UniCEUB – Centro Universitário de Brasília

FATECS - Faculdade de Tecnologia e Ciências Sociais Aplicadas

Curso de Engenharia da Computação

Uilian Gonçalves de Sales

CRIAÇÃO DE UM SOFTWARE, BASEADO EM UM SISTEMA DE VISÃO, PARA RECONHECIMENTO DA POSIÇÃO DO VEÍCULO NA RODOVIA.

Brasília

2008

Uilian Gonçalves de Sales

CRIAÇÃO DE UM SOFTWARE, BASEADO EM UM SISTEMA DE VISÃO, PARA RECONHECIMENTO DA POSIÇÃO DO VEÍCULO NA RODOVIA.

Trabalho apresentado ao Centro Universitário de Brasília (UniCEUB/FATECS) como pré-requisito para a obtenção de Certificado de Conclusão de Curso de Graduação, na área de Engenharia da Computação.

Orientador: Thiago Toríbio

Brasília

2008

Uilian Gonçalves de Sales

CRIAÇÃO DE UM SOFTWARE, BASEADO EM UM SISTEMA DE VISÃO, PARA RECONHECIMENTO DA POSIÇÃO DO VEÍCULO NA RODOVIA.

Trabalho apresentado ao Centro Universitário de Brasília (UniCEUB/FATECS) como pré-requisito para a obtenção de Certificado de Conclusão de Curso de Graduação, na área de Engenharia da Computação.

Orientador: Thiago Toríbio

Brasília, _____ de _____ de _____.

Banca examinadora

_____

Professor: Nome Completo

_____

Professor: Nome Completo

RESUMO

Todos os anos muitas pessoas morrem em acidentes de trânsito, e os principais fatores incluem abuso na direção e desatenção às mudanças no ambiente. As circunstâncias estimulam o desenvolvimento de sistemas de apoio à condução para melhorar a segurança durante a direção dos veículos.

Este trabalho propõe-se a desenvolver um *software*, baseado em visão computacional, que tem como objetivo alertar o condutor caso o veículo inicie uma saída de pista à esquerda. A saída de pista é detectada quando o veículo inicia uma trajetória sobre a faixa contínua localizada no centro da estrada.

Para identificar a posição do veículo na pista o *software* acessa os parâmetros iniciais que definem uma relação entre a posição do veículo e o local onde a faixa é visualizada na imagem do vídeo. Para localizar a posição da faixa, na imagem, são analisados dois métodos, o reconhecimento da faixa por redes neurais e a localização da faixa por detecção de bordas usando o operador de Sobel e a transformada de Hough.

Palavras chaves:

Visão computacional, processamento de imagens.

ABSTRACT

Every year many people die in traffic accidents, and the main factors include abuse and neglect toward the changes in the environment. The circumstances stimulate the development of support systems to improve driving safety during the direction of vehicles.

This work is proposed to develop a software, based on computational vision, which aims to alert the driver if the vehicle starts an output of runway left. The output of the runway is detected when the vehicle starts a continuous path on the track located in the center of the road.

To identify the position of the vehicle on the track the software accesses the initial parameters that define a relationship between the position of the vehicle and where the track is visible in the video image. To locate the position of the track, in the image, two methods are considered, the recognition of the track by neural networks and location of the track by detecting edges using Sobel operator, and Hough transform.

Keywords

Computational vision, image processing.

SUMÁRIO

LISTA DE FIGURAS

LISTA DE TABELAS

1. INTRODUÇÃO

1.1. Motivação

A Polícia Rodoviária Federal estima que 29,8% de todos os acidentes ocorridos no ano de 2005 em rodovias federais tenham sido causados pela falta de atenção dos motoristas. Técnicos da instituição acreditam ainda que 9,57% das colisões são provocadas por excesso de velocidade e que 8,79% dos acidentes ocorrem porque os motoristas não mantêm uma distância segura do carro à frente. Apenas em 2,96% dos casos, a precariedade das estradas teria sido responsável pelas colisões (SAN 06).

Segundo estatísticas da Brigada Militar do Rio Grande do Sul grande parte dos acidentes que ocorreram nas rodovias do Rio Grande do Sul, entre janeiro e agosto de 2008, são do tipo "Colisão"[1]. A Tabela 1-1 apresenta a quantidade de acidentes, do ano de 2008, por Tipo de acidente e Mês (BRI 08).

Tabela 1-1: Quantidade de acidentes por Tipo x Mês/2008

TIPO	JAN	FEV	MAR	ABR	MAI	JUN	JUL	AGO	TOTAL
ABALROAMENTO	285	245	272	288	290	300	277	267	2224
ATROPELAMENTO DE ANIMAL	13	14	23	25	28	29	23	27	182
ATROPELAMENTO DE PESSOA	16	24	28	38	45	37	36	32	256
CAPOTAMENTO	74	50	50	65	71	47	49	44	450
CHOQUE	219	236	260	218	228	237	186	195	1779
COLISÃO	156	162	157	162	192	150	124	115	1218
NÃO ESPECIFICADO	41	28	32	18	35	32	24	28	238
TOMBAMENTO	50	55	60	61	53	47	46	52	424

Veículos com sistemas de apoio embarcados permitem uma analise do ambiente e do estado do veículo o que possibilita ao sistema realizar ajustes no veículo para se adaptar ao ambiente ou alertar o motorista caso exista uma situação de perigo.

[1] Segundo o DETRAN "Colisão" significa colisão frontal ou traseira, "Abalroamento" significa colisão lateral e "Choque" significa colisão contra um objeto fixo (BRI 08).

Câmeras digitais são ótimos sensores para monitorar o ambiente, visto que as imagens fornecem uma grande quantidade de informação. Neste projeto é desenvolvido um *software* que usa uma câmera de vídeo para filmar a rodovia e, com base nas imagens adquiridas, identifica a posição do veículo na pista.

Se o veículo iniciar uma trajetória sobre a faixa contínua, à esquerda, o *software* emite um alerta sonoro para o motorista, evitando uma saída de pista. O escopo deste projeto envolve apenas saídas de pista à esquerda. É sugerido, na seção 6.1, o desenvolvimento de um *software* que trate saídas de pista para a esquerda e para a direita.

O desenvolvimento do software não seria possível sem o aprendizado adquirido nas disciplinas de Técnicas de Programação e Engenharia de Programas. O conhecimento adquirido na disciplina de Arquitetura de Computadores foi essencial para um completo entendimento do hardware utilizado. As disciplinas de Cálculo e Física serviram de base para entender a teoria necessária para o desenvolvimento deste projeto.

1.2. Estado da arte

As empresas automotivas estão investindo, com grande interesse, em sistemas inteligentes que possam auxiliar o processo de condução dos veículos. Aplicações baseadas em radares e sensores a laser são dedicadas a realizar medidas precisas, mas não podem fornecer uma completa descrição do ambiente por onde o veículo é conduzido. Um sistema baseado em visão computacional, pode compensar a falta de informações, já que a visão pode fornecer resultados confiáveis sobre a descrição do ambiente.

A seção 1.3 apresenta algumas aplicações criadas para auxiliar a condução de veículos.

1.3. Exemplos de aplicações

1.3.1. Sensor ultra-sônico da Bosh

A *BOSH* criou um sistema que previne colisões frontais ao alerta o motorista sobre a proximidade de outros veículos à frente. O sistema é baseado em dois sensores ultra-sônico localizado em cada lado do pára-choque dianteiro. Estes dois sensores ultra-sônicos medem o espaço e podem registrar a velocidade media do veículo à frente (VRU 08).

1.3.2. Assistente de estacionamento da BMW

O assistente de estacionamento está sendo testado e já faz parte do protótipo da nova *BMW X5*. O sistema usa sensores para estimar se há um espaço de estacionamento grande o bastante para o carro. Quando o sensor encontra um espaço adequado ele executa as manobras necessárias para estacionar o veículo. O sistema dirige até que o veículo esteja estacionado (BMW 08).

1.3.3. Assistente de visibilidade

O carro *BMW Série 7* possui um sistema que alerta o condutor caso o sensor, baseado em visão computacional, detecte um ser humanos a até 100 metros à frente do veículo. Durante a noite o sistema usa a imagem térmica para detectar os pedestres que estão na estrada. O sistema usa redes neurais artificiais para reconhecer os pedestres evitando falsos alertas (BMW 08).

1.3.4. Aviso de colisão com suporte de freios da *VOLVO*

"*Para alertar os motoristas distraídos a Volvo criou um sistema no qual a área frontal do veículo é monitorada continuamente por um radar. Se o sistema percebe que o carro se aproxima de outro rapidamente e o motorista não reage, uma luz vermelha se acende no pára-brisa e um aviso sonoro é ativado. Mas, se, mesmo assim, o motorista demorar a reagir e o risco de colisão aumentar, o suporte de freio é ativado.*

As pastilhas ficam mais próximas dos discos e a pressão do freio é reforçada hidraulicamente, facilitando a frenagem e reduzindo o efeito de uma possível colisão. Caso o motorista pise no pedal de freio com muita força, as luzes da lanterna traseira começam a piscar para avisar quem vem atrás" (VRU 08).

A Volvo possui outros diversos equipamentos de segurança em seus modelos:

1.3.5. Sistema inteligente de informação (IDIS)

"*Sistema eletrônico que não deixa o motorista se distrair com informações desnecessárias durante situações de estresse. Ele checa continuamente algumas funções do carro, como movimento das rodas, posição do pedal do acelerador, controles dos freios etc. Em momentos de estresse, qualquer informação de menor importância, como chamada telefônica, é atrasada*" (VRU 08).

1.3.6. Faróis ativos bi-xenônio

"*Os faróis podem girar até 15 graus em cada sentido, ampliando o campo de visão em curvas. Para evitar o ofuscamento da visão do motorista que trafega em sentido contrário, o ângulo dos faróis é ajustado de acordo com o peso transportado ou situação (se o carro está acelerando ou freando)*" (VRU 08).

1.3.7. Assistência hidráulica do freio (HBA)

A pressão no freio é reforçada hidraulicamente e, em situação de emergência (caso o motorista não pise forte no pedal), o HBA faz com que o ABS atue de forma eficiente, encurtando a distância de frenagem (VRU 08).

1.3.8. City Safety

Dispositivo que será item de série no novo XC60. Estudos feitos pela Volvo revelam que 75% dos acidentes urbanos se referem a colisões em velocidade inferior a 30km/h. Muitas vezes, o motorista nem chega a frear.

Quando o motorista da frente freia repentinamente, o City Safety identifica o risco de colisão por meio de um radar ótico, integrado na parte superior do pára-brisa, e aciona o freio de forma automática, mesmo que o condutor não pise no pedal. Caso o carro esteja em velocidade de até 15km/h, o sistema evita o acidente. De 16km/h a 30km/h, o impacto da colisão é atenuado, preservando a integridade dos ocupantes e reduzindo os danos nos veículos (VRU 08).

1.3.9. Assistência de emergência em estradas

Usando radar e câmeras, o sistema monitora o caminho à frente, indicando a aproximação de carros e a disciplina do motorista em sua faixa. Caso o veículo apresente uma tendência de bater nos veículos da frente, ou sair da faixa, um aviso sonoro é ativado. Se o motorista não reagir, uma força é aplicada ao volante de direção automaticamente, ajudando o motorista a retornar para a faixa pretendida. Sistema semelhante equipa o novo VW Passat CC (VRU 08).

1.3.10. Pre-Crash da Toyota

O mecanismo usa uma câmera instalada na frente do banco do condutor que é capaz de detectar o movimento das pálpebras com precisão. Quando há qualquer perda de visão incomum (a detecção de pálpebras superiores e inferiores), o microcontrolador será capaz de interpretá-lo corretamente e alertar o motorista. Para reconhecer o estado das pálpebras o sistema usa uma rede neural artificial (TOY 08).

1.3.11. Leitor de sinalização de trânsito

A empresa Hella Aglaia criou um sistema que identifica sinais de trânsito. Para reconhecer os sinais de trânsito a empresa usa um sistema baseado em redes neurais artificiais, desta forma os condutores são sempre alertados sobre a sinalização da rodovia evitando multas por excesso de velocidade causadas pela desatenção do motorista (VRU 08).

1.4. Objetivo do trabalho

Este projeto se refere ao desenvolvimento de um *software* de apoio ao motorista baseado em visão computacional. O objetivo é desenvolver um *software* que trabalhe em tempo real para realizar a detecção de saída de pista.

A saída de pista é detectada quando a distância entre o veículo e a faixa contínua no centro da estrada é menor ou igual a zero. Este projeto analisa dois métodos para localizar a faixa na imagem do vídeo, o reconhecimento da faixa através de uma rede neural e a detecção dos contornos de objetos (faixa) com o operador de Sobel e a transformada de Hough.

Também é objetivo deste trabalho analisar as vantagens e desvantagens deste tipo de sistema, bem como a forma como foi desenvolvido. A implementação das funções usadas para realizar o treinamento da rede neural, a implementação do operador de Sobel e a transformada de Hough estão na biblioteca *OpenCV*.

1.5. Detalhes do projeto

O sistema é constituído de uma câmera de vídeo e uma unidade de processamento embarcado em um veículo. A câmera de vídeo é fixada em um ponto onde se pode filmar a rodovia, as imagens adquiridas pela câmera são encaminhadas para a unidade de processamento em tempo real.

O *software* é dividido em dois módulos, módulo de configuração e módulo de processamento. O módulo de configuração define os parâmetros iniciais necessários para que o módulo de processamento possa calcular e informar a distância que o veículo está da faixa central da estrada. O modulo de processamento recebe as imagens e realiza o processamento necessário para detectar a faixa na rodovia e exibir a distância que o veículo está desta faixa.

Parar localizar a posição do veículo na imagem o *software* usa conceitos de visão computacional. Neste projeto são analisados dois métodos para localizar a faixa na rodovia, reconhecimento através de redes neurais e o operador de Sobel com a transformada de Hough. O operador de Sobel gera uma imagem, a partir da

imagem original, com destaque para os contornos das bordas dos objetos encontrados e a transformada de Hough é usada para localizar formas geométricas (retas e círculos) nas imagens. A implementação de Hough retorna um vetor de pontos que é usado para localizar a posição da faixa.

1.6. Resumo dos capítulos

A organização do restante deste trabalho está organizado da seguinte forma:

O capítulo 2 apresenta a teoria sobre o processamento de imagens, teoria necessária para realizar o tratamento da imagem adquirida da câmera de vídeo. Descreve também os métodos utilizados para a detecção de linhas. O capítulo 3 apresenta a teoria necessária para usar uma rede neural artificial. A utilização de uma rede neural é uma das formas analisadas para localizar a posição da faixa na imagem. O capítulo 4 mostra como foi realizada a implementação e os problemas encontrados durante o desenvolvimento do *software*. O capítulo 5 exibe os resultados experimentais após a realização dos testes. O capítulo 6 apresenta a conclusão e os principais pontos deste trabalho. Este capítulo sugere também os projetos futuros que podem ser desenvolvidos a partir deste.

2. PROCESSAMENTO DE IMAGENS

Uma das tarefas do *software* baseado em visão computacional, objetivo deste trabalho, é realizar o tratamento adequado à imagem antes de encaminhá-la para processamento. Este capítulo aborda a teoria necessária para a realização desta tarefa.

2.1. Definição de imagem

Uma imagem pode ser definida como uma função bidimensional f(x, y) onde o valor da amplitude de *f* nas coordenadas espaciais *(x, y)* é um valor positivo cujo significado é determinado pela origem da imagem. Como uma imagem é gerada a partir de um processo físico, os seus valores são proporcionais à energia irradiada por uma fonte física. Como conseqüência, *f(x, y)* deve ser finito e não nulo, ou seja (GON 02):

$$0 < f(x,y) < \infty \tag{2-1}$$

Para uma imagem que possui diversas informações de bandas de diferentes freqüências, é necessária uma função *f(x, y)* para cada banda. É o caso de imagens coloridas que são formadas pela informação de cores primárias, como o vermelho, verde e azul (GON 02).

A função *f(x, y)* pode ser caracterizada por duas componentes: (1) a quantidade de luz que incide sobre a fonte observada (iluminação). (2) a quantidade de luz refletida pelos objetos na fonte (refletância). A iluminação e a refletância são denotadas por *i(x, y)* e *r(x, y)*, respectivamente. O produto das duas funções *i(x, y)* e *r(x, y)* resulta *f(x, y)*,

$$f(x,y) = i(x,y)r(x,y) \tag{2-2}$$

$$0 < i(x,y) < \infty \tag{2-3}$$

$$0 < r(x,y) < 1 \tag{2-4}$$

A equação indica que a refletância é limitada por *0* (total absorção) e *1* (total refletância). A natureza de *i(x, y)* é determinada pela fonte de luz, e *r(x, y)* é determinada pelas características dos objetos na imagem (GON 02).

2.1.1. Imagem *Gray-scale*

Em computação uma imagem *gray-scale* é uma imagem da qual os valores de cada *pixel* é uma variação entre a cor preta e a cor branca. As imagens *gray-scale* podem contar com diversos tons de cinza em sua formação.

2.2. Processamento digital de imagens

O processamento de uma imagem envolve ferramentas multidisciplinares e tem como objetivo fins diversos. O processamento de imagem sofre uma grande variação em função da área de trabalho e esta variação ocorre segundo:

A natureza das imagens

Depende da área onde as imagens foram adquiridas, exemplo: Área médica, industrial ou de laboratório.

A qualidade das imagens

O nível da qualidade de uma imagem varia de acordo com a aplicação. Características como o número e distribuição dos detalhes e a presença de áreas com variação gradual de brilho influem no grau subjetivo atribuído à qualidade da imagem.

Conhecimento do ambiente

Ter um conhecimento inicial do ambiente é importante para o processo de análise. O trabalho onde o conhecimento inicial é fraco exige um processo de análise mais rico para suprir a falta de conhecimento (FAC 02).

O processamento digital de imagens pode ser divido, quanto ao grau de abstração, em três níveis: Baixo, médio e alto (FAC 02).

No processamento de baixo nível os dados de entrada são *pixels* da imagem original e os dados de saída representam propriedades da imagem, na forma de

valores numéricos associados a cada pixel. No processamento de nível médio este conjunto de valores produz como resultado uma lista de características. O processamento de alto nível produz a partir destas características, uma interpretação do conteúdo da imagem (FAC 02). Uma estrutura funcional completa de um sistema de processamento e analise de imagens é apresentado na Figura 2-1:

Figura 2-1: Estrutura funcional de um sistema de visão computacional

No baixo nível temos as fases de Aquisição e Pré-processamento.

Aquisição: A imagem capturada é transformada em uma imagem digital sobre a forma de uma tabela de valores discretos inteiros chamados pixels. Neste trabalho é usada uma câmera de vídeo digital, portanto é necessária somente uma interface com o computador.

Pré-processamento: Esta etapa permite corrigir defeitos na imagem que podem surgir durante a etapa de aquisição. Os defeitos podem ter como causa características físicas do sistema, condições de iluminação, entre outros.

As duas etapas do nível médio são:

Segmentação: O objetivo desta etapa é dividir a imagem em partes constitutivas. Em uma imagem natural a segmentação é efetuada pela detecção de descontinuidades (contornos) e / ou similaridades (regiões) na imagem.

Representação: Nesta etapa é elaborada uma estrutura adequada, agrupando resultados das etapas anteriores ao armazenamento dos padrões que contem o conhecimento.

O alto nível contém a etapa de interpretação:

Interpretação: Permite a compreensão da imagem. Esta é a parte que apresenta para o usuário do sistema o resultado da operação com base no conhecimento adquirido durante as fases anteriores (CAN 06).

2.3. Segmentação de imagens

O processo de segmentação de uma imagem pode ser realizado objetivando extrair uma determinada região da imagem ou dividir a imagem num conjunto de regiões distintas.

Uma região de uma imagem pode ser descrita como um conjunto de pontos “ligados” onde, de qualquer ponto da imagem pode se chegar a outro ponto através de um caminho contido nessa região. As regiões que se deseja encontrar, geralmente, são regiões homogêneas, ou seja, apresentam alguma propriedade local aproximadamente constante. Comumente esta propriedade é a continuidade do nível de cinza (FAC 02).

2.3.1. Bordas

As bordas caracterizam limites, em uma imagem são as áreas com fortes contrastes de intensidade. A detecção das bordas em uma imagem reduz significativamente a quantidade de dados e facilita a remoção de informações inúteis, ao preservar as propriedades estruturais importantes em uma imagem (GIG 08).

Neste trabalho é usado o operador de *Sobel* para detectar as bordas dos objetos contidos em uma imagem.

2.4. Operador de Sobel

Existem técnicas que permitem detectar a descontinuidade dos níveis de cinza e fornecer contornos entre as diferentes regiões. Mas raramente o conjunto de *pixels* traduz contornos completos. Por causa do ruído, dos cortes no contorno devido a iluminância não uniforme e de outras perturbações, o contorno aparece sendo um conjunto descontínuo. A partir desses fatos, alguns procedimentos de enlace de bordas são usados junto aos algoritmos de detecção de contornos para reunir *pixels* de bordas em um conjunto significativo de contornos de regiões (FAC 02). Uma imagem digital pode ser olhada como o resultado de um processo de amostragem de um sinal contínuo (FIL 08), a Figura 2-2 mostra uma imagem com uma borda ideal e a Figura 2-3 mostra uma imagem com uma borda do tipo rampa onde o declive da rampa é proporcional ao grau de indefinição na borda (GON 02).

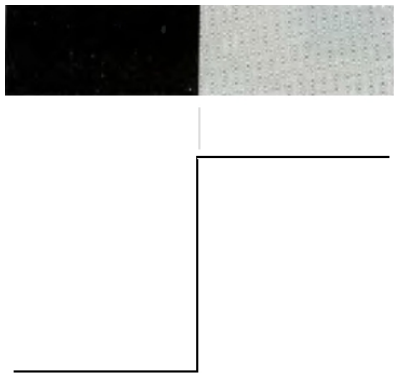


Figura 2-2: Modelo de uma borda ideal

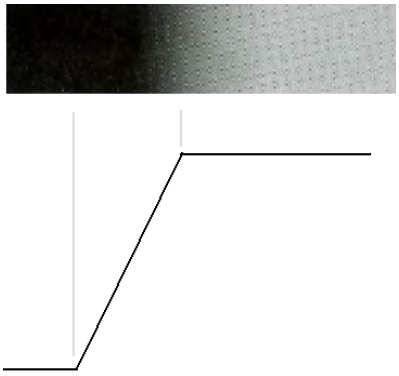


Figura 2-3: Modelo de uma borda do tipo rampa

2.4.1. Detecção de Bordas

As bordas caracterizam limites e seu tratamento é conseqüentemente um problema de importância fundamental no processamento de imagem. As bordas nas imagens são as áreas com contrastes fortes de intensidade, um salto na intensidade de um *pixel* ao seguinte. Os diversos métodos existentes para realizar a detecção de borda podem ser classificados em duas categorias, gradiente e laplaciano. Neste trabalho usaremos o operador de Sobel, um método do tipo gradiente.

Os métodos do tipo gradiente detectam as bordas procurando o valor máximo e mínimo na primeira derivada do sinal (GRE 02). A Figura 2-4 apresenta um sinal e a sua derivada na Figura 2-5 com um ponto máximo encontrado no centro da figura. (GRE 02).

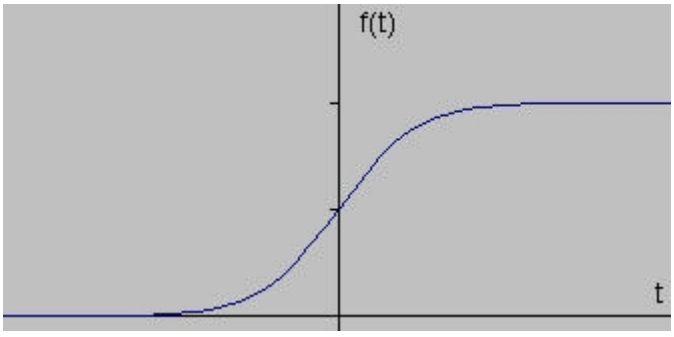


Figura 2-4: Sinal de uma imagem com borda (GRE 02)

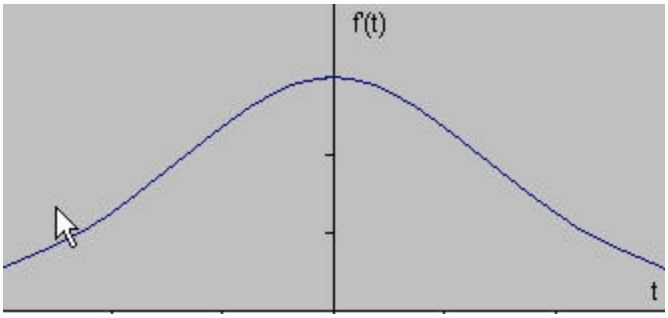


Figura 2-5: Resultado da primeira derivada do sinal (GRE 02)

Assim em uma imagem, mostrada na Figura 2-6, é possível detectar através da primeira derivada, o ponto onde a borda se localiza.

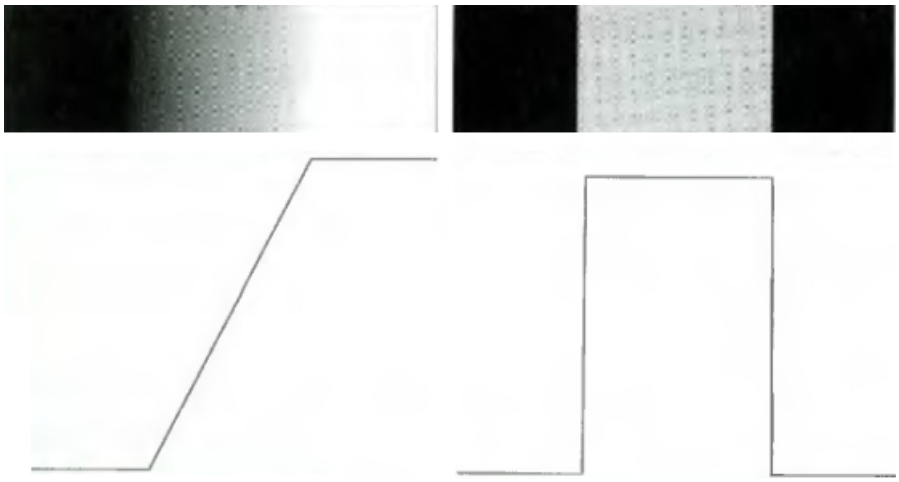


Figura 2-6: Resultado da primeira derivada (GON 02)

O detector de borda de Sobel usa um par de "máscaras" *3x3* que são convoluídas com a imagem original para calcular uma aproximação da derivada do sinal. Uma "máscara" que estima a inclinação no sentido *X* (colunas), e outra máscara que estima a inclinação no sentido *Y* (linhas). As "máscaras" mostradas na Figura 2-7 são chamadas de operador de Sobel (GRE 02):

-1	0	+1
-2	0	+2
-1	0	+1

Gx

+1	+2	+1
0	0	0
-1	-2	-1

Gy

Figura 2-7: Máscaras de Sobel (GRE 02)

Assim sendo *I* uma imagem inicial e *Gx*, *Gy* duas imagens que possuem uma aproximação às derivadas horizontal e vertical de *I* (GRE 02).

$$Gx = \begin{bmatrix} -1 & 0 & +1 \\ -2 & 0 & +2 \\ -1 & 0 & +1 \end{bmatrix} * A \qquad Gy = \begin{bmatrix} +1 & +2 & +1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix} * A \tag{2-5}$$

Então a magnitude *G* e a direção *D* do gradiente são dadas por (GRE, 02):

$$G = \sqrt{Gx^2 + Gy^2} \tag{2-6}$$

$$D = \arctan\left[\frac{Gx}{Gy}\right] \tag{2-7}$$

É importante observar que *pixels* nas primeiras e últimas fileiras não podem ser manipulados por uma máscara *3x3*, isto porque ao colocar o centro da máscara sobre um *pixel* na primeira fileira (por exemplo) a máscara estará fora dos limites da imagem (GRE 02).

O processo de convolução é apresentado na seção 2.4.2.

2.4.2. Convolução

A convolução é uma operação matemática fundamental a muitos operadores de processamento de imagens. A convolução fornece uma maneira de multiplicar duas disposições de números, geralmente de tamanhos diferentes, mas da mesma extensão, para produzir uma terceira disposição de números. Isto pode ser usado no processamento de imagem para executar os operadores cujos valores dos *pixels* da saída são combinações lineares de determinados valores dos *pixels* da entrada (FIS 00).

No contexto do processamento de imagem, uma das disposições da entrada é normalmente uma imagem *gray-scale*. A segunda disposição é geralmente menor, e é igualmente bidimensional também conhecida como "máscara". A Figura 2-8 mostra à representação de uma imagem na forma de uma matriz e uma matriz menor a "máscara" (FIS 00).

I_{11}	I_{12}	I_{13}	I_{14}	I_{15}	I_{16}	I_{17}	I_{18}	I_{19}
I_{21}	I_{22}	I_{23}	I_{24}	I_{25}	I_{26}	I_{27}	I_{28}	I_{29}
I_{31}	I_{32}	I_{33}	I_{34}	I_{35}	I_{36}	I_{37}	I_{38}	I_{39}
I_{41}	I_{42}	I_{43}	I_{44}	I_{45}	I_{46}	I_{47}	I_{48}	I_{49}
I_{51}	I_{52}	I_{53}	I_{54}	I_{55}	I_{56}	I_{57}	I_{58}	I_{59}
I_{61}	I_{62}	I_{63}	I_{64}	I_{65}	I_{66}	I_{67}	I_{68}	I_{69}

K_{11}	K_{12}	K_{13}
K_{21}	K_{22}	K_{23}

Figura 2-8: Exemplo de uma imagem (Esq.) e uma máscara (Dir.)

A convolução é executada "deslizando" a "máscara" sobre a imagem de entrada, começando geralmente no canto esquerdo superior, o movimento da máscara é feito por todas as posições onde a máscara se ajuste dentro dos limites da imagem de entrada, cada posição da máscara corresponde a um único *pixel* de saída. O valor de saída é calculado multiplicando o valor do item da "máscara" e o valor subjacentes do *pixel* na imagem de entrada. A saída é a soma de todas as células que foram multiplicadas (FIS 00). Assim, em nosso exemplo, da Figura 2-8, o valor do *pixel* de saída S_{57} é dado por:

$$S_{57} = I_{57}K_{11} + I_{58}K_{12} + I_{59}K_{13} + I_{67}K_{21} + I_{68}K_{22} + I_{69}K_{23} \quad (2\text{-}8)$$

Se a imagem tem fileiras *M* e colunas *N*, e a máscara tem fileiras *m* e colunas *n*, o tamanho da imagem da saída terá *M - m + 1* linhas, e *N - n + 1* colunas.

$$Saída(i,j) = \sum_{k=1}^{m}\left(\sum_{l=1}^{n}\left(I(i+k-1, j+l-1)K(k,l)\right)\right) \quad (2\text{-}9)$$

Onde *i* está entre *1* e *M - m + 1* e *j* está entre *1* e *N - n + 1* (FIS 00).

2.5. Transformada de Hough

A transformada de Hough é um método usado para detectar formas que podem ser parametrizadas, como linhas e círculos, em imagens digitais. O principal conceito da transformada de Hough está em criar um mapeamento entre a imagem e o espaço de parâmetros. Cada borda, previamente detectada, de uma imagem é transformada pelo mapeamento para determinar células no espaço de parâmetros. Essas células são incrementadas, e indicarão no final do processo, através da máxima local do acumulador, quais parâmetros são correspondentes a forma especificada (JAM 00).

Há várias parametrizações possíveis para o espaço de linhas. Hough usou a equação *declive-intercepte*, definida por $y = ax + b$, como a representação paramétrica de uma linha mostrado na Figura 2-9 são paralelas ao eixo *y* (JAM 00).

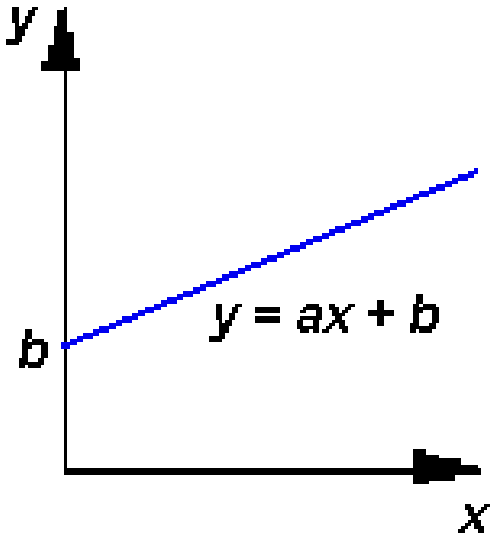


Figura 2-9: representação da equação da reta (declive-intercepte)

Para exemplificar a transformada de Hough é feito o cálculo apenas para dois *pixels* pertencentes a uma linha (reta) de uma imagem, mostrada na Figura 2-10.

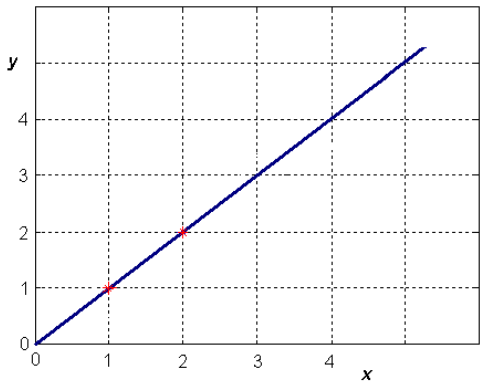


Figura 2-10: Plano (x, y) ou Espaço Real (JAM 00)

Para iniciar o processo é necessária uma matriz acumuladora que corresponde aos valores “quantizados” para *a* e *b,* onde *a* é estipulado entre [-2, 3]. Assim, usando uma matriz acumuladora A, o procedimento de Hough examina cada *pixel* e calcula os parâmetros da equação especificada que passa por cada *pixel* (JAM 00).

Calculados os parâmetros de um determinado *pixel*, eles são “quantizados” para um valor correspondente *a* e *b*, e o acumulador A(*a*, *b*) é incrementado. Quando todos os *pixels* tiverem sido processados, é procurado no acumulador A os máximos locais, estes valores indicam os parâmetros de prováveis linhas na imagem.

Na Figura 2-11 (plano a-b ou espaço de Hough), é apresentado as linhas geradas pelos parâmetros calculados, onde a linha verde refere-se ao ponto (1,1) e a azul ao ponto (2,2) do plano x-y (JAM 00).

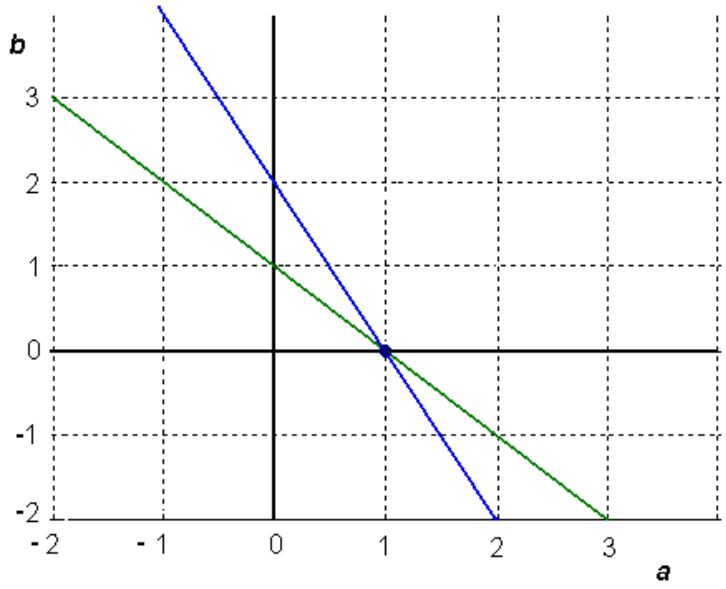


Figura 2-11: Plano a-b (parâmetros) (JAM 00)

Para este exemplo foi estipulado o intervalo [-2, 3] para *a*, e empregado os valores de *x* e *y* do pixel que está sendo examinado na equação $b = -ax + y$, possibilitando o cálculo do valor de *b*. O valor de *a* é escolhido empiricamente, entretanto a posição x_i de cada *pixel* do Plano Real (Figura 2-10) deve estar dentro do intervalo de *a*. Os valores são utilizados para incrementar o referido elemento do acumulador *A(a , b)* e traçar uma reta no plano a-b . A Tabela 2-1 mostra os valores utilizados e os parâmetros obtidos (JAM 00).

Tabela 2-1: Cálculo do parâmetro b e indicação e o acumulador A (JAM 00).

X	Y	a	b = -a*x+y	Acumulador A(a,b)
1	1	-2	3	***A* (-2, 3)**
1	1	-1	2	***A* (-1, 2)**
1	1	0	1	***A* (0, 1)**
1	1	1	0	***A* (1, 0)**
1	1	2	-1	***A* (2,-1)**
1	1	3	-2	***A* (3,-2)**
2	2	-2	6	***A* (-2, 6)**
2	2	-1	4	***A* (-1, 4)**
2	2	0	2	***A* (0, 2)**
2	2	1	0	***A* (1, 0)**
2	2	2	-2	***A* (2,-2)**
2	2	3	-4	***A* (3,-4)**

Ao procurar o máximo no acumulador *A(a , b)*, verifica-se que este indicará o elemento referenciado por *a = 1* e *b = 0*. Isto significa que a equação *y = 1.x + 0*, ou simplesmente *y = x*, é a equação que melhor define uma reta que passa sobre os pontos *(1,1)* e *(2,2)* da imagem. Tendo esta informação faz-se um pós-processamento para determinar onde começa e termina a reta encontrada, caso haja interesse. Um limiar pode ser utilizado quando se procura o(s) máximo(s) no acumulador, a fim de determinar um valor mínimo de pontos colineares. Se o valor do acumulador não for superior ao do limiar então é considerado um ruído. As detecções de outras formas utilizando a transformada de Hough usam o mesmo princípio, há somente alteração no número de parâmetros da equação que é empregada, e em conseqüência na dimensão do acumulador (JAM 00).

2.6. OpenCV

OpenCV é uma biblioteca para visão computacional de código aberto disponíveis a partir da Internet, a biblioteca é escrita em C e C + + e roda em Linux, Windows e Mac OSX e possui interfaces de desenvolvimento para as linguagens *Python*, *Ruby* e *Matlab*. *OpenCV* foi projetado para obter eficiência computacional em aplicações que de tempo real. Se *OpenCV* estiver sobre uma arquitetura *Intel* ela pode tirar proveitos de desempenho das funções primitivas do processador (BRA 08).

Os objetivos da *OpenCV* é fornecer uma infra-estrutura simples de usar que ajude as pessoas a construir aplicativos rápidos e sofisticados baseados em visão computacional. A biblioteca *OpenCV* contém mais de 500 funções que abrangem várias áreas de visão, incluindo inspeção de produtos em fabricas, imagens médicas, segurança, interface do usuário, calibração de câmera e robótica. *OpenCV* também possui funções para auxiliar no trabalho de aprendizagem de maquina, um item extremamente útil para a visão computacional.

3. REDES NEURAIS

Neste capítulo são apresentados os conceitos básicos de redes neurais. O propósito da utilização de uma rede neural é fazer uma comparação deste método com o método de detecção de bordas no que se refere à localização da faixa na rodovia.

3.1. Reconhecimento de padrões

Um padrão é basicamente um conjunto de objetos que devido as semelhanças de suas características se agrupam em uma determinada categoria. Reconhecer é uma característica básica de todos os seres humanos; quando uma pessoa vê um objeto, ela reúne todas as propriedades e comportamentos deste objeto, depois ela compara com algo parecido previamente armazenado na mente. O objeto visto é reconhecido se, na mente for encontrado algo similar (PAN 96).

O ato de reconhecer é um conceito simples e familiar para todos no mundo real, mas no mundo da inteligência artificial reconhecer um objeto é uma incrível façanha; isso porque a funcionalidade do cérebro humano é surpreendente e não é comparável a qualquer software ou maquina artificial. O ato de reconhecer pode ser dividido em duas grandes categorias: reconhecimento de algo concreto e reconhecimento de algo abstrato. O reconhecimento de algo concreto envolve elementos espaciais e temporais como impressões digitais, mapas meteorológicos, fotos ou uma forma de onda. Reconhecimento de algo abstrato refere-se a itens que não existem fisicamente (JES 08).

Neste trabalho é tratado apenas o reconhecimento concreto. Entre as aplicações que fazem uso do reconhecimento concreto incluem: reconhecimento de impressão digital, reconhecimento vocal, reconhecimento de face, reconhecimento de caracteres, e reconhecimento de assinaturas.

3.2. Redes neurais artificiais

Um neurônio em “desenvolvimento” é comparado a um cérebro plástico: a plasticidade permite que o sistema nervoso em desenvolvimento adapte-se ao seu meio. Assim como a plasticidade parece ser essencial para o funcionamento dos neurônios como unidades de processamento de informação do cérebro humano, também ela o é com relação às redes neurais constituídas com neurônios artificiais. Na sua forma mais geral, uma rede neural é uma máquina que é projetada para modelar a maneira como o cérebro realiza uma tarefa particular ou função de interesse; a rede é normalmente implementada utilizando-se componentes eletrônicos ou é simulada por programação em um computador digital (HAY 01).

3.3. Modelo de um neurônio

Um neurônio é uma unidade de processamento de informação que é fundamental para a operação de uma rede neural. Um elemento de processamento, de uma rede neural, é mostrado na Figura 3-1; São mostradas também as entradas, os pesos (*Wi*) de cada entrada e a saída. O valor associado aos nodos é chamado de valor de ativação, o qual é a soma de cada valor da entrada multiplicado por seu respectivo valor do peso (CAN 06).

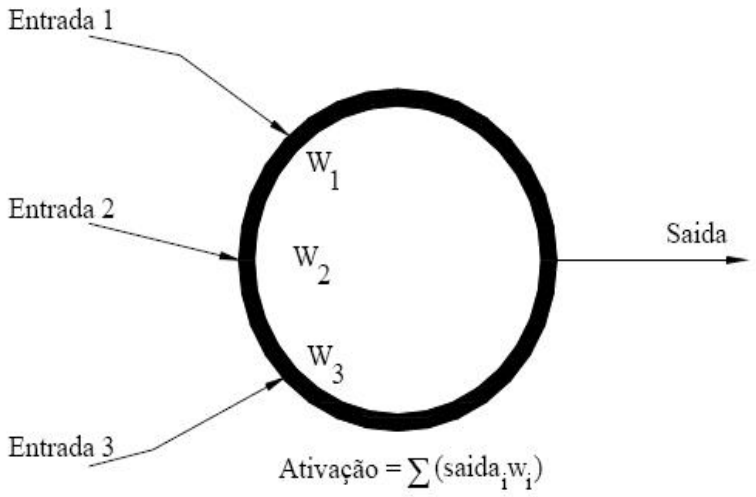


Figura 3-1: Modelo de um neurônio (CAN 06)

Uma rede neural artificial é formada pela conexão de neurônios, ou nós; Os nós são organizados em camadas, como se ilustra na Figura 3-2. As entradas na primeira camada (camada de entradas) são recebidas pelos neurônios da camada de entrada. As saídas destes neurônios são direcionadas para as entradas dos neurônios da camada seguinte até a saída da rede (CAN 06).

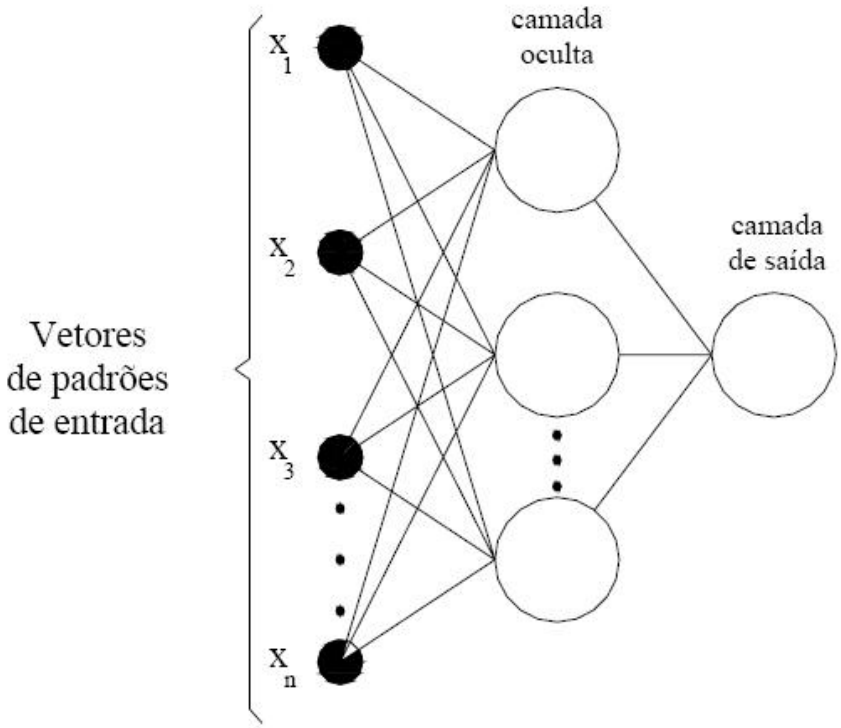


Figura 3-2: Estrutura de uma rede neural artificial (CAN 06)

3.4. Benefícios das Redes Neurais

É evidente que uma rede neural extrai seu poder computacional através, primeiro, de sua estrutura maciçamente paralela e distribuída e segundo de sua habilidade de aprender e, portanto, de generalizar. A generalização se refere ao fato de a rede neural produzir saídas adequadas para entradas que não estavam presentes durante o treinamento (aprendizagem). Estas duas capacidades de processamento de informação tornam possível para a rede neural resolver problemas complexos que são atualmente intratáveis por outros métodos computacionais (HAY 01).

3.5. Exemplo do funcionamento de um neurônio

O objetivo do *software*, produto deste trabalho, é reconhecer uma faixa na rodovia. Neste capítulo é mostrado como uma rede neural pode realizar o reconhecimento de um padrão de imagem.

Na Figura 3-3 vê se uma representação de uma rede com três neurônios (*N1*, *N2* e *N3*). Cada entrada (*Xij*) representa o estado de um pixel.

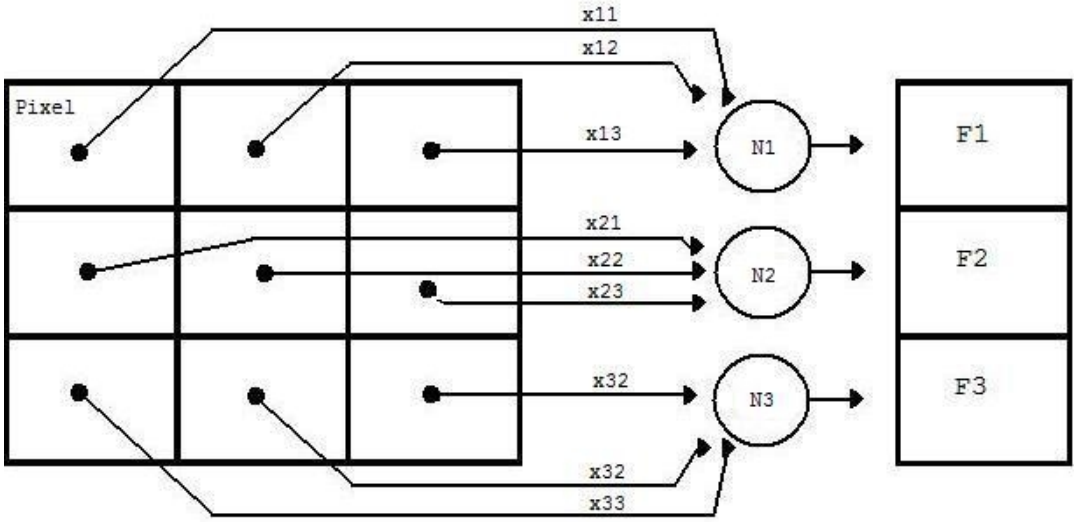


Figura 3-3: Rede neural com três neurônios

A rede neural da Figura 3-4 está treinada para reconhecer os padrões T e H. A associação dos padrões são: tudo preto e tudo branco, respectivamente, como são mostrado na Figura 3 4.

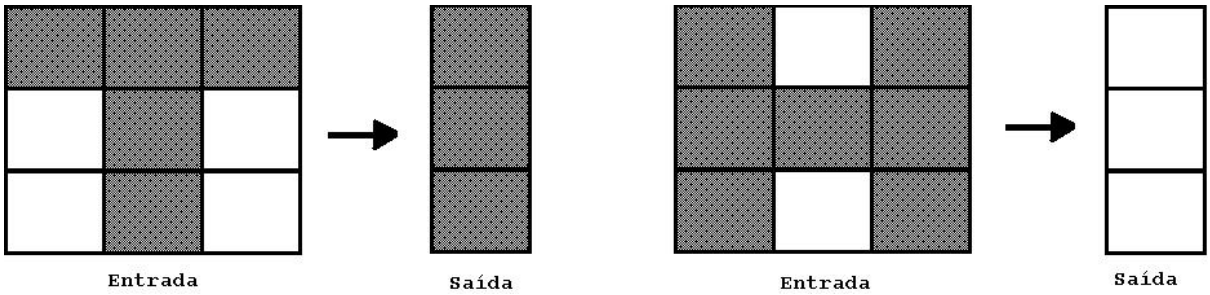


Figura 3-4: Associação Entrada X Resultado

Fazendo cada pixel preto corresponder ao valor 0 (zero) e cada pixel branco corresponder ao valor 1 (um), então a tabela verdade para os três neurônios após a generalização é mostrada nas tabelas: Tabela 3-1, Tabela 3-2 e Tabela 3-3.

Tabela 3-1: Tabela verdade (neurônio N1)

X11:	0	0	0	0	1	1	1	1
X12:	0	0	1	1	0	0	1	1
X13:	0	1	0	1	0	1	0	1
Saída	0	0	1	1	0	0	1	1

Tabela 3-2: Tabela verdade (neurônio N2)

X21:	0	0	0	0	1	1	1	1
X22:	0	0	1	1	0	0	1	1
X23:	0	1	0	1	0	1	0	1
Saída	1	0/1	1	0/1	0/1	0	0/1	0

Tabela 3-3: Tabela verdade (neurônio N3)

X21:	0	0	0	0	1	1	1	1
X22:	0	0	1	1	0	0	1	1
X23:	0	1	0	1	0	1	0	1
Saída	1	0	1	1	0	0	1	0

Dos resultados apresentados nas tabelas: Tabela 3-1, Tabela 3-2 e Tabela 3-3, vê-se que as os resultados, da Figura 3-5, podem ser extraídos.

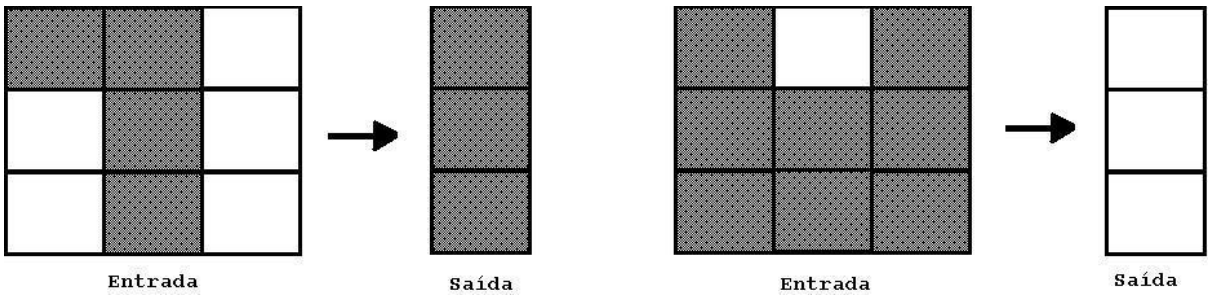


Figura 3-5: Resultados do reconhecimento com a RN

Os resultados obtidos, mostrados na Figura 3-5, mostram que a rede pode tratar entradas diferentes daquelas para as quais foi treinada.

3.6. Processos de aprendizagem

O procedimento utilizado para realizar o processo de aprendizagem é chamado de algoritmo de aprendizagem, cuja função é modificar os pesos da rede de uma forma ordenada para alcançar um objetivo desejado (HAY 01).

A propriedade que é de importância primordial para uma rede neural é a sua habilidade de aprender a partir de seu ambiente e de melhorar o seu desempenho através da aprendizagem. A melhoria do desempenho ocorre com o tempo de acordo com alguma medida preestabelecida. Uma rede neural aprende acerca do seu ambiente através de um processo interativo de ajustes aplicados a seus pesos. Idealmente, a rede se torna mais instruída sobre o seu ambiente após cada iteração do processo de aprendizagem (HAY 01).

Portanto, aprendizagem é um processo pelo qual os parâmetros livres de uma rede neural são adaptados através de um processo de estimulação pelo ambiente no qual a rede está inserida. O tipo de aprendizagem é determinado pela maneira pela qual a modificação dos parâmetros ocorre. Esta definição do processo de aprendizagem implica a seguinte seqüência de eventos.

- A rede neural é estimulada por um ambiente
- A rede neural sofre modificações nos seus parâmetros livres como resultado desta estimulação.
- A rede neural responde de uma maneira nova ao ambiente, devido às modificações ocorridas na sua estrutura interna (HAY 01).

3.7. Aprendizagem com um professor

Também conhecida como aprendizagem supervisionada, este tipo de aprendizagem considera o professor como tendo conhecimento sobre o ambiente, com este conhecimento sendo representado por um conjunto de exemplos de entrada-saída. Entretanto, o ambiente é desconhecido pela rede neural de interesse. Suponha agora que o professor e a rede neural sejam expostos a vetor de treinamento retirado do ambiente. Em virtude de seu conhecimento prévio, o professor é capaz de fornecer à rede neural uma resposta desejada para aquele vetor de treinamento. Na verdade, a resposta desejada representa a ação ótima a ser realizada pela rede neural. Os parâmetros da rede são ajustados sob a influência combinada do vetor de treinamento e do sinal de erro. O sinal de erro é definido como a diferença entre a resposta desejada e a resposta real da rede. Este ajuste é realizado passo a passo, interativamente, com o objetivo de fazer a rede neural emular o professor. Desta forma, o conhecimento do ambiente disponível ao professor é transferido para a rede neural através de treinamento (HAY 01).

3.8. Perceptron

O *perceptron* é a forma mais simples de uma rede neural usada para classificação de padrões. Basicamente o *perceptron* é constituído de um único neurônio com pesos sinápticos ajustáveis (HAY 01).

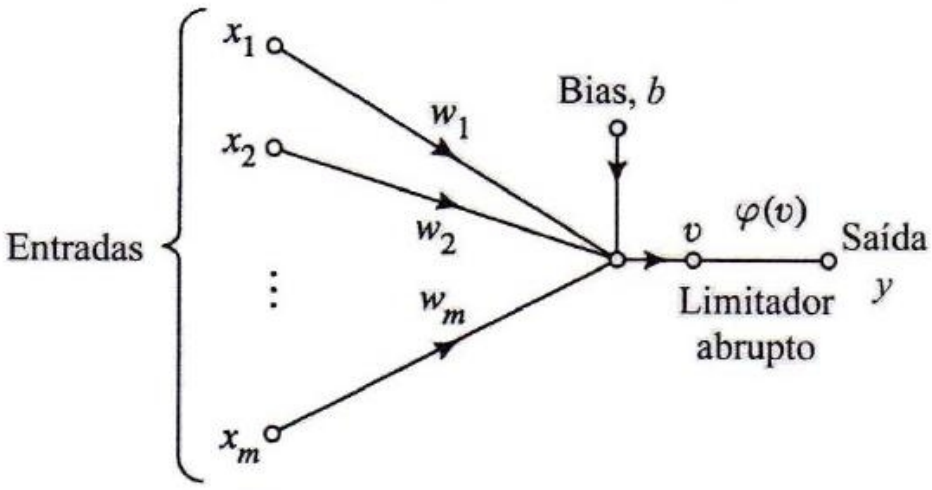


Figura 3-6: Grafo de fluxo de sinal do perceptron (HAY 01)

No modelo de grafo de fluxo de sinal da Figura 3-17, os pesos sinápticos do perceptron são representados por $w_1, w_2, \ldots w_m$. Correspondentemente, as entradas aplicadas ao perceptron são representadas por $x_1, x_2, \ldots, x_m$. O *bias* aplicado externamente é representado é por *b*. Do modelo constatamos que a entrada do limitador abrupto ou o campo local induzido no neurônio é

$$v = \sum_{i=1}^{m} w_i x_i + b \qquad (3\text{-}1)$$

O objetivo do perceptron é classificar corretamente o conjunto de estímulos aplicados externamente $x_1, x_2, \ldots, x_m$ em uma de duas classes C_1 e C_2. A regra de decisão para a classificação é atribuir o ponto representada palas entradas $x_1, x_2, \ldots, x_m$ à classe C_1 se a saída do perceptron *y* for *+1* e à classe C_2 se ela for *-1*.

Para compreender melhor o comportamento de um classificador de padrões, normalmente se traça um mapa das regiões de decisão no espaço de sinal m-dimensional abrangido pelas m variáveis de entrada $x_1, x_2, \ldots, x_m$. Na forma mais simples do perceptron, existem duas regiões separadas por um hiperplano definido por

$$\sum_{i=1}^{m} w_i x_i + b = 0 \qquad (3\text{-}2)$$

Isto está ilustrado na Figura 3-18 para o caso de duas variáveis *x1* e *x2*, para o qual a fronteira de decisão toma a forma de uma linha reta. Um ponto (*x1*, *x2*) que se encontra acima da linha de fronteira é atribuído à classe C1 e um ponto (*x1*, *x2*) que está abaixo da linha de fronteira é atribuído a C2. Note também que o efeito do bias *b* é meramente de deslocar a fronteira de decisão em relação à origem (HAY 01).

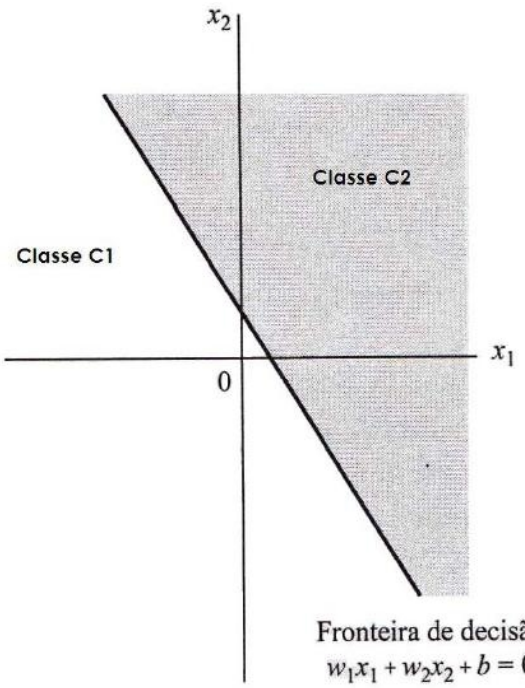


Figura 3-7: Fronteira de decisão (HAY 01)

3.9. Perceptron multicamada

As redes de uma só camada resolvem apenas problemas linearmente separáveis. A solução de problemas não linearmente separáveis passa pelo uso de redes com uma ou mais camadas intermediárias ou escondidas (BRA 00).

As redes de múltiplas camadas, tipicamente, consistem de um conjunto de unidades sensoriais (nós de fonte) que constituem a camada de entrada, uma ou mais camada ocultas de nós computacionais e uma camada de saída de nós computacionais. O sinal de entrada se propaga para frente através da rede, camada por camada. Estas redes, *perceptrons* de múltiplas camadas (MLP, *multilayer* perceptron) representam uma generalização do perceptron de camada única (HAY 01). A Figura 3-19 mostra o grafo arquitetural de um perceptron de múltiplas camadas com a camada de entrada, uma camada intermediaria e uma camada de saída.

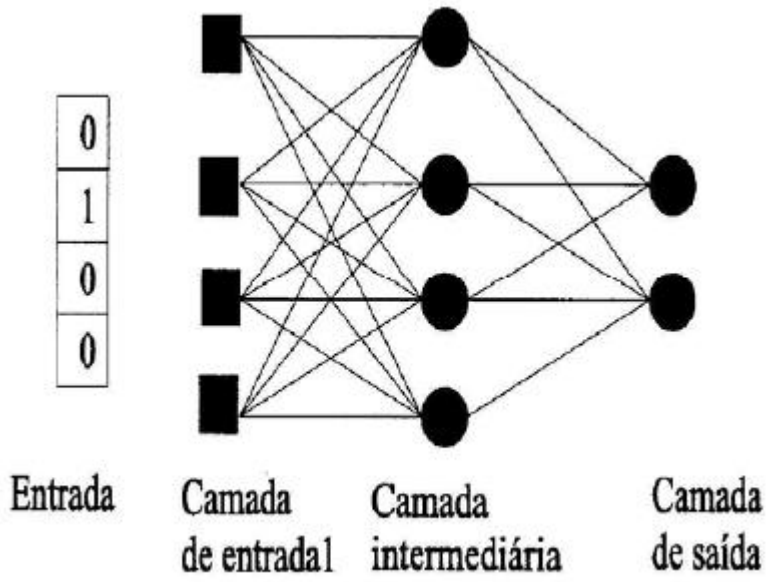


Figura 3-8: Grafo arquitetural de um perceptron (HAY 01)

3.10. O Algoritmo Back-propagation

Uma rede com uma camada intermediária pode implementar qualquer função contínua e a utilização de duas camadas intermediárias permite a aproximação de qualquer função. Porém deve ser observado que permitir a implementação (aprendizado) da função não implica, como é o caso da perceptron, a garantia de implementação da função. Dependendo da distribuição dos dados, a rede pode convergir para um mínimo local ou pode demorar demais para encontrar a solução desejada. O problema passa a ser então como treinar estas redes (BRA 00).

O algoritmo de aprendizado mais conhecido para treinamento destas redes é o algoritmo *back-propagation*. O algoritmo *back-propagation* é um algoritmo supervisionado que utilizam pares (entrada, saída desejada) para, por meio de um mecanismo de correção de erros, realizarem ajustes nos pesos da rede. O treinamento ocorre em duas fases, em que cada fase percorre a rede em um sentido.

Estas duas fases são chamadas de fase *forward* (para frente) e fase *backward* (para trás). A fase *forward* é utilizada para definir a saída da rede para um dado padrão de entrada. A fase *backward* utiliza a saída desejada e a saída fornecida para atualizar os pesos de suas conexões. A Figura 3 9 ilustra estas duas fases (BRA 00).

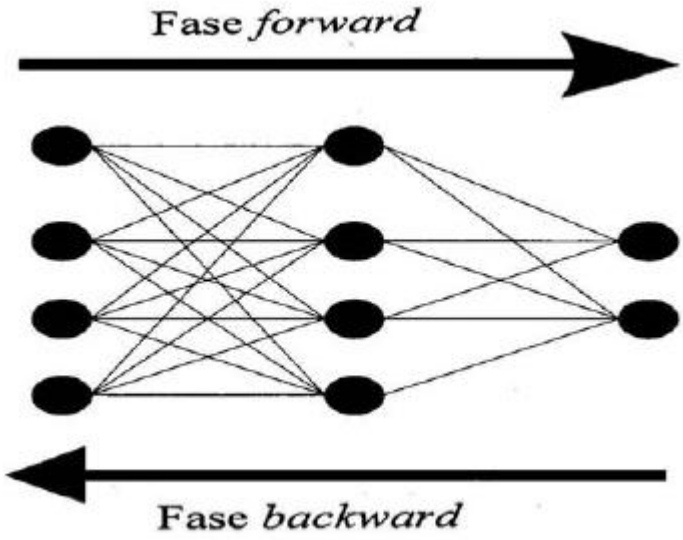


Figura 3-9: Fases do Back-propagation (BRA 00)

Figura 3-9: Fluxo de processamento do algoritmo back-propagation. Os dados seguem da entrada para a saída no sentido *forward*, e os erros, da saída para a entrada no sentido *backward*.

4. DESENVOLVIMENTO DO SISTEMA

Este capítulo apresenta a estrutura topológica do sistema, as ferramentas utilizadas e o processo de desenvolvimento do software.

4.1. Topologia do sistema

O sistema criado é constituído por uma câmera de vídeo e uma unidade de processamento embarcado em um veículo. A unidade de processamento contém o *software* que é o responsável pelo reconhecimento da posição do veículo na rodovia.

A câmera de vídeo, posicionada próximo ao retrovisor, captura imagens da rodovia e as encaminha por meio de um cabo *USB* para a unidade de processamento. A topologia é mostrada na Figura 4-1.

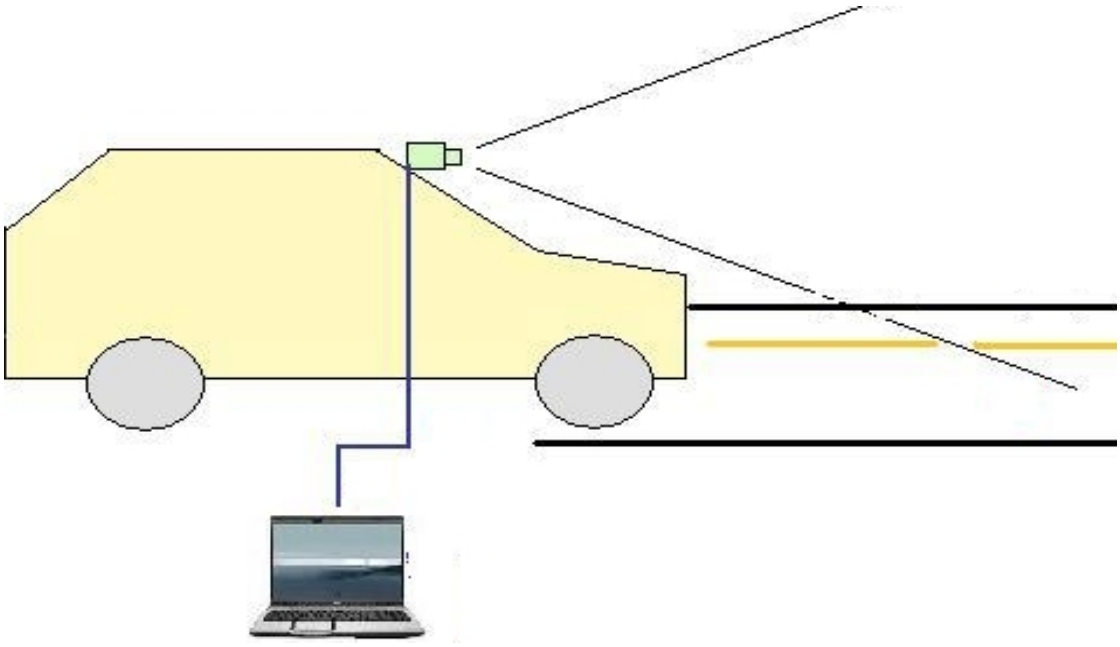


Figura 4-1: Topologia do sistema

O *software*, contido na unidade de processamento, “lê” as imagens e inicia os procedimentos necessários para informar a posição do veículo e emitir um alerta sonoro caso o veículo inicie uma saída de pista.

O *software* é dividido em dois módulos, o módulo de configuração e o módulo de processamento. O módulo de configuração é usado para definir os parâmetros iniciais que são usados pelo módulo de processamento, a Figura 4-2 mostra a topologia do *software*.

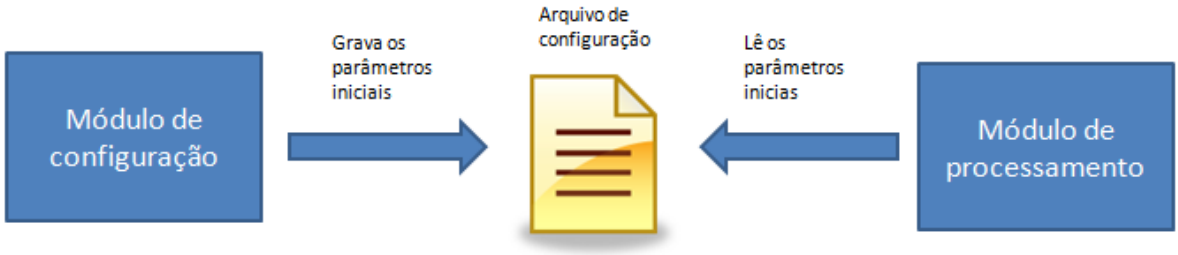


Figura 4-2: Topologia do software

4.1.1. Módulo de configuração

O módulo de configuração é o responsável por definir os parâmetros iniciais do software, que são:

- Definir, na imagem, a posição que a faixa, à esquerda do veículo, é visualizada quando o veículo está no centro da pista.
- Definir a distância (em metros) entre a roda esquerda do veículo e a faixa que divide as pistas na estrada.
- Definir, na imagem, a posição que a faixa é visualizada quando o veículo está sobre ela.
- Gravar as configurações no arquivo de configuração.

4.1.2. Módulo de processamento

O módulo de processamento realiza o trabalho necessário para informar a posição do veículo e alertar o condutor caso o veículo inicie uma saída de pista. As principais tarefas do módulo de processamento são:

- Tratar as informações do arquivo de configuração.
- Tratar as imagens recebidas pela unidade de processamento

- Realizar o pré-processamento, a segmentação e a interpretação da imagem.
- Informar a posição do veículo na pista e alertar o condutor caso haja o inicio de uma saída de pista.

4.2. *Hardware* utilizado

O sistema é composto por dois *hardwares*, uma câmera de vídeo e uma unidade de processamento. A câmera de vídeo usada é uma *WebCam* ligada à unidade de processamento através de um cabo *USB,* neste projeto a unidade de processamento é um *Notebook* cuja configuração é apresentada na Tabela 4-1.

Tabela 4-1: Configuração da unidade de processamento

Marca / Modelo	HP Pavilion dv6433
Memória	2 GB
Processador	Intel Centrino Duo 1.7 GHz

A configuração da câmera é apresentada na Tabela 4-2.

Tabela 4-2: Configuração da câmera de vídeo

Marca / Modelo	Creative WebCam Live Ultra
Resolução do sensor	1.3 MP
Frames por segundo	30
Tamanho da imagem gerada	640 x 480 pixels

4.3. Ferramentas utilizadas

Esta seção mostra as linguagens de programação, *interface* de desenvolvimento e bibliotecas utilizadas no desenvolvimento do software.

4.3.1. Linguagens de programação

O software foi desenvolvido usando duas linguagens de programação, *C* e *Java*. A *interface* gráfica do módulo de configuração foi desenvolvida usando a linguagem *Java* e o módulo de processamento foi totalmente desenvolvimento usando a linguagem *C*.

A linguagem *Java* foi escolhida para desenvolver o módulo de configuração porque esta linguagem possui um conjunto de funções especializadas em gerar *interfaces* gráficas. O conjunto de funções, chamado *Swing,* flexibiliza o desenvolvimento por possuir todos os componentes utilizados para interagir com o usuário.

A linguagem *C* é uma linguagem de programação para uso geral, amplamente utilizado em diversas arquiteturas de computadores. *C* foi escolhida para desenvolver o módulo de processamento devido ao seu desempenho extremamente rápido se comparado com a linguagem *Java*.

4.4. Desenvolvimento

Esta seção mostra como foi realizado o desenvolvimento do software, apresenta também os problemas encontrados e as soluções escolhidas para resolvê-los.

4.4.1. Módulo de configuração

O módulo de configuração define os parâmetros iniciais que o módulo de processamento necessita para trabalhar. Os casos de uso do módulo de configuração são:

1. O usuário define a posição da faixa quando o veículo está com a roda esquerda sobre a faixa.
2. O usuário define a posição da faixa quando o veículo está no centro da pista, e a distância entre a roda esquerda e a faixa, quando o veículo está no centro da pista.

A Tabela 4-3 apresenta o caso de uso 1 “O usuário define a posição da faixa quando o veículo está com a roda esquerda sobre a faixa”

Tabela 4-3: Caso de uso 1.

Especificação de caso de uso "Definir a posição da faixa quando o veículo está no centro da pista"	
Pré-condições:	O veículo deve estar posicionado com a roda esquerda sobre a faixa. O sistema deve possuir uma câmera de vídeo. A câmera de vídeo deve estar filmando a área à frente do veículo.
Fluxo de eventos	
1. Este caso de uso se inicia quando o usuário inicia o modulo de configuração. 2. Na tela do modulo de configuração o usuário aciona o botão "Capturar 1" 3. O sistema captura a imagem da câmera de vídeo e exibe esta imagem para o usuário. 4. O usuário marca na imagem o ponto (na área inferior da imagem) que a faixa é visualizada. 5. O usuário aciona o botão "Salvar". 6. O sistema salva as informações da tela no arquivo de configuração. 7. O usuário aciona o botão "Fechar" 8. O caso de uso é finalizado.	

A Tabela 4-3 apresenta o caso de uso 2 “O usuário define a posição da faixa quando o veículo está no centro da pista”.

Tabela 4-4: Caso de uso 2.

Especificação de caso de uso “O usuário define a posição da faixa quando o veículo está no centro da pista, e a distância entre a roda esquerda e a faixa, quando o veículo está no centro da pista”	
Pré-condições:	O veículo deve estar posicionado no centro da pista. O sistema deve possuir uma câmera de vídeo. A câmera de vídeo deve estar filmando a área à frente do veículo.
Fluxo de eventos	
1. Este caso de uso se inicia quando o usuário inicia o modulo de configuração. 2. Na tela do modulo de configuração o usuário aciona o botão “Capturar 1” 3. O sistema captura a imagem da câmera de vídeo e exibe esta imagem para o usuário. 4. O usuário marca na imagem o ponto (na área inferior da imagem) que a faixa é visualizada. 5. O entra com o valor da distância entre a roda esquerda e a faixa. 6. O usuário aciona o botão “Salvar”. 7. O sistema salva as informações da tela no arquivo de configuração. 8. O usuário aciona o botão “Fechar” 9. O caso de uso é finalizado.	

Com base nos casos de uso, os requisitos que o módulo deve atender são:

1. O módulo deve permitir a captura de imagens da câmera de vídeo.
2. O módulo deve permitir que seja marcada, na imagem, a posição que o usuário vê a imagem.
3. O módulo deve possibilitar que o usuário informe a distância entre a roda (esquerda) do veículo e a faixa da rodovia.

4. O módulo deve permitir que o usuário grave as opções selecionadas no arquivo de configuração.

A Figura 4-3 mostra a tela do protótipo da *interface* com o usuário.

Figura 4-3: Protótipo da interface com o usuário do módulo de configuração

Com o objetivo de atender os requisitos deste módulo é apresentado abaixo os trechos de códigos mais importantes:

Requisito: O módulo deve permitir a captura de imagens da câmera de vídeo.

Para realizar a captura da imagem o software faz uma chamada ao aplicativo "CAPTURE". Este aplicativo foi desenvolvido em *C* e usa a funções da biblioteca para capturar uma imagem da câmera de vídeo e salva-la como um arquivo *JPG*.

```
Process p = Runtime.getRuntime().exec(CAPTURE_APP)
```

Após a execução do aplicativo "CAPTURE" o módulo atualiza a exibição da imagem. A visualização da imagem é atualizada usando o seguinte código:

```
ResourceMap resourceMap = Application.getInstance(
ldwcfg.LDWCfgApp.class).getContext().getResourceMap(ConfigurarBox.class);
ImageIcon icone = new ImageIcon(IMAGEMF);
java.awt.Image aux = icone.getImage();
aux.flush();
imagemF.setIcon(new ImageIcon(aux));
```

O módulo deve permitir que seja marcada, na imagem, a posição que a imagem é visualizada. Para permitir que o usuário marque a posição é usado um objeto do tipo *Slider*, que permite marcar uma posição entre um intervalo de valores. A declaração deste objeto é feita da seguinte forma:

```
private javax.swing.JSlider posicaoCentro
```

Requisito: O módulo deve possibilitar que o usuário informe a distância entre a roda (esquerda) do veículo e a faixa da rodovia.

O usuário irá informar o valor em num objeto chamado *JTextField*, este objeto é declarado da seguinte forma:

```
private javax.swing.JTextField distanciaPontosText;
```

Requisito: O módulo deve permitir que o usuário grave as opções selecionadas no arquivo de configuração.

Para realizar a gravação no arquivo de configuração é usado um objeto do tipo *BufferedWriter*. O seguinte trecho de código mostra como este objeto é usado:

```
// Cria um arquivo para escrita
BufferedWriter file = new BufferedWriter(new FileWriter( ARQ_CONFIG))
```

Os trechos de códigos mais importantes do módulo de configuração estão no Apêndice A.

4.4.2. Módulo de processamento

O módulo de processamento informa a posição do veículo na pista e alerta o condutor caso o veículo inicie uma saída de pista. Os requisitos que o módulo deve atender são:

1. O módulo deve capturar as imagens da câmera de vídeo.
2. O módulo deve informar o usuário sobre a posição do veículo em relação a faixa contínua que está no centro da estrada.
3. O módulo deve alertar o usuário caso o veículo inicie uma saída de pista.

Para atender os requisitos deste módulo o desenvolvimento segue o fluxo apresentado na Figura 2-1: Estrutura funcional de um sistema de visão computacional, apresentado na seção 2.2.

4.4.3. Aquisição da imagem

A biblioteca *OpenCV* provê ferramentas utilitárias para ler o conteúdo de imagens ou os *frames* de um vídeo. Estas ferramentas são parte do pacote *HighGUI* que faz parte do *OpenCV*. O Exemplo 1 mostra como é carregada uma imagem a partir do *frame* de um vídeo.

```
int main( int argc, char** argv ) {
cvNamedWindow( “Exemplo1”, CV_WINDOW_AUTOSIZE );
CvCapture* capture = cvCaptureFromCAM(0);
IplImage* frame;
while(1) {
frame = cvQueryFrame( capture ); if( !frame ) break;
cvShowImage( “Exemplo1”, frame );
char c = cvWaitKey(20); if( c == 27 ) break; }
cvReleaseCapture( &capture );
cvDestroyWindow( “Exemplo1” );
}
```

Exemplo 1: Lendo uma imagem da câmera

A função *cvNamedWindow* cria uma janela que pode mostrar uma imagem. Esta função é fornecida pelo pacote *HighGUI*. A função *cvNamedWindow* requer dois parâmetros, o nome da janela e seu tamanho. No Exemplo 1 o nome da janela é “Exemplo1” e o tamanho é ajustado conforme o tamanho da imagem quando esta for carregada pela função *cvShowImage*. A função *cvShowImage* mostra a janela com uma imagem, caso a janela já tenha sido mostrada a função apenas atualiza a janela com a nova imagem.

A função *cvWaitKey* executa uma parada no processamento, esta função requer apenas um parâmetro. Se o parâmetro for positivo o programa irá esperar pelo tempo correspondente ao parâmetro passado, em milissegundos. Se o parâmetro for 0 ou um valor negativo o programa irá esperar que seja pressionado uma tecla antes de continuar.

Uma vez carregada na memória a imagem pode ser removida com a função *cvReleaseImage*. OpenCV espera que seja passado para esta função um ponteiro do tipo *IplImage**. Finalmente a janela criada pode ser destruída com o comando *cvDestroyWindow*.

A função *cvCaptureCam* retorna um ponteiro para uma estrutura do tipo *CvCapture*. Esta estrutura contém todas as informações lidas da câmera. A função *cvQueryFrame* faz a leitura de um frame, esta função retorna um ponteiro para um item do tipo *IplImage*.

Por fim *while*(1) inicia um *loop* infinito que pode ser cancelado caso o usuário tecle a teclar *ESC*, que em código *ASCII* corresponde ao valor *27*.

Após adquirir uma imagem o programa segue com o pré-processamento da imagem.

4.4.4. Pré-processamento

Esta etapa visa preparar a imagem para que o trabalho de segmentação seja otimizado. Caso seja necessário alguns ajustes na imagem podem ser realizados nesta etapa.

Para detectar a saída de pista é suficiente interpretar apenas algumas partes da imagem que contém características importantes dela, não sendo necessário analisar todas as outras partes da imagem. Geralmente, só pequenas áreas da imagem contem características relevantes, onde deve estar concentrada toda a potência do processamento, que deve ser eficiente (CAN 06). A Figura 4-4 ilustra este conceito aplicado a detecção da faixa na rodovia.

A região definida como “região de interesse” foi escolhida empiricamente. A região é a área esquerda inferior da imagem porque neste local está a faixa da rodovia que é analisada. A largura da imagem é de 230 *pixels,* a metade da largura da imagem capturada pela câmera de vídeo.

Figura 4-4: Ilustração da imagem com destaque para a área de interesse

O módulo de processamento usa o seguinte código para copiar a área que é processada pelo módulo de processamento.

```
cvSetImageROI(imagem, area);
```

A função *cvSetImagemROI* define uma região na imagem que irá sofrer o tratamento. Os função *cvSetImagemROI* requer dois parâmetros, a imagem e a região de interesse. O parâmetro área é do tipo retângulo (*CvRect*). Após definir uma região de interesse o processo pode ser desfeito com a função *cvResetImageROI(IplImage* image)*.

O processo de segmentação, deste software, necessita que a imagem seja uma imagem *Gray-scale.* Para converter a imagem é usado a função *cvCvtColor,* esta função converte imagens de um tipo para outro. Os parâmetros esperados são imagem original, imagem de destino e o tipo de conversão. Alguns tipos de conversões são mostrados na Tabela 4-5.

Tabela 4-5: Opções de conversão da função cvCvtColor

Constante	**Descrição**
CV_RGB2GRAY	Converte RGB para *gray-scale*
CV_GRAY2RGBA	Converte de *gray-scale* para RGB

Após delimitar a região de interesse e converter a imagem para Gray-scale o próximo passo é segmentar a imagem.

4.4.5. Segmentação e Representação

O processo de segmentação tem por objetivo extrair uma determinada região da imagem ou dividir a imagem num conjunto de regiões distintas. Nesta seção é mostrado como foi desenvolvido o processo de localização da faixa na rodovia.

4.4.6. Segmentação e Representação por detecção de bordas

A transformada de Hough, vista na seção 2.5, retorna um conjunto de pontos que usamos para localizar a posição da faixa na rodovia, entretanto a transformada de Hough obtém melhores resultados quando é executada sobre uma imagem cujas bordas já foram destacadas. Para destacar as bordas é usado o algoritmo de Sobel.

A biblioteca *OpenCV* implementa o algoritmo de Sobel através da função *cvSobel(const CvArr* src, CvArr* dst, int xorder, int yorder, int aperture_size = 3)*; Onde *src* é a imagem de origem, *dst* é a imagem de destino, *xorder* e *yorder* são

ordens da derivada na direção *x, y*. A Figura 4-5 ilustra uma imagem antes e depois de aplicar o algoritmo de Sobel.

Figura 4-5: Exemplo de uma imagem após aplicar Sobel

A função cvSobel é usada no software da seguinte forma:

```
cvSobel(imagemCinza, imagemSobel, 1 , 1, 1);
```

A transformada de Hough não mostra o seu resultado explicitamente, em vez disso ele retorna um conjunto de pontos que (*x0, y0)* da origem e (*x1 y1)* do destino para formar as retas encontradas. *OpenCV* suporta diversos tipos de transformada de Hough, neste projeto é usado a transformada probabilística de Hough. A transformada probabilística de Hough é uma variação da transformada de Hough, "probabilística", porque, em vez de acumular uma extensão de linhas individuais no acumulador ela acumula apenas uma fração deles, a idéia é reduzir o tempo de processamento com a redução do número de cálculos realizados.

A função que implementa a transformada de Hough tem a seguinte assinatura:

```
CvSeq* cvHoughLines2(CvArr* image, void* line_storage, int method, double rho, double theta, int threshold, double param1 = 0, double param2 = 0);
```

O primeiro argumento é a imagem de entrada, o segundo argumento é um ponteiro para o local onde os resultados serão armazenados, dentro de um objeto do tipo *CvMemoryStorage*, o terceiro argumento é o valor que indica qual variação da transformada de *Hough* é usada. Os argumentos *rho* e *theta* são a resolução, quantidade de *pixels*, entre as linhas. O valor do *threshold* faz com que uma linha seja retornada para a função se o valor do acumulador for maior que *threshold*. Para a transformada probabilista de Hough *Param1* é o tamanho mínimo da linha, *Param2* é o espaço mínimo que pode existir entre um seguimento de várias linhas para que elas sejam consideradas uma única linha.

A função *cvHoughLines2* retorna um vetor de objetos CvSeq, cada objeto CvSeq possui um vetor com dois objetos CvPoint indicando o início e o final da linha encontrada. O trecho de código abaixo exemplifica esta tarefa:

```
for (int i = 0; i < linhasHough->total; i++) {
        ponto = (CvPoint*) cvGetSeqElem(linhasHough, i);
        // desenha a linha encontrada
        cvLine(img,ponto[0],ponto[1],cvScalar(255,0,0), 5);
}
```

A função *cvGetSeqElem* extrai do vetor de objetos *CvSeq* o elemento inçado por i. Como o objeto *CvSeq* possui dois elementos *cvPoint* é usado a função cvLinha para desenhar, na imagem, uma linha entre estes dois pontos.

Para desconsiderar as linhas transversais o módulo de processamento esta tratando apenas as linhas que possuem uma inclinação superior a 30 graus. O valor de 30 graus é empírico, este valor foi escolhido porque é o valor que o maior número de acertos mesmo quando o veículo está em uma curva. O trecho de código abaixo mostra como foi tratado o ângulo da linha encontrada:

```
double x = (ponto[1].x-ponto[0].x);
double y = (ponto[0].y-ponto[1].y);
double t = y/x;
double angulo = atan(t);
angulo = 180*angulo/PI;    // rad -> grau
if (angulo > ANGULO) {
.....
```

Melhoramentos na localização da faixa na imagem

O fator do software realizar o processamento de 30 frames por segundo possibilita a utilização de uma média simples para detectar a posição da faixa na imagem, visto que existe pouca variação do ambiente neste espaço de tempo.

Para realizar o controle do fluxo de processamento o software usa uma constante chamada FRAME_MEDIA e um contador de frames. Sempre que o contador de frames for igual a FRAME_MEDIA o software realiza uma média das posições em que a faixa foi localizada. Caso a média for maior que 0 o software mostra a posição para o usuário do software.

```
int FRAME_MEDIA = 10;

if (contador >= FRAME_MEDIA) {
if (linhasEncontradas > 0) {
```

Para reduzir o erro, que pode ocorrer caso o software não localize a posição da faixa em uma quantidade muito grande de frames, é utilizado a constante MAX_FRAME_SEM_LINHA. A constante MAX_FRAME_SEM_LINHA “zera” o valor da variável que informa a posição da linha caso o software não localize a faixa dentro de uma quantidade de frames maior que o valor da constante MAX_FRAME_SEM_LINHA.

```
frameSemLinha ++;
if (frameSemLinha>=MAX_FRAME_SEM_LINHA) {
        uponto0x = 0;
        uponto0y = 0;
        uponto1x = 0;
        uponto1y = 0;
}
```

4.4.7. Segmentação e Representação por redes neurais

Além das funções internas a biblioteca *OpenCV* possui um conjunto de ferramentas para trabalhar com reconhecimento de objetos. As ferramentas podem ser encontradas na pasta *bin* do diretório de instalação do *OpenCV*. Para realizar o treinamento de uma rede é recomendável possuir amostras negativas e amostras positivas do objeto que se deseja localizar.

4.4.8. Amostras negativas

É recomendável ter um grande número de amostras positivas, ao menos 800. Amostras positivas são imagens do mundo real que não possuem uma instância do objeto a ser detectado. Para trabalhar com as amostras negativas é preciso criar um arquivo texto com o nome de cada imagem.

4.4.9. Amostras positivas

O tutorial do *OpenCV* recomenda o uso de pelo menos 4000 amostras positivas, as amostras positivas são diversas imagens do objeto que se deseja detectar. A ferramenta usada para treinar a rede requer um arquivo texto informando o nome do arquivo da amostra, a quantidade de instâncias do objeto que existe na amostra e a localização de cada instância.

A Figura 2-26 apresenta um arquivo que contêm uma lista de amostras positivas. O nome do primeiro arquivo da lista é *Img1.jpg*, ele possui duas instâncias de objetos e a posição destes objetos está definida onde é mostrado o comentário *(x1,y1,width1, height1)*.

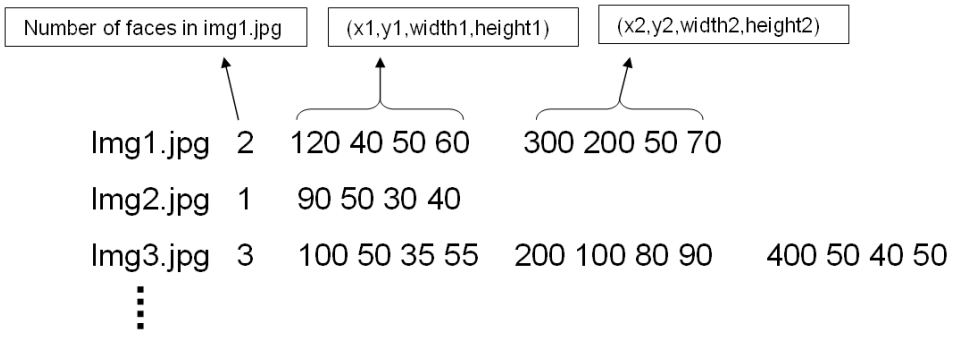


Figura 4-6: Arquivo com a lista de amostras positivas

Antes de iniciar o treinamento da rede o conjunto de amostras positivas deve ser compactada em um arquivo .vec, para realizar esta tarefa é usado a ferramenta createsamples.

createsamples -info info.txt -vec infovec.vec

onde *info.txt* é o nome do arquivo que contêm a lista das imagens positivas e *infovec.vec* é o nome do arquivo de saída.

Para realizar o treinamento da rede é usada a ferramenta *haartrainning* da seguinte forma:

Haartraining -data data-file -vec infovec.vec -bg negatives\neginfo.txt

onde *data-file* é o nome do arquivo que contêm o resultado do treinamento e *neginfo.txt* é o nome do arquivo que contêm a lista de amostras negativas.

A ferramenta *haartrainning* gera um arquivo *XML* com as informações do treinamento, o trecho de código seguinte mostra como este arquivo pode ser usado.

```
const char* cascade_name =    "data-file.xml";
static CvHaarClassifierCascade* cascade = 0;
cascade = (CvHaarClassifierCascade*)
        cvLoad( cascade_name, 0, 0, 0 );
CvSeq* faces = cvHaarDetectObjects( imagem, cascade, storage,
1.1, 2, 0, cvSize(30, 30) );
```

A função *cvHaarDetectObjects* retorna um conjunto de objetos *CvRect* que trata-se de um retângulo para cada objeto encontrado.

4.4.10. Interpretação

A interpretação consiste em tratar as informações geradas pela segmentação e representação a fim de obter o resultado final do processo, informar a posição atual do veículo e alertar o usuário caso o veículo inicie uma saída de pista à esquerda.

Após obter o valor do início de uma linha o sistema verificar se o início desta linha é maior que o valor do ponto limite definido pelo módulo de configuração. O ponto limite é o local, na imagem, que o usuário informou que a faixa é visualizada quando o veículo está iniciando uma saída de pista (está sobre a faixa). O trecho de código abaixo verifica se o *pontoEncontrado* (inicio da linha) é maior que o *pontoLimite* (valor definido pelo usuário), se o valor for verdadeiro o programa emite um alerta sonoro.

```
if (pontoEncontrado>pontoLimite) {
        Beep(300,100);
}
```

Se o inicio da faixa ainda não estiver sobre o pontoLimite o sistema apenas exibe sobre a imagem a distância que o veículo está da faixa.

```
sprintf(buffer, ">> %5.1fm ", distFaixa );
cvPutText(imagem,buffer,cvPoint(100,100,&font,cvScalar(200));
```

Toda a interpretação é mostrado na imagem do vídeo, portanto é necessário converter as coordenadas x_0, y_0 da imagem de interesse para as coordenadas *x1,* y1 da imagem real. Abaixo é mostrado como foi desenhada a linha encontrada sobre a imagem do vídeo.

```
cvLine(imagem,cvPoint(ponto[0].x+pontoX0, ponto[0].y+pontoY0),
cvPoint(ponto[1].x+pontoX0, ponto[1].y+pontoY0), cvScalar(255, 0, 0), 5);
```

onde *ponto[0].x* é a posição X_0 que do inicio da linha, *pontoX*$_0$ é o ponto, na horizontal, que a imagem de interesse foi recortada do vídeo. *Ponto[0].y* é a posição Y_0 do inicio da linha, *pontoY*$_0$ é a posição vertical que a imagem de interesse foi recortada.

O código do módulo de processamento pode ser visto no Apêndice B.

5. RESULTADOS EXPERIMENTAIS

Este capítulo apresenta como os testes foram realizados e os resultados encontrados.

5.1. Módulo de configuração

O módulo de configuração deve permitir que o usuário capture uma imagem da câmera de vídeo e possibilitar a seleção da posição que a faixa é visualizada. Todas as informações fornecidas pelo usuário são gravadas em um arquivo de configuração.

Os seguintes testes foram realizados no módulo de configuração:

Teste 1 – Entrada de dados

Tabela 5-1: Módulo de configuração – Teste 1

Valores selecionados		**Valores contidos no arquivo de configuração**
Item	**Valor**	
Posição da faixa quando o veículo está no centro da pista.	10 *pixels*	10
Posição da faixa quando o veículo está sobre a faixa	150 *pixels*	150
Valor entre a roda do veículo e a faixa	0.8 metros	0.8

Teste 2 – Reabertura do módulo para conferir os dados

Tabela 5-2: Módulo de configuração – Teste 2

Valores selecionados no Teste 1		**Valores visualizados**
Item	**Valor**	
Posição da faixa quando o veículo está no centro da pista.	10 *pixels*	10 *pixels*
Posição da faixa quando o veículo está sobre a faixa	150 *pixels*	150 *pixels*
Valor entre a roda do veículo e a faixa	0.8 metros	0.8 metros

Teste 3 – Alteração dos dados

Tabela 5-3: Módulo de configuração – Teste 1

Valores selecionados		**Valores contidos no arquivo de configuração**
Item	**Valor**	
Posição da faixa quando o veículo está no centro da pista.	25 *pixels*	25
Posição da faixa quando o veículo está sobre a faixa	200 *pixels*	200
Valor entre a roda do veículo e a faixa	1 metros	1

O módulo de configuração finalizou com sucesso todos os testes realizados. A captura da imagem de vídeo possibilita ao usuário marcar a posição que a faixa é visualizada e informar a distância entre a roda do veículo e a faixa da rodovia.

5.2. Detecção da faixa

Esta seção apresenta os resultados obtidos com a detecção da faixa na rodovia usando os algoritmos de Sobel e Hough. O esperado é que o software detecte corretamente a posição do veículo em relação a faixa no centro da estrada. Os seguintes testes foram realizados:

Teste 1 – Detecção da posição do veículo em uma rodovia em bom estado[2].

Configuração:

- A faixa é visualizada na imagem quando o veículo está sobre a mesma na posição 260 pixels.
- Quando o veículo está no centro da pista, a faixa é visualizado na imagem na posição 50 *pixels*.
- Quando o veículo está no centro da pista, a distância entre a roda esquerda e a faixa que divide a estrada é de 0,70 metros.
- A rodovia está com a faixa central em bom estado.

Tabela 5-4: Detecção da posição do veículo – Teste 1

Posição real entre a roda do veículo e a faixa (metros)	Posição da faixa na imagem (pixels)	Posição informada pelo software	Erro (metros)
1,0	401	1,54	0,54
0,9	294	1,13	0,23
0,8	402	1,55	0,75
0,7	258	0,99	0,29
0,6	293	1,13	0,53
0,5	86	0,33	-0,17
0,4	209	0,80	0,40
0,3	81	0,31	0,01
0,2	142	0,55	0,35
0,1	-12	-0,05	-0,15
0,0	73	0,28	0,28
-0,1	63	-0,15	0,05

[2] Rodovia em bom estado significa que a pista esta limpa e as faixas bem destacadas.

Teste 2 – Realizado nas mesmas condições que o teste 1.

Tabela 5-5: Detecção da posição do veículo – Teste 2

Posição real entre a roda do veículo e a faixa (metros)	Posição da faixa na imagem (pixels)	Posição informada pelo software	Erro (metros)
1	389	1,50	0,50
0,9	248	0,95	0,05
0,8	372	1,43	0,63
0,7	184	0,71	0,01
0,6	321	1,23	0,63
0,5	99	0,38	-0,12
0,4	247	0,95	0,55
0,3	99	0,38	0,08
0,2	172	0,66	0,46
0,1	-49	-0,19	-0,29
0	108	0,42	0,42

O software emitiu um alerta sonoro quando a roda do veículo estava sobre a faixa.

Teste 3 – Realizado nas mesmas condições que o teste 1.

Tabela 5-6: Detecção da posição do veículo – Teste 3

Posição real entre a roda do veículo e a faixa (metros)	Posição da faixa na imagem (pixels)	Posição informada pelo software	Erro (metros)
1	349	1,34	0,34
0,9	267	1,03	0,13
0,8	382	1,47	0,67
0,7	237	0,91	0,21
0,6	321	1,23	0,63
0,5	181	0,70	0,20
0,4	205	0,79	0,39
0,3	101	0,39	0,09
0,2	99	0,38	0,18
0,1	25	0,10	0,00
0	78	0,30	0,30

O software emitiu um alerta sonoro quando a roda do veículo estava sobre a faixa.

Os resultados apresentados nas Tabelas 5-4, 5-5 e 5-6 mostram que não existe um padrão de erros durante a detecção da posição do veículo. Este trabalho identificou os seguintes problemas quer geram a variação dos erros.

- Instabilidade da câmera causada pela trepidação.
- Pequenas variações na direção do veículo.
- Iluminação da rodovia.

Teste 4 – Detecção da posição do veículo em uma rodovia com as faixas desbotadas.

Configuração:

- A faixa é visualizada na imagem quando o veículo está sobre a mesma na posição 240 pixels.
- Quando o veículo está no centro da pista, a faixa é visualizado na imagem na posição 60 *pixels*.
- Quando o veículo está no centro da pista, a distância entre a roda esquerda e a faixa que divide a estrada é de 0,80 metros.
- A rodovia está com a faixa central em bom estado.

Tabela 5-7: Detecção da posição do veículo – Teste 4

Posição real entre a roda do veículo e a faixa (metros)	Posição da faixa na imagem (pixels)	Posição informada pelo software	Erro (metros)
1	Faixa não encontrada		-1,00
0,7	187	0,78	0,08
0,5	Faixa não encontrada		-0,50
0,3	Faixa não encontrada		-0,30
0	78	0,33	0,33

Teste 5 – Usando as mesmas configurações do teste 4.

Tabela 5-8: Detecção da posição do veículo – Teste 5

Posição real entre a roda do veículo e a faixa (metros)	Posição da faixa na imagem (pixels)	Posição informada pelo software	Erro (metros)
1	Faixa não encontrada		-1,00
0,7	Faixa não encontrada		-0,70
0,5	110	0,46	-0,04
0,3	Faixa não encontrada		-0,30
0,1	Faixa não encontrada		-0,10
0	98	0,41	0,41

Os resultados apresentados nas tabelas 5-7 e 5-8 mostram que o software falhou na detecção da faixa quando a mesma varia a sua posição, entretanto quando a faixa é localizada a posição informada não difere muito dos resultados dos testes 1 a 3.

Teste 6 – Resultados do teste com cálculo da posição média da faixa.

Os resultados apresentados na tabela 5-9 mostram que o software obteve melhores resultados usando o cálculo da posição através de uma média das posições encontradas.

Tabela 5-9: Detecção da posição do veículo – Teste 6

Posição real entre a roda do veículo e a faixa (metros)	**Posição da faixa na imagem (pixels)**	**Posição informada pelo software (metros)**	**Erro (%)**
1,0	81,0	0,8	16%
0,9	86,0	0,8	11%
0,8	72,0	0,9	14%
0,7	98,0	0,7	1%
0,6	112,0	0,6	1%
0,5	122,0	0,6	5%
0,4	138,0	0,5	9%
0,3	159,0	0,4	13%
0,2	154,0	0,4	24%
0,1	181,0	0,4	28%
0,0	172,0	0,4	40%

Neste conjunto de testes percebe-se que a melhor alternativa é a utilização de uma média da posição da faixa. Os testes mostram que é necessário considerar o erro informado pelo software antes de enviar um alerta sonoro para o motorista.

Neste teste foi usado o limite de 30cm entre a roda do veículo e a faixa da rodovia.

5.3. Redes neurais artificiais

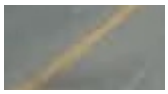
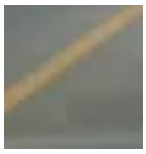
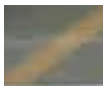
O treinamento da rede neural usou 4000 amostras positivas e 4000 amostras negativas, o treinamento da rede foi realizado usando o computador descrito na Tabela 5-10. O treinamento levou cerca de 30 horas.

Tabela 5-10: Configuração do computador para treinamento da RN

Marca / Modelo	HP Pavilion dv6433
Memória	2 GB
Processador	Intel Centrino Duo 1.7 GHz

O resultado final não foi satisfatório visto que o software não conseguiu reconhecer a pista na rodovia. A Tabela 5-11 apresenta algumas imagens que foram testadas durante o processo de reconhecimento.

Tabela 5-11: Imagens usadas para testar a RN

	Faixa não reconhecida
	Faixa não reconhecida
	Faixa não reconhecida
	Faixa não reconhecida

O principal problema encontrado para usar uma rede neural foi a pouca informação disponível acerca das ferramentas utilizadas pela biblioteca *OpenCV* bem como material didático abordando a utilização de redes neurais para o reconhecimento de imagens.

6. CONCLUSÃO

Com o desenvolvimento deste projeto foi possível reconhecer a posição da faixa na rodovia usando a detecção de bordas, mas não houve sucesso usando redes neurais artificiais. Após detectar a faixa na rodovia o software conseguiu informar a posição do veículo na rodovia.

O desenvolvimento de softwares para auxiliar a condução de veículos é uma área em expansão e possui grande potencial de crescimento. Este trabalho visou ser uma parte do estudo necessário para a criação de grandes projetos que envolvam visão computacional.

Por fim, este trabalho agregou conhecimentos acadêmicos e profissionais, e como conseqüência despertou o interesse à continuação dos estudos através de uma pós-graduação.

6.1. Sugestões para trabalhos futuros

Com base nos resultados obtidos podem ser listadas as seguintes sugestões para trabalhos futuros:

Evolução deste projeto:

- Usar outras técnicas para melhorar a imagem antes de processá-la.
- Usar um microcontrolador Atmel AP7000 ao invés de um computador pessoal para realizar o processamento.
- Um sistema para detecção das duas faixas da rodovia, a fim de evitar saída de pista para a direita e para a esquerda.
- Desenvolvimento de um sistema, baseado em visão computacional, para detectar veículos à frente com a finalidade de evitar colisões.
- Desenvolvimento de um sistema, baseado em visão computacional, para reconhecer placas de trânsito.

BIBLIOGRAFIA

[BMW 08] BMW Site. 2008. BMW LDW. *Acessado em 01*/08/2008, Disponível em http://www.bmw.com/com/en/newvehicles/5series/sedan/2007/allfacts/ergonomics/ldw.html.

[BRA 08] BRADSKI, Gary e KAEHLER, Adrian. 2008. *Learning OpenCV.* Sebastopol, CA: O'Reilly Media, Inc.,, 2008. ISBN: 978-0-596-51613-0.

[BRA 00] BRAGA, Antônio, CAVALHO, André e LUDERMIR, Teresa. 2000. *Redes neurais artificiais - Teoria e aplicações.* Rio de Jeneiro: LTC, 2000.

[BRI 08] Brigada Militar, 2008, Disponível em http://www.brigadamilitar.rs.gov.br/crbm/acidentes/perfil_acidente.asp, acessado em 10/10/2008.

[CAN 06] CANO, C. E. V. 2006. Técnica de Navegação de um Robô Móvel Baseado em um Sistema de Visão para Integrá-lo em uma Célula Flexível de Manufatura (FMC). Dissertação de Mestrado. Brasília: Universidade Federal de Brasília, 2006.

[FAC 02] FACON, Jacques. 2002. Processamento e Análise de Imagens. *PUC Paraná - Curso de Mestrado em Informática Aplicada. Acessado em* 21/10/2002 Disponível em http://www.ppgia.pucpr.br/~facon/CursoProcImagem.pdf.

[FIL 08] FILHO, Kepler de Souza Oliveira. 2008. Instituto de Física da UFRGS. Acessado em 08/10/2008, disponível em http://www.if.ufrgs.br/ast/med/imagens/.

[FIS 00] FISHER, Robert, et al. 2000. *Convolution. Acessado em* 10/10/2008, disponível em http://homepages.inf.ed.ac.uk/rbf/HIPR2/convolve.htm.

[GIG 08] GIGORO, Tom. *Canny Edge Detector Implementation. Acessado em* 20/10/2008, disponível em http://www.tomgibara.com/computer-vision/canny-edge-detector.

[GON 02] GONZALES, Rafael C e WOODS, Richard E. 2002. *Digital Image Processing.* New Jersey: Prentice-Hall, Inc., 2002. ISBN 0-201-18075-8.

[GRE 02] GREEN, Bill. 2002. Edge Detection Tutorial. Acessado em 06/08/2008, Disponível em http://www.pages.drexel.edu/~weg22/edge.html.

[HAY 01] HAYKIN, Simon. 2001. *Redes neurais: príncipio e prática.* [trad.] Paulo Martins Engel. 2 ed. Porto Alegre: Bookman, 2001.

[JAM 00] JAMUNDÁ, Teobaldo. 2000. Seminário Visão Computacional - CPGCC/UFSC - 2000.2. *Reconhecimento de Formas: A Transformada de Hough. Acessado em* 4/9/2008, Disponível em http://www.inf.ufsc.br/~visao/2000/Hough/.

[JES 08] JESAN, John Peter. 2008. *The neural approach to pattern recognition. Acessado em* 09/09/2008, Disponível em http://www.acm.org/ubiquity/views/v5i7_jesan.html.

[PAN 96] PANDYA, Abhijit S e MACY, Robert B. 1996. *Pattern Recognition with Neural Networks in C++.* s.l.: CRC Press, 1996.

[SAN 06] SANTANA, Alex. 2006. Os trechos da morte. *Associação Nacional dos Delegados da Polícia Federal. Acessado em* 13/08/2008, Disponível em http://www.adpf.org.br/modules/news/article.php?storyid=22124.

[TOY 08] Toyota Site, 2008. Site da Toyota (Japão), *Acessado em 01*/08/2008, Disponível em http://www.toyota.co.jp/en/tech/its/comfortable/mx.html.

[VRU 08] VRUM.COM.BR, 2008. Segurança ativa *Acessado em* 16/08/2008, Disponível em http://noticias.vrum.com.br/veiculos_nacional/template_interna_noticias,id_noticias=26606&id_sessoes=1/template_interna_noticias.shtml.

APÊNDICE A

```
// ========================================================
// MÓDULO DE CONFIGURAÇÃO
// ========================================================
@Action
public void salvar() {
salvar(true);
}

public void salvar(boolean msg) {
String novaLinha = System.getProperty("line.separator");
String pontoLimite = null;
String pontoCorreto = null;
String distanciaInformada = null;
int distanciaNaImagem = -1;
double fator = -1;

try {
BufferedWriter file = new BufferedWriter(new FileWriter(
ARQ_CONFIG));

pontoCorreto = pontoCorretoSlider.getValue() + novaLinha;
pontoLimite = pontoLimiteSlider.getValue() + novaLinha;
distanciaInformada = distanciaPontosText.getText();
distanciaNaImagem         =         (pontoLimiteSlider.getValue()         -
pontoCorretoSlider.getValue());

fator = Double.parseDouble(distanciaInformada)/distanciaNaImagem;

file.write(pontoCorreto);
file.write(pontoLimite);
file.write(distanciaInformada + novaLinha);
file.write(fator + novaLinha);
file.write(distanciaNaImagem + novaLinha);

file.close();
if (msg) {
JOptionPane.showMessageDialog(null, "Configurações salvas!");
}
} catch (Exception e) {
e.printStackTrace();
}}
public void carregaConfiguracao() {
// carrega a configuracão
String pontoCorreto = null;
String pontoLimite = null;
String distancia = null;
try {
BufferedReader file = new BufferedReader(new FileReader(
ARQ_CONFIG));
pontoCorreto = file.readLine();
pontoLimite = file.readLine();
distancia = file.readLine();
file.close();

pontoCorretoSlider.setValue(Integer.parseInt(pontoCorreto));
pontoLimiteSlider.setValue(Integer.parseInt(pontoLimite));
```

```
distanciaPontosText.setText(distancia);
} catch (Exception e) {
e.printStackTrace();
}}

@Action
public void carregaImagem() {
// carrega a imagem do recurso
org.jdesktop.application.ResourceMap              resourceMap              =
org.jdesktop.application.Application.getInstance(ldwcfg.LDWCfgApp.class).ge
tContext().getResourceMap(ConfigurarBox.class);
ImageIcon icone = new ImageIcon(ARQ_IMAGEM);
java.awt.Image aux = icone.getImage();
aux.flush();
imagemLabel.setIcon(new ImageIcon(aux));}
@Action
public void fechar() {
setVisible(false);}

@Action
public void executar() {
try {
Process p = Runtime.getRuntime().exec(ARQ_EXE);
salvar(false);
setVisible(false);
}catch                              (IOException                                ex)
{Logger.getLogger(ConfigurarBox.class.getName()).log(Level.SEVERE,    null,
ex);
}}
```

APÊNDICE B

```
// =========================================================
// MÓDULO DE PROCESSAMENTO
// =========================================================

#include <cv.h>
#include <highgui.h>
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <windows.h>
#include <time.h>

#include <iostream>

using namespace std;
int main(int argc, char** argv) {

int posicaoSobreFaixa;
int posicaoCentroPista;
float distanciaRodaFaixa;
float fatorPM;

FILE *fp;
if ((fp = fopen("c:/temp/test4/config/ldw.txt", "r")) == NULL) {
printf("Erro abrindo o arquivo de config!!");
exit(EXIT_FAILURE);
} else {
char buffer[100];
fgets(buffer, 100, fp);
posicaoSobreFaixa = atoi(buffer);
fgets(buffer, 100, fp);
posicaoCentroPista = atoi(buffer);
fgets(buffer, 100, fp);
distanciaRodaFaixa = atof(buffer);
fgets(buffer, 100, fp);
fatorPM = atof(buffer);
}

CvCapture *video;
//    video = cvCaptureFromCAM(0);
video = cvCaptureFromFile("C:/Temp/test4/teste-sobre-pista-1.avi");
if (!cvGrabFrame(video)) {
printf("Erro inicializando a fonte de vídeo!!");
exit(EXIT_FAILURE);
}

IplImage *frame;
IplImage *imagem;
IplImage *imagemCinza;
IplImage *imagemSobel;
IplImage *imagemHough;

int ALTURA = 350; // altura da regiao de interesse

CvRect retangulo = cvRect(0, ALTURA, 320, 100);
CvSeq *linhasHough;
```

```
CvPoint* ponto;
CvMemStorage *memoria = cvCreateMemStorage(0);

int contador = 0;
int iframe = 0;
int linhasEncontradas = 0;
int ANGULO = 10;              ///
int FRAME_MEDIA = 0;                ///
int MAX_FRAME_SEM_LINHA = 5;  //
int frameSemLinha = 0;
float LIMITE = 0.4;

int ponto0x = 0;
int ponto0y = 0;
int ponto1x = 0;
int ponto1y = 0;

int uponto0x = -1;
int uponto0y = -1;
int uponto1x = -1;
int uponto1y = -1;

CvFont font;
cvInitFont(&font, CV_FONT_HERSHEY_DUPLEX,1, 0.5, 0, 0, 1);

cvNamedWindow("1", 1);
//    cvNamedWindow("2", 1);
//    cvNamedWindow("3", 1);

for (;;) {
frame = cvQueryFrame(video);
if (!frame)
break;

contador++;
iframe++;

//
// define a area de interesse
//
imagem = cvCloneImage(frame);
cvSetImageROI(imagem, retangulo);

//
// cria uma imagem grayscale
//
imagemCinza = cvCreateImage(cvGetSize(imagem), IPL_DEPTH_8U, 1);
cvCvtColor(imagem, imagemCinza, CV_BGR2GRAY);

//
// cria uma imagem Sobel a partir da imagem grayscale
//
imagemSobel = cvCreateImage(cvGetSize(imagemCinza), imagemCinza->depth,
imagemCinza->nChannels);
cvSobel(imagemCinza, imagemSobel, 1, 1, 1);

//
// gera as linhas de hough e escreve na imagemHough
//
imagemHough = cvCreateImage(cvGetSize(imagem), imagem->depth,
imagem->nChannels);
```

```
linhasHough = cvHoughLines2(imagemSobel, memoria,
CV_HOUGH_PROBABILISTIC, 1, CV_PI/180, 50, 50, 10);

int i;
for (i = 0; i < MIN(linhasHough->total,5); i++) {
ponto = (CvPoint*) cvGetSeqElem(linhasHough, 0);
double x = (ponto[1].x - ponto[0].x);
double y = (ponto[0].y - ponto[1].y);
double t = y / x;
double angulo = atan(t);
angulo = 180 * angulo / CV_PI;

if ((angulo > ANGULO)) {
ponto0x += ponto[0].x;
ponto0y += ponto[0].y;
ponto1x += ponto[1].x;
ponto1y += ponto[1].y;
linhasEncontradas++;
}
}

char buffer[100];
sprintf(buffer, "C:/Temp/x/i%d.jpg", iframe);
cvSaveImage(buffer, imagem);

if (uponto0x > 0) {
cvLine(frame, cvPoint(uponto0x, uponto0y+ALTURA), cvPoint(uponto1x,
uponto1y+ALTURA), CV_RGB(255,0,0), 3, CV_AA, 0);


cvLine(frame,    cvPoint(uponto0x,    uponto0y+ALTURA),    cvPoint(uponto0x,
uponto0y+ALTURA), CV_RGB(0,255,0), 3, CV_AA,0 );

char buffer[100];
float tmp = uponto0x;
float distancia =  fatorPM/tmp;

sprintf(buffer, " << %f", fatorPM );
cvPutText(frame, buffer, cvPoint(1, 20), &font, CV_RGB(0,255,0));

sprintf(buffer, " << %f", tmp );
cvPutText(frame, buffer, cvPoint(1, 50), &font, CV_RGB(0,255,0));

sprintf(buffer, " << %f", distancia );
cvPutText(frame, buffer, cvPoint(1, 90), &font, CV_RGB(0,0,255));

printf("%f    %f \n", distancia, LIMITE);
if (distancia < LIMITE) {
Beep(70,70);
}


}

if (contador >= FRAME_MEDIA) {
if (linhasEncontradas > 0) {
ponto0x /= linhasEncontradas;
ponto0y /= linhasEncontradas;
ponto1x /= linhasEncontradas;
ponto1y /= linhasEncontradas;
```

```
uponto0x = ponto0x;
uponto0y = ponto0y;
uponto1x = ponto1x;
uponto1y = ponto1y;
} else {
frameSemLinha ++;
if (frameSemLinha>=MAX_FRAME_SEM_LINHA) {
frameSemLinha = 0;
uponto0x = 0;
uponto0y = 0;
uponto1x = 0;
uponto1y = 0;
}
}
ponto0x = 0;
ponto0y = 0;
ponto1x = 0;
ponto1y = 0;

contador = 0;
linhasEncontradas = 0;
}

//          char buffer[100];
//          sprintf(buffer, "C:/Temp/test2/frames/i%d.jpg", iframe);
//          cvSaveImage(buffer, imagem);


cvShowImage("1", frame);
//          cvShowImage("2", imagemSobel);
//          cvShowImage("3", imagem);

cvReleaseImage(&imagem);
cvReleaseImage(&imagemCinza);
cvReleaseImage(&imagemSobel);
cvReleaseImage(&imagemHough);
//          if (cvWaitKey(1) == 13) {
//                cvWaitKey(0);
//          }
if (cvWaitKey(1) == 27) {
break;
}
}
cout << "\nFim" << endl;
cvReleaseCapture(&video);
cvDestroyAllWindows();
return EXIT_SUCCESS;
}
// ************************************************
/// Usando redes neurais para identificar a image
#include <stdio.h>
#include <stdlib.h>
#include <iostream>
#include <math.h>
#include <windows.h>
#include <time.h>
#define PI 3.141592654;
int main(int argc, char** argv) {
/*  Lê o arquivo de configuração */
FILE *fp;
```

```
if ((fp = fopen("c:\\temp\\ldw\\ldw.cfg", "r")) == NULL) {
printf("Erro abrindo o arquivo de config!!");
exit(EXIT_FAILURE);
}
/** Área de interesse*/
int pontoX0 = 110; int pontoY0 = 370; int pontoX1 = 320; int pontoY1 = 440;
int largura = pontoX1 - pontoX0; int altura = pontoY1 - pontoY0;
/* * Ponto correto e ponto limite */
int pontoCorreto;
int pontoLimite;
double fator;
char lac[100];
double distLimiteFaixa;
fgets(lac, 100, fp); // melhor ponto
pontoCorreto = atoi(lac);
fgets(lac, 100, fp); // menor ponto
pontoLimite = atoi(lac);
fgets(lac, 100, fp); // pula
fgets(lac, 100, fp); // pixels / metro
fator = atof(lac);
/** Captura o video*/
CvCapture *video;
video = cvCaptureFromAVI("C:\\temp\\ldw\\m.avi");
//    video = cvCaptureFromCAM(0);
if (!cvGrabFrame(video)) {
printf("Erro abrindo o arquivo de video!!");
exit(EXIT_FAILURE);
}
/** vars*/
IplImage *frame; // frame
IplImage *imagem; // imagem gerada a partir do video
IplImage *imagemInteresse; // imagem com a area de interesse
IplImage *imagemCinza; // imagem com 1 canal, cinza
IplImage *imagemSobel; // bordas de sobel
CvRect retangulo; // um retangulo
CvMemStorage *memoria = cvCreateMemStorage(0); // memoria
CvSeq *linhasHough; // linhas de hough
CvPoint* ponto; // um ponto qualquer
int qtLinhas;
/* * Fontes */
CvFont font;
cvInitFont( &font, CV_FONT_HERSHEY_DUPLEX,1,0.5,0,0,1);
/**/
frame = cvQueryFrame(video);
cvNamedWindow("video1", 1);
int fanterior = 1;
int parar = 0;
for (;;) {
frame = cvQueryFrame(video);
if (!frame)
break;
/* * frame p/ imagem */
imagem = cvCloneImage(frame); // copia a imagem do frame
imagem->origin = 0; // como a imagem é formada -> topo-esquerda
cvFlip(imagem, imagem); // corrige a posição da imagem
/* * area de interesse */
retangulo = cvRect(pontoX0, pontoY0, largura, altura); // cria um obj
retangulo
imagemInteresse = cvCreateImage(cvGetSize(imagem), imagem->depth,
imagem->nChannels); // inicializa var
cvCopyImage(imagem, imagemInteresse); // faz uma cópia
```

```
cvSetImageROI(imagemInteresse, retangulo); // recorta
/* * converte a imagem de interesse para níveis de cinza */
//imagemCinza  =  cvCreateImage(cvGetSize(imagemInteresse),  imagem->depth,
      1); // faz uma copia com 1 canal
imagemCinza   =   cvCreateImage(cvGetSize(imagemInteresse),   IPL_DEPTH_8U,
      1); // faz uma copia com 1 canal
cvCvtColor(imagemInteresse, imagemCinza, CV_BGR2GRAY); // converte

rectangulo = cvImagemDetect(ARQUIVO_XML);
//
// neste projeto "retangulo" esta sempre retornando null,
// para maiores detalhes veja o capítulo sobre os resultados
// utilizando redes neurais
//


// ** fim ** //
```