foi desenvolvida, através de um programa escrito em linguagem C#. Nela utilizou-se a biblioteca *Fuzzy* aproveitando as funções que disponibiliza. O exemplo foi a criação de um jogo de nave espacial em 2D cuja inteligência é toda baseada em Lógica *Fuzzy*, denominado *Collision Game*.

O sistema que gerencia a inteligência artificial do jogo é composto por quatro variáveis de entrada e uma de saída. As variáveis de entrada dividem-se em duas categorias. Categoria “MET” e “ALVO”. A variável de saída está em uma terceira categoria, denominada “OUT”.

As variáveis da categoria “MET” são: Direção_met e Distância_met. A primeira indica a direção do meteoro, a segunda indica a distância em relação à nave. As variáveis da categoria “ALVO” são: Direção_alvo e Distancia_alvo, por fim da categoria “OUT” tem-se a variável Saida_out, responsável pela resposta do sistema.

Como mencionado, é no sistema Fuzzy que se entra com as grandezas, *memberships*, que são os membros e suas funcionalidades do sistema [25], e regras de controle. Empregamos este sistema para entrar graficamente de forma eficaz

com os dados necessários. Este gera um arquivo de dados com as variáveis e regras.

O arquivo é importado pela aplicação *Collision Game*, a qual emprega a biblioteca *fuzzy* utilizando suas funções para controlar o jogo.

A seguir apresentar-se os passos que devem ser realizados no sistema *Fuzzy.* Também serão ilustradas todas as possibilidades do Sistema.

- Passos que devem ser realizados no Sistema Fuzzy:

1. Criar novo sistema.

2. Definir categoria.

3. Definir variáveis de entrada, saída, determinando valor máximo, mínimo, tipo e unidade destas grandezas.

4. Definir *memberships* de cada variável.

5. Definir regras de atuação.

2.4. O Jogo - Aplicação Collision Game

A finalidade deste jogo é aplicar a biblioteca contendo a matemática da Lógica Fuzzy. Para o desenvolvimento da aplicação foram empregados:

- A ferramenta de desenvolvimento da Lógica Fuzzy (Sistema Fuzzy) apresentada nas seções anteriores.

- A biblioteca Fuzzy, que foi a .dll (*direct link library*) gerada. A .dll é a biblioteca com as definições e métodos de um programa ou parte dele, para uma "ligação", *link*, com algum código.

- A linguagem de programação C#, utilizada como ferramenta para criação do sistema proposto.

O jogo é constituído por uma nave espacial, cujo objetivo é chegar à Terra. Entretanto existem os meteoros (obstáculos). Para chegar ao seu objetivo, a nave precisa desviar dos meteoros. Portanto o jogo trata-se de controlar a posição da nave de modo que ela possa atingir seu objetivo. Se a nave colidir com algum obstáculo, gera uma explosão. Caracteriza-se como o fim do jogo quando a nave atinge a Terra.

Como característica do jogo pode-se destacar que a posição inicial da nave é sempre a mesma. A velocidade tem um valor inicial (V0). Para acelerar ou frear, as setas do teclado devem ser pressionadas, no controle manual do jogo, (direita: acelerar, esquerda: frear, seta para cima: virar à esquerda, para baixo: virar à direita). Entretanto, para as simulações a nave orienta-se segundo regras Fuzzy.

A direção que a nave espacial deve seguir é calculada de acordo com a distância e posição em relação à Terra e aos meteoros. A nave desvia dos meteoros de acordo com a direção e distancia dos mesmos. A nave segue sempre na direção contrária em relação à posição dos meteoros. O desvio lateral é dado pela distância, quanto mais perto, mais a nave vira. Como tem o objetivo chegar à Terra, a nave mira o alvo e vira de acordo com a posição que esta em relação a ele.

O sistema é composto de quatro classes, três responsáveis pela aplicação em si, e a última responsável pelo método de defuzzificação, definido em [25], que é usada para o cálculo do valor de saída. Esta é gerada pelos valores obtidos nas conclusões das várias regras, que são agregados em uma única ação de controle através de uma média ponderada, como mostra a equação abaixo. Os pesos são valores que mostram o grau de compatibilidade do valor.

$$M = \frac{\left(\sum (peso * valor)\right)}{\sum peso}$$

Equação - 1 [25]: Valor de saída da defuzzuficação.

2.5. A Substituição do Fuzzy pelo Método Potencial

Neste trabalho optou-se pela substituição da lógica fuzzy pelo método da função potencial pelos seguintes fatores:

- A função potencial é simples algoritmo que tem a função de perseguir e desviar ou seja, não é preciso outras condições e controles lógicos associados a este algoritmo. O fuzzy envolve várias regras e lógicas associada aos controles de entrada e saída;

- A demanda computacional é extremamente baixa, como a função potencial representa um algoritmo simples, a matemática envolvida é rapidamente calculada ;

- É muito simples a implementação, é uma simples função do 2º. grau.

Diferente do simulador original, nesta versão modificada, o veiculo pode ser iniciado de qualquer posição, esta é escolhida de maneira casual e aleatória. O capitulo 6 descreve as funcionalidades entre outros detalhes deste método potencial, como por exemplo suas regras e requisitos. Nas Figuras 1, 2 e 3 observa-se o simulador sendo executado na largada, no meio do percurso e na chegada ao alvo, respectivamente.

Figura 1 – Nave espacial na Largada

Figura 2 – Nave espacial no meio do percurso

Figura 3 – Nave chegando ao objetivo

3. Processamento Digital de Imagens

3.1. Introdução

O Processamento de imagens é certamente uma área em crescimento. O rápido progresso na indústria de computadores, observado nos últimos tempos, fez com que o acesso a essas máquinas por mais e mais pessoas tenha aumentado em grande escala. CPUs estão cada vez mais rápidas e mais baratas, possibilitando que pesquisas de uma forma geral sejam beneficiadas, devido a maior disponibilidade de equipamentos. Diversos temas científicos são abordados e em alguns casos de caráter interdisciplinar. Entre eles a compreensão de imagens, a análise em multi-resolução e em multi-frequência, a análise estatística, a codificação e a transmissão de imagens, etc [4]. A disciplina “processamento de imagens” vem na realidade do Processamento de Sinais. Os sinais, como as imagens, são, na realidade, um fenômeno físico que traz no seu interior uma determinada informação. Esta informação pode estar associada a uma medida (neste caso o sinal é representado por um fenômeno físico), ou pode estar associada a um nível cognitivo (neste caso o sinal é representado por uma forma de conhecimento). Processar uma imagem consiste em transformá-la sucessivamente, por exemplo, com a aplicação de filtro de ruídos, filtro de cores [4], com o objetivo de extrair mais facilmente as informações nela presentes.

Cabe neste momento fazer uma comparação entre o Processamento de Imagem e a Computação Gráfica. A computação gráfica é uma técnica aplicada através de seqüências animadas na televisão ou em filmes de cinema. A Computação Gráfica pode ser definida como a criação, gravação e manipulação de modelos de objetos e subseqüente imagens, através de computador e dispositivos de interação. Ela parte de uma informação para obter uma imagem ou um filme, aqui é o termo filme é irrelevante, visto que um filme é uma seqüência de imagens. O Processamento de Imagens parte da imagem (de uma informação inicial que é

geralmente capturada por uma câmera) ou de uma seqüência de imagens para obter a “informação” desejada por exemplo, a posição de determinado objeto no espaço [10]. Deste ponto de vista o Processamento de Imagens e a Computação Gráfica são exatamente métodos opostos, mas isto não quer dizer que as técnicas envolvidas em cada caso não possam ser as mesmas ou pelo menos complementares [4, 10]. É evidente que neste sentido processar uma imagem, como é feito pelo sistema visual humano (SVH), é extremamente complexo [4, 10]. Realizar as mesmas tarefas que o SVH, com a ajuda de máquinas, exige por antecedência uma compreensão “filosófica” do mundo ou dos conhecimentos humanos. Esta característica faz com que o processamento de imagens seja uma disciplina com extrema dependência do sistema no qual ele está associado, não existindo, no entanto, uma solução única e funcional para todos os problemas. Daí a não existência, até o momento, de sistemas de análise de imagens complexos e que funcionem para todos os casos. A analise quantitativa e a interpretação de imagens representa atualmente um ponto de apoio importante em diversas disciplinas científicas. Tal é o caso, por exemplo, na ciência dos materiais, onde o computador pode identificar diferentes composições químicas de elementos, na medicina, onde, por exemplo, pode-se identificar através da coloração o tipo sanguíneo de uma pessoa, na engenharia, onde através de uma imagem podem-se obter os modelos estruturais de um projeto, dentre outras áreas do conhecimento. Na realidade, a diversidade de aplicações do processamento de imagens está associada diretamente a análise da informação [10]. Em todas estas disciplinas existe a busca por informações quantitativas que representem um fenômeno estudado. Do ponto de vista da óptica, uma imagem é um conjunto de pontos que convergem para formar um todo, mas pode-se dizer de uma maneira mais ampla que uma imagem é o suporte para efetuar-se a troca de informações entre a imagem e o computador [10]. O termo imagem estava inicialmente associado ao domínio da luz visível, porém atualmente é muito freqüente falar de imagens quando uma grande quantidade de dados está representada sob a forma bidimensional (por exemplo: as imagens acústicas, sísmicas, de satélites, infravermelhas, magnéticas, etc). Os métodos recentes de exploração automática desta informação permitiram o desenvolvimento de técnicas complexas, que podem ser globalmente classificadas em duas grandes linhas. A primeira está associada a uma análise da informação e a segunda

representa as técnicas que permitem obter uma melhoria (do termo em inglês "*enhancement*") significativa da imagem [10].

O termo análise está relacionada a parte do tratamento onde existe uma descrição da informação presente na imagem. Esta parte é chamada de parametrização e é nela que várias medidas quantitativas (parâmetros) são utilizadas para descrever diferentes informações dentro de uma imagem [10]. Algumas aplicações típicas são: a determinação do número de células presentes em um tecido biológico, o cálculo das formas dos contornos de uma célula ou ainda a determinação da distribuição de uma população específica de um conjunto de células.

As técnicas dedicadas à análise de imagens podem variar significativamente segundo a sua complexidade e a necessidade em tempo de processamento. Nesta área de analise de imagens encontra-se um nível elevado de complexidade no tratamento da informação. Um exemplo prático pode ser visto, por exemplo, na classificação automática de células doentes dentro de um conjunto de células observadas em microscopia [10]. Esta análise específica demanda soluções dadas pelas técnicas de "classificação e reconhecimento de formas". Neste caso é necessário medir vários parâmetros, pertinentes ao problema, na imagem, como por exemplo: a superfície, a forma de cada célula, sua quantidade, o número de células vizinhas a uma dada célula e a densidade de células em uma dada região. Destas medidas, com várias classes de células organizadas em uma base de dados, catalogadas anteriormente, obtêm-se então uma classificação das células com uma dada probabilidade de serem células doentes ou normais.

O termo "*enhancement*" está associado à melhoria da qualidade de uma imagem, com o objetivo posterior de ser julgado por um observador humano. Alguns exemplos deste tipo de técnica são: a subtração da imagem por uma imagem referência, por exemplo, com o intuito de identificar algum objeto diferente naquela imagem, ou seja, acrescentado depois; a utilização de cores-falsas, para, por exemplo, forçar a diferenciação entre determinados pontos da imagem; a utilização de filtros passa-alta ou passa-baixa [4, 10], com o objetivo de retirar ruídos indesejados na imagem; a correção de deformações espaciais devido a óptica ou devido a uma variação da iluminação de fundo, por exemplo, para uma real

identificação do objeto em questão na imagem [10]. Os sistemas dedicados a melhorar a qualidade da imagem trabalham, geralmente, muito rápido, pois são construídos em "*hardware*", ou seja o hardware desses computadores, dedicados ao processamento de imagens, são totalmente dimensionados para esta tarefa, como exemplo as placas de vídeo 3D que fazem a vetorização das imagens por *hardware*. Essa velocidade permite ao usuário um julgamento sobre várias imagens processadas, segundo o tipo de tratamento. Praticamente todos os programas, existentes hoje no mercado, de tratamento de imagens trabalham com algumas funcionalidades abstratas via hardware, por exemplo a parte de vetorização 2D ou 3D. O DirectX é o software responsável por essa interação com o hardware.

Neste trabalho o interesse se concentra em procedimentos para extrair de uma imagem informação de uma forma adequada para o processamento computacional. Frequentemente essa informação apresenta poucas semelhanças com as características utilizadas pelo homem na interpretação do conteúdo de uma imagem. Isso pode ser observado em, por exemplo, uma imagem binária onde um computador estará processando uma cadeia de 0´s e 1´s, que para ele não passará de números. Contudo entende-se que esse cadeia de 0´s e 1´s representará, por exemplo, imagem e não imagem respectivamente.

Problemas típicos em percepção por máquina, que rotineiramente usam técnicas de processamento de imagens, são, o reconhecimento automático de caracteres, visão computacional industrial para a montagem e inspeção de produtos, reconhecimento militar, processamento automático de impressões digitais, analise de resultados de raios X e amostras de sangue em tela, processamento de imagens aéreas e de satélites para previsão do tempo e monitoração do plantio [4]. Neste trabalho o problema de percepção por maquina será o reconhecimento automático de todos os objetos na área de testes, ou seja o veículo inteligente, os obstáculos e o alvo, extraindo desses objetos as informações relativas a área ocupada por eles e à posição relativa na área de testes. O processamento de imagens digitais abrange uma ampla escala de hardware, software e fundamentos teóricos e, portanto essa técnica é fundamental para o sucesso do trabalho proposto.

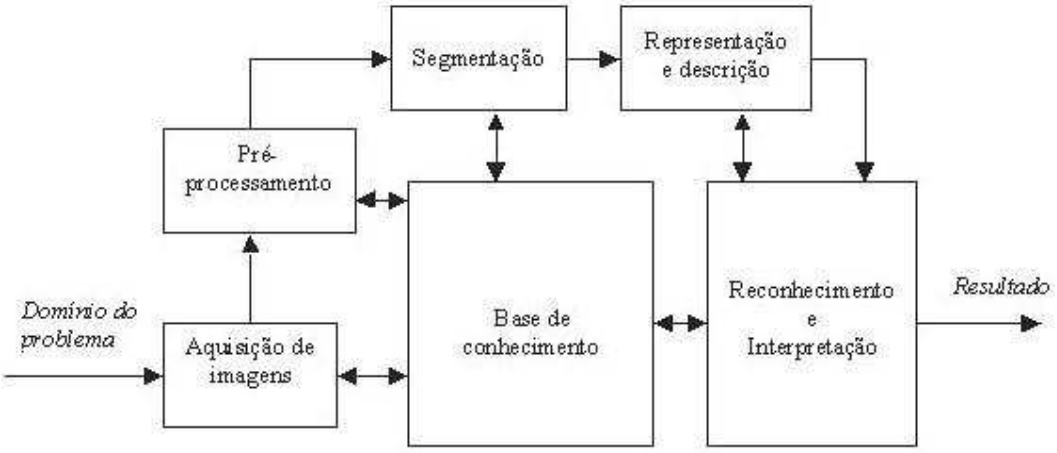

Figura 4 – Passos fundamentais em processamento de imagens digitais.

A figura 4, de acordo com [4], mostra que o objetivo do processamento de imagens digitais (PDI) é produzir um resultado a partir do domínio do problema por meio de processamento de imagens. O domínio do problema é a identificação dos objetos carro, bola e etc. O primeiro passo do PDI é a aquisição da imagem – isto é, adquirir uma imagem digital. Neste trabalho isso é feito através de uma câmera de alta definição conectada diretamente ao computador, através de um cabo coaxial de duas vias, comum, como mostra a figura 5.

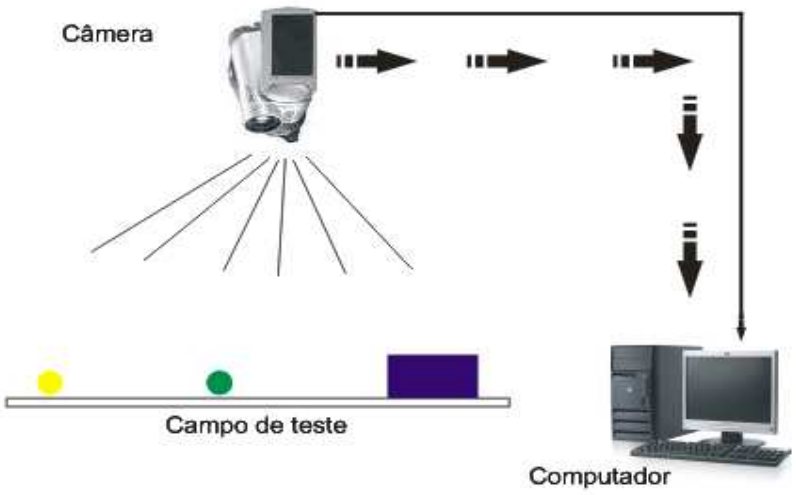

Figura 5 – Esquema de aquisição da imagem digital.

Após a aquisição da imagem digital, o próximo passo trata de pré-processar a imagem. A função chave no pré-processamento é melhorar a imagem de forma a aumentar as chances para o sucesso dos processos seguintes. Neste trabalho o pré-processamento consiste em filtrar a imagem, remover ruídos e isolar as regiões com maior probabilidade de informações.

O próximo estágio trata da segmentação. Definida em termos gerais, a segmentação divide uma imagem de entrada em partes ou objetos constituintes [4]. Em geral, a segmentação automática é uma das tarefas mais difíceis no processamento de imagens digitais [4]. Por um lado, um procedimento de segmentação robusto favorece substancialmente a solução de uma maneira bastante satisfatória para um problema de imageamento [4]. Em contrapartida algoritmos de segmentação fracos ou erráticos quase sempre ocasionam falhas no processamento. No caso deste trabalho, o papel básico da segmentação é extrair os objetos individualmente. Objetos esses que serão subdivididos em carro, obstáculo 1, obstáculo 2 e alvo.

A saída do estágio no processo de segmentação é constituída tipicamente por dados na forma de *pixels* ("*raw pixel data*"), correspondendo tanto a fronteira de uma região como a todos os pontos dentro da mesma, ou seja, isso significa que a região formada pelo objeto, e sua fronteira têm um valor definido X e todas as outras regiões não pertencentes a esse objeto, ou a sua fronteira, têm um valor diferente de X. Em ambos os casos são necessárias conversões dos dados para uma forma adequada ao processamento computacional. A primeira pergunta que deve ser feita é se os dados devem ser representados como fronteiras ou regiões completas. A representação por fronteira é adequada quando o interesse se concentra nas características da forma externa, tais como cantos ou pontos de inflexão. A representação por região é adequada quando o interesse se concentra em propriedades internas, tais como a textura ou a forma [4]. Em algumas aplicações, entretanto, essas representações coexistem, como é o caso deste trabalho.

A escolha de uma representação é apenas parte da solução para transformar os dados iniciais em uma forma adequada para o subseqüente

processamento computacional. Um método para descrever os dados também deve ser especificado, de forma que as características de interesse sejam destacadas e expostas. O processo de descrição também chamado seleção de características, procura extrair características que resultem em alguma informação quantitativa de interesse ou que seja básica para a discriminação entre classes de objetos. As cores têm esse papel de "características" de identificação dos objetos.

O último estágio mostrado na figura 4 envolve reconhecimento e interpretação. Reconhecimento é o processo que atribui um rótulo ao objeto, baseado na informação fornecida pelo seu descritor. A interpretação envolve a atribuição de significado a um conjunto de objetos reconhecidos. A cor azul, por exemplo, representará o objeto "carro", a cor vermelha a frente do carro, a cor verde representará o objeto obstáculo, e a amarela o objetivo.

A base de conhecimento é a informação que necessita extrair para que o problema seja solucionado, ou seja, nela estão contidos dados para que a interpretação seja guiada a certa referência [4]. Como exemplo pode-se dizer que, neste caso, a base de conhecimento possui informações referentes à cor azul associada ao objeto carro, ou seja, para o sistema de processamento de imagens a cor azul significa carro.

3.2. Imagens

De acordo com [4, 8], o termo imagem refere-se a uma função de intensidade luminosa bidimensional denotada por *f(x, y),* em que o valor da amplitude de f nas coordenadas espaciais *(x, y)* dá a intensidade (brilho) da imagem naquele ponto. Como a luz é uma forma de energia*, f(x, y)* deve ser positiva e finita, isto é,

$0 < f(x, y) < \infty$ *(*Equação 3.2.1*).*

Complementando a definição, segundo [10], uma imagem digital é uma imagem *f(x, y)* discretizada tanto em coordenadas espaciais quanto em brilho. Uma imagem digital pode ser considerada como sendo uma matriz cujos índices de linhas

e de colunas identificam um ponto na imagem, e o correspondente valor do elemento da matriz identifica o nível de cinza naquele ponto. Os elementos dessa matriz digital são chamados de elementos da imagem, elementos da figura*, "pixels"* ou *"pels",* estes dois últimos, abreviações de *"picture elements"* (elementos de figura). Quanto mais *pixels* uma imagem tiver melhor é a sua resolução e qualidade. A convenção dos eixos para representação de imagens digitais no Processamento de Imagens é diferente da convenção usada na Computação Gráfica. Como mostrado na figura 6.

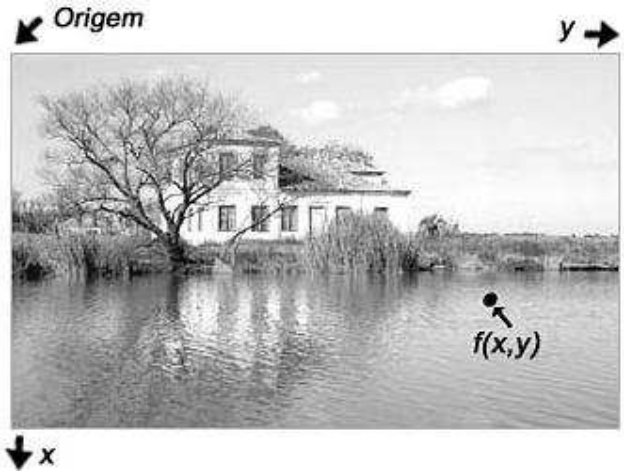

Figura 6 – Convenção dos eixos para representação de imagens digitais.

Para ser adequada ao processamento computacional, a função *f(x, y),* precisa ser digitalizada tanto espacialmente quanto em amplitude. A digitalização das coordenadas espaciais (x, y) é denominada amostragem da imagem e a digitalização da amplitude é chamada quantização em níveis de cinza.

Suponha que uma imagem contínua *f(x, y*) é aproximada por amostras igualmente espaçadas, arranjadas na forma de uma matriz N x M, em que cada elemento é uma quantidade discreta. O lado direito dessa equação representa o que normalmente é denominado imagem digital. Os termos imagem e *pixels* são usados a seguir para denotar uma imagem digital e seus elementos.

Em certas ocasiões, torna-se útil exprimir amostragens e quantização em termos matemáticos mais formais. Sejam Z e R os conjuntos dos números inteiros e

reais, respectivamente. O processo de amostragem pode ser compreendido como a partição do plano *xy* em uma grade, com coordenadas de cada cruzamento da grade sendo um par de elementos obtidos do produto cartesiano Z x Z (também representado por Z^2), que é o conjunto de todos os pares ordenados (a, b) com a e b sendo elementos de Z. Portanto *f(x, y)* é uma imagem digital se (x, y) forem elementos de Z x Z e *f* uma função que atribui valor de nível de cinza (isto é, um numero real) a cada par de coordenadas (x, y) distinto. Essa atribuição funcional é obviamente o processo de quantização descrito anteriormente. Se os níveis de cinza são também inteiros, Z toma o lugar de R, e uma imagem digital torna-se então uma função bidimensional (2-D), com valores inteiros de coordenada e amplitude.

Esse processo de digitalização envolve decisões a respeito dos valores para N, M e o numero de níveis de cinza discretos permitidos para cada *pixel.* A prática comum em processamento de imagens digitais é assumir que essas quantidades são potencias inteiras de dois; isto é:

$N = 2^n$, $M = 2^k$ (Equação 3.2.2) e $G = 2^m$ (Equação 3.2.3).

Em que G é o numero de níveis de cinza. O número, B, de bits necessários para armazenar uma imagem digitalizada é obtido pela equação 3.2.2:

B = N x M x m. (Equação 3.2.4).

Por exemplo, uma imagem de 128 x 128 *pixels* com 64 níveis de cinza requer 98.304 bits para armazenamento.

3.3. Os vizinhos de um *pixel*

Conforme definição de [4], um *pixel p* nas coordenadas (x, y), possui quatro vizinhos, dois horizontais e dois verticais, cujas coordenadas são dadas por:

(x + 1, y), (x – 1, y), (x, y + 1), (x, y -1). (Equação 3.3.1)

Esse conjunto chamado de vizinhança 4 de p. é representado por $N_4(p)$, conforme a figura 7.

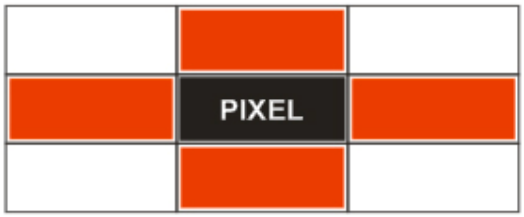

Figura 7 – Vizinhança 4.

Cada *pixel* está a uma unidade de distancia (x, y), sendo que alguns dos vizinhos de p estarão fora da imagem se (x, y) estiver na borda da imagem, como ilustra a figura 8.

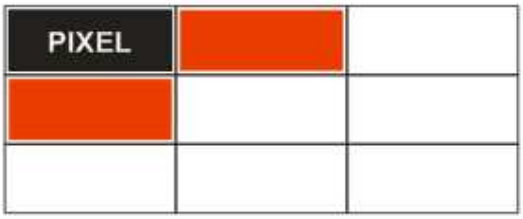

Figura 8 – Vizinhança 4 com vizinhos fora da imagem.

Os quatro vizinhos diagonais de p possuem como coordenadas:

(x + 1, y + 1), (x – 1, y+1), (x+1, y - 1), (x - 1, y -1). Neste caso representado na figura 9 pelos blocos na cor cinza.

São denotados por $N_D(p)$. Esses pontos, unidos com a vizinhança 4, são chamados de vizinhança 8 de p, representada por $N_8(p)$, vizinhança 8, conforme figura 9, onde todos os blocos, vermelhos e cinzas são os *pixels* vizinhos na N_8.

Figura 9 – Vizinhança 8.

Como antes alguns pontos de $N_D(p)$ e $N_8(p)$ permanecerão fora da imagem quando (x, y) se encontrar na borda da imagem, como observado na figura 8.

3.4. Conectividade

A conectividade entre *pixels* é um conceito importante usado no estabelecimento de bordas de objetos e componentes de regiões em uma imagem. De acordo com [4, 8] para estabelecer se dois *pixels* estão conectados, é preciso determinar se eles são, de alguma forma, adjacentes e se seus níveis de cinza satisfazem um certo critério de similaridade.

Em uma imagem binária, por exemplo, dois *pixels* podem ser de vizinhança 4, mas somente serão conectados se possuírem o mesmo valor de níveis de cinza. A conectividade deve ser definida segundo alguns parâmetros, por exemplo se um pixel tem alguma relação de vizinhança com outro. Existem varias definições de conectividade, mas elas podem ser, para este caso, como visto anteriormente, 4-Conectado, 8-Conectado.

Para a detecção destes elementos, pixels, pode-se utilizar, por exemplo, operações de convolução (a partir de duas funções produz-se uma terceira) [4, 8] com máscaras que são operações orientadas à vizinhança bastante utilizadas no processamento de imagens digitais. A máscara percorre a imagem desde o seu canto superior esquerdo até seu canto inferior direito. A cada posição relativa da máscara sobre a imagem, o *pixel* central da sub-área da imagem no momento será substituído pelo valor calculado na operação, esta operação é denominada de convolução. A idéia das operações com máscara é fazer com que o valor atribuído a um *pixel* seja uma função de seu nível de cinza e do nível de cinza dos seus

vizinhos, com o objetivo de determinar a conexão, ou não, de cada *pixel*, por exemplo, seja uma sub-área de uma imagem onde z_1 até z_9 são os valores de tons de cinza de cada *pixel*. Conforme ilustrado na tabela 1.

Tabela 1 – Valores do tom de cinza de cada *pixel*.

z_1	z_2	z_3	...
z_4	z_5	z_6	...
z_7	z_8	z_9	...
...	...	...	

Pode-se aplicar uma operação com máscara utilizando essa sub-área de imagem substituindo por exemplo o valor de z_5 pela média dos pixels desta sub-área. Para tal realiza-se a seguinte operação:

$z_5 = (z_1 + z_2 + \ldots + z_9)/9$ (Equação 3.4.1)

Pode-se também realizar uma média ponderada utilizando para tal uma máscara com um peso específico para cada *pixel* da sub-área.

Tabela 2 – Valores de média ponderada para usar como máscara.

p_1	p_2	p_3
p_4	p_5	p_6
p_7	p_8	p_9

Neste caso a operação seria:

$$z_5 = (z_1 + z_2 + \ldots + z_9) = \sum_{i=1}^{9} p_i z_i \qquad \text{(Equação 3.4.2)}$$

Se $p_i = 1/9$ esta operação tem o mesmo resultado da operação anterior.

Esta equação é bastante utilizada em processamento digital de imagens nos casos de testes de conectividade. A seleção apropriada de coeficientes e a possibilidade de aplicação da máscara em cada *pixel* na imagem torna possível uma variedade de operações úteis como redução de ruído, afinamento de regiões e detecção de bordas [4, 8], pois, a partir dos resultados, pode-se obter as comparações necessárias para determinar, por exemplo o que é borda e o que não é, o que é ruído e o que é imagem.

O deslocamento da máscara (convolução) sobre toda a imagem é uma tarefa que exige bastante esforço computacional. Por exemplo, aplicando uma máscara 3 x 3 em uma imagem 512 x 512 requer 9 multiplicações e oito adições para cada pixel totalizando 2.359.296 multiplicações e 2.097.152 adições.

- Detecção dos pontos isolados:

Em [4] tem-se a máscara a seguir que é um exemplo de operador de convolução que, quando aplicado a uma imagem, destacará pixels brilhantes circundados por pixels mais escuros. Este operador corresponde a um filtro passa-altas, que é um filtro que permite a passagem de freqüências acima de certo valor.

$$\begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix}$$

Máscara como exemplo de operador de convolução.

- Detecção de Linhas:

As máscaras a seguir, retiradas de [4], podem ser usadas para a detecção de linhas horizontais e verticais (acima) e diagonais (abaixo).

$$\begin{bmatrix} -1 & -1 & -1 \\ 2 & 2 & 2 \\ -1 & -1 & -1 \end{bmatrix} \quad \begin{bmatrix} -1 & 2 & -1 \\ -1 & 2 & -1 \\ -1 & 2 & -1 \end{bmatrix}$$

$$\begin{bmatrix} -1 & -1 & 2 \\ -1 & 2 & -1 \\ 2 & -1 & -1 \end{bmatrix} \quad \begin{bmatrix} 2 & -1 & -1 \\ -1 & 2 & -1 \\ -1 & -1 & 2 \end{bmatrix}$$

- Detecção de Bordas:

Detecção de bordas (*edge detection*), a saber, vem desafiando os pesquisadores da área de processamento de imagens há muitos anos e sobre ele continuam sendo experimentadas novas técnicas, cujos resultados são publicados ainda hoje nos mais conceituados periódicos científicos mundiais. Trata-se, portanto, de um tema em aberto, pois as tarefas são consideradas “difíceis” [4]. Neste trabalho não é diferente, a “borda” dos objetos é a região mais crítica, pois é a mais difícil de ser identificada devido a vários fatores como, por exemplo, a refração e a reflexão da luz.

3.5. Filtros

Em processamento de imagens, os filtros lineares são geralmente descritos através de matrizes de "convolução” de acordo com [4, 8]. Um filtro numérico influencia a variação da freqüência espacial em uma imagem. Na freqüência temporal a escala usada é geralmente o Hertz (S^{-1}), em uma imagem é usada o 1/metro (m^{-1}) ou 1/pels (pix^{-1}). O termo freqüência espacial é análoga ao termo freqüência temporal e ela descreve a velocidade de modificação de uma luminosidade em uma direção em uma imagem.

Na prática, é necessário escolher uma matriz de dimensão n x n com valores que dependem do filtro que será utilizado, seja ele passa baixa (filtrando as altas freqüências), passa faixa (filtrando uma região específica de freqüências espaciais) ou passa alta (filtrando as baixas freqüências). Em uma imagem, as altas freqüências correspondem as modificações abruptas dos níveis de cinza, i.e., as

bordas dos objetos. As baixas freqüências correspondem às variações suaves dos níveis de cinza. Para evidenciar os contornos de um determinado objeto são utilizados filtros do tipo passa-alta. Em outros casos o interesse pode ser na forma da iluminação de fundo, onde, para isso, é utilizado filtros passa-baixa para eliminar todas as altas freqüências correspondendo a borda dos objetos, e chegar a iluminação de fundo.

3.6. "Bounding Box"

A definição de *bounding box*, ou "caixas envolventes", em uma tradução literal, pode ser entendida a partir de seu nome. Em imagens digitais, o espectro de freqüência de uma área onde há objeto e outra que não existe objeto, fornece a definição de bordas daquele objeto. Seguindo essa borda delimitando um elemento pode-se definir o *bounding box* desse elemento, ou seja, o *bounding box* é a delimitação do que envolve a imagem propriamente dita [4, 8]. Observe o exemplo na figura 10.

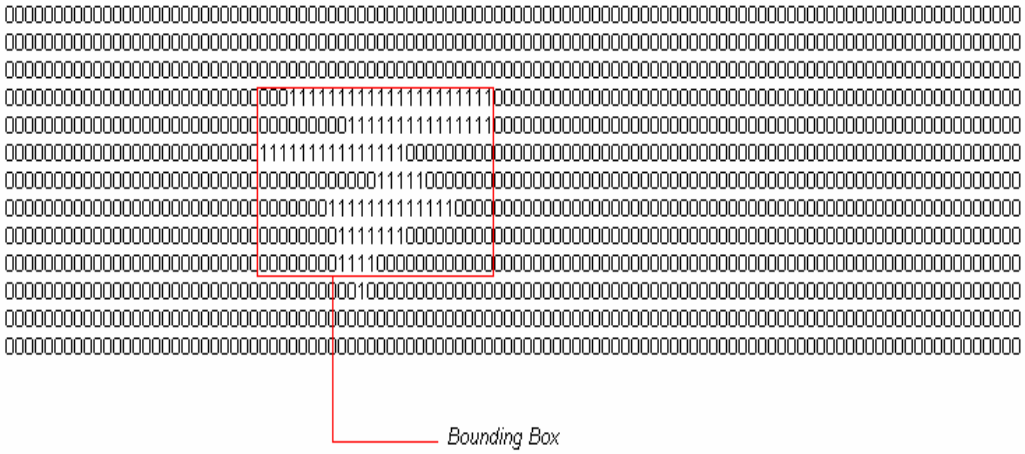

Figura 10 – Exemplo de bits da *bounding box*

3.7. Histograma

O histograma de uma imagem indica o percentual de pixels naquela imagem, que apresentam um determinado nível de cinza [4 e 8]. Estes valores são normalmente representados por um gráfico de barras que fornece para cada nível de

cinza um número (ou percentual) de *pixels* correspondentes na imagem, conforme ilustra a figura 11.

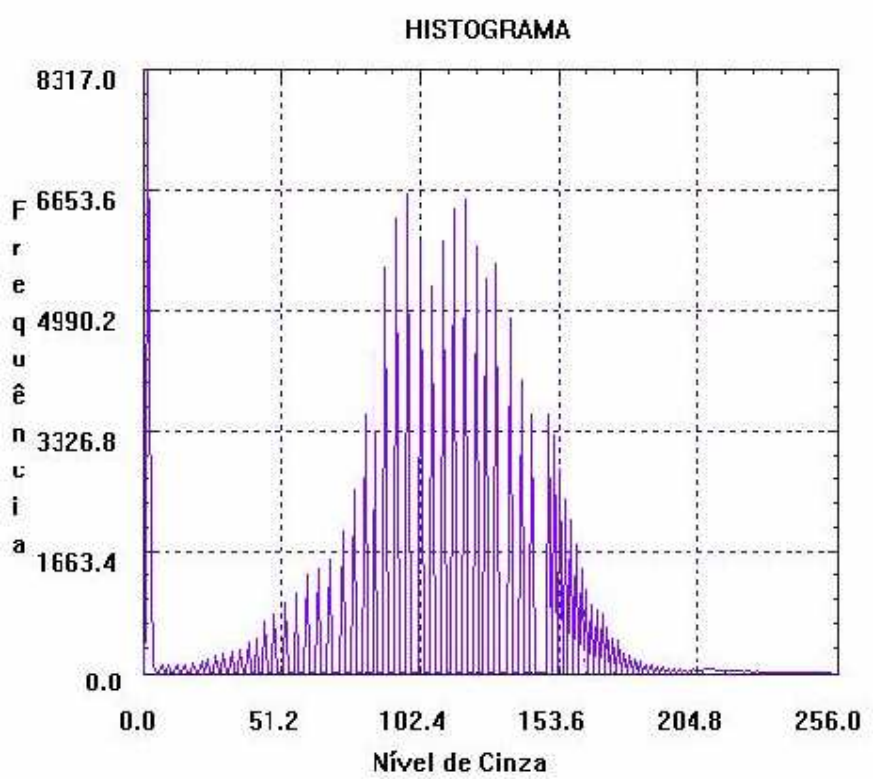

Figura 11 – Exemplo de um histograma.

Através da visualização no histograma de uma imagem obtêm-se uma indicação de sua qualidade quanto ao nível de contraste e quanto ao seu brilho médio, se a imagem é predominante clara ou escura, ou seja, quanto maior for a tendência do histograma em determinados valores pode-se ter uma idéia se a imagem tende a ser mais clara ou mais escura, ou seja pela disposição da freqüência dos pontos no histograma podemos, por exemplo, dizer se ele tem mais "regiões" brancas, sem objetos, ou não. Ele pode ser calculado, de acordo com [4, 8], como:

$$\Pr(Rk) = \frac{nk}{n} \quad \text{(Equação 3.7.1)}$$

Onde:

Pr = Probabilidade do k-ésimo nível de cinza;

nk = número de pixels cujo nível de cinza corresponde a k;

n =número total de pixels na imagem.

Um histograma apresenta várias características importantes. A primeira delas é que cada Pr(*rk*) fornece, como sugere a notação, a probabilidade de um pixel da imagem apresentar nível de cinza *rk*. Portanto um histograma nada mais é que uma função de distribuição de probabilidades e como tal deve obedecer aos axiomas e teoremas da teoria da probabilidade [4]. Por exemplo, a soma dos valores Pr(*rk*) é 1, o que já era esperado. Na figura 12 é mostrado o histograma real do campo de testes.

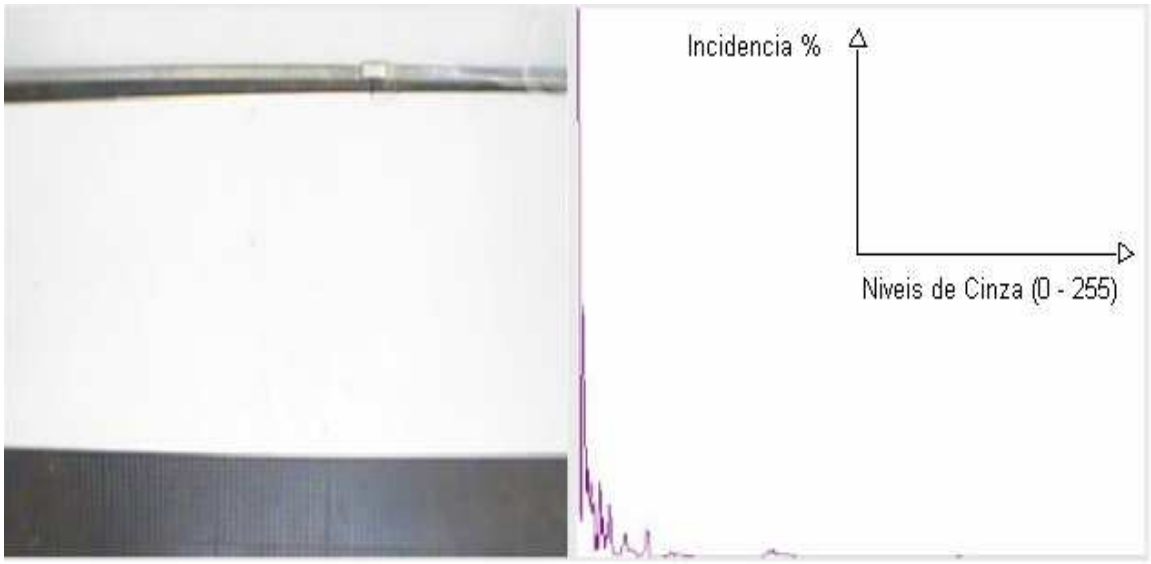

Figura 12 – Imagem do campo de testes em “aberto” e seu respectivo histograma.

3.8. Processamento de Imagens Coloridas - Padrão de Cores RGB e HSB/HSV

O uso de Cores em processamento de imagens é motivado por dois fatores principais: primeiramente em análise de imagens automatizadas, a cor é um descritor poderoso que frequentemente simplifica a identificação do objeto e a extração de uma cena [4]. Em segundo lugar, em análise de imagens

desempenhada por seres humanos, a motivação para o uso de cores é que o olho humano pode discernir milhares de tons e intensidades de cores, comparando a cerca de apenas duas dúzias de tons de cinza [4]. Em compensação o uso de cores, devido a suas diversas possibilidades, torna o processamento mais complexo, ou seja, será mais “trabalhoso” para o computador fazer essa identificação de cores do que, por exemplo, trabalhar com imagens binárias (preto e branco). Contudo devido a grande capacidade de hardware e software disponível hoje em dia este “problema” é facilmente contornável, fazendo do computador um ótimo “olho humano eletrônico”.

O processamento de imagens coloridas é dividido em duas áreas principais: processamento de cores reais e pseudo-cores [4]. Na primeira Categoria as imagens são tipicamente adquiridas com um sensor de cores reais, tal como uma câmera colorida ou um scanner colorido. Na segunda categoria, o problema é a atribuição de um tom de cor para uma intensidade monocromática particular ou a uma variação de intensidades. Até recentemente, grande parte de processamento de imagens coloridas era feito com pseudo-cores. O progresso significativo feito nos anos 80 tornou sensores de cores e hardware para processamento de imagens coloridas disponíveis a preços razoáveis [4]. Como resultado desses avanços, técnicas de processamento de imagens em cores reais estão se tornando significativas numa ampla escala de aplicações, como é o caso deste trabalho.

É muito importante a compreensão de como cada cor é descrita e utilizada por diversos aplicativos e dispositivos, sejam eles de captura como scanners e maquinas digitais, de visualização, como monitores e projetores ou mesmo dispositivos de impressão como impressoras digitais, rotativas entre tantas outras. Cada um dos dispositivos trabalha com uma linguagem própria, descrevendo e utilizando a cor a partir de um modelo; os modelos de cores mais conhecidos são: RGB (vermelho, verde, azul), HSB (matiz, saturação, brilho), HSV (é o mesmo do HSB é apenas no nome: matiz, saturação, valor), CMYK (*ciano*, *magenta*, amarelo e preto) e CIE L*a*b* (L sendo para luminosidade e os valores de a* e b* para dois componentes cromáticos). Neste trabalho serão utilizados os padrões *RGB e HSB/HSV.*

RGB é a abreviatura do sistema de cores aditivas formado por Vermelho (*Red*), Verde (*Green*) e Azul (*Blue*) [4 e 8]. É o sistema aditivo de cores, ou seja, de projeções de luz, como monitores e *datashows*, A escala de RGB varia de 0 (mais escuro) a 255 (mais claro). Nos programas de edição de imagem, esses valores são habitualmente representados por meio de notação hexadecimal, indo de 00 (mais escuro) até FF (mais claro) para o valor de cada uma das cores. Assim, a cor #000000 é o preto, pois não há projeção de nenhuma das três cores; em contrapartida, #FFFFFF representa a cor branca, pois as três cores estarão projetadas em sua intensidade máxima. A figura 13 mostra o padrão RGB de cores. Neste trabalho a imagem é adquirida, inicialmente, neste formato, entretanto isso gerou um problema, pois era necessário que o sistema fosse numericamente contínuo, e o único que possui essa característica é o sistema HSB/HSV.

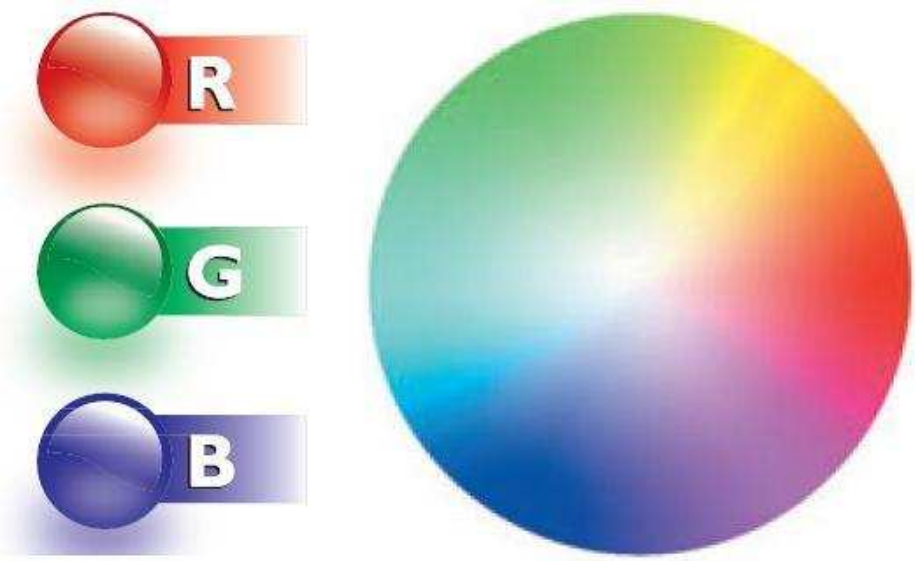

Figura 13 – Padrão RGB de Cores [11]

De acordo com [11] HSV é a abreviatura para o sistema de cores formadas pelas componentes *Hue* (tonalidade), *Saturation* (Saturação) e *Value* (Valor). Esse sistema também é conhecido como HSB (*Hue*, *Saturation* e *Brightness* - Tonalidade, Saturação e Brilho, respectivamente). Esse sistema de cores define o espaço de cor conforme descrito abaixo, utilizando seus três parâmetros:

- Tonalidade: Verifica o tipo de cor, abrangendo todas as cores do espectro, desde o vermelho até o violeta, mais o *magenta*. Atinge valores de 0 a 360, mas para algumas aplicações, esse valor é normalizado de 0 a 100%.

- Saturação: Também chamado de "pureza". Quanto menor esse valor, mais o tom de cinza aparecerá a imagem. Quanto maior o valor, mais "pura" é a imagem. Atinge valores de 0 a 100%.

- Valor, ou brilho: Define o brilho da cor, ou seja do mais escuro (cor preta), para o mais claro (cor branca). Atinge valores de 0 100%.

Esse sistema foi inventado no ano de 1978, por Alvy Ray Smith. É caracterizada por ser uma transformação não-linear do sistema de cores RGB.

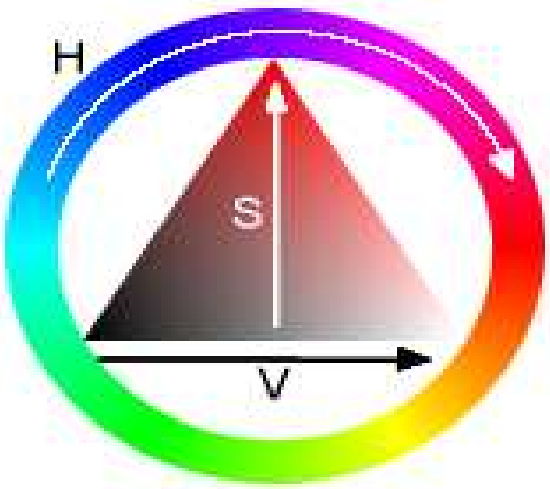

Figura 14 - O sistema de cores HSV/HSB sendo visto como um círculo [11]

Como citado anteriormente foi necessário fazer a conversão de RGB para HSB. Por necessidade de limitar faixas de cores continuas e pré-estabelecidas para distinção dos objetos carro, obstáculo e objetivo. Portanto seja uma cor definida por (*R*, *G*, *B*), onde *R*, *G* e *B* estão entre 0.0 e 1.0, onde 0.0 e 1.0 são, respectivamente, o maior e o menor valor possível para cada. A transformação para os parâmetros (*H*, *S*, *V*) dessa cor pode ser determinada pelas fórmulas abaixo.

Seja *MAX* e *MIN* os valores máximo e mínimo, respectivamente, dos valores (*R*, *G*, *B*):

$$H = \begin{cases} 60 \times \frac{G-B}{MAX-MIN} + 0, & \text{if } MAX = R \\ & \text{and } G \geq B \\ 60 \times \frac{G-B}{MAX-MIN} + 360, & \text{if } MAX = R \\ & \text{and } G < B \\ 60 \times \frac{B-R}{MAX-MIN} + 120, & \text{if } MAX = G \\ 60 \times \frac{R-G}{MAX-MIN} + 240, & \text{if } MAX = B \end{cases}$$

$$S = \frac{MAX - MIN}{MAX}$$

$$V = MAX$$

Os resultados dão a tonalidade variando de 0 a 360, indicando o ângulo no circulo onde a tonalidade (*H*) está definida, e a saturação e o brilho variando de 0.0 a 1.0, representando o menor e o maior valor possível.

Estes conceitos de *RGB e HSB/HSV* são fundamentais, pois neste trabalho tiveram um importante papel na identificação dos objetos. O RGB é o formato padrão, ou seja, o formato em que a imagem é adquirida e tratada pelo software. Entretanto devido a não-linearidade de cores no formato RGB fez-se a conversão para HSB/HSV, com a finalidade de conseguir parâmetros de cores mais próximos e delimitar faixas de cores para a identificação dos objetos. Na seção 7.3 deste trabalho é mostrado em detalhes todas as aplicações e implementações dos conceitos de visão computacional demonstrados neste capítulo.

4. Robótica - Veículos Inteligentes

4.1. Introdução

A robótica é uma área do conhecimento que tem evoluído de forma muito rápida nos últimos anos, entretanto, o estudo, o projeto e a implementação de robôs e autômatos (primeiros robôs) vêm sendo desenvolvidos há vários séculos. Na verdade, é difícil estabelecer a data precisa do surgimento dos primeiros autômatos (ancestrais dos atuais robôs), onde podemos considerar como um marco na história da humanidade a construção da primeira máquina de calcular, criada por Pascal em 1642 [12]. A partir desta época inúmeros autômatos foram desenvolvidos, mas foi somente a partir de 1923 que o termo "*robot*" começou a ser empregado, tendo sido usado pela primeira vez por [13].

Os primeiros robôs eram, na verdade, autômatos complexos, verdadeiras obras de arte mecânicas, que executavam tarefas de modo repetitivo. Estes robôs deram origem aos atuais braços manipuladores de base fixa, que atualmente são largamente adotados na indústria (e.g. indústria automotiva). Somente mais recentemente surgiram os robôs móveis, que se caracterizam pela sua capacidade de se deslocar de modo guiado, semi-autônomo ou totalmente autônomo [12]. Apesar de existirem referências mais antigas a dispositivos móveis comandados à distância, pode-se considerar que sua origem foi fortemente influenciada pelo desenvolvimento de artefatos militares, como por exemplo, veículos inteligentes teleguiados ou mesmo auto-guiados como as bombas V1 e V2 desenvolvidas pela Alemanha durante a 2ª. Guerra Mundial [13].

Portanto, o desenvolvimento da robótica evoluiu dos autômatos, passando pelos robôs manipuladores de base fixa, pelos dispositivos móveis guiados a distância, chegando mais recentemente aos robôs móveis semi-inteligentes e mesmo os totalmente inteligentes, estes se denominam veículos inteligentes, ou veículos não tripulados, ou Robôs Móveis Inteligentes (RMA´s). Um exemplo de sucesso do desenvolvimento de veículos inteligentes de alta tecnologia é o *rover Soujourner* que explorou o planeta Marte em 1997 [12]. Entretanto, cabe destacar que mesmo os robôs de exploração espacial como o *Soujourner,* e seus sucessores

Spirit e *Opportunity,* são dispositivos de autonomia limitada, necessitando ainda de comandos enviados a distância por seres humanos que controlam a missão. Em meio a tantos tipos distintos de robôs e veículos com diferentes recursos e níveis de autonomia, é importante que se busque identificar e organizá-los em categorias. Na Tabela 3 é apresentada uma classificação dos robôs, de acordo com diferentes critérios, considerando, entre outros aspectos: estrutura, funcionalidade. mobilidade e autonomia.

Tabela 3 – Tipos de Robôs

Funcionalidade: Tipos de Aplicações	• Mecanismos eletro-mecânicos com movimento repetitivo pré-definido e fixo (autômatos) • Manipuladores: Braços de base fixa (e.g. robôs industriais) • Manipuladores: Braços de base móvel (e.g. gruas robóticas) • Robôs Móveis (ver abaixo os tipos de **mobilidade**): o Terrestres: *indoor*, *outdoor off-road* (qualquer terreno), veículos convencionais *outdoor* (automóveis) o Aquáticos, Submarinos, Aéreos o Exploração Espacial (e.g. *Soujourner*)
Mobilidade: Dispositivos de Locomoção	• Robôs móveis com deslocamento com rodas • Robôs móveis com deslocamento com esteiras • Robôs com deslocamento por propulsão (e.g. no ar, na água) • Robôs com deslocamento por pernas (e.g. *animats*, bípedes) Dispositivo/Modelo Cinemático: diferencial, síncrono, triciclo ou *Ackermann* Dudek
Autonomia: Grau de Inteligência do Robô	• Totalmente comandados a distância (e.g. carro de "controle remoto" = tele-comandado, tele-operado) • Veículos guiados: dependem de informações externas, sem as quais não podem operar (AGV – *Automated Guided Vehicle*) • Veículos semi-autônomos: operam controlados remotamente, mas possuindo um certo grau de autonomia (e.g. podem parar de modo a evitar o choque com obstáculos) • Veículos autônomos: operam sem intervenção humana, de forma totalmente automatizada. Uma vez iniciada sua operação, executam tarefas de modo autônomo.

4.2. Propriedades dos Veículos Inteligentes

Os Veículos inteligentes possuem como características fundamentais, as capacidades de locomoção e de operação de modo semi ou completamente autônomo. Também deve ser considerado que maiores níveis de autonomia serão

alcançados à medida que o robô passe a integrar dois outros aspectos considerados da maior importância: robustez (capacidade de lidar com as mais diversas situações) e inteligência (de modo a resolver e executar tarefas por mais complexas que sejam).

Os Veículos inteligentes possuem algumas propriedades que demonstram o possível grau de complexidade que podem chegar esses tipos de robôs como Ação e Locomoção, Percepção, Controle e Inteligência, comunicação [13].

• **Ação e Locomoção**: Como o robô irá se deslocar no ambiente. Os dispositivos de locomoção são de grande importância na caracterização de um robô móvel;

• **Percepção**: Como o robô irá perceber o ambiente (e.g. sensores de contato – *bumpers* - e de distância em relação a obstáculos) e inclusive monitorar parâmetros próprios dele e de seu comportamento (e.g. carga da bateria, hodometria);

• **Controle e Inteligência**: Como o robô irá transformar suas percepções e conhecimentos prévios adquiridos (e.g. mapa do ambiente) em ações, ou seqüências de ações, a serem executadas;

• **Comunicação**: Como o robô irá se comunicar com um operador humano, ou mesmo com outros dispositivos robóticos;

As pesquisas em robótica móvel têm avançado na direção de uma maior integração e aperfeiçoamento destes quatro aspectos descritos acima, cuja meta principal é o desenvolvimento de modelos cada vez mais sofisticados e robustos dos veículos inteligentes.

4.3. Ação e Locomoção de Veículos Inteligentes

Os robôs e veículos móveis devem ser dotados de atuadores, que serão os responsáveis pela execução de suas ações e, principalmente, pelo seu deslocamento no ambiente. Existem diferentes tipos de dispositivos usados na locomoção dos robôs, e conforme a solução ou implementação adotada o veículo terá diferentes tipos de comportamento. Podem ser citados como exemplo os robôs providos com duas rodas laterais e acionadores independentes acoplados a cada

uma delas, obtendo-se, assim, um veículo de *tração diferencial* [13, 14], onde a rotação (giro do robô) é obtida pelo acionamento de modo diferenciado de cada um dos motores. Um caso particular de veículos com acionamento diferencial é o dos veículos dotados de esteiras.

Um outro tipo de acionamento é o do "carro tradicional", que é dotado de um motor capaz de acionar ao mesmo tempo rodas de um mesmo eixo, tração dianteira ou traseira, com acionamento e controle de velocidade próprio *(throttle control*), onde este possui também um acionamento independente para girar a barra da direção (*steering control*). Este tipo de veículo usualmente terá um comportamento seguindo a cinemática *Ackermann* [13, 14]. Deve-se destacar que o conhecimento do modelo de acionamento, e por conseqüência do modelo cinemático, é essencial para que se possa aplicar um mecanismo adequado de controle, ou mesmo, simular o funcionamento de um robô móvel ou veículo. O "veículo-robô" deste trabalho usa esse ultimo tipo. Porém existe uma pequena diferença, a barra de direção tem comportamento de três estados apenas, ou seja, ou ela está na posição "reta" ou "esquerda" ou "direita".

O acionamento de atuadores pode, em certos equipamentos, não se restringir apenas ao controle do deslocamento do veículo, podendo também ser usado para acionar outras partes do robô, como por exemplo um braço robótico, ou mesmo ser usado para provocar ações sobre o ambiente (e.g. robô bombeiro) ou melhor posicionar um de seus sensores (e.g. uso de uma câmera acoplada a um braço robótico).

4.4. Percepção de Veículos Inteligentes

A percepção é um dos componentes principais dos robôs móveis, pois é através dos sensores que pode ser garantido ao sistema um maior nível de autonomia e robustez. Os sensores empregados junto à robótica móvel são de diferentes tipos, onde alguns servem para se ter uma realimentação mais direta do resultado de uma ação, como por exemplo, os *encoders* (equipamentos eletromecânicos, utilizados para conversão de movimentos rotativos ou deslocamentos lineares em impulsos elétricos de onda quadrada, que geram uma quantidade exata de impulsos por volta em uma distribuição perfeita dos pulsos ao

longo dos 360 graus do giro do eixo). A "ordem" de execução de um comando não garante que o resultado perfeito e completo desta ação seja atingido, por exemplo, uma mesma tensão de acionamento aplicada a um motor, durante um mesmo período de tempo, poderá ser afetada pela inclinação do terreno, em termos da distância total percorrida (isto sem considerar questões relacionadas à curva de resposta a aceleração e o torque do motor). O uso de um *encoder* permite que seja medido o real deslocamento (leia-se giro) das rodas de um robô móvel. Na Tabela 4 são listados os principais dispositivos sensores que vêm sendo utilizados junto aos veículos inteligentes.

Tabela 4 – Tipos de sensores usados em veículos inteligentes

Sensor	**Função**
Encoder (de rodas)	Mede o número de rotações das rodas, e permite que se obtenha informações de odometria (medida de deslocamento).
Sensor de Infravermelho	Mede a distância entre o sensor e um obstáculo posicionado em frente a ele, através da estimativa da distância calculada em função do retorno da reflexão da luz na superfície (medida de distância).
Sensor Laser	Mede a distância entre o sensor e um obstáculo posicionado em frente – sensor direcional de alta precisão (medida de distância).
Sonar (ultra-som)	Estima a distância entre o sensor e os obstáculos ao seu redor, através da reflexão sonora. É um sensor mais sensível a perturbações, obtendo medidas aproximadas (medida de distância).
Sensores de Contato (bumbers)	Identifica quando ocorre uma colisão entre o veículo (sensor) e um ponto de contato (medida de contato/pressão).
Bússola Eletrônica	Identifica a orientação do veículo em relação ao campo magnético da terra (medida de orientação – posição relativa).
GPS	Identifica a posição absoluta do veículo no globo terrestre, baseando-se na rede de satélites GPS – *Global Positioning System* (medida de posicionamento absoluto).
Imagens: Visão Artificial	O uso de imagens permite que sejam adquiridas a partir de uma ou mais câmeras (visão monocular, estéreo ou omnidirecional) descrições do ambiente (imagem monocromática ou colorida). As imagens permitem que se implemente técnicas de determinação de posicionamento relativo, posicionamento absoluto, detecção e estimativa de deslocamento, assim como detecção de obstáculos.
Outros sensores	Acelerômetros, Giroscópios, Sensores de Inclinação, Radar, ...

O adequado uso dos sensores, bem como a correta interpretação dos dados fornecidos por estes (modelo sensorial, incluindo precisão, distância alcançada,

suscetibilidade ao ruído, etc), é que permitirá a correta implementação, ou simulação, de sistemas de veículos inteligentes mais robustos. Um outro tema de grande relevância na robótica é a integração de sensores (fusão sensorial), de modo a integrar e explorar a complementaridade entre os diferentes sensores, que possuem diferentes escalas de sensibilidade e confiabilidade (e.g. fusão de dados de um radar, sonar e sensor laser).

A percepção do veículo não tripulado se dá através de uma câmera de vídeo colorida, fixada no alto do campo de testes. Esta câmera consegue capturar toda a área necessária para o objetivo deste trabalho. Consegue também a captura das imagens em tempo real, imprescindível para o sucesso da solução do problema proposto.

4.5. Controle e Inteligência de Veículos Inteligentes

Um sistema robótico móvel pode ter diferentes níveis de autonomia e inteligência, o que será definido pelo tipo de sistema de controle integrado ao sistema. Existem veículos inteligentes que possuem um nível de autonomia e inteligência extremamente limitado como, por exemplo, os AGV's (*Automatic Guided Vehicles – Veículos guiados automaticamente*) convencionais industriais.

De acordo com [12] um AGV industrial é um robô móvel que usualmente segue uma marcação preestabelecida (e.g. uma faixa pintada no solo), e mesmo se este possui integrada uma capacidade de se deslocar de modo autônomo (locomoção) e de perceber o ambiente (sensor da faixa), seu sistema de controle depende desta informação para poder continuar atuando. Se um pedaço da faixa for danificado, o robô usualmente não terá como prosseguir em sua tarefa e deverá parar de se movimentar. O mesmo acontece caso este robô detecte um obstáculo em seu caminho, ele irá parar e aguardar que o obstáculo seja removido. Nota-se que os níveis de robustez, autonomia e inteligência de um veículo como este é bastante reduzida, não sendo capaz de suportar falha na marcação ou de desviar de obstáculos e retornar para a execução de sua tarefa.

Em função disto, as pesquisas em robótica móvel autônoma tem avançado na direção de propor sistemas de controle, que incluam "comportamentos inteligentes",

de modo a tornar os robôs mais confiáveis, robustos e menos dependentes da intervenção de seres humanos. Este “comportamento inteligente” é implementado através de sistemas de controle que devem, entre outras funções, realizar tarefas como as descritas a seguir [12]:

• Garantir a preservação da integridade física dos seres humanos, bem como não causar danos ao ambiente onde está inserido;

• Garantir a preservação da integridade física do robô: evitar colisões contra pessoas e objetos (obstáculos estáticos ou móveis);

• Garantir a manutenção de informações que permitam uma melhor execução das tarefas, o que pode incluir: construção e atualização de um mapa do ambiente, determinação da localização do robô em relação a este mapa, definição de trajetórias que permitam que ele se desloque de um ponto a outro (considerando o mapa disponível);

• Integrar as informações provenientes de diferentes sensores, interpretando estas informações e levando em consideração problemas de ruído e erros associados às mesmas;

• Gerar os comandos, na seqüência correta, de modo a realizar as tarefas que lhe são atribuídas;

• Definir soluções alternativas para situações imprevistas que se apresentem e que por ventura possam prejudicar a execução das ações que estavam planejadas;

• Capacidade de se adaptar, ser “treinado” com as experiências passadas e corrigir seus erros e, quem sabe no futuro, ter a capacidade de evoluir.

Concluindo, para que se possam desenvolver adequadamente projetos na área de robótica móvel e veículos inteligentes, são necessários primeiramente:

(i) obter modelos robustos do comportamento dos diversos componentes destes sistemas (atuadores e sensores);

(ii) modelar o comportamento do sistema, projetando um sistema de controle que seja capaz de integrar os dados sensoriais, gerar um plano de ação e executar este plano;

(iii) simular o sistema completo – sensores, controle, atuadores, de forma a validar e aperfeiçoar o sistema de controle;

(iv) integrar o sistema de controle em um robô/veículo real de modo a validar a proposta e verificar a sua robustez perante situações reais.

No veículo autônomo deste trabalho o controle é feito através de um método de inteligência artificial denominado método da função potencial. Este método recebe a informação do sensor, câmera, e determina qual a ação que o veículo deverá executar, neste caso, a direção em que o mesmo deve se locomover.

4.6. Comunicação de Veículos Inteligentes

A comunicação é uma ferramenta de grande importância, tanto nos sistemas robóticos que dependem de informações externas (e.g. controle externo), ou mesmo, sendo usada de forma a estabelecer uma troca de informações entre múltiplos robôs e veículos. Um exemplo de comunicação entre controlador e dispositivo controlado, é o sistema adotado para comunicação entre a NASA e o *rover Soujourner*, que, devido às grandes distâncias, necessitava de certa autonomia para se manter íntegro durante a espera de novas instruções, ou mesmo, quando havia uma perda de sinal [12].

Em relação à comunicação entre veículos, este tipo de comunicação tem se apresentado como de grande importância, principalmente em situações relacionadas ao controle de fluxo em comboios nas estradas, conforme será apresentado posteriormente. Atualmente a comunicação é um tema de grande interesse e um importante foco de pesquisa na área de robótica móvel, estando fortemente relacionada às pesquisas em Inteligência Artificial.

Neste trabalho a comunicação do veiculo autônomo acontece da seguinte forma. O sistema computacional recebe a informação da câmera, que neste caso é o sensor do veiculo. Este sistema computacional, depois dos devidos cálculos, envia,

via porta paralela, para o “controle remoto” do carrinho a informação necessária para a movimentação do mesmo. Este controle remoto se comunica com o mesmo através de radiofreqüência. A figura 15 ilustra o esquema de comunicação usado.

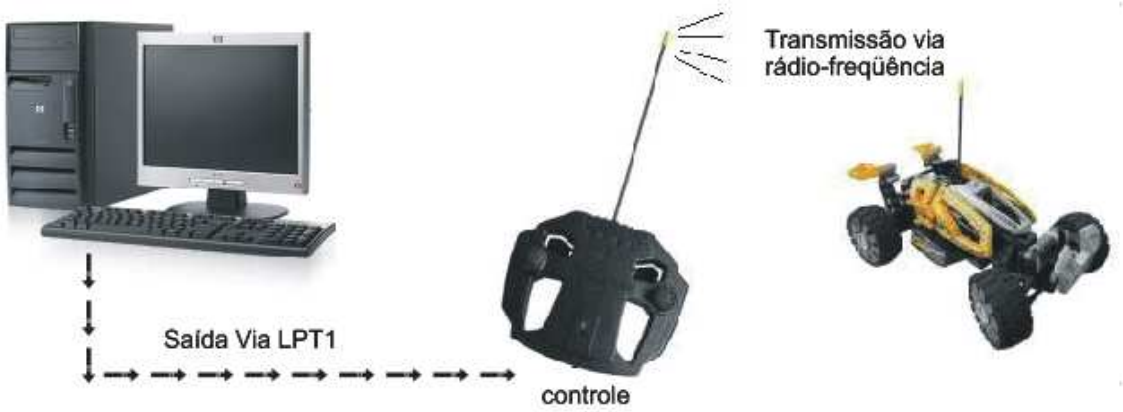

Figura 15 – Esquema da comunicação do veículo

5. Sistemas de Controle Aplicado a Veículos Inteligentes

5.1. Introdução

O sistema computacional de controle, dependendo do problema proposto, deve preservar a integridade do veículo a ser controlado, preservar a integridade dos objetos e entes presentes no ambiente onde este veículo atua, deve planejar, executar e encontrar soluções para a execução de tarefas propostas, e em alguns casos deve inclusive interagir com outros sistemas. As características de tal sistema de controle impõe o estudo de técnicas que vem sendo abordadas junto ao domínio da Inteligência Artificial (I.A.), e principalmente no que diz respeito a técnicas de controle de “agentes artificiais inteligentes”.

O desenvolvimento de um sistema de controle de um veículo autônomo é uma tarefa bastante complexa. Esta tarefa envolve inicialmente o projeto e implementação das camadas inferiores da pirâmide de controle (e.g. interface com o hardware e dispositivos eletromecânicos), e posteriormente é necessária também a implementação de um sistema computacional que irá gerenciar os diversos componentes e módulos deste sistema. Um sistema computacional de controle pode envolver tarefas mais simples, que podem ser gerenciadas apenas por CLPs (controladores lógicos programáveis). Entretanto, sistemas mais complexos, planejados para controlar a execução de tarefas mais complexas (e.g. estacionar um carro, dirigir de modo autônomo), requerem uma arquitetura computacional de controle [22]. As tarefas dessa arquitetura computacional de controle podem envolver:

- Capacidade para ler e interpretar os sinais recebidos dos sensores do veículo;
- Capacidade de evitar obstáculos presentes no caminho do veículo;
- Capacidade para reagir a eventos, inclusive eventos inesperados, como a aparição imprevista de obstáculos móveis;

- Capacidade de planejamento de trajetórias e de execução de tarefas, como por exemplo: definição de rotas de um ponto A até um ponto B, tendo conhecimento, ou não, de um mapa do ambiente;

- Capacidade de gerenciar os diversos componentes do sistema, de modo gerar os comandos na ordem adequada e com os parâmetros corretos, a fim de que a tarefa que foi planejada possa ser executada;

As arquiteturas computacionais de controle de veículos inteligentes são as mais diversas, e mesmo na literatura são encontrados diferentes enfoques e abordagens [3, 22], entretanto, podem ser citadas algumas das arquiteturas de controle que se tornaram as mais conhecidas e reconhecidas pelas suas características e potencialidades: controle reativo, controle deliberativo, controle hierárquico e controle híbrido [14, 22].

5.2. Controle Reativo

O controle reativo consiste de um sistema de reação sensorial-motora, onde este tipo de controle normalmente é o mais simples de ser implementado (reativo puro), não necessitando de muitos recursos computacionais para sua implementação. No controle reativo existe um laço de: *(i)* leitura dos sensores; *(ii)* imediato processamento destas informações; *(iii)* geração de um comando de resposta para os atuadores. Usualmente um esquema de controle reativo considera apenas as leituras sensoriais realizadas no presente para fins de tomada de decisão e geração de comandos de ação [22]. Um sistema reativo é bastante útil para implementar comportamentos como: desviar de obstáculos (*avoid collision behaviour*: reage a presença de um obstáculo), e seguir um objeto (*wall-following/lane-follow behaviour*: acompanhar um elemento guia). A Figura 16 apresenta um esquema que pode ser considerado como um sistema de controle reativo, e é um exemplo pratico do trabalho com o veículo não tripulado, os obstáculos, o objetivo e a rota a ser seguida. Caso a tomada de decisão do agente-robô não se utilize de mapas, de memória, ou de outras informações adicionais, fazendo uso apenas das informações sensoriais a fim de gerar uma nova ação.

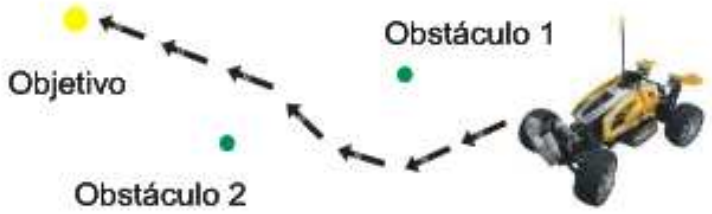

Figura 16 – O carro no campo de testes e sua rota estática.

5.3. Controle Deliberativo

O controle deliberativo (ou cognitivo) consiste da aplicação de um mecanismo de planejamento das ações, podendo ser estabelecido um plano prévio de execução de uma seqüência de ações, baseado nos conhecimentos que o sistema possui sobre o problema a ser resolvido (e.g. mapa do ambiente, rotas disponíveis). No controle deliberativo é assumida a existência de um processo de alto nível de raciocínio e tomada de decisões, usualmente mais complexo de ser implementado do que o controle reativo. Este processo permite que sejam planejadas ações de modo a tratar e executar tarefas que exigem um nível de controle mais sofisticado, como por exemplo, definir (traçar uma rota) e executar a tarefa de se deslocar de um ponto a outro do ambiente considerando um mapa deste, como apresentado na Figura 17.

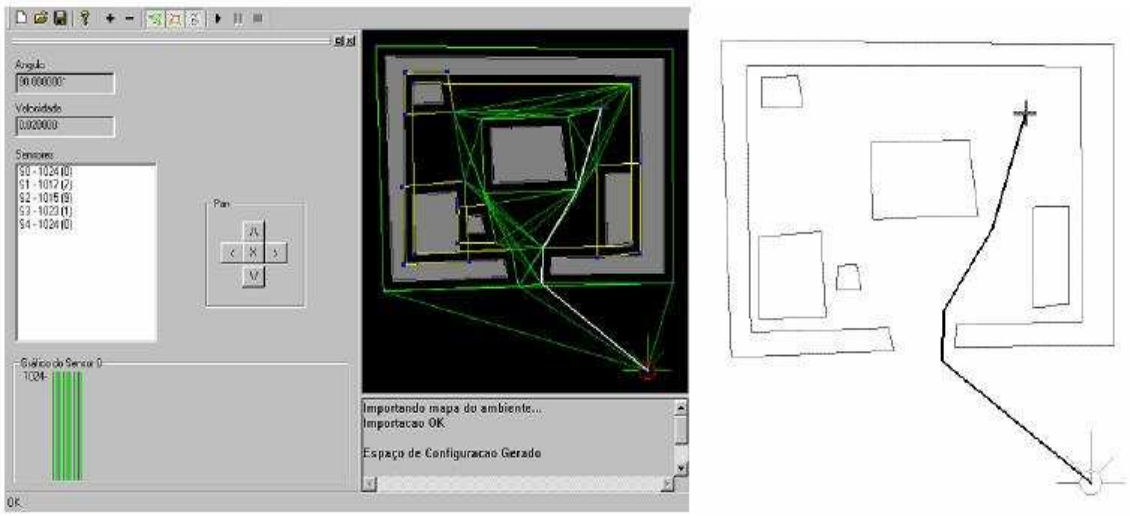

Figura 17 - Comportamento deliberativo com o planejamento e execução de trajetória considerando o mapa do ambiente [14].

Entretanto, o controle “deliberativo puro” possui limitações quando colocado frente a eventos imprevistos, como por exemplo, um obstáculo que se moveu obstruindo a sua rota [14]. Neste caso, o controle deliberativo puro irá ter dificuldades de reagir a uma nova configuração do ambiente, que não havia sido prevista em seu planejamento inicial. Pode-se dizer que o ideal em um sistema de controle seria que este tivesse a capacidade de reação de um sistema reativo, com a capacidade de planejamento e execução de tarefas complexas de um sistema deliberativo. Desta combinação entre reativo e deliberativo surgem os sistemas hierárquicos e, ou híbridos.

5.4. Controle Hierárquico e Controle Híbrido

O controle hierárquico-híbrido consiste da combinação de múltiplos módulos de controle reativo e, ou, deliberativo em camadas dispostas de modo que estes possam operar de modo hierárquico. A combinação dos diferentes módulos de controle leva à adoção de um esquema de prioridades em relação às múltiplas camadas do sistema, onde é comum encontrarmos estes sistemas de controle classificados como: sistemas hierárquicos com decomposição vertical e decomposição horizontal [14]. Os sistemas hierárquico-híbridos apresentam a vantagem de poderem combinar os comportamentos obtidos de seus diferentes módulos a fim de obter um comportamento mais robusto e uma execução de tarefas mais complexas. Um exemplo de aplicação deste tipo de sistema de controle seria a composição do planejamento e execução de uma tarefa de navegação do ponto A ao ponto B, inicio e fim respectivamente (controle deliberativo – de menor prioridade), onde um módulo reativo estaria encarregado, de preservar a integridade do robô, desviando dos obstáculos que forem percebidos através de seus sensores (controle reativo – de maior prioridade). Neste exemplo, pode-se imaginar que o robô passa para um controle reativo ao detectar um obstáculo que esteja dificultando seu deslocamento, e uma vez “resolvido o problema”, ele retorna a execução de sua tarefa principal, onde pode inclusive solicitar ao módulo deliberativo um novo planejamento de trajetória (nova rota), já pré-estabelecida.

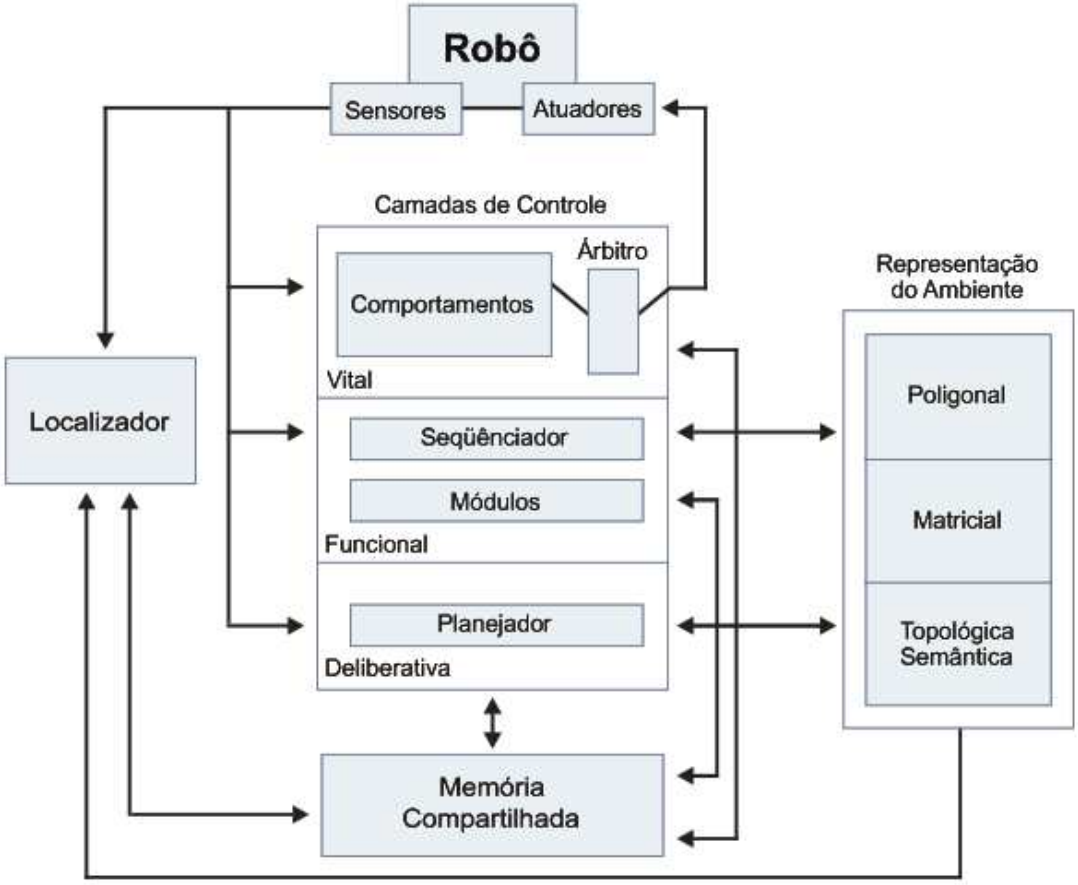

Figura 18 - Arquitetura de Controle Híbrido COHBRA [14]

O controle híbrido é implementado através de uma série de módulos que operam em paralelo e que se comunicam. Um exemplo de sistema de controle híbrido de veículos inteligentes é a arquitetura proposta por [14] – Arquitetura COHBRA/HyCAR, a qual implementa módulos de controle de localização, de manipulação da representação do ambiente e uma hierarquia de módulos em camadas de controle do robô (camada vital, funcional e deliberativa), onde estes módulos se comunicam através de uma memória compartilhada, entre os módulos (tipo *blackboard*) [14].

6. Técnicas de Inteligência Artificial Aplicadas a Veículos Inteligentes

6.1. Introdução

Inteligência Artificial (IA) é uma das ciências mais recentes, que atualmente abrange uma variedade enorme de subcampos, que vão desde áreas de uso geral, como aprendizado e percepção, até tarefas mais específicas, como jogos de xadrez [23]. Para [24] a inteligência artificial é a parte da ciência da computação que compreende o projeto de sistemas computacionais que exibam características associadas, quando presentes no comportamento humano, à inteligência. A Inteligência artificial sistematiza e automatiza tarefas intelectuais e, portanto, é potencialmente relevante para qualquer esfera da atividade intelectual humana, que neste caso, será abordada técnicas de IA para auxiliar o processo de desenvolvimento de veículos inteligentes.

Num sentido o mais amplo, a maioria dos veículos inteligentes incorpora alguma forma de inteligência artificial [23]. Alguns desenvolvedores consideram tarefas tais como “encontro de caminho” (*pathfinding*), métodos para detecção de colisão serem passiveis de implementação de algoritmos de IA [23].

6.2. Definições

Algumas técnicas de IA podem ser citadas como Lógica Fuzzy, redes neurais, algoritmos genéticos, sistemas especialistas, sistemas baseado no conhecimento, e combinações entre esses.

A lógica *fuzzy* ou lógica difusa é uma generalização da lógica *booleana. A* lógica *booleana* admite apenas dois estados, em contrapartida a lógica *fuzzy* admite valores lógicos intermediários entre a falsidade e a verdade (como o talvez). Como existem várias formas de se implementar um modelo *fuzzy*, a lógica *fuzzy* deve ser vista mais como uma área de pesquisa sobre tratamento da incerteza, ou uma família de modelos matemáticos dedicados ao tratamento da incerteza, do que uma lógica propriamente dita [23]. As implementações da lógica difusa permitem que

estados indeterminados possam ser tratados por dispositivos de controle, como por exemplo, avaliar conceitos como morno, médio, quente dentre outros. Ao trabalhar com a lógica *fuzzy* é comum chamar a lógica *booleana* de lógica nítida. Muitos pesquisadores de versões *booleanas* de lógica não aceitam a lógica *fuzzy* como uma verdadeira lógica [23]. Isso pode ser associado a diferentes fatos, entre eles o fato de muitos modelos permitirem soluções aproximadas que não correspondem a uma "verdade" lógica.

Os modelos de redes neurais procuram aproximar o processamento dos computadores ao cérebro humano (neurônios). As redes neurais possuem um grau de interconexão similar a estrutura do cérebro e em um computador convencional moderno a informação é transferida em tempos específicos dentro de um relacionamento com um sinal para sincronização, simulando a ação de um neurônio humano real. Muitas aplicações as redes neurais artificiais são compostas de alguns, muitos ou apenas um neurônio [23]. Isto é muito simples se comparado com o cérebro humano. Algumas aplicações específicas usam as redes neurais compostas, talvez, de milhares dos neurônios, contudo até estas são simples na comparação a nossos cérebros. Não se pode comparar o poder de processamento do cérebro humano com redes neurais; entretanto, para problemas específicos, as redes neurais podem ser ferramentas muito poderosas.

Algoritmos Genéticos são métodos computacionais de busca e otimização inspirados nos mecanismos de evolução natural e da genética [23]. Os Algoritmos Genéticos operam sobre uma população de soluções dos problemas codificados usando transições probabilísticas e não determinística. Os três aspectos mais importantes do desenvolvimento de um Algoritmo Genético são: (1) definição e implementação da representação genética, (2) definição da função objetivo, e (3) definição e implementação dos operadores genéticos. Uma vez que se tenha definido estes três aspectos, o algoritmo genético pode ser um grande auxilio na solução de problemas com uso de inteligência artificial.

Um sistema especialista pode ser visto como uma subárea da Inteligência Artificial, desenvolvido a partir da necessidade de se processar informações não numéricas, um sistema especialista é capaz de apresentar conclusões sobre um determinado tema, desde que devidamente orientado e “alimentado”. Um sistema

especialista, de acordo com [24], é uma forma de sistema baseado no conhecimento especialmente projetado para emular a especialização humana de algum domínio específico. Um sistema especialista irá possuir uma base de conhecimento formada de fatos, regras e heurísticas sobre o domínio, tal como um especialista humano faria, e deve ser capaz de oferecer sugestões e conselhos aos usuários e, também, adquirir novos conhecimentos e heurísticas com essa interação. O objetivo do sistema especialista é bastante restrito, se considerar o objetivo dos modelos psicológicos: os sistemas especialistas são concebidos para reproduzir o comportamento de especialistas humanos na resolução de problemas do mundo real, mas o domínio destes problemas é altamente restrito [24].

Os sistemas baseados no conhecimento foram alvos de várias pesquisas em Inteligência Artificial, realizadas com sucesso. Esses sistemas são baseados num modelo explícito de conhecimento destinado a solucionar problemas. O conhecimento deve ser representado em forma de regras ou modelos de objetos. Sistemas Baseados no Conhecimento, *Knowledge-based systems*, são sistemas que aplicam mecanismos automatizados de raciocínio para a representação e inferência de conhecimento [24]. Esses sistemas costumam ser identificados como simplesmente "de inteligência artificial aplicada" e representam uma abrangente classe de aplicações da qual todas as demais seriam aproximadamente subclasses. Existe uma série de formalismos que podem ser utilizados para modelar o conhecimento de sistemas baseados no conhecimento, tais como, regras de produção, raciocínio baseados em casos, redes probabilísticas, entre outros.

6.3. Método de Perseguição e Desvio (Chasing/Evading)

Existem varias técnicas de perseguição de alvo e desvio de obstáculos, técnicas essas que podem envolver lógica *fuzzy* como em [25], ou outros algoritmos envolvendo inteligência artificial. Neste trabalho foi usada a técnica de funções potenciais. Os benefícios de se usar funções potenciais aqui são vários, dentre eles:

- Um simples algoritmo tem a função de perseguir e desviar ou seja, não é preciso outras condições ou controles associados a este algoritmo;

- A demanda computacional é extremamente baixa;

- É muito simples a implementação.

Basicamente este algoritmo calcula a força entre cada item envolvido, carro, obstáculos, alvo. Neste ponto é imprescindível descrever este método.

Muitos livros foram escritos sobre função potencial aplicados a diversos fenômenos físicos como, por exemplo, energia magnética. Entretanto neste trabalho será usado o resultado desta, visto que a adaptação da função potencial para este uso [23], é chamada de função potencial de Lenard-Jones (*Lenard-Jones potential function*) [23]. A equação da função potencial de Lenard-Jones é:

$$U = -\frac{A}{r^n} + \frac{B}{r^m}$$ (Equação 6.3.1)

A figura 19 mostra três gráficos desta função para diferentes valores para os expoentes *n* e *m.*

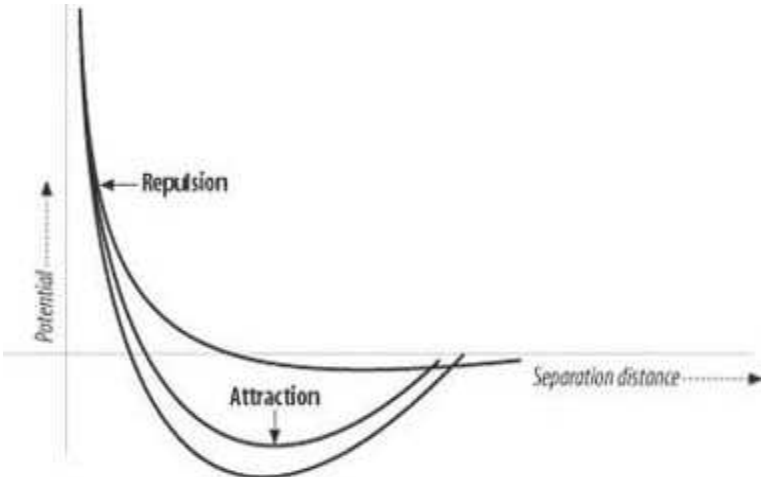

Figura 19 – Função potencial de Lenard-Jones

Na Física a função potencial de Lenard-Jones representa a energia potencial de atração ou repulsão entre moléculas. Na equação *U* representa a energia potencial interatômica, o qual é inversamente proporcional à distância de separação *r* entre, neste trabalho, objetos. *A* e *B* são parâmetros, assim como os expoentes *n* e *m.* Se esta equação da função potencial for derivada teremos uma função que representará a força. A função força produz tanto forças de atração quanto forças de repulsão dependendo da proximidade desses objetos. É essa habilidade de

representar atração e repulsão, ao mesmo tempo, que beneficia a escolha deste método para o problema proposto.

Para controlar essa habilidade de atração e repulsão os parâmetros devem ser inseridos de modo a definir o que será atraído e o que será repelido. Neste caso o objeto “alvo” interage, com o veículo autônomo, com uma força de atração e os objetos “obstáculos” interagem com uma força de repulsão. Essa “força de repulsão” será usada na evasão (*evading*), ou seja, no desvio dos obstáculos (*obstacle avoidance*), como ilustrado na figura 21, e a força de atração será usada para a perseguição do alvo (*chasing*), como ilustrado na figura 20. Com esta tarefa de perseguir e desviar o método de função potencial de Lenard-Jones pode ser considerado um método inteligente [23].

Figura 20 – Força de atração (*chasing*)

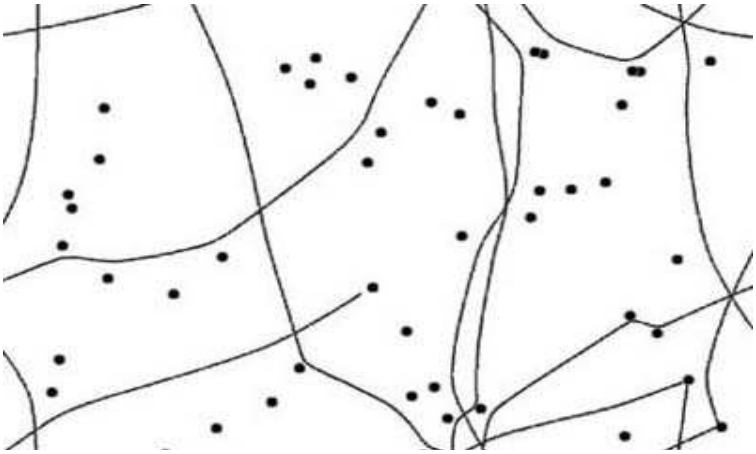

Figura 21 – Força de repulsão com uma série de obstáculos (*obstacle avoidance)*

7. Aplicação e Resultados

7.1. Proposta e Arquitetura do Sistema

Este trabalho tem a proposta de traduzir para o mundo real os resultados obtidos pelo simulador de [25], que, resumidamente, elaborou um programa de simulação de veículo inteligente, usando técnicas de inteligência artificial, neste caso, lógica *fuzzy*. Para esta proposta era necessário, para sua execução, um veiculo autônomo, obstáculos, alvo, campo de testes, sensores (neste caso a câmera) e um sistema de controle e processamento (neste caso um computador convencional do tipo *personal computer*). O objetivo é fazer com que o veiculo autônomo persiga o alvo, até atingi-lo, desviando de possíveis obstáculos em sua rota.

Para representar o veiculo autônomo foi escolhido um carrinho de controle remoto, de brinquedo, genérico. Este é alimentado por baterias e seu "controle remoto" é feito por radiofreqüência. Os obstáculos e o alvo são representados por bolinhas de borracha convencionais. Como campo de testes foi escolhido uma sala na Universidade Federal de Itajubá, com espaço e dimensões adequadas para este propósito. O sistema formado por esses objetos é ilustrado na figura 22.

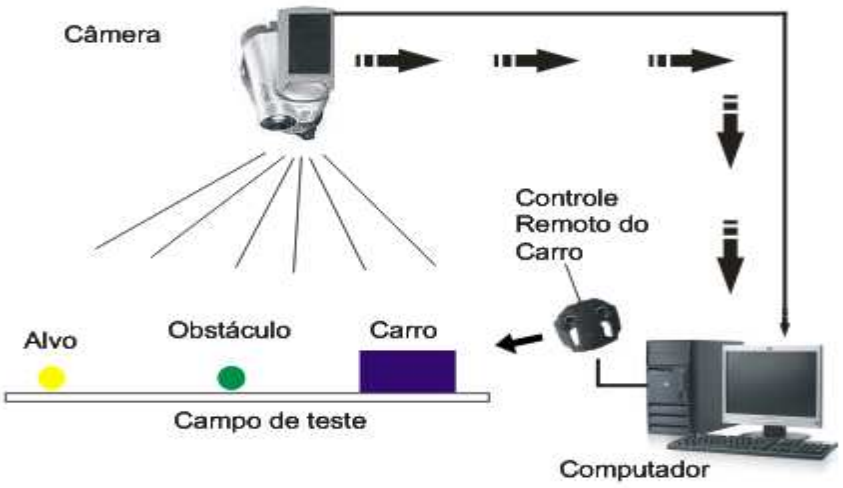

Figura 22 – O sistema Proposto por este trabalho

Para que esse sistema funcione é necessária a implantação da intercomunicação entre seus objetos. O computador é o centro de controle e gerenciamento das funcionalidades do sistema, ou seja, todos estão ligados entre si através dele. A câmera é ligada ao microcomputador, via cabo coaxial blindado, em uma placa de captura, genérica, inserida no *slot* PCI (extensão do barramento da unidade central de processamento, CPU, do microcomputador). O controle remoto do carrinho foi modificado de forma a receber um simples circuito, baseado em relés, a qual está ligado ao microcomputador, via cabo UTP CAT 5, em sua porta paralela, LPT1. Os objetos carrinho, obstáculos e alvo estão situados no campo de testes. A inserção e interação desses objetos com o sistema se dão através da câmera, que identifica suas posições, e através do controle remoto, que envia informações para o carro, dependendo da situação dos outros objetos, alvo e obstáculos. Dessa forma a intercomunicação do sistema e os requisitos de *hardware* estão completos.

Estando completo todos os requisitos de hardware do sistema proposto é necessário a preparação do software de baixo nível desse sistema ou seja, o software com a função de controlar a entrada e saída de informações. As imagens recebidas da câmera chegam ao computador (sistema de entrada) através do *driver* do fabricante da placa de captura (código proprietário). O computador envia (sistema de saída) as informações para o controle remoto, através da porta paralela, por um software de código aberto, denominado *inpout32.dll.*

7.2. O Software de Controle, Interpretação e Gerenciamento

Estando toda a arquitetura do sistema preparada, hardware e software de baixo nível, é necessário, agora o desenvolvimento do software de alto nível, responsável pelo gerenciamento, interpretação e controle do sistema, que tem as funcionalidades:

- Interface gráfica para melhor relação com o usuário;
- Aquisição das imagens provenientes da câmera;
- Tratamento das imagens, com o objetivo de separar todos os objetos (carro, obstáculos e alvo) e identificá-los;

- Processar as informações das imagens, e a partir dessas, gerar uma informação única, naquele intervalo de tempo, ou seja a direção em que o carro deve se movimentar naquele instante;
- Enviar a “informação” da direção ao controle remoto do carro;
- Fazer uso de iterações, de forma que, a cada movimento do carrinho ele gere outro, até que seja atingido seu objetivo.
- Trabalhar em tempo real;
- Ser rápido de tal maneira que todos os processamentos paralelos, resumidamente, aquisição de imagem, calculo da direção e envio do comando, sejam feitos de forma suficientemente rápidas a fim de não comprometer a próxima iteração.

Este software foi construído utilizando a ferramenta VISUAL STUDIO 2005®, da MICROSOFT©, através da linguagem C#. As vantagens na opção por esta linguagem são inúmeras como a portabilidade, a interface gráfica amigável com o usuário, uso da orientação a objetos, robustez e bom desempenho. O software deve, em um primeiro momento, executar as tarefas mostradas pela figura 23.

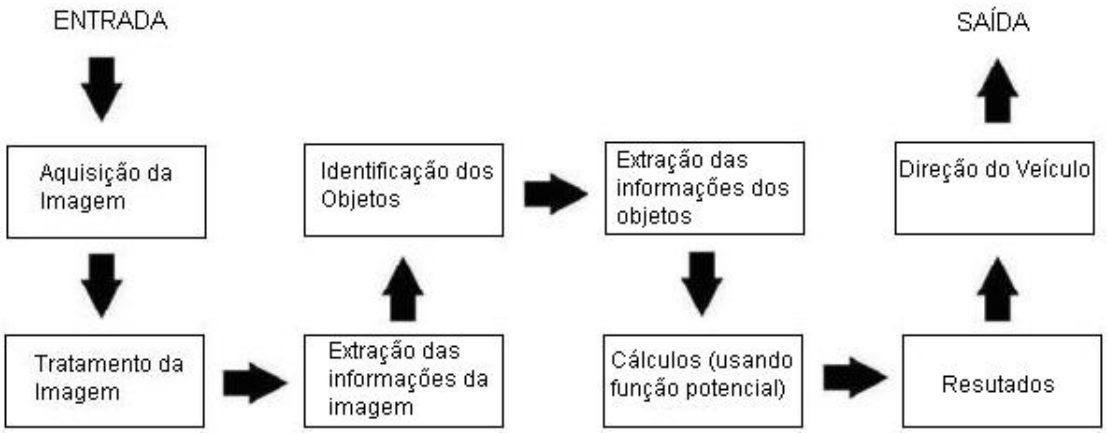

Figura 23 – Proposta do Software

Para otimização da aquisição e do tratamento das imagens o software usa bibliotecas desenvolvidas para o uso do DIRECTX®, da MICROSOFT. O DIRECTX

é uma ferramenta que oferece uma interface padrão para interagir com elementos gráficos, placas de som e dispositivos de entrada, entre outros. Ele se aproxima do hardware específico e traduz um conjunto comum de instruções em comandos específicos de hardware [27]. Isso faz com que a imagem tratada no programa continue na velocidade natural da câmera, ou seja, em tempo real imprescindível para execução da tarefa proposta por este trabalho. As bibliotecas utilizadas são de código aberto, baseadas na GNU (Licença Pública Geral) e são denominadas DShowNET.dll e DShowHlpr.dll.

Uma vez inserida no programa as imagens passam por diversas etapas até que o sistema computacional possa "entendê-las". Primeiramente a imagem é recebida como um *bitmap* (padrão de imagens). Depois da aplicação de uma série de filtros e algoritmos de visão computacional essa imagem está com os objetos, contidos na mesma, identificados. Com essa identificação consegue-se extrair a informação mais importante nesta etapa que é a localização destes. A localização, posição e a área desses objetos, é o objetivo de todo os processo de tratamento de imagens e visão computacional. A localização e necessária a próxima etapa, o controle.

Com a informação da posição dos objetos o software deve efetuar os cálculos baseados no algoritmo de função potencial (*Lenard-jones potencial function*). Esses cálculos terão como resultados a direção que o carrinho deverá seguir. Existem quatro possíveis valores para esses resultados: ir para frente, ir para esquerda, ir para a direita ou ficar parado, este ultimo caso o carrinho tenha atingido seu objetivo, o alvo. Essa informação é então passada a saída (*inpout32.dll*) e chega ao controle remoto. Que por sua vez transmite ao carrinho e ele executa a tarefa.

O tempo de execução, como citado anteriormente, é fundamental para este trabalho. Um possível atraso na execução de qualquer tarefa, por parte do computador, torna a tarefa proposta impossível de ser realizada. Por isso o software usa uma ferramenta muito útil em sistemas de tempo real as *threads*. *Threads*, ou processos, são tarefas que determinado programa executa paralelamente. Neste caso é necessário que enquanto uma *thread* captura, trata, extrai informações das imagens, faz os cálculos para determinar as posições dos objetos, outra *thread* envie os dados para o controle remoto do carrinho, isso ao mesmo tempo, e de forma

iterativa até que o objetivo seja alcançado (atingir o alvo). De forma análoga percebe-se que a sincronia entre as *threads* nesse caso também é fundamental.

7.3. Visão Computacional Aplicada

O objetivo deste trabalho é controlar um "veículo" em tempo real, para isso a etapa de processamento de imagens deve ser a mais rápida possível e gerar como resultado a identificação dos objetos, ou seja, a informação relativa à posição e área dos mesmos. Na primeira etapa do trabalho foi utilizado um software protótipo que tratava a imagem como um todo, fazendo todos os seus cálculos em cima de objetos bi*tmap*. A figura 24 escreve essa primeira interface "protótipo".

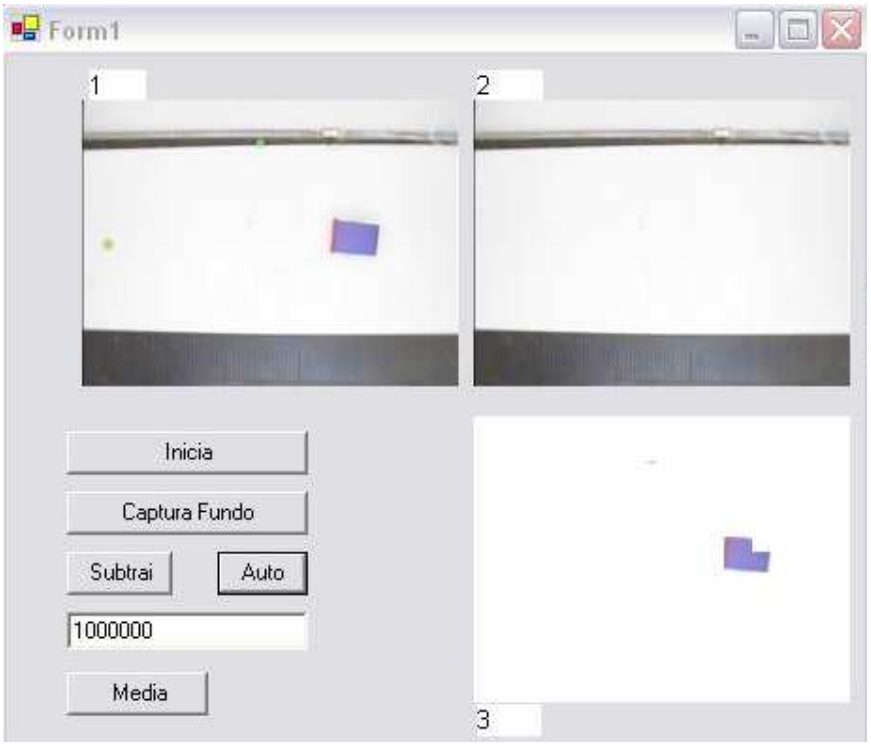

Figura 24 – Primeiro software desenvolvido para identificação dos objetos.

Entretanto, esse protótipo, não funcionou, porque o desempenho era muito baixo. O programa perdia muitos quadros da imagem, ocasionando "saltos", pois a classe, que realizava a aquisição desses *bitmaps,* fazia as operações no próprio *bitmap*, o que necessitava varias operações matemáticas, como por exemplo a cópia

do *bitmap* inteiro para uma outra posição de memória. Consequentemente desperdiçando muito tempo de processamento, tão crucial para a solução do problema proposto. Este protótipo funciona com os seguintes princípios, Na figura 24 a tela 1 é a imagem real captada diretamente da câmera. A tela 2 é o fundo, que representa a área de teste sem nenhum objeto. A tela 3 é a imagem resultante da subtração do *bitmap* da tela 1 menos o *bitmap* da tela 2 (denominado RetiraFundo), funcionando como um "filtro" inicial, obtendo então, apenas os objetos identificados. Como foi dito anteriormente essa técnica não funcionou, pois ela era demasiadamente lenta e exigia do microcomputador uma série de cálculos que o tornava excessivamente lento e consequentemente inútil ao propósito deste trabalho. A partir deste problema, foi utilizada a ferramenta *Directx*, da *MICROSOFT.* Esta ferramenta foi escolhida, pois ela trata a imagem com acessos diretos à memória e referencias à mesma, tornando a etapa de pré-processamento bem mais rápida e útil a esta proposta com o uso, da já citada, biblioteca *DshowNet.dll e* DShowHlpr.dll*.* Estas bibliotecas melhoraram consideravelmente a etapa do processamento inicial da imagem, ou seja aquisição e disposição em tela, permitindo que houvesse acesso direto à memória para manipulação dos dados da imagem recebida da câmera. Isso diminuiu o numero de cálculos, do protótipo inicial, já que com o acesso direto a memória (DMA), não são necessárias cópias a cada quadro e consequentemente, é incrementando o desempenho, conseguindo, a partir deste ponto, adquirir as imagens, para o futuro tratamento, em tempo real. Partindo do método RetiraFundo, e usando as bibliotecas de acesso direto a memória, obteve-se o primeiro "filtro" do sistema.

O método RetiraFundo obtém, a partir da imagem limpa (sem nenhum objeto) subtraído da imagem original (com todos os objetos), uma primeira diferenciação dos objetos. A figura 25 mostra que a imagem 1 é a imagem da câmera que é atualizada em tempo real. A imagem 2 é o fundo, que é estático e é obtido quando não existe nenhum objeto na área de testes. A imagem 3 é a resultante da operação de subtração da imagem 2 da imagem 1.

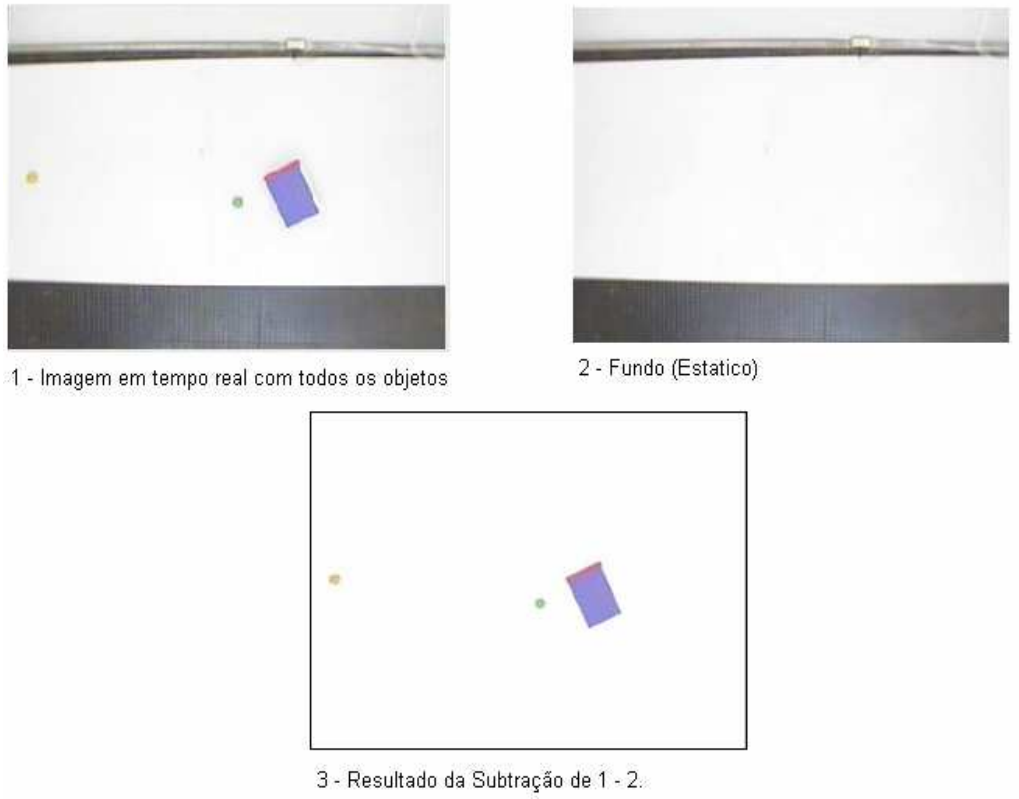

Figura 25 – Imagens obtidas pelo método RetiraFundo

Contudo o método RetiraFundo não é totalmente eficiente devido a problemas de variação de iluminação.

Neste ponto surge um outro problema, a ser tratado, a iluminação da área de testes. Esse problema ocorre, pois objetos na área de testes, não estão sobre nenhum controle de iluminação, ou seja, a área de teste nada mais é do que uma sala com iluminação convencional. Portanto havia, além da variação de freqüência da iluminação do campo de testes (a mesma da rede elétrica 60 Hz), a entrada de outras fontes de luz através das janelas e portas da sala, onde estava inserido o sistema. Essas fontes de luz, luz solar e lâmpadas de outras salas, não são desejadas, visto que a cada instante elas têm uma intensidade, ocasionando ruídos. Esses ruídos se manifestavam nas imagens impedindo uma correta distinção entre elas. Ruídos esses que tinham varias conseqüências, como por exemplo, reflexão e refração em bordas dos objetos, áreas mais iluminadas, sombra nos objetos e gerada por eles, dentre outros fatores. Para tentar minimizar este problema foram inseridas lâmpadas incandescentes com luz amarela sobre a área de testes. Essas lâmpadas têm a potencia nominal de 100 W e são utilizadas quatro delas

posicionadas, cada uma, nas extremidades da área de testes. Essas lâmpadas não eliminaram todo o ruído gerado no sistema, porém contribuíram para que esses diminuíssem consideravelmente, possibilitando, através da criação e uso de outros filtros de software, realizar a tarefa de identificação dos objetos. Elas contribuíram, também, para diminuir o processamento computacional, visto que, evitaram a necessidade da implementação de alguns filtros, podendo então, essa iluminação ser considerada uma filtragem de ruídos inicial, não totalmente eficiente, mas, muito importante no quesito tempo real e ganho de processamento computacional.

Estando a imagem inserida no sistema e pronta para a etapa de pré-processamento é necessário conhecer as características dessa imagem para criação de filtros e reconhecimento dos objetos. Aqui o histograma da imagem ajuda nesta tarefa de extração das características da imagem. A figura 26 representa o histograma com o sistema vazio, ou seja, sem objetos. A figura 27 representa o histograma do sistema completo, ou seja, os objetos estão dispostos no sistema.

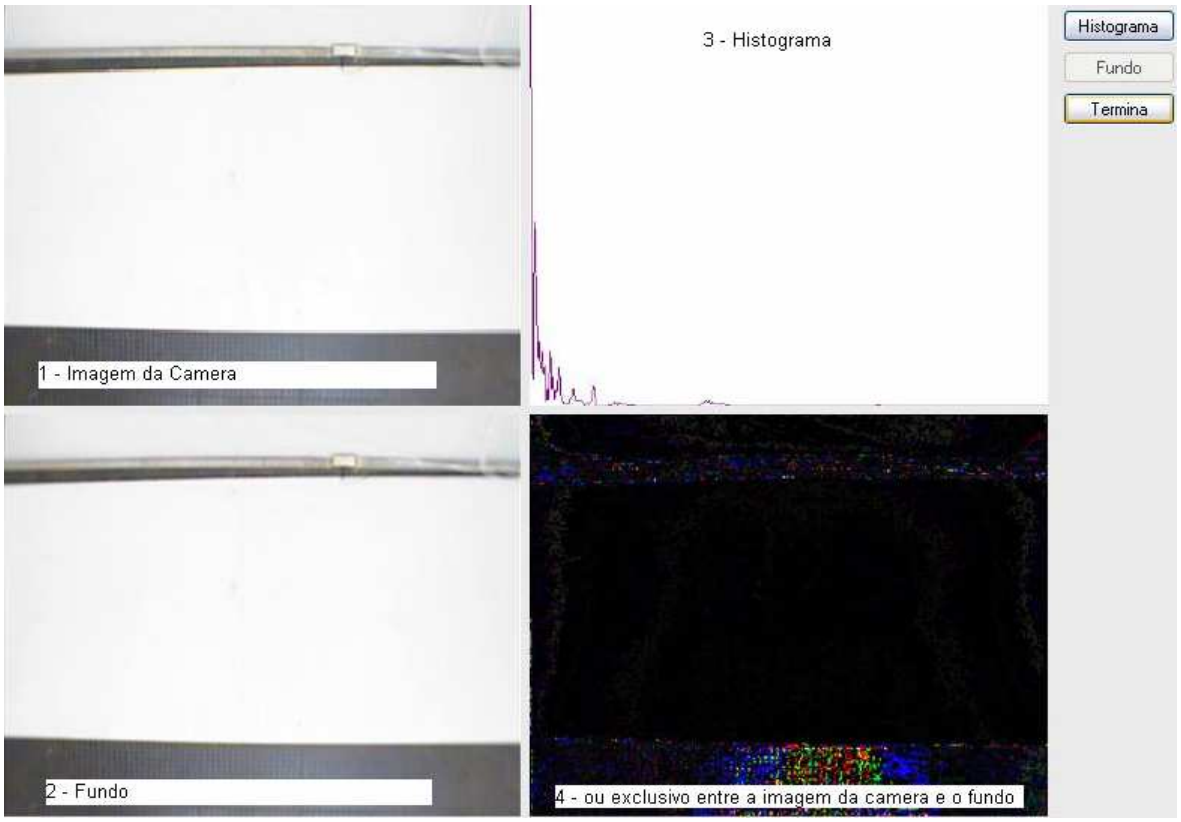

Figura 26 – Histograma com o sistema sem objetos.

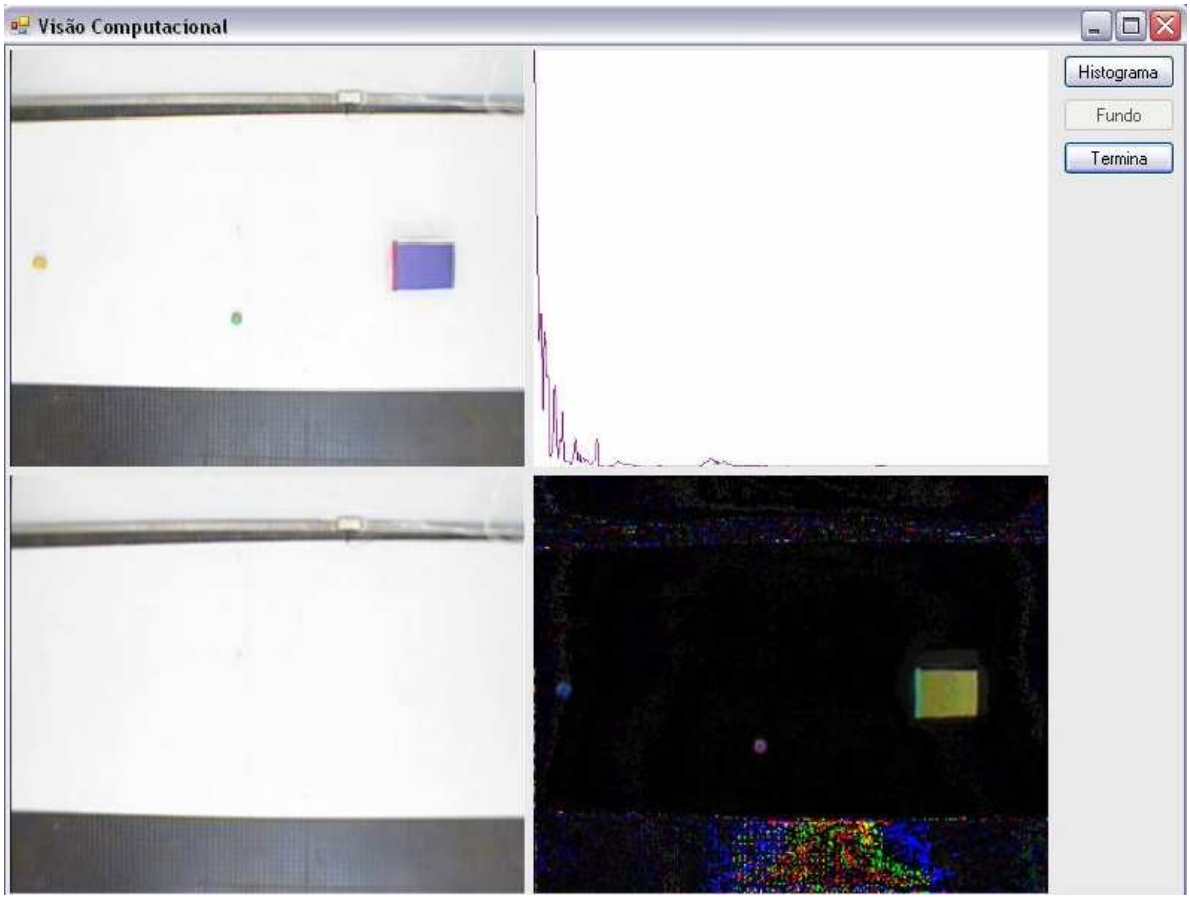

Figura 27 – Histograma com o sistema com objetos.

A Informação do histograma foi muito útil para identificar as faixas de freqüências dos objetos. Com essas faixas de freqüências determinadas, os filtros passa-alta e passa-baixa puderam ser usados para retirar alguns ruídos do sistema, partindo do principio que se não é carrinho, nem alvo, nem obstáculos é um ruído. O resultado foi bastante positivo.

A próxima etapa do trabalho é definir, portanto, quem era objeto e quem não era, ou seja fazer a identificação. A idéia inicial era binarizar toda a informação para que o sistema trabalhasse apenas com preto e branco e assim diminuíssem as operações matemáticas. Identificaríamos as *bounding boxes* dos objetos por área, ou seja, o que tivesse a área maior seria o “carro” e os que tivessem as menores áreas seriam os objetos. Assim como ilustra a figura 28.

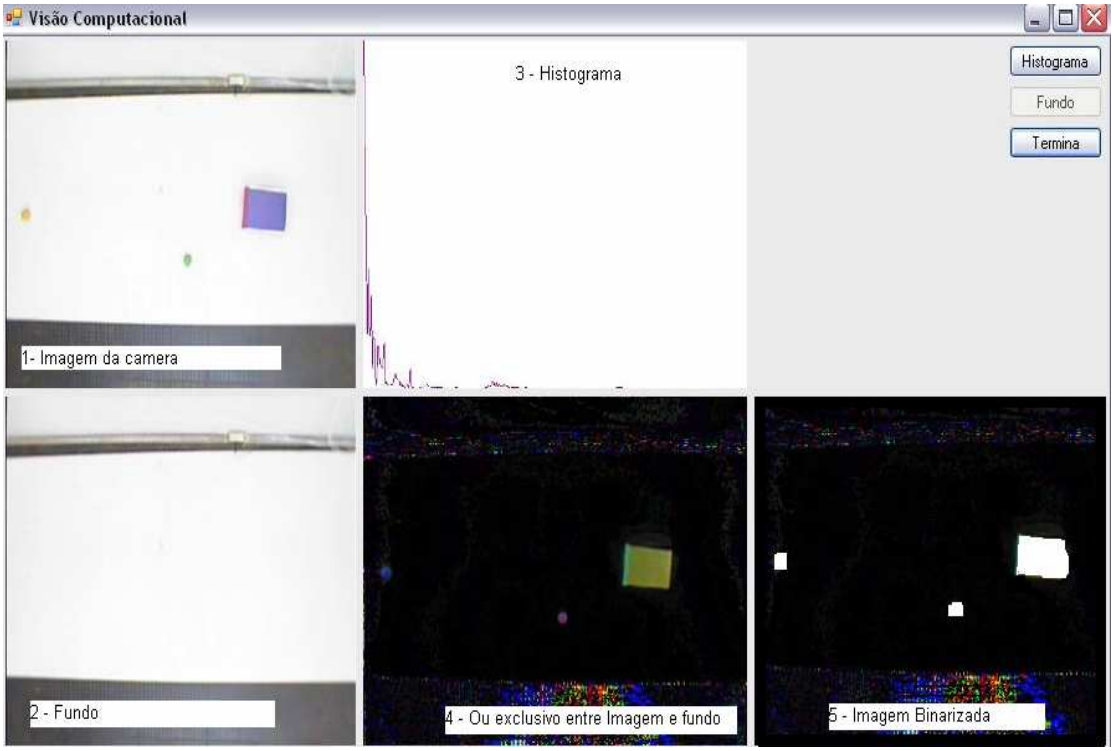

Figura 28 – Etapas para Binarização.

Entretanto essa idéia foi implementada, mas logo foi abandonada. Por dois motivos importantes. Se houvesse um obstáculo maior que o "carro" esse método falharia e em trabalhos futuros esse método seria pouco proveitoso visto que, haveria várias limitações de tamanho. Então, optou-se pelo método de identificação por cores, dessa forma as limitações anteriormente citadas estariam superadas e a escalabilidade (indica sua habilidade de manipular uma porção crescente de trabalho de forma uniforme, ou estar preparado para o crescimento do mesmo) do sistema seria mantida.

O primeiro problema enfrentado ao utilizar o método de identificação por cores foi o padrão de cores *RGB*, utilizado pela linguagem de programação. O problema do padrão *RGB,* nesta aplicação*,* é que ele não é linear. Por esse motivo não há como delimitar as faixas de cores. Através de estudo do sistema de cores ficou constatado que o sistema *HSB* é continuo e não apresenta "saltos", podendo então ser utilizado para esse propósito. Esse sistema foi o escolhido para se determinar as faixas de cores utilizadas neste trabalho. Então o software faz a conversão de cores de *RGB* para *HSB,* baseado na equação demonstrada no capitulo 2, seção 8. Com o bitmap definido neste novo padrão, HSB/HSV, é

necessário determinar as faixas de freqüência de cada objeto, dependendo das suas cores. Utilizando métodos empíricos, através do software gratuito (freeware) ANY COLOR PICKER, de [28], foram definidas as faixas, conforme ilustra a tabela 5.

Tabela 5 – Faixas para os objetos no padrão HSB

Objeto	**Valor H**	**Valor S**	**Valor B**
Carro – Cor Azul	(240 +- 11) graus	(40 +-28)%	(75 +- 28)%
Frente do carro – Cor Vermelha	(105 +- 35) graus	(25+-30)%	(82 +- 30)%
Obstáculo – Cor Verde	(320 +- 33) graus	(40+-30)%	(95 +- 30)%
Chegada – Cor Amarela	(50 +-12) graus	(40+-20)%	(88 +- 18)%

Pela tabela 5 é possível observar os parâmetros de cada cor, dos objetos e suas respectivas tolerâncias. Pois devido à freqüência da iluminação, e outros fatores anteriormente citados, como a incidência, por exemplo, da luz solar, a reflexão da imagem para a câmera tende a variar. Com essas tolerâncias a identificação ocorreu sem maiores problemas para o propósito deste trabalho, onde essas variações, que são perceptíveis não comprometem o objetivo do carro em atingir o alvo. Com esses parâmetros, é possível então buscar as *bounding boxes* de cada objeto, completando desta forma a identificação dos mesmos.

Tendo a imagem convertida no padrão HSB e com seus parâmetros identificados é possível agora fazer a varredura na imagem para se conseguir identificar os objetos. Essa varredura utiliza os seguintes métodos de [26]:

- AABB (*Axis-aligned bounding boxes*) - onde os volumes escolhidos para encapsular os objetos são caixas alinhadas com os eixos de orientação.
- SDM (*Signed Distance Map*) - Consiste em inscrever o objeto dentro de uma caixa alinhada com os eixos e dividi-la de maneira a formar um grid em duas dimensões.
- ORIENTED BOUNDING BOX (OBB) - emprega caixas para aumentar o desempenho da colisão, mas neste caso a caixa possui uma orientação que descreve o seu próprio espaço.
- TESTE DE COLISÃO EM ÁRVORES OBB´S.
- COVARIÂNCIA E BOUNDIG BOX 2D.

Após a execução dos métodos de [26], o software obtêm as informações, referentes à localização relativa na área de testes, de todos os objetos. Essas informações são, formalmente, descritas por posição X e Y do centro do objeto, e a partir do centro seu raio em X e Y, RX e RY. Como na tabela 6.

Tabela 6 – Descrição dos Objetos

OBJETO	COR	CENTRO	RAIO	OCUPAÇÃO
Carro	AZUL	carroRX, carroRY	carroRaioX, carroRaioY	Veiculo não tripulado
Frente do Carro	Vermelha	frenteCarroRX, frenteCarroRY	frenteCarroRaioX, frenteCarroRaioY	Extensão do veiculo não tripulado
Bolinha Verde	Verde	bolinhaVerdeRX, bolinhaVerdeRY	bolinhaVerdeRaioX, bolinhaVerdeRaioY	Obstáculos
Bolinha Amarela	Amarela	bolinhaAmarelaRX, BolinhaAmarelaRY	bolinhaAmarelaRaioX, bolinhaAmarelaRaioY	Alvo

Com as informações *bounding boxes* definidas, são adotadas cores a fim de delimitar e facilitar a visualização por parte do usuário.dos objetos. A figura 29 ilustra as cores adotadas, a saber, carro – preto; frente do carro – laranja; obstáculos – rosa; alvo – dourado.

Figura 29 – Delimitações e cores das *bounding boxes*

Com esse método a identificação dos objetos foi um sucesso, porém percebe-se que as *bounding boxes* podem sofrer variações em determinadas posições, devido aos problemas de iluminação já mencionados. Contudo essas variações não comprometem o objetivo do trabalho sendo assim a etapa de visão computacional está concluída.

7.4. Aplicação da Função Potencial

Estando, as informações de posição, inseridas no sistema é necessário, agora, um método inteligente para tratar essas informações e controlar os objetos de forma a cumprirem sua tarefa. Como citado no decorrer deste trabalho o método utilizado foi o de função potencial. Em contrapartida o simulador apresentado por [25], que foi o ponto de partida, utilizava como método inteligente a lógica *fuzzy. O* simulador sofreu uma substituição no método de inteligência artificial, substituiu-se a lógica *fuzzy* pela função potencial, com o objetivo de aperfeiçoar este trabalho, isso, devido às características da função potencial (1 - Um simples algoritmo tem a função de perseguir e desviar ou seja, não é preciso outras condições e controles lógicos associados a este algoritmo; 2 - A demanda computacional é extremamente baixa; 3 - É muito simples a implementação).

O método da função potencial, nesta aplicação, atua da seguinte forma: Com os dados de todos os objetos do sistema é feito um calculo para achar a força resultante. Esta força depende da distancia do veiculo autônomo para os obstáculos ou alvo. Quanto mais próximo o carro se aproximar de um obstáculo maior será a força de repulsão entre eles. Caso contrário, quando o carro estiver mais próximo do alvo, mais forte será à força de atração entre eles. Este conceito é ilustrado na figura 30.

Figura 30 – “Forças” atuando sobre o veículo

Conforme observado na figura 30, é válido afirmar, também, que a cada instante de tempo, desde que o veículo tenha se movimentado, os valores resultantes das forças envolvidas serão distintos, gerando outra possível direção para a movimentação do veículo, sendo necessário, portanto um novo cálculo. Esse é um dos motivos que o programa trabalha utilizando iterações entre cada movimento do veículo. Outro motivo, para uso das iterações, é o tempo de resposta do veículo que é muito baixo, pois o carrinho é um brinquedo, adaptado para esse trabalho, e como tal seu circuito é bem simples e não oferece muitos recursos nem respostas rápidas aos comandos.

Como saída da função potencial temos um ângulo. Esse ângulo é formado entre direção do carrinho, vetor direção, com a força resultante dos cálculos da função potencial. O vetor direção é obtido dos pontos do centro do carro (carroRX,

carroRY), até os pontos do centro de sua frente (frenteCarroRX, frenteCarroRY), como mostra a figura 31.

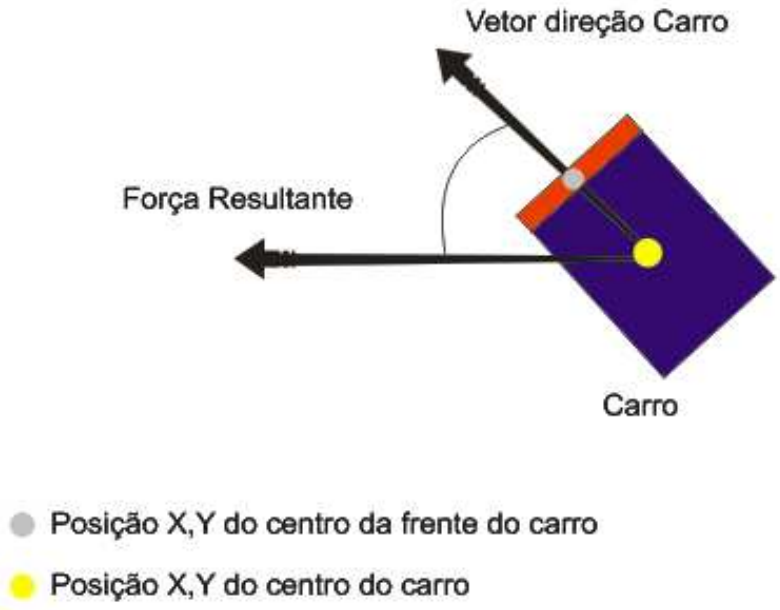

Figura 31 – Ângulo entre a força resultante e o vetor direção do carro

Esse ângulo é o resultado esperado para a decisão da direção do carro. Ele foi obtido de forma empírica através de testes comparativos de desempenho A partir dele pode-se criar a regra para a direção, ou seja:

- Se -5°< Ângulo < 5°-> seguir em frente;
- Se Ângulo < -5°-> virar a esquerda;
- Se ângulo > 5°-> Virar á direita.

Com este resultado o software envia a informação para o carro, este executa a ação. Caso o software constate que o carro ainda não atingiu seu objetivo, chegar ao alvo, outra iteração será feita, e assim sucessivamente, até o sucesso da tarefa.

7.5. Controle e Dinâmica do Veículo

Quando o sinal de radiofreqüência chega até o veiculo autônomo este se movimenta conforme a instrução recebida do computador. Essa instrução tem

apenas quatro valores: frente, frente-esquerda, frente-direita e não fazer nada. A dinâmica do veículo autônomo adotado neste trabalho é muito limitada, dado a sua simplicidade (é um brinquedo). Essa limitação foca-se principalmente no esterço das rodas dianteira que não é controlado, ou seja, ou a roda está reta, ou toda virada para a esquerda ou toda virada para a direita. Isso é contornado utilizando-se as iterações com uma freqüência pré-estabelecida, pulsos a cada 3 segundos, que foi obtida empiricamente através de testes. Por esse motivo a área de testes comportou apenas dois obstáculos, visto que para o uso de um terceiro, ou mais, seriam necessárias posições pré-estabelecidas para esses obstáculos, o veículo e o alvo. O alvo e os obstáculos precisam estar separados por uma distancia de pelo menos duas vezes o comprimento do carrinho para que o sistema funcione perfeitamente. Esta é a única ressalva para que esse sistema funcione perfeitamente. Caso o carrinho pudesse “andar de lado”, ou seja, para cima, para baixo, para frente e para trás essa limitação não existiria e poderíamos até colocar mais objetos.

O sistema, hoje, comporta apenas obstáculos estáticos por dois motivos: a limitação dinâmica do carrinho e a pequena área do campo de testes. O software está apto, a qualquer instante, a tratar esse sistema dinamicamente, com os objetos em movimento, como no simulador de [25], desde que existam condições físicas para isso, isto é, um carrinho com um controle mais preciso e uma área de testes com uma dimensão maior. Porém para proposta deste trabalho, a tarefa do carrinho em se atingir o alvo, desviando dos obstáculos, pode ser considerada completa.

8. Conclusão e Trabalhos Futuros

8.1. Conclusão

Este trabalho de dissertação propôs a criação de uma aplicação prática a partir de um sistema funcional em um simulador. Utilizando ferramentas de visão computacional e de inteligência artificial, foi possível controlar um veículo autônomo com a finalidade de desviar de obstáculos e atingir um alvo. Apesar de todas as dificuldades encontradas para traduzir as informações da simulação para o mundo real, o sistema consegue ser suficientemente estável e funcional para execução da tarefa proposta. O cenário utilizado para validação do método foi um ambiente físico real, área de testes, e que, apesar da simplicidade, foi suficiente para validar a aplicação proposta. O resultado apresentado corrobora a metodologia ao conseguir solucionar o problema através de objetos do mundo real (carro, obstáculos e alvo). O sistema se mostrou extremamente flexível, e com grande poder de aplicabilidade podendo ser aprimorado e melhorado futuramente.

Como este trabalho tem resultados práticos algumas imagens e vídeos colaboram com as afirmações aqui apresentadas. As figuras 32, 33 e 34 mostram a rota usada pelo carrinho para atingir seu objetivo, vista do software. Em [29] (http://www.youtube.com/watch?v=0Ind3EN0n2Q) é possível encontrar um vídeo que contém várias simulações, em diversas situações diferentes da posição do alvo, obstáculos e saída do carrinho, que comprovam o sucesso da aplicação.

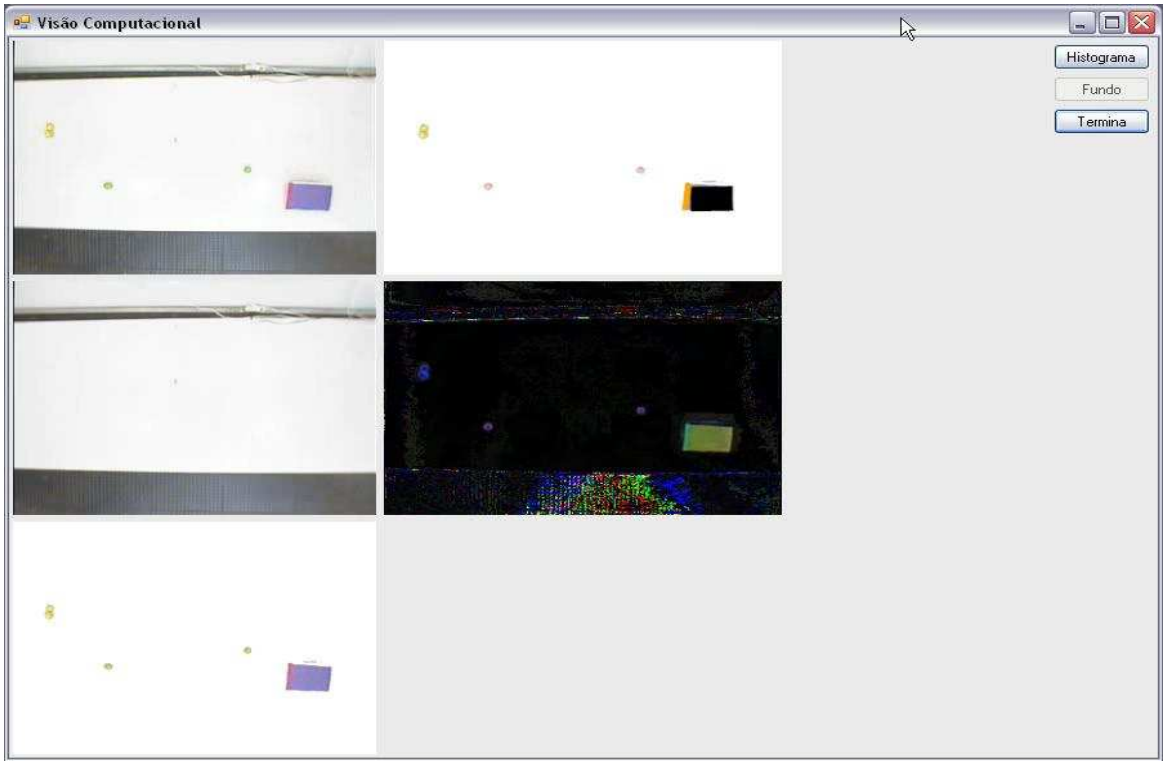

Figura 32 – Carrinho Na posição de largada

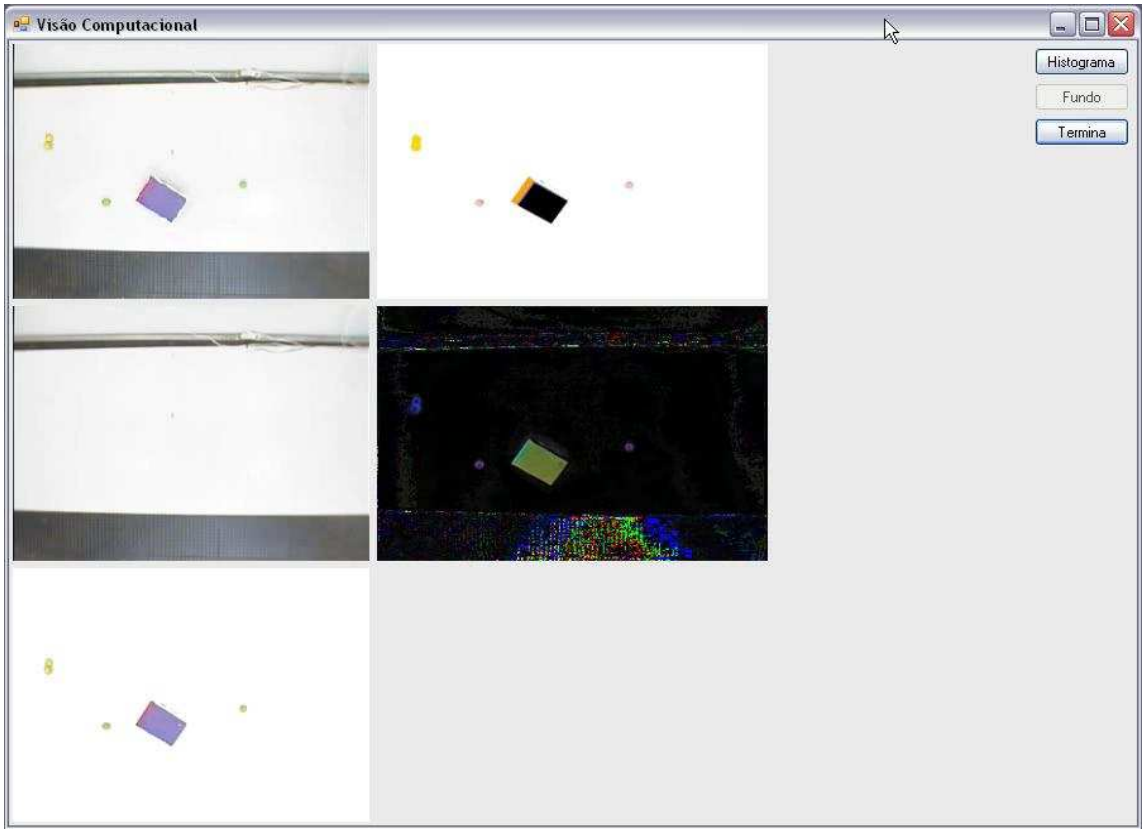

Figura 33 – Carrinho em uma posição intermediária

Figura 34 – Carrinho atingindo o alvo

8.2. Trabalhos Futuros

O sistema construído exigiu um grande conhecimento da linguagem C#, visão computacional, programação em DirectX, conhecimento de inteligência artificial, entre outros. Contudo, graças à orientação a objetos, a tarefa de futuras modificações no programa, ou, simplesmente, a extração de funcionalidades, fica bastante simples, oferecendo uma ótima escalabilidade para possíveis mudanças e outras aplicações.

Com um veículo melhor, com controle mais robusto, uma área com o sistema de iluminação controlado e com maiores dimensões físicas, as possibilidades são inúmeras. Podendo ser elaborados veículos inteligentes com diversas funções. Como exemplo, poder-se-ia ser um veículo autônomo em uma fábrica, com a função de transporte, de insumos, material, etc. Este exemplo abrange todas as situações deste trabalho ou seja, na fábrica existe um alvo, que é

o lugar onde o veiculo deve chegar, existe também obstáculos nesse caminho, paredes, pessoas, maquinas etc. Poderiam também ser utilizados os método de visão computacional com a função de identificar objetos em outras situações e para diferentes aplicações visto que o mesmo mostrou-se muito eficaz no mundo real.

Seria interessante também, a adequação deste sistema para funcionar em três dimensões, *oriented bounding box 3D*, citado por [26]. Com essa funcionalidade a identificação e controle poderiam ser estendidos para sistemas multicâmeras em tempo real. E com isso, poderia, por exemplo, fazer o controle de um helicóptero.

Referências Bibliográficas

[1] POMERLEAU, D. Neural network based autonomous navigation. In: Thorpe, C.(Ed). Vision and Navigation: The CMU Navlab. Kluwer Academic Publihers, 1990.

[2] DAVID M. BOURG, GLENN SEEMANN.: AI for Game Developers**,** Ed. O'Reilly, 2004.

[3] MEDEIROS, ADELARDO. Introdução a Robótica. ENA-98 Encontro Nacional de Automática (50º Congresso da SBPC). Natal, RN. pp.56-65. 1998.

[4] RAFAEL C. GONZALES, RICHARD E. WOODS.: Processamento de Imagens Digitais. Brasil, 2000. Ed. EDGARD BLUCHER LTDA.

[5] STONE, H. W.: Mars Pathfinder Microver. A low-cost, low-power Spacecraft. Proceeding of the 1996 AIAA. Forum on advanced developments in Space Robotics. Madison, WI. August, 1996.

[6] LEMONICK, MICHEL.: Dante Tours the Inferno. Time Magazine – Time Domestic/Science. Vol. 144, No. 7. August 15, 1994.

[7] SHEUER, A. & LAUGIER, C.: Planning Sub-Optimal and Continuous-Curvature Paths for Car-Like Robots. IEEERSJ International Conference on Intelligent Robots and Systems. Victoria, British-Columbia, Canada. Oct. 1998.

[8] MARQUES FILHO, OGÊ & VIEIRA NETO, HUGO.: Processamento Digital de Imagens. Rio de Janeiro, 1999. Ed. BRASPORT.

[9] MARIE FARINES, JEAN & SILVA FRAGA, JONI & SILVA OLIVEIRA, ROMULO.: Sistemas de tempo real. UFSC, Brasil, 2000. Ed. Escola de Computação.

[10] MÁRCIO PORTES DE ALBUQUERQUE & MARCELO PORTES DE ALBUQUERQUE.: Processamento de Imagens: Métodos e Análises. Artigo. Centro Brasileiro de Pesquisas Físicas – CBPF/MCT.

[11] Sistema De Cores. Wikipedia, 2006. Disponível em: <http://pt.wikipedia.org/wiki/HSV/>. Acesso em: 12 Abr. 2006.

[12] CLÁUDIO ROSITO JUNG, FERNANDO SANTOS OSÓRIO, CHRISTIAN ROBERTO KELBER E FARLEI JOSÉ HEINEN.: Computação Embarcada: Projeto e Implementação de Veículos Inteligentes Inteligentes. Rio Grande do Sul, Brasil, 2005. CSBC 2005 - XXV Congresso da Sociedade Brasileira de Computação.

[13] DUDEK, G. AND JENKIN, M. (2000). *Computational Principles of Mobile Robotics. Cambridge University Press, Cambridge, UK.*

[14] HEINEN, F. J. (2000). *Robótica Autônoma: Integração entre Planificação e Comportamento Reativo*. Série Produção Discente, Editora Unisinos. São Leopoldo, RS. Web: http://ncg.unisinos.br/robotica/ (acessado: maio de 2007).

[15] FARINES, JEAN-MARIE & SILVA FRAGA, JONI & SILVA OLIVEIRA, RÔMULO.: Sistemas de Tempo real. Santa Catarina, Brasil, 2000. DAS – UFSC.

[16] J. A. STANKOVIC, *Misconceptions about real time computing*, IEEE Computer, vol 21 (10), October 1988.

[17] G.C. BUTTAZZO, *Hard Real Time Computing Systems: Predictable Scheduling Algorithms and Applications*, Ed. Kluwer Academics Publishers, 1997.

[18] DELTA 4, *Real Time Concepts, on Delta 4 Architecture Guide*, Cap.5, pp.102 124, 1991.

[19] M. JOSEPH.: *Problems, Promisse and Performance: Some Questions For Real Time Specifications, On proceedings of Rex Workshop on Real-Time: Theory in Practice, Lecture notes in computer science No. 600, June, 1991, pp.315-324. Ed. Spring Verlag.*

[20] R. MILNER, *A Calculus of Communicating Systems, Lecture Notes in Computer Science, vol 92, 1980, Ed. Springer Verlag*.

[21] G. BERRY, *Real Time Programming: Special Purpose or General Purpose Languages, In Information Processing 89, pp.11-17, Ed. Elsevier Science Publishers, 1989.*

[22] MEDEIROS, ADELARDO A.D. (1998). *A Survey of Control Architectures for Autonomous Mobile Robots. JBCS* - Journal of the Brazilian Computer Society, special issue on Robotics, vol. 4, n. 3.

[23] DAVID M. BOURG, GLENN SEEMAN.: *AI for Game Developers. Publisher: O'Reilly Pub Date: July* 2004.

[24] BARR, A.; FEIGENBAUM, E. (Ed.). *The Handbook of Artificial Intelligence*. Los Altos, California: William Kaufmann Inc., 1981. v.I.II.

[25] HONÓRIO, Leonardo de Mello ; BARBOSA, Daniele Alcântara ; SOUZA, Luis Edival de . Sistema de Desenvolvimento de Lógica Fuzzy Orientado a Objetos. In: Congresso Nacional de Tecnologia da Informação, SUCESU2004, 2004, Florianóplois, 2004.

[26] HONÓRIO, Leonardo de Mello ; DIAS, Welinton ; FREIRE, Muriel ; SOUZA, Luiz Edival de . Ambiente de Manufatura Virtual 3D: Desenvolvimento de Equipamentos e Testes de Colisões em Tempo Real. In: XVI Congresso Brasileiro de Automática, 2006, Salvador. CBA2006, 2006. v. XVI. p. 911-916.

[27] Introdução ao DirectX. Microsoft Corporation, 2005. Disponível em: <http://www.microsoft.com/brasil/msdn/Tecnologias/arquitetura/DirectX.mspx/>. acesso em: 03 Abr. 2007.

[28] ANY COLOR PICKER. Anry 2003-2007. Disponível em: <http://www.anryhome.com/software/colorpicker/>. Acesso em 03 Fev, 2006.

[29] You Tube. Google inc 2007. Vídeo com os resultados do trabalho. Ref.: <http://www.youtube.com/watch?v=0Ind3EN0n2Q>

[30] JESUS, Edison Oliveira de. SRIDE – Sistema Reconhecedor de Imagens de Diagramas Elétricos. Tese de Doutorado. Campinas, Brasil 1997.

APÊNDICE

Neste apêndice estão inseridos os principais códigos do programa desenvolvido, o que pode complementar o entendimento dos métodos utilizados neste trabalho. Estes códigos estão caracterizados com as mesmas definições de cores utilizadas pelo VISUAL STUDIO 2005. O que facilita o entendimento do mesmo.

Arquivo FormPrincipal.cs

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Drawing.Imaging;
using System.Drawing.Drawing2D;
using System.Text;
using System.Windows.Forms;
using DShowNET;
using DShowHlpr;
using RetiraFundo;
using System.Threading;

namespace Principal
{
    public partial class FormPrincipal : Form
    {
        public static float posicaoXObjetivo;
        public static float posicaoYObjetivo;
        public static float ObjetivoRX;
        public static float ObjetivoRY;

        public static float posicaoXFrenteCarrinho;
        public static float posicaoYFrenteCarrinho;
        public static float posicaoXCarrinho;
        public static float posicaoYCarrinho;
        public static float CarrinhoRX;
        public static float CarrinhoRY;

        public static float posicaoXObstaculo1;
        public static float posicaoYObstaculo1;
        public static float Obstaculo1RX;
        public static float Obstaculo1RY;

        public static float posicaoXObstaculo2;
        public static float posicaoYObstaculo2;
        public static float Obstaculo2RX;
        public static float Obstaculo2RY;

        private DShowHelper dh;
        private bool        primeiro   = true;
        private bool        capturando = false;
        private System.Windows.Forms.Timer timercaptura;
```

```
        private System.Windows.Forms.Timer tempoThread;

        ControleVeiculo controle;

        float[] anguloCarro = new float[11];
        float[] anguloDesejado = new float[11];
        int contador = 0;
        float anguloCarroMedia = 0;
        float anguloDesejadoMedia = 0;

        int condicaoDeParada = 0;

        float contador2 = 0;
        float[] pXInicioCarro = new float[5];
        float[] pXFimCarro = new float[5];
        float[] pYInicioCarro = new float[5];
        float[] pYFimCarro = new float[5];

        float pXImedio = 0;
        float PXFmedio = 0;
        float pYImedio = 0;
        float pYFmedio = 0;
        //SaidaPortaParalela estadoInicialCarro = new
SaidaPortaParalela(0);
        //System.Threading.Thread threadCarrinho = new
System.Threading.Thread(saidaPortaParalela.direcaoCarro);

        public FormPrincipal()
        {
            InitializeComponent();
        }

        private void FormPrincipal_Activated(object sender, EventArgs e)
        {

            if (primeiro)
            {

                primeiro = false;

                dh = new DShowHelper(panelCaptura);
                dh.Ativa();

                //limitação para "filtragem"
                float limiteInferior = 0.0f;
                float limiteSuperior = 255.0f;

                BinarizaComBoundingBox.limiteinferior = limiteInferior;
                BinarizaComBoundingBox.limitesuperior = limiteSuperior;
                BinarizaPegandoACorOriginal.limiteinferior =
limiteInferior;
                BinarizaPegandoACorOriginal.limitesuperior =
limiteSuperior;

            }

        }
        /*
        private void buttonQuadro_Click(object sender, EventArgs e)
        {
```

```
        }
         * */

        private void buttonFundo_Click(object sender, EventArgs e)
        {


            if (pictureBoxFundo.Image != null)
            {

                pictureBoxFundo.Image.Dispose();

            }

            if (System.IO.File.Exists("fundo.bmp"))
            {

                pictureBoxFundo.Image = Bitmap.FromFile("fundo.bmp");

            }
            else
            {

                pictureBoxFundo.Image = dh.CapturaQuadro();
                pictureBoxFundo.Image.Save("fundo.bmp",
System.Drawing.Imaging.ImageFormat.Bmp);

            }

        }

        private void buttonInicia_Click(object sender, EventArgs e)
        {

            //estadoInicialCarro.parar();
            if (!capturando)
            {

                buttonInicia.Text    = "Termina";
                buttonFundo.Enabled  = false;
                buttonHistograma.Enabled = true;

                timercaptura = new System.Windows.Forms.Timer();
                timercaptura.Interval = 33; //33fps
                timercaptura.Tick += new EventHandler(timercaptura_Tick);
                timercaptura.Enabled = true;

                tempoThread = new System.Windows.Forms.Timer();
                tempoThread.Interval = 1; // .9 segundos //default .9
segundos
                tempoThread.Tick += new EventHandler(tempoThread_Tick);
                tempoThread.Enabled = true;



            }
            else
            {
```

```
                buttonInicia.Text    = "Inicia";
                timercaptura.Dispose();
                tempoThread.Dispose();
                buttonFundo.Enabled  = true;
                buttonHistograma.Enabled = true;

            }

            capturando = !capturando;

        }

        void timercaptura_Tick(object sender, EventArgs e)
        {




            if (pictureBoxImagemTransformada.Image != null)
            {

                pictureBoxImagemTransformada.Image.Dispose();

            }

            Bitmap b = dh.CapturaQuadro();
            pictureBoxImagemTransformada.Image = Retira.Executa(b,
(Bitmap)pictureBoxFundo.Image, false);




            if (pictureBoxHistograma.Image != null)
            {

                pictureBoxHistograma.Image.Dispose();

            }

            //pictureBoxHistograma.Image =
Histograma.Executa((Bitmap)pictureBoxImagemTransformada.Image);

            if (pictureBoxBinarizacao.Image != null)
            {

                pictureBoxBinarizacao.Image.Dispose();

            }

            //pictureBoxBinarizacao.Image =
BinarizaComBoundingBox.Transforma((Bitmap)pictureBoxImagemTransformada.Imag
e);


            if (pictureBoxImagemComCor.Image != null)
            {
                pictureBoxImagemComCor.Image.Dispose();
            }
```

```
            pictureBoxImagemComCor.Image =
BinarizaPegandoACorOriginal.recuperaCorOriginal(b,
(Bitmap)pictureBoxImagemTransformada.Image);

            //de posse da imagem filtrada e perfeita .. hehehe calcula as
bounding boxes dos elementos
            //presentes :-)
            OrientedBoundingBox2D[] bbs;

            //objetos da cena


            DadoCorObjeto[] dco = new DadoCorObjeto[] { new
DadoCorObjeto(new CorHSB(246.0f, 0.335f, 0.74f) , new CorHSB(17.0f , 0.15f,
0.15f)),
                                                        new
DadoCorObjeto(new CorHSB(87.0f, 0.45f, 0.85f) , new CorHSB(14.0f , 0.18f,
0.14f)),
                                                        new
DadoCorObjeto(new CorHSB(332.0f, 0.34f , 0.68f) , new CorHSB(20.0f , 0.2f,
0.2f)),
                                                        new
DadoCorObjeto(new CorHSB (54.0f, 0.5f, 0.88f) , new CorHSB(12.0f , 0.3f,
0.2f))};
            float[] tolerancias2 = new float[] { 100.0f * 100.0f,
                                                 15.0f *  15.0f  ,
                                                 15.0f *  15.0f  ,
                                                 15.0f *  15.0f };

            pictureBoxHistograma.Image =
DivideObjetos.Executa((Bitmap)pictureBoxImagemComCor.Image, dco,
tolerancias2, out bbs);

            b.Dispose();


        }

        void tempoThread_Tick(object sender, EventArgs e)
        {

            if (condicaoDeParada == 0)
            {
                if (DivideObjetos.posicaoXObjetivo != 0 &&
DivideObjetos.posicaoXObstaculo1 != 0 && DivideObjetos.posicaoXObstaculo2
!= 0)
                {
                    posicaoXObjetivo = DivideObjetos.posicaoXObjetivo;
                    posicaoYObjetivo = DivideObjetos.posicaoYObjetivo;
                    ObjetivoRX = DivideObjetos.ObjetivoRX;
                    ObjetivoRY = DivideObjetos.ObjetivoRY;

                    posicaoXObstaculo1 = DivideObjetos.posicaoXObstaculo1;
                    posicaoYObstaculo1 = DivideObjetos.posicaoYObstaculo1;
                    Obstaculo1RX = DivideObjetos.Obstaculo1RX;
                    Obstaculo1RY = DivideObjetos.Obstaculo1RY;

                    posicaoXObstaculo2 = DivideObjetos.posicaoXObstaculo2;
                    posicaoYObstaculo2 = DivideObjetos.posicaoYObstaculo2;
                    Obstaculo2RX = DivideObjetos.Obstaculo2RX;
                    Obstaculo2RY = DivideObjetos.Obstaculo2RY;
```

```
                    condicaoDeParada++;
                }
                else
                    condicaoDeParada = 0;


            }

            if (DivideObjetos.CarrinhoRX != 0 && posicaoXObjetivo != 0)
            {
                /*
                posicaoXObjetivo = DivideObjetos.posicaoXObjetivo;
                posicaoYObjetivo = DivideObjetos.posicaoYObjetivo;
                ObjetivoRX = DivideObjetos.ObjetivoRX;
                ObjetivoRY = DivideObjetos.ObjetivoRY;
                 * */

                    posicaoXFrenteCarrinho =
DivideObjetos.posicaoXFrenteCarrinho;
                    posicaoYFrenteCarrinho =
DivideObjetos.posicaoYFrenteCarrinho;
                    posicaoXCarrinho = DivideObjetos.posicaoXCarrinho;
                    posicaoYCarrinho = DivideObjetos.posicaoYCarrinho;
                    CarrinhoRX = DivideObjetos.CarrinhoRX;
                    CarrinhoRY = DivideObjetos.CarrinhoRY;
                    /*
                    posicaoXObstaculo1 = DivideObjetos.posicaoXObstaculo1;
                    posicaoYObstaculo1 = DivideObjetos.posicaoYObstaculo1;
                    Obstaculo1RX = DivideObjetos.Obstaculo1RX;
                    Obstaculo1RY = DivideObjetos.Obstaculo1RY;

                    posicaoXObstaculo2 = DivideObjetos.posicaoXObstaculo2;
                    posicaoYObstaculo2 = DivideObjetos.posicaoYObstaculo2;
                    Obstaculo2RX = DivideObjetos.Obstaculo2RX;
                    Obstaculo2RY = DivideObjetos.Obstaculo2RY;

                    */
                    controle = new ControleVeiculo(posicaoXObjetivo,
posicaoYObjetivo, ObjetivoRX, ObjetivoRY, posicaoXFrenteCarrinho,
                    posicaoYFrenteCarrinho, posicaoXCarrinho,
posicaoYCarrinho, CarrinhoRX, CarrinhoRY, posicaoXObstaculo1,
posicaoYObstaculo1, Obstaculo1RX,
                    Obstaculo1RY, posicaoXObstaculo2, posicaoYObstaculo2,
Obstaculo2RX, Obstaculo2RY);


                    anguloCarro[contador] = controle.carroAlinhamento();
                    anguloDesejado[contador] =
controle.direcaoDejesadaCarro();
                    if (contador == 10)
                    {

                        for (int i = 0; i < 11; i++)
                        {
                            anguloCarroMedia += anguloCarro[i];
                            anguloDesejadoMedia += anguloDesejado[i];
                        }
                        anguloCarroMedia /= 10;
                        anguloDesejadoMedia /= 10;
```

```
                        RetornaMediaControle valorDirecao = new
RetornaMediaControle(anguloCarroMedia, anguloDesejadoMedia,
posicaoXObjetivo, posicaoXCarrinho);

                        //define da direcao que o carrinho irá tomar como
resposta a classe de controle do veiculo!
                        ThreadControle controleCarrinho = new
ThreadControle(valorDirecao.direcao());
                        Thread threadSecundaria = new
Thread(controleCarrinho.iniciaControle);
                        threadSecundaria.Start();

                        contador = 0;
                        anguloCarroMedia = 0;
                        anguloDesejadoMedia = 0;

                    }
                    else
                        contador++;

                }


        }

        private void panelCaptura_Paint(object sender, PaintEventArgs e)
        {

        }

        private void buttonHistograma_Click(object sender, EventArgs e)
        {

            //Histograma histograma = new
Histograma((Bitmap)pictureBoxImagemTransformada.Image);

            //if (pictureBoxHistograma.Image != null)
            //{

            //    pictureBoxHistograma.Image.Dispose();

            //}

            //pictureBoxHistograma.Image = histograma.Executa();
        }

    }
}
```

Fim do arquivo FormPrincipal.cs

Arquivo CorHSB.cs

```
using System;
using System.Collections.Generic;
using System.Text;
using System.Drawing;


namespace Principal
{

    public class DadoCorObjeto
    {

        public CorHSB c1;
        public CorHSB c2;

        public DadoCorObjeto(CorHSB c1, CorHSB c2)
        {

            this.c1 = c1;
            this.c2 = c2;

        }

        //retorna true se a cor especificada pertence à faixa
        public bool CorDentroFaixa(CorHSB c)
        {

            // c2 é tolerância ... e não mais uma cor ... :D
            float v1 = c1.H - c2.H;
            float v2 = c1.H + c2.H;

            bool dentrofaixa;
            if (v1 < 0)
            {

                dentrofaixa = !(c.H >= v2 && c.H <= (360.0f + v1));

            }
            else if (v2 > 360.0f)
            {

                dentrofaixa = !(c.H >= (v2 - 360.0f) && c.H <= v1);

            }
            else
            {

                dentrofaixa = (c.H >= v1 && c.H <= v2);

            }

            v1 = c1.S - c2.S;
            v2 = c1.S + c2.S;

            dentrofaixa = dentrofaixa && (c.S >= v1 && c.S <= v2);

            v1 = c1.B - c2.B;
            v2 = c1.B + c2.B;
```

```
            dentrofaixa = dentrofaixa && (c.B >= v1 && c.B <= v2);

            return dentrofaixa;

        }

    }

    public class CorHSB
    {

        public float H;
        public float S;
        public float B;

        public CorHSB(float h, float s, float b)
        {

            H = h;
            S = s;
            B = b;

        }

        public static CorHSB DeRGB(int r, int g, int b)
        {

            Color c = Color.FromArgb(r, g, b);
            return new CorHSB(c.GetHue(), c.GetSaturation(),
c.GetBrightness());

        }

    }

}
```

Fim do Arquivo CorHSB.cs

Arquivo BoundingBox2D.cs

```
using System;
using System.Collections.Generic;
using System.Text;
using System.Drawing;

namespace RetiraFundo
{

    public class OrientedBoundingBox2D
    {

        //atributos privados
        private Matriz2x2 orientacao;
        private Matriz2x2 orientacaot;
        private Vetor2D   centro;
        private float rx;
        private float ry;

        //filhos
        public OrientedBoundingBox2D[] filhos = new
OrientedBoundingBox2D[2];

        //construtores
        public OrientedBoundingBox2D()
        {
        }

        //propriedades públicas
        public Matriz2x2 Orientacao
        {

            get
            {

                return orientacao;

            }

            set
            {

                orientacao  = value;
                orientacaot = orientacao.Transposta();

            }

        }

        public Vetor2D Centro
        {

            get
            {

                return centro;

            }
```

```
            set
            {

                centro = value;

            }

        }

        public float Rx
        {

            get
            {

                return rx;

            }

            set
            {

                rx = value;

            }

        }

        public float Ry
        {

            get
            {

                return ry;

            }

            set
            {

                ry = value;

            }

        }

        //métodos públicos
        public void Processa(Vetor2D v)
        {

            Vetor2D temp;

            temp = v - centro;
            temp = orientacaot * temp;

            temp = new Vetor2D((float)Math.Abs(temp.X),
(float)Math.Abs(temp.Y));

            //acerta os raios
            if (temp.X > rx)
```

```
            {

                rx = temp.X;

            }

            if (temp.Y > ry)
            {

                ry = temp.Y;

            }

        }

        public void Processa(Vetor2D[] pontos)
        {

            foreach (Vetor2D v in pontos)
            {

                Processa(v);

            }

        }

        public PointF[] RetornaPontosMundo()
        {

            PointF[] temp = new PointF[4];
            Vetor2D  v;

            v = orientacao * new Vetor2D(-rx, -ry) + centro;
            temp[0] = new PointF(v.X, v.Y);

            v = orientacao * new Vetor2D(+rx, -ry) + centro;
            temp[1] = new PointF(v.X, v.Y);

            v = orientacao * new Vetor2D(+rx, +ry) + centro;
            temp[2] = new PointF(v.X, v.Y);

            v = orientacao * new Vetor2D(-rx, +ry) + centro;
            temp[3] = new PointF(v.X, v.Y);

            return temp;

        }


        public static void DivideOBB(OrientedBoundingBox2D obb, Vetor2D[]
pontos, out Vetor2D[] pontosA, out Vetor2D[] pontosB)
        {
            List<int> idx1 = new List<int>();
            List<int> idx2 = new List<int>();

            bool eixoX = false;
            if (obb.Rx > obb.Ry) eixoX = true;

            Matriz2x2 iM = obb.Orientacao.Inversa();
```

```
            for (int i = 0; i < pontos.Length; i++)
            {
                Vetor2D Pt = pontos[i] - obb.Centro;
                Vetor2D iPt = iM * Pt;

                if (eixoX)
                {
                    if (iPt.X < 0.0f) idx1.Add(i);
                    else idx2.Add(i);
                }
                else
                {
                    if (iPt.Y < 0.0f) idx1.Add(i);
                    else idx2.Add(i);
                }
            }

            pontosA = new Vetor2D[idx1.Count];
            for (int i = 0; i < idx1.Count; i++)
                pontosA[i] = pontos[idx1[i]];

            pontosB = new Vetor2D[idx2.Count];
            for (int i = 0; i < idx2.Count; i++)
                pontosB[i] = pontos[idx2[i]];

        }


    }

}
```

Arquivo BoundingBox2D.cs Fim

Arquivo DivideObjetos.cs

```
using System;
using System.Collections.Generic;
using System.Text;
using System.Drawing;
using System.Drawing.Imaging;
using System.Drawing.Drawing2D;
using System.Threading;


using RetiraFundo;

namespace Principal
{

    //reconhece os objetos na imagem e retorna as coordenadas das bounding
boxes que os descrevem
    class DivideObjetos
    {
        //public float x;

        public static float posicaoXObjetivo;
        public static float posicaoYObjetivo;
        public static float ObjetivoRX;
        public static float ObjetivoRY;

        public static float posicaoXFrenteCarrinho;
        public static float posicaoYFrenteCarrinho;
        public static float posicaoXCarrinho;
        public static float posicaoYCarrinho;
        public static float CarrinhoRX;
        public static float CarrinhoRY;

        public static float posicaoXObstaculo1;
        public static float posicaoYObstaculo1;
        public static float Obstaculo1RX;
        public static float Obstaculo1RY;

        public static float posicaoXObstaculo2;
        public static float posicaoYObstaculo2;
        public static float Obstaculo2RX;
        public static float Obstaculo2RY;



        public static unsafe OrientedBoundingBox2D[] Executa(Bitmap imagem,
DadoCorObjeto[] objetos)
        {

            List<Vetor2D>[] pontosobjetos = new
List<Vetor2D>[objetos.Length];


            for (int i = 0; i < objetos.Length; i++)
            {

                pontosobjetos[i] = new List<Vetor2D>();

            }
```

```
            BitmapData bd = imagem.LockBits(new Rectangle(0, 0,
imagem.Width, imagem.Height), ImageLockMode.ReadOnly,
PixelFormat.Format24bppRgb);
            byte* p = (byte*)bd.Scan0;
            int stride = imagem.Width * 3;

            //varre o bitmap
            for (int i = 0; i < (imagem.Width * imagem.Height * 3); i += 3)
            {

                byte bbO = (*p++);
                byte ggO = (*p++);
                byte rrO = (*p++);

                if (bbO != 0xff && ggO != 0xff && rrO != 0xff)
                {

                    for (int j = 0; j < objetos.Length; j++)
                    {

                        DadoCorObjeto dc = objetos[j];
                        if (dc.CorDentroFaixa(CorHSB.DeRGB(rrO, ggO, bbO)))
                        {

                            int cbsx = (int)(j % stride) / 3;
                            int cbsy = (int)(j / stride);

                            pontosobjetos[j].Add(new Vetor2D(cbsx, cbsy));
                            break;


                        }

                    }

                }

            }

            imagem.UnlockBits(bd);

            OrientedBoundingBox2D[] resultado = new
OrientedBoundingBox2D[objetos.Length];

            for (int i = 0; i < objetos.Length; i++)
            {

                if (pontosobjetos[i].Count > 0)
                {

                    //calcula as bounding boxes
                    Vetor2D[] pontos = pontosobjetos[i].ToArray();

                    OrientedBoundingBox2D obb = new
OrientedBoundingBox2D();
                    Vetor2D media2;
                    Vetor2D centrobb;
                    obb.Orientacao =
MatrizCovariancia.CalculaOrientacao(pontos, out media2, out centrobb);
                    obb.Centro = centrobb;
```

```
                    obb.Processa(pontos);

                    resultado[i] = obb;

                }

            }

            return resultado;

        }

        public static unsafe Bitmap Executa(Bitmap imagem, DadoCorObjeto[]
objetos, float[] tolerancias2, out OrientedBoundingBox2D[] boundingboxes)
        {

            List<Vetor2D>[] pontosobjetos = new
List<Vetor2D>[objetos.Length];
            Coordenadas pontosCoordenadas = new Coordenadas();

            for (int i = 0; i < objetos.Length; i++)
            {

                pontosobjetos[i] = new List<Vetor2D>();

            }

            BitmapData bd = imagem.LockBits(new Rectangle(0, 0,
imagem.Width, imagem.Height), ImageLockMode.ReadOnly,
PixelFormat.Format24bppRgb);
            byte* p = (byte*)bd.Scan0;
            int stride = imagem.Width * 3;

            //varre o bitmap
            for (int i = 0; i < (imagem.Width * imagem.Height * 3); i += 3)
            {

                byte bbO = (*p++);
                byte ggO = (*p++);
                byte rrO = (*p++);

                if (bbO != 0xff && ggO != 0xff && rrO != 0xff)
                {

                    for (int j = 0; j < objetos.Length; j++)
                    {

                        DadoCorObjeto dc = objetos[j];
                        if (dc.CorDentroFaixa(CorHSB.DeRGB(rrO, ggO, bbO)))
                        {

                            int cbsx = (int)(i % stride) / 3;
                            int cbsy = (int)(i / stride);

                            pontosobjetos[j].Add(new Vetor2D(cbsx, cbsy));
                            break;


                        }

                    }
```

```
                }

            }

            //remove os pontos muito longe da média ...
            for (int i = 0; i < objetos.Length; i++)
            {

                //acha a média ...
                Vetor2D media = new Vetor2D(0.0f, 0.0f);
                for (int j = 0; j < pontosobjetos[i].Count; j++)
                {

                    media += pontosobjetos[i][j];

                }

                media.X /= pontosobjetos[i].Count;
                media.Y /= pontosobjetos[i].Count;

                List<Vetor2D> pontospararemover = new List<Vetor2D>();
                foreach (Vetor2D v2d in pontosobjetos[i])
                {

                    float dx = v2d.X - media.X;
                    float dy = v2d.Y - media.Y;

                    float distancia2 = dx * dx + dy * dy;

                    if (distancia2 > tolerancias2[i])
                    {

                        pontospararemover.Add(v2d);

                    }

                }

                foreach (Vetor2D v2d in pontospararemover)
                {
                    if (i != 1)
                    pontosobjetos[i].Remove(v2d);

                }

            }

            imagem.UnlockBits(bd);

            OrientedBoundingBox2D[] resultado = new
OrientedBoundingBox2D[objetos.Length];

            for (int i = 0; i < objetos.Length; i++)
            {

                if (pontosobjetos[i].Count > 0)
                {

                    //calcula as bounding boxes
                    Vetor2D[] pontos = pontosobjetos[i].ToArray();
```

```
                    OrientedBoundingBox2D obb = new
OrientedBoundingBox2D();
                    Vetor2D media2;
                    Vetor2D centrobb;

                    obb.Orientacao =
MatrizCovariancia.CalculaOrientacao(pontos, out media2, out centrobb);
                    obb.Centro = centrobb;
                    obb.Processa(pontos);

                    resultado[i] = obb;

                    if (i == 1)
                    {
                        Vetor2D[] pontos1, pontos2;
                        OrientedBoundingBox2D.DivideOBB(obb, pontos, out
pontos1, out pontos2);

                        OrientedBoundingBox2D obb1 = new
OrientedBoundingBox2D();
                        OrientedBoundingBox2D obb2 = new
OrientedBoundingBox2D();
                        Vetor2D media2_1, media2_2;
                        Vetor2D centrobb1, centrobb2;

                        obb1.Orientacao =
MatrizCovariancia.CalculaOrientacao(pontos1, out media2_1, out centrobb1);
                        obb1.Centro = centrobb1;
                        obb1.Processa(pontos1);

                        obb2.Orientacao =
MatrizCovariancia.CalculaOrientacao(pontos2, out media2_2, out centrobb2);
                        obb2.Centro = centrobb2;
                        obb2.Processa(pontos2);

                        obb.filhos[0] = obb1;
                        obb.filhos[1] = obb2;

                        //continue;
                    }

                }

            }

            boundingboxes = resultado;

            Bitmap b = (Bitmap)imagem.Clone();
            Graphics g = Graphics.FromImage(b);
            g.CompositingMode = CompositingMode.SourceOver;
            g.CompositingQuality = CompositingQuality.HighQuality;
            g.InterpolationMode = InterpolationMode.HighQualityBicubic;
            g.SmoothingMode = SmoothingMode.AntiAlias;

            for (int i = 0; i < objetos.Length; i++)
            {

                OrientedBoundingBox2D obb = boundingboxes[i];
```

```
                if (boundingboxes[0] != null)
                    pontosCoordenadas.Carrinho(boundingboxes[0].Centro.X,
boundingboxes[0].Centro.Y, boundingboxes[0].Rx, boundingboxes[0].Ry);
                if (boundingboxes[1] != null)
                {

pontosCoordenadas.bolinhaVerde1(boundingboxes[1].filhos[0].Centro.X,
boundingboxes[1].filhos[0].Centro.Y, boundingboxes[1].filhos[0].Rx,
boundingboxes[1].filhos[0].Ry);

pontosCoordenadas.bolinhaVerde2(boundingboxes[1].filhos[1].Centro.X,
boundingboxes[1].filhos[1].Centro.Y, boundingboxes[1].filhos[1].Rx,
boundingboxes[1].filhos[1].Ry);
                    //else pontosCoordenadas.bolinhaVerde(0, 0, 0, 0);
                }

                if (boundingboxes[2] != null)
                    pontosCoordenadas.vermelho(boundingboxes[2].Centro.X,
boundingboxes[2].Centro.Y, boundingboxes[2].Rx, boundingboxes[2].Ry);
           //else pontosCoordenadas.vermelho(0, 0, 0, 0);

                if (boundingboxes[3] != null)

pontosCoordenadas.bolinhaAmarela(boundingboxes[3].Centro.X,
boundingboxes[3].Centro.Y, boundingboxes[3].Rx, boundingboxes[3].Ry);


               // setando os parametros para enviar para o form1 para
enviar para a 2a. Thread
                posicaoXObjetivo = pontosCoordenadas.bolinhaAmarelaCentroX;
                posicaoYObjetivo = pontosCoordenadas.bolinhaAmarelaCentroY;
                ObjetivoRX = pontosCoordenadas.bolinhaAmarelaRX;
                ObjetivoRY = pontosCoordenadas.bolinhaAmarelaRY;

                posicaoXCarrinho = pontosCoordenadas.carrinhoCentroX;
                posicaoYCarrinho = pontosCoordenadas.carrinhoCentroY;
                posicaoXFrenteCarrinho = pontosCoordenadas.vermelhoCentroX;
                posicaoYFrenteCarrinho = pontosCoordenadas.vermelhoCentroY;
                CarrinhoRX = pontosCoordenadas.carrinhoRX;
                CarrinhoRY = pontosCoordenadas.carrinhoRY;


                posicaoXObstaculo1 =
pontosCoordenadas.bolinhaVerde1CentroX;
                posicaoYObstaculo1 =
pontosCoordenadas.bolinhaVerde1CentroY;
                Obstaculo1RX = pontosCoordenadas.bolinhaVerde1RX;
                Obstaculo1RY = pontosCoordenadas.bolinhaVerde1RY;
                posicaoXObstaculo2 =
pontosCoordenadas.bolinhaVerde2CentroX;
                posicaoYObstaculo2 =
pontosCoordenadas.bolinhaVerde2CentroY;
                Obstaculo2RX = pontosCoordenadas.bolinhaVerde2RX;
                Obstaculo2RY = pontosCoordenadas.bolinhaVerde2RY;
```

```
                if (obb != null)
                {

                    PointF[] pontosobb = obb.RetornaPontosMundo();

                    //desenha a oobb
                    //Cores dos respecivos carrinho, Cor verde (total), cor
vermelha, Bolinha tenis(amarela), bolinha verde
                    Brush[] brushes = new Brush[] { Brushes.Black,
Brushes.Transparent, Brushes.Orange, Brushes.Gold, Brushes.Pink };
                    g.FillPolygon(brushes[i], pontosobb);

                    if (obb.filhos[0] != null)
                    {
                        pontosobb = obb.filhos[0].RetornaPontosMundo();
                        g.FillPolygon(brushes[4], pontosobb);
                    }

                    if (obb.filhos[1] != null)
                    {
                        pontosobb = obb.filhos[1].RetornaPontosMundo();
                        g.FillPolygon(brushes[4], pontosobb);
                    }

                }

            }

            g.Dispose();

            return b;

        }

    }

}
```

Arquivo DivideObjetos.cs Fim

Arquivo ControleVeiculo.cs

```
using System;
using System.Collections.Generic;
using System.Collections;
using System.Text;
using System.Drawing;
using System.Drawing.Drawing2D;

namespace Principal
{
    class ControleVeiculo
    {

        public Carro carrinho;
        public Obstaculo bolinhasVerdes;
        //chamando os obstaculos de "pedras"
        public List<Obstaculo> Pedras;
        int numeroObstaculos = 2;
        public Objetivo bolinhaAmarela;

        public bool colisao = false;

        public int Height = 0;
        public int Width = 0;

        private float[] anguloDeEsterco;

        public double[] valorAngulo;
        public double[] valorDistancia;
        public double[] valorDxObs;
        public double[] valorDyObs;

        public double valorDxAlv;
        public double valorDyAlv;
        public double AlvoAngulo;
        public double AlvoDistancia;

        float posicaoXObjetivo;
        float posicaoYObjetivo;
        float ObjetivoRX;
        float ObjetivoRY;

        float posicaoXFrenteCarrinho;
        float posicaoYFrenteCarrinho;
        float posicaoXCarrinho;
        float posicaoYCarrinho;
        float CarrinhoRX;
        float CarrinhoRY;

        float posicaoXObstaculo1;
        float posicaoYObstaculo1;
        float Obstaculo1RX;
        float Obstaculo1RY;

        float posicaoXObstaculo2;
        float posicaoYObstaculo2;
        float Obstaculo2RX;
        float Obstaculo2RY;
```

```
        float teste;

        float AlinhamentoCarro;
        float DirecaoDesejada;

        //variaveis do controle



        //define o numero de obstaculos nesse caso 2

        int direcao;


        public ControleVeiculo(
                                    float posicaoXObjetivo,
                                    float posicaoYObjetivo,
                                    float ObjetivoRX,
                                    float ObjetivoRY,

                                    float posicaoXFrenteCarrinho,
                                    float posicaoYFrenteCarrinho,
                                    float posicaoXCarrinho,
                                    float posicaoYCarrinho,
                                    float CarrinhoRX,
                                    float CarrinhoRY,

                                    float posicaoXObstaculo1,
                                    float posicaoYObstaculo1,
                                    float Obstaculo1RX,
                                    float Obstaculo1RY,

                                    float posicaoXObstaculo2,
                                    float posicaoYObstaculo2,
                                    float Obstaculo2RX,
                                    float Obstaculo2RY
                                )
        {
           this.posicaoXObjetivo = posicaoXObjetivo;
           this.posicaoYObjetivo = posicaoYObjetivo;
           this.ObjetivoRX = ObjetivoRX;
           this.ObjetivoRY = ObjetivoRY;

           this.posicaoXFrenteCarrinho = posicaoXFrenteCarrinho;
           this.posicaoYFrenteCarrinho = posicaoYFrenteCarrinho;
           this.posicaoXCarrinho = posicaoXCarrinho;
           this.posicaoYCarrinho = posicaoYCarrinho;
           this.CarrinhoRX = CarrinhoRX;
           this.CarrinhoRY = CarrinhoRY;

           this.posicaoXObstaculo1 = posicaoXObstaculo1;
           this.posicaoYObstaculo1 = posicaoYObstaculo1;
           this.Obstaculo1RX = Obstaculo1RX;
           this.Obstaculo1RY = Obstaculo1RY;

           this.posicaoXObstaculo2 = posicaoXObstaculo2;
           this.posicaoYObstaculo2 = posicaoYObstaculo2;
           this.Obstaculo2RX = Obstaculo2RX;
           this.Obstaculo2RY = Obstaculo2RY;
```

```
            valorDistancia = new double[numeroObstaculos];
            valorDxObs = new double[numeroObstaculos];
            valorDyObs = new double[numeroObstaculos];
            valorAngulo = new double[numeroObstaculos];
            Pedras = new List<Obstaculo>();
            anguloDeEsterco = new float[5];

            start();
            AtualizarFuncaoExponencial(0, 1.0f, 4000, 3);
        }

        private void carro()
        {
            PointF centroCarro = new PointF(posicaoXCarrinho,
posicaoYCarrinho);
            PointF velocidade = new PointF(0.0f, 0.0f);
            //estou setando o raio do carro manualmente
            //carrinho = new Carro(centroCarro, velocidade, CarrinhoRX,
0.0f, 0.5f);
            carrinho = new Carro(centroCarro, velocidade, 30, 0.0f, 0.5f);
        }

        private void objetivo()
        {
            PointF centroObjetivo = new PointF(posicaoXObjetivo,
posicaoYObjetivo);
            PointF velocidade = new PointF(0.0f, 0.0f);
            bolinhaAmarela = new Objetivo(centroObjetivo, velocidade,
ObjetivoRX, 0.0f, 0.0f);
        }

        private void obstaculo()
        {
            Pedras.Clear();
            PointF velocidade = new PointF(0.0f, 0.0f);
            PointF[] centroObstaculos;

            centroObstaculos = new PointF[numeroObstaculos];
            centroObstaculos[0] = new PointF(posicaoXObstaculo1,
posicaoYObstaculo1);
            centroObstaculos[1] = new PointF(posicaoXObstaculo2,
posicaoYObstaculo2);

            for (int i = 0; i < numeroObstaculos; i++)
            {
                //definindo
                //vou setar o raio padrao e igual a 10 nese caso 5
                Pedras.Add(new Obstaculo(centroObstaculos[i], velocidade,
5, 0.0f, 0.0f));
            }

        }
        //

        public void retorna()
        {

            AlvoDistancia = carrinho.Distancia(posicaoXFrenteCarrinho,
posicaoYFrenteCarrinho, bolinhaAmarela.centro, bolinhaAmarela.raio);
            valorDxAlv = -(bolinhaAmarela.x - posicaoXFrenteCarrinho);
```

```
            valorDyAlv = -(bolinhaAmarela.y - posicaoYFrenteCarrinho);
            AlvoAngulo = 180 * Math.Atan2(valorDyAlv, valorDxAlv) / Math.PI
- carrinho.alfa;

            for (int i = 0; i < numeroObstaculos; i++)
            {
                //valorDistancia[i] = carrinho.Distancia(Pedras[i].centro,
Pedras[i].raio);
                //aqui cou setar o raio das bolinhas manualmente nesse caso
5 //10 é o padrao
                valorDistancia[i] =
carrinho.Distancia(posicaoXFrenteCarrinho, posicaoYFrenteCarrinho,
Pedras[i].centro, 5);
                valorDxObs[i] = -(Pedras[i].x - posicaoXFrenteCarrinho);
                valorDyObs[i] = -(Pedras[i].y - posicaoYFrenteCarrinho);
                valorAngulo[i] = 180 * Math.Atan2(valorDyObs[i],
valorDxObs[i]) / Math.PI - carrinho.alfa;
            }

        }


        public void AtualizarFuncaoExponencial(float d, float uf, float a,
float n)
        {
            // original cos -> sen
            retorna();
            float u, ux, uy;

            u = uf;
            ux = u * (float)Math.Cos(AlvoAngulo * Math.PI / 180);
            uy = u * (float)Math.Sin(AlvoAngulo * Math.PI / 180);

            for (int i = 0; i < numeroObstaculos; i++)
            {
                u = (float)(a / Math.Pow(valorDistancia[i], n));
                ux -= u * (float)Math.Cos(valorAngulo[i] * Math.PI / 180);
                uy -= u * (float)Math.Sin(valorAngulo[i] * Math.PI / 180);
                Pedras[i].Atualizar(d);
            }
            carrinho.SetV(new PointF(ux, uy));
            //para atualizar o angulo com o eixo do carrinho
            AtualizarComAFrente(ux, uy);

            teste = uy;


        }

        public void AtualizarComAFrente(float ux, float uy)
        {
            PointF vetorCarroInicio = new PointF(posicaoXCarrinho,
posicaoYCarrinho);
            PointF vetorCarroFinal = new PointF(posicaoXFrenteCarrinho,
posicaoYFrenteCarrinho);

            PointF vetorDirecaoInicio = new PointF(posicaoXCarrinho,
posicaoYCarrinho);
            PointF vetorDirecaoFinal = new PointF(ux, uy);
```

```
        float v1x = (vetorCarroInicio.X - vetorCarroFinal.X);
        float v1y = (vetorCarroInicio.Y - vetorCarroFinal.Y);

        float AlinhamentoCarro = (float)Math.Atan2(v1y, v1x);
        float DirecaoDesejada = (float)Math.Atan2(uy, ux);

        this.AlinhamentoCarro = AlinhamentoCarro;
        this.DirecaoDesejada = DirecaoDesejada;

        //Console.WriteLine(AlinhamentoCarro.ToString());

        // frente = 1
        // frente esquerda = 9
        //frente direita = 3
        //ré = 4
        //ré + esquerda = 12
        //ré + direita = 6

        //isso é apenas um teste para obter um angulo medio


            float AngEsterce = DirecaoDesejada - AlinhamentoCarro;


    }



    public void AtualizarMet(float d)
    {
        //Nv1.Atualizar(d);

        for (int i = 0; i < numeroObstaculos; i++)
        {
            Pedras[i].Atualizar(d);
        }

    }



    public void start()
    {
        obstaculo();
        carro();
        objetivo();
    }
    /*
    public void Iteracao(ref Graphics G, float d)
    {
        AtualizarFuncaoExponencial(d, 2.0f, 1000, 3);
        AtualizarMet(d);

        //  colidir();

    }
     * */

    public float carroAlinhamento()
    {
```

```
            return AlinhamentoCarro;
        }


        public float direcaoDejesadaCarro()
        {
            return DirecaoDesejada;
        }




        public int direcaoVeiculo()
        {
            // frente = 1
            // frente esquerda = 9
            //frente direita = 3
            //ré = 4
            //ré + esquerda = 12
            //ré + direita = 6

            /*a saber (nao usa para nada)
             * esquerda = 8
             * direita = 2
             * */
            if ((posicaoXFrenteCarrinho - posicaoXObjetivo) > 8 ||
posicaoXFrenteCarrinho > 20)
            {
                return direcao;
            }
            else
                direcao = 0;

                return direcao;

        }

    }// fim da classe controle veiculo

    public class Carro : ObjMoveis
    {
        public Carro(PointF mycentro, PointF myvelocidade, float myraio,
float mydirecao, float mydeltat)
        {
            raio = myraio;
            direcao = mydirecao;
            deltat = mydeltat;
            centro = mycentro;
            velocidade = myvelocidade;

        }


    }

    public class Objetivo : ObjMoveis
    {
        public Objetivo(PointF mycentro, PointF myvelocidade, float myraio,
float mydirecao, float mydeltat)
        {
            raio = myraio;
```

```
            direcao = mydirecao;
            deltat = mydeltat;
            centro = mycentro;
            velocidade = myvelocidade;


        }
    }

    public class Obstaculo : ObjMoveis
    {

        public Obstaculo(PointF mycentro, PointF myvelocidade, float
myraio, float mydirecao, float mydeltat)
        {


            if (imagem == null)
            {

                raio = myraio;
                direcao = mydirecao;
                deltat = mydeltat;
                centro = mycentro;
                velocidade = myvelocidade;

            }

        }

    }



    /// <summary>
    /// Summary description for ObjMoveis.
    /// </summary>
    public class ObjMoveis
    {
        public PointF velocidade;
        public PointF centro;
        public float raio;
        public float direcao;
        public float deltat;
        public float Rot;
        public ArrayList novPedra = new ArrayList();


        public float virar = 0.9f;

        public Bitmap imagem;

        public PointF CentroReal
        {
            get
            {
                return new PointF(px, py);
            }
        }

        public float px
```

```
        {
            get
            {
                return (x + raio / 2f);
            }
        }

        public float py
        {
            get
            {
                return (y + raio / 2f);
            }
        }

        public float x
        {
            get { return centro.X; }
        }

        public float y
        {
            get { return centro.Y; }
        }

        public float r
        {
            get { return raio; }
        }

        public float alfa
        {
            get { return (float)(180f * direcao / Math.PI); }
        }



        public ObjMoveis() { }

        public ObjMoveis(PointF mycentro, PointF myvelocidade, float
myraio, float mydirecao, float mydeltat)
        {
            raio = myraio;
            direcao = mydirecao;
            deltat = mydeltat;
            centro = mycentro;
            velocidade = myvelocidade;

        }

        public void Acelerar(float acc)
        {
            velocidade.X *= acc;
            velocidade.Y *= acc;
            if (velocidade.X > 3) velocidade.X = 3;
            if (velocidade.Y > 3) velocidade.Y = 3;
            if (velocidade.X < -3) velocidade.X = -3;
            if (velocidade.Y < -3) velocidade.Y = -3;

            centro.X += velocidade.X / deltat;
            centro.Y += velocidade.Y / deltat;
```

```
        }

        public void ColisaoPanel(float Hei, float Wid)
        {

            if ((centro.X <= 0) || (centro.X >= (Wid - raio)))
            {
                //Girar(0.1f);

            }
            if ((centro.Y <= 0) || (centro.Y >= (Hei - raio)))
            {
                //Girar(0.1f);
            }
            if ((centro.X < 0) & (centro.Y < 0))
            {
                //Girar(0.1f);}
            }

        }
        /*
        public bool TestaColisao(PointF CentroObjeto2, float RaioObjeto2)
        {
            if (Distancia(CentroObjeto2, RaioObjeto2) < 0)
                return true;
            return false;
        }
         * */
        public float Distancia(float carrinhoFrenteX, float
carrinhoFrenteY, PointF CentroObjeto2, float RaioObjeto2)
        {
            return (float)((Math.Sqrt(Math.Pow(carrinhoFrenteX -
CentroObjeto2.X, 2) + Math.Pow(carrinhoFrenteY - CentroObjeto2.Y, 2)))) -
(raio / 2f + RaioObjeto2 / 2f);
        }


        public void DeltaV(PointF delta)
        {
            velocidade = new PointF(velocidade.X + delta.X, velocidade.Y +
delta.Y);
            if (velocidade.X > 2) velocidade.X = 2;
            if (velocidade.Y > 2) velocidade.Y = 2;
            if (velocidade.X < -2) velocidade.X = -2;
            if (velocidade.Y < -2) velocidade.Y = -2;
        }

        public void SetV(PointF delta)
        {
            velocidade = new PointF(delta.X, delta.Y);

        }


        public void Girar(float ang)
        {

            direcao += ang;

        }
```

```
        public void Atualizar(float deltat)
        {
            float nv = velocidade.X;
            PointF newvel = new PointF((float)(nv * Math.Cos(direcao)),
(float)(nv * Math.Sin(direcao)));
            centro = new PointF(centro.X + newvel.X * deltat, centro.Y +
newvel.Y * deltat);
        }

        public void Atualizar2(float deltat)
        {
            centro.X = centro.X + deltat * velocidade.X;
            centro.Y = centro.Y + deltat * velocidade.Y;
        }


    }
}
```

Arquivo ControleVeiculo.cs fim

Arquivo ThreadControle.cs

```
using System;
using System.Collections.Generic;
using System.Text;
using System.Threading;
using System.Runtime.InteropServices;

namespace Principal
{
    class ThreadControle
    {

        [DllImport("inpout32.dll", EntryPoint = "Out32")]
        public static extern void Output(int adress, int value);
        //esta classe funciona como a thread secundaria para o controle
        //do carrinho!
        private int valorPorta;
        private volatile bool _pare;
        //Coordenadas coordenadas = new Coordenadas();
        //ControleVeiculo controleVeiculo = new ControleVeiculo();

        public ThreadControle(int valorPorta)
        {
            this.valorPorta = valorPorta;
        }
        public void iniciaControle()
        {

            if (valorPorta != 0)
            {
                Output(888, valorPorta);
                Thread.Sleep(100);// ajusto a frequencia do carro default
150

                if (valorPorta == 9)
                    Output(888, 8);
                if (valorPorta ==3)
                    Output(888, 2);
                if(valorPorta ==1)
                    Output(888, 0);

                //Output(888, 0);
                Thread.CurrentThread.Abort();
            }
            else
            {
                Output(888, 0);
                Thread.CurrentThread.Abort();
            }



        }
    }
}
```

Arquivo ThreadControle.cs Fim