CENTRO FEDERAL DE EDUCAÇÃO TECNOLÓGICA DE MINAS GERAIS

CAMPUS DIVINÓPOLIS

GRADUAÇÃO EM ENGENHARIA MECATRÔNICA

João Otávio Teixeira de Miranda

DIREÇÃO AUTÔNOMA DE UM VEÍCULO DIFERENCIAL DE PEQUENO PORTE UTILIZANDO VISÃO COMPUTACIONAL

Divinópolis.

2014.

João Otávio Teixeira de Miranda

DIREÇÃO AUTÔNOMA DE UM VEÍCULO DIFERENCIAL DE PEQUENO PORTE UTILIZANDO VISÃO COMPUTACIONAL

Monografia de Trabalho de Conclusão de Curso apresentada ao Colegiado de Graduação em Engenharia Mecatrônica como parte dos requisitos exigidos para a obtenção do título de Engenheiro Mecatrônico.

Áreas de integração: Mecânica, Programação e Eletrônica.

Orientador: Prof. Me. Alan Mendes Marotta

Divinópolis.

2014.

João Otávio Teixeira de Miranda

DIREÇÃO AUTÔNOMA DE UM VEÍCULO DE PEQUENO PORTE UTILIZANDO VISÃO COMPUTACIONAL

Monografia de Trabalho de Conclusão de Curso apresentada ao Colegiado de Graduação em Engenharia Mecatrônica como parte dos requisitos exigidos para a obtenção do título de Engenheiro Mecatrônico.

Áreas de integração: Mecânica (Tecnologia de Fabricação Mecânica), Programação (Visão Computacional, Linguagem C/C++, Comunicação Serial), e Eletrônica (Ponte H).

Comissão Avaliadora

_____

Prof. Me. Alan Mendes Marotta (Orientador)

Centro Federal de Educação Tecnológica de Minas Gerais - CEFET MG

_____

Prof. Dr. Renato de Sousa Dâmaso

Centro Federal de Educação Tecnológica de Minas Gerais - CEFET MG

_____

Prof. Dr. Luiz Claudio Oliveira

Centro Federal de Educação Tecnológica de Minas Gerais - CEFET MG

Divinópolis.

2014.

Dedico este trabalho aos meus pais, pela confiança e apoio.

Agradecimentos

A Deus por ter me dado luz, paciência e sabedoria ao longo do trabalho.

Aos meus pais, Marcus Teixeira e Letânia Marta Teixeira de Miranda, pela confiança, pelo apoio, pelo incentivo e pela qualidade de vida que foi me fornecida ao longo de minha vida.

Ao professor Alan Mendes Marotta pela orientação, pelos conhecimentos passados, pela paciência, pelo incentivo e pela contribuição que me foi fornecido ao longo do trabalho .

Aos colegas que me ajudaram e me incentivaram na realização do trabalho.

"Acredite em si próprio e chegará um dia em que os outros não terão outra escolha senão acreditar com você."

(Cynthia Kersey)

Resumo

A temática que será empregada durante este trabalho é o desenvolvimento de um veículo diferencial de pequeno porte com direção autônoma, utilizando visão computacional. Com esta tecnologia de automação em automóveis, o ser humano terá várias vantagens e benefícios, sendo estas a diminuição de acidentes proveniente de falhas humanas, o aumento da mobilidade para pessoas que possuem pouca ou nenhuma disponibilidade de tempo para outras atividades. Estes projetos já vêm sendo realizados no Brasil como CARINA 1 e CARINA 2 da Escola de Engenharia de São Carlos, ByWire XGV da Universidade Federal do Espírito Santo e CADU da Universidade Federal de Minas Gerais. Para se realizar este trabalho teve-se que projetar e produzir um veículo diferencial com pequenas dimensões para efetivar testes e verificar a eficiência da tecnologia de automação da direção por meio de visão computacional. Após construído o veículo, foi feito o desenvolvimento do *software* de visão computacional, sendo ele constituído por: uma parte responsável pela comunicação do *software* com os atuadores responsáveis pelo movimento do carrinho, uma parte de captura de imagens provindos pela webcam embarcada no carrinho, uma parte de segmentação e representação de regiões e dados importantes para identificação do caminho a ser percorrido, e uma parte de interpretação dos dados e de envio de comandos para a realização do correto direcionamento. Como meio de interpretação dos comandos enviados pelo *software* para os atuadores, foi utilizado o Arduino. Os resultados mostram que a utilização da tecnologia de visão computacional no controle da direção de um veículo diferencial pequeno é eficiente, conseguindo detectar a sinalização do caminho e se direcionar corretamente durante a sua movimentação.

Palavras-chave: Visão computacional. Direção autônoma. OpenCV. Veículo autônomo. Arduino.

Abstract

The theme that will be used for this work is the development of a small differential vehicle with autonomous direction, using computer vision. With this car in automation technology, the human has several advantages and benefits, which are to reduce accidents from human error, increase of mobility for persons who have little or no time available for other activities. These projects have already been carried out in Brazil as CARINA 1 and CARINA 2 of the School of Engineering of São Carlos, ByWire XGV the Federal University of Espírito Santo and CADU the Federal University of Minas Gerais. To perform this study was that design and produce a vehicle differential with small to effect tests and verify the efficiency of automation technology direction through computer vision. After built the vehicle, was made the development of computer vision *software*, it consisting of: one party responsible for *software* communication with the actuators responsible for the cart movement, an image capture part stemmed the webcam embedded in the cart, part of segmentation and representation of regions and important data to identify the way to go, and a part of interpretation of the data and sending commands to perform the correct direction. As a means of interpreting the commands sent by the *software* to the actuators, Arduino was used. The results show that the use of computer vision technology to control the direction of a small differential vehicle is efficient, achieving detect the signaling path and directing correctly when they are moving.

Keywords: Computer vision. Autonomous direction. OpenCV. Autonomous vehicle. Arduino.

Lista de ilustrações

Lista de tabelas

Lista de notações e acrônimos

Letras Latinas

$\mathrm{Km/h}$ - Quilômetro por hora

Km - Quilômetro

Bit - Binário

Byte - Unidade de informação digital equivalente a oito bits

G_σ - Operador gaussiano

G'_σ - Derivada do operador gaussiano

Hz - Unidade de frequência dada em Hetz

MHz - Mega Hetz (1.000 Hetz)

GHz - Giga Hetz (1.000.000 Hetz)

GB - Giga Bits (1.000.000 Bits)

V - Unidade de medida que representa Volts

rpm - Unidade de medida que representa rotação por minuto

A - Unidade de medida que representa corrente elétrica

kg/cm - Quilograma por centímetro

mm - Milímetros

h - Unidade de medida dada em hora

fps - Frame por segundo

Letras Gregas

σ - Desvio padrão

π - Valor correspondente à aproximadamente 3,1415.

e - Função exponencial

Acrônimos

EUA - Estados Unidos da América

CADU - Carro Autônomo Desenvolvido na Universidade Federal de Minas Gerais

USB - *Universal Serial Bus* (Porta Universal)

ONSV - Observatório Nacional de Segurança Viária

TCC - Trabalho de Conclusão de Curso

DARPA - *Defense Advanced Research Projects Agency* (Agência de Projetos de Pesquisa Avançada de Defesa)

CARINA - Carro Robótico Inteligente para Navegação Autônoma

EESC-USP - Escola Engenharia de São Carlos - Universidade de São Paulo

UFES - Universidade Federal do Espírito Santo

UFMG - Universidade Federal de Minas Gerais

GPS - *Global Positioning System* (Sistema de Posicionamento Global)

CC - Corrente Contínua

A/D - Analógico/Digital

RGB - *Red Green Blue* (Representação Vermelho Verde Azul)

HSV - *Hue Saturation Value* (Representação Tonalidade Saturação Valor)

OpenCV - *Open Source Computer Vision Library* (Biblioteca Aberta Recursos Visão Computacional)

PWM - *Pulse-Width Modulation* (modulação por largura de pulso)

ICSP - In-Circuit Serial Programming (Programação Serial No Circuito)

IDE - Integrated Development Environment (Ambiente Integrado de Desenvolvimento)

GND - *Ground* (Terra)

ASCII - *American Standard Code for Information Interchange* (Código Padrão Americano para o Intercâmbio de Informação)

DCB - *Device Control Block* (Bloco de Controle de Dispositivos)

Sumário

Capítulo 1- Introdução

Nesse capítulo será abordado a definição do problema que será tratado por este trabalho, a motivação por ter escolhido esse tema e a sua importância na atualidade, o objetivo geral e os específicos, e a síntese dos capítulos que serão apresentados posteriormente.

1.1. Definição do problema

Este Trabalho de Conclusão de Curso tem como objetivo a automação da direção de um veículo diferencial de pequeno porte, utilizando para isso recursos de visão computacional. Para realizar este trabalho foram integradas as seguintes áreas do curso de Engenharia Mecatrônica: eletrônica, computação e mecânica.

O tema do Trabalho de Conclusão do Curso surgiu com base no projeto desenvolvido na disciplina de "Introdução à prática experimental" que foi ministrada pelo professor Dr. Renato de Sousa Dâmaso, no segundo período do Curso de Engenharia Mecatrônica.

Esse projeto citado anteriormente consistiu em utilizar o trilho de ar do laboratório de física e melhorar a detecção do tempo que o carrinho gastava para se deslocar ao longo do trilho de ar. Para isso o grupo utilizou recursos de visão computacional no âmbito de obter o exato momento que o carrinho chegava nos pontos fixos no trilho.

1.2. Motivação

Com o desenvolvimento de tecnologias ao redor do mundo, o mercado consumidor tende a procurar mais inovação nos produtos. Um destes é o automóvel que vem sendo procurado desde a sua criação por ser um objeto que auxilia no

transporte de pessoas e de mercadorias. Como a procura pelo consumidor pela aquisição de veículos só vem aumentando com o passar do tempo, o fabricante vem melhorando as qualidades deste para que o mercado continue neste nível de procura. Um desses aprimoramentos que estão sendo estudados e desenvolvidos pelas empresas e por instituições de ensino e de pesquisa é a automação dos automóveis. Para que isso ocorra, um dos principais componentes do carro, que é a direção, deve passar por estes aperfeiçoamentos.

Para que o automóvel consiga se orientar sem nenhum auxílio humano, este deve identificar a trajetória e escolher o melhor caminho que deve ser adotado em cada situação. Visto isso, um dos meios que se pode utilizar como fonte de auxílio para sensoriamento e de interpretação de imagens é a visão computacional, que tem o objetivo de mostrar “que uma análise automática é possível... e um espectro de tarefas do processamento de informação visual podem ser compreendidos e automatizados” (TANIMOTO e KLINGER, 1980). Adicionando essa tecnologia nos automóveis atuais, produzindo um *software* que tome decisões de orientação com base nesta técnica, isto poderia facilitar o dia a dia das pessoas no mundo inteiro.

Este tipo de projeto, que pode ser visualizado na Figura 1.1, já vem sendo desenvolvido por algumas empresas e instituições (isso será citado na Seção 2.1 deste trabalho). Porém, mais complexo que o proposto neste trabalho. Uma dessas instituições é o exército dos Estados Unidos, que vem promovendo competições entre instituições e empresas (*Grand Challenge*). Isso é comentado por Neto (2007): “A principal intenção na organização do desafio foi estimular o desenvolvimento de veículos *off-road* autônomos, já que o objetivo do governo dos EUA é transformar um terço de sua frota de veículos militares em autônomos até 2015. Neste intuito, a competição foi aberta para universidades, colégios e empresas de dentro e fora dos EUA, além de serem permitidos veículos de todos os tipos para a competição”.

Figura 1.1 – CADU, o veículo autônomo mineiro (TECHTUDO, 2012).

Caso esta tecnologia de automação em automóveis fique disponível, o ser humano terá várias vantagens e benefícios importantes. Um desses benefícios seria com relação a diminuição de acidentes no trânsito devido às falhas humanas ou negligência humana.

Segundo notícia publicada no portal do DETRAN-MA (2013): "De acordo com uma pesquisa do Observatório Nacional de Segurança Viária (ONSV), divulgada em agosto de 2013, 98% dos acidentes de trânsito são causados por negligência ou erro humano. Dentre essas negligências, o consumo da bebida alcoólica combinada com a direção ocupa o segundo lugar no *ranking* das causas de morte por imprudência no trânsito, segundo a ONSV." Estes dados também podem ser observados na Tabela 1.1.

Tabela 1.1 – Números de acidentes por tipo e gravidade em 2011 (DPRF, 2011).

MINISTÉRIO DOS TRANSPORTES
DEPARTAMENTO NACIONAL DE INFRAESTRUTURA DE TRANSPORTES
DIRETORIA DE INFRAESTRUTURA RODOVIÁRIA
COORDENAÇÃO GERAL DE OPERAÇÕES RODOVIÁRIAS

MINISTÉRIO DA JUSTIÇA
DEPARTAMENTO DE POLÍCIA RODOVIÁRIA FEDERAL
COORDENAÇÃO GERAL DE OPERAÇÕES
COORDENAÇÃO DE CONTROLE OPERACIONAL

Quadro 0102 - NÚMERO DE ACIDENTES POR TIPO E GRAVIDADE

TOTAIS GERAIS - BRASIL **Ano de 2011**

TIPO DO ACIDENTE	DISTRIBUIÇÃO SEGUNDO A GRAVIDADE DO ACIDENTE				
	TOTAL	C/ Morto	C/ Ferido	S/ Vítima	Não Inf.
Choque com objeto fixo	14.699	327	4.190	9.905	277
Capotagem	7.352	317	3.849	2.995	191
Atropelamento	6.221	1.348	4.699	167	7
Atropelamento de animal	4.365	77	1.076	3.197	15
Choque com veiculo estacionado	546	13	83	447	3
Colisão traseira	54.999	590	11.691	42.688	30
Abalroamento no mesmo sentido	30.549	361	6.643	23.529	16
Colisão frontal	6.218	1.734	3.232	1.240	12
Abalroamento em sentido oposto	1.717	115	703	899	0
Abalroamento transversal	19.065	589	9.173	9.292	11
Tombamento	6.150	158	2.774	3.176	42
Saída de pista	24.933	665	8.967	14.844	457
Atropelamento e fuga	1.133	394	732	4	3
Queda de veiculo	5.927	237	5.327	322	41
Outros tipos	5.051	83	841	4.086	41
Total	188.925	7.008	63.980	116.791	1.146

Outro benefício para a população seria o aumento da mobilidade de algumas pessoas que possuem pouca ou nenhuma, como idosos e deficientes físicos. Isto também beneficiaria as empresas por acrescentar consumidores que antes não podiam usar estas mercadorias.

Outra vantagem seria o melhor aproveitamento do tempo que os condutores teriam, pois não teriam mais que ficar prestando atenção na direção e no ambiente em volta, fazendo com que as pessoas obtenham mais tempo em outras atividades. Como atualmente as pessoas vêm necessitando de meios para otimizar o uso do tempo, este seria de grande utilidade principalmente nas cidades onde o trânsito é muito conturbado e denso. Essas vantagens foram abordadas em uma notícia da Cayres (2013).

1.3. Objetivos do trabalho

Objetivo geral:

Este TCC tem como finalidade produzir e testar um sistema de direção autônoma que possa ser utilizado em veículos com direção diferencial de pequeno porte, utilizando para isso os recursos de visão computacional.

Objetivos específicos:

- Coletar e sistematizar informações bibliográficas acerca do tema abordado.

- Desenvolver um *software* que capture e analise o caminho que o automóvel trafega, e assim tome decisões autônomas sobre que direção se deve adotar.

- Construir um veículo em pequenas dimensões, compondo em si todos os acessórios para o sistema de visão computacional.

- Projetar e desenvolver uma comunicação do *software* com os motores do veículo, utilizando um Arduino Uno como meio de interlocução.

- Aplicar o *software* e a comunicação no veículo.

- Realizar testes com a finalidade de obter resultados com relação à estrutura do carrinho, ao *software* de visão computacional e à comunicação do *software* com o Arduino Uno.

1.4. Síntese dos capítulos posteriores

Este trabalho está organizado em 6 partes, sendo elas organizadas pela seguinte ordem:

- Capítulo 2: apresenta a revisão de literatura do trabalho, como também os principais fundamentos teóricos necessários para a compreensão deste trabalho.

- Capítulo 3: apresenta toda a parte de metodologia empregada para a realização deste trabalho, detalhando os meios e modos que foram utilizados para realizá-lo, desde a construção do veículo de pequeno porte até a programação do *software* de visão computacional.

- Capítulo 4: mostra e discute os resultados dos testes realizados para verificar o cumprimento dos objetivos citados anteriormente.

- Capítulo 5: aborda as conclusões que foram alcançadas ao final do trabalho.

- Capítulo 6: propõem trabalhos futuros com base nos resultados e conclusões encontradas no final do trabalho.

Capítulo 2: Fundamentos

Neste capítulo serão abordados a revisão da literatura, que foi base para o desenvolvimento do trabalho proposto, e também os principais fundamentos teóricos necessários para a compreensão e a realização do trabalho.

2.1. Revisão da literatura

O começo da robótica se destacou pelo surgimento da robótica industrial, tendo suas maiores aplicações com robôs manipuladores. Estes "têm a característica particular de serem fixos e grandes e só podem realizar tarefas de manipulação, sendo classificados como máquinas ferramentas" (DA SILVA, 2003).

A partir destes conceitos, o ser humano começou a procurar meios para fazer estes robôs se locomoverem, pois para certos tipos de trabalho, necessitava de mais versatilidade nos processos. Com essa necessidade, surgiram os robôs móveis que têm a função de poder se locomoverem em lugares de difícil acesso ao ser humano (por limitações humanas ou geográficas). Estes tinham a capacidade de operação autônoma com a execução de *software*s de alto nível que permitiam a esses dispositivos locomoverem e realizarem a percepção do ambiente através de seus sensores.

Com o passar do tempo, o potencial interesse e crescimento da robótica móvel resultaram na geração das classificações pelas formas de locomoção e de navegação. Conforme dito por Dudek e Jenkin (2000), os tipos de locomoção podem ser classificadas em quatro: terrestres, aquáticos, aéreos e espaciais, enquanto que de acordo com as formas de navegação, podem ser classificadas de autônomos e semiautônomos.

Este tipo de sistema tem sido assunto de pesquisas e projetos por mais de quarenta anos. Os primeiros grupos de pesquisa surgiram na universidade de Stanford, e, como foi dito por Da Silva (2003), foram nestes grupos que se

desenvolveu o primeiro robô móvel no início dos anos sessenta e foi chamado de Shakey, conforme mostra a Figura 2.1. Este robô estava equipado com uma rede sensorial composta por uma câmera e sensores de torque que lhe capacitava a se locomover sobre uma trajetória reta numa superfície plana.

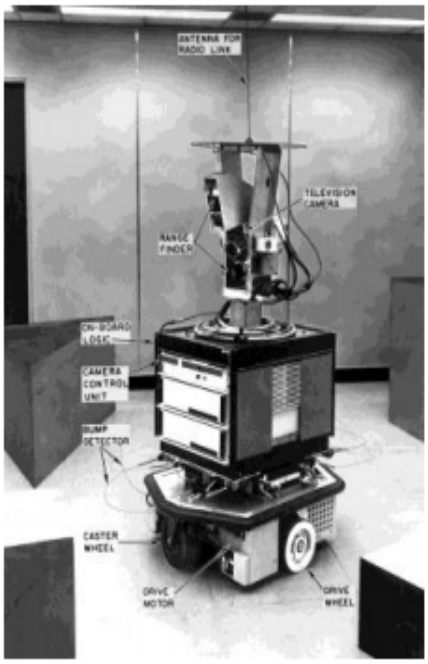


Figura 2.1 - Robô Shakey da Universidade Stanford (SHAKEY, 1968).

Segundo Gomes (2010), em 1977 no Japão, cientistas do Laboratório de Engenharia Mecânica de Tsukuba desenvolveram e construíram o primeiro veículo autônomo que podia se locomover a uma velocidade de até 30 km/h seguindo o caminho claramente marcado.

Três anos mais tarde, Ernst Dickmanns da Universidade Bundeswehr de Munique desenvolveu um automóvel da Mercedes Benz equipado com câmeras e sensores, que conseguia chegar a uma velocidade de 98 km/h seguindo as faixas brancas situadas na estrada (DICKMANNS, 2004).

Com o desenvolvimento e resultados nesta área, várias empresas e instituições começaram a estudar e projetar mais aprofundadamente sobre esse tema.

Na década de 90, a Comissão Européia fundou o programa *Eureka Prometheus Project*, com o único intuito de projetar carros autônomos. Dois projetos deste programa atingiram os melhores resultados, sendo eles os veículos gêmeos Vamp e Vita e o veículo ARGO.

Os veículos gêmeos Vamp e Vita foram desenvolvidos pelo grupo de investigação da Universidade Bundeswehr de Munique com a orientação do professor Ernst Dickmanns. De acordo com Dickmanns (2004) estes veículos apresentaram um ótimo desempenho em uma via pública de três faixas, pois tinham a capacidade de mudar de faixa, capacidade de detectar outros automóveis e de atingir a velocidade de 130 km/h.

Já o veículo ARGO produzido pela Universidade de Parma na Itália mostrado na Figura 2.2, apesar de algumas características semelhantes aos veículos gêmeos, alcançou uma velocidade média menor, 90 km/h, atingindo 94% da rota planejada. Contudo, como foi mencionado por Broggi *et al.* (1999), este veículo apresentava um sistema equipado com apenas duas câmeras de vídeo em preto e branco e algoritmos de visão estereoscópica utilizado para seguir seu caminho.

Figura 2.2 - Projeto do Veículo ARGO (GINGICHASHVILI, 2007).

Contudo, foi a partir do ano 2000 que a área de automação veicular começou a se destacar pelas parcerias entre as instituições de ensino e as indústrias. Uns dos eventos que ajudaram a ocorrer esse destaque foram os promovidos pela agência americana de investigação do departamento de defesa dos EUA, DARPA *Grand Challenge*. Como foi citado por Ferreira (2012), esse evento tinha como finalidade criar e desenvolver tecnologias para fins militares, no intuito de todas as tarefas

perigosas serem realizadas por uma máquina em vez de um humano, de modo a proteger os soldados e otimizar as ações.

Na sua primeira versão, o objetivo foi a conclusão total de um percurso de 228,5 km de comprimento sem obstáculo situado no deserto Mojave nos EUA. Nesta edição, nenhuma das equipes participantes conseguiu completar a prova, e o melhor resultado percorreu somente 11,84 km. Já na segunda edição, das vinte e três equipes participantes na etapa final, cinco das equipes terminaram a prova e apenas uma entre todas as equipes participantes não conseguiu ultrapassar a melhor marca da edição anterior.

No ano de 2007, a DARPA organizou um outro evento chamando de *Urban Challenger* (Desafio Urbano), que consistia que as equipes deveriam projetar e construir um veículo autônomo capaz de se locomover em trânsito congestionado, e efetuar algumas manobras como estacionar e ultrapassar.

Algumas destas tecnologias desenvolvidas nos projetos e estudos citados anteriormente, já estão disponíveis em alguns carros produzidos atualmente. Entre elas, podem-se encontrar os sistemas para auxílio de estacionamento, sistema para controle de velocidade e de sistemas de alerta de colisão.

Atualmente, no Brasil encontram-se instituições educacionais pesquisando sobre a área de veículo autônomo e obtendo resultados muito satisfatórios.

Entre os vários projetos produzidos no Brasil vale destacar: CARINA 1 e CARINA 2 da Escola de Engenharia de São Carlos (EESC-USP), ByWire XGV da Universidade Federal do Espírito Santo (UFES) e CADU da Universidade Federal de Minas Gerais (UFMG).

Como foi dito por Fernandes *et al.* (2012), CARINA 1 é um pequeno veículo elétrico do tipo de carro de golfe produzido pelo Grupo de Robótica Móvel do Laboratório de Mecatrônica e que tem o acelerador, o freio e a direção modificados para poderem ser controlados por computador.

Já o CARINA 2, como é comentado por De Oliveira (2013), é um Fiat Adventure, desenvolvido pela mesma equipe que projetou o CARINA 1, e nele possui total automação da direção, do freio e do acelerador. O veículo contém computadores embarcados, GPS, câmeras e *lasers*. Estes são usados para

localização, direção, detecção de obstáculos e tomadas de decisões. O CARINA 2 foi o primeiro veículo autônomo da América Latina a trafegar em vias públicas.

O ByWire XGV, descrito por Dias (2013) é um Ford Escape Híbrido comercialmente preparado para receber comandos de controle por computador. Ele foi obtido pelo Laboratório de Computação de Alto Desempenho e foi desenvolvido nele um sistema de inteligência artificial que possibilita que o veículo seja capaz de se guiar sozinho, com ajuda somente de câmeras e de sensores de *lasers*.

Já o CADU, mencionado por De Oliveira (2013) é um Chevrolet Astra Sedan 2.4 e nele foi feita a total instalação da parte de automação pelo Grupo de Pesquisa e Desenvolvimento de Veículos Autônomos. Ele possui automação nos componentes de atuação e sensoriamento. Ele também vem sendo testado somente dentro do campus por ser um veículo doado e não apresentar número de chassis.

2.2. Fundamentação teórica

Nesta seção serão abordados conceitos básicos para a compreensão e a realização do trabalho.

2.2.1. Ponte H

Resumidamente um circuito de ponte H consiste em quatro chaves, mecânicas ou eletrônicas, situadas numa forma que assemelha a letra "H", como pode ser vista na Figura 2.3. A função desse circuito é voltada, principalmente, ao acionamento de motores CC, de forma a proporcionar a alimentação do motor e permitir o controle de seu sentido.

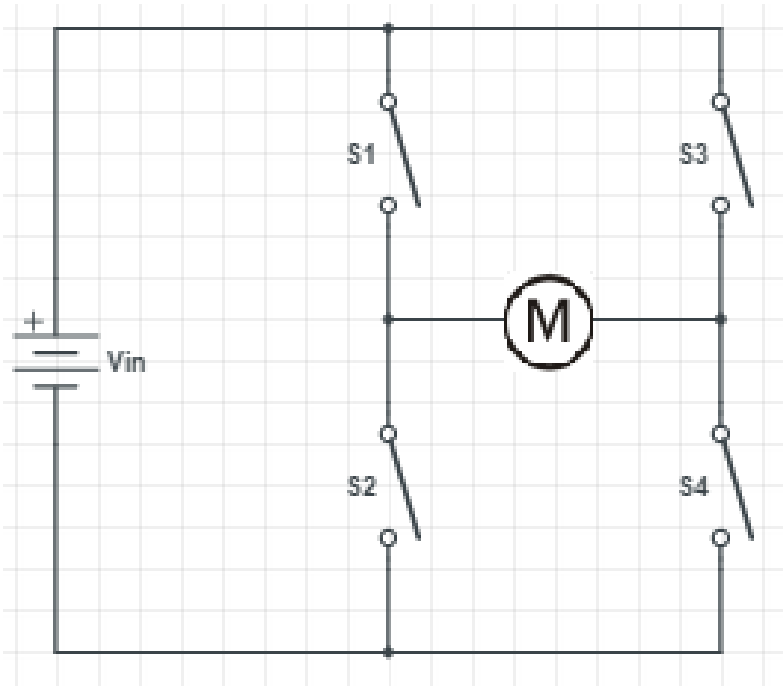


Figura 2.3 - Modelo de circuito Ponte H (ELETROMANÍACOS, 2014).

As chaves ficam localizadas em cada um dos extremos e são acionadas par a par, interligando a fonte de tensão (pólo positivo) ao motor e este ao terra (pólo negativo). Para operar o motor, é necessário acionar um par de chaves diagonalmente opostas, e dependendo do par escolhido, pode-se estabelecer o sentido de rotação do motor.

Se as duas chaves superiores ou as inferiores forem acionadas simultaneamente, o circuito fará com que o motor freie. Isso se deve pelo fato do motor, como todo componente indutivo, gerar uma tensão entre os seus terminais. Quando os terminais do motor são conectados ao mesmo pólo, ocorre um “curto-circuito” no motor, e a tensão gerada por ele força-o a girar na direção oposta, fazendo-o parar instantaneamente. Entretanto, se todas as chaves forem abertas, o circuito será desligado e caso o motor estivesse girando, ele pararia suavemente. Quando as chaves S1 e S2 ou S3 e S4 são fechadas simultaneamente produzem um curto circuito da fonte de alimentação, sendo essas condições proibidas.

2.2.2. Visão Computacional

O processamento de imagens é a área da computação que se refere à manipulação e operações com imagens. Ao processamento e transformação de dados de imagens aplicado à computação de forma inteligente, interpretativa, de modo a extrair informações, novas representações e/ou conclusões a partir dos dados da imagem, dá-se o nome de visão computacional. Esta definição foi mencionada por Forsyth e Ponce (2003).

Imagens são consideradas sinais bidimensionais presentes no domínio espacial (que podem ou não variar no tempo). Como qualquer sinal, computacionalmente é impossível representar o sinal análogo, que é infinito. Já a imagem digital pode ser representada. Estas, segundo Gonzales e Woods (2008), podem ser definidas como funções bidimensionais compostas por um conjunto de elementos finitos, chamados de *pixels*, que possuem localização e valor particulares.

Quando uma fotografia é obtida através de uma câmera digital, em algum momento, a informação do ambiente foi transferida do formato analógico para o formato digital. O sinal analógico, ou a luz que incide através do obturador da câmera, precisa ser convertido em uma sequência de *bits* que possa ser lida por um computador. O mesmo processo é necessário quando uma fotografia revelada de um filme de câmera analógica é digitalizada. Esse processo envolve duas etapas principais, presentes em qualquer conversão A/D (analógico para digital) em processamento de sinais: a amostragem e a quantização, como citado por Haykin e Van Venn (2002).

Na amostragem, o sinal contínuo obtido, no caso a luz, deve ser discretizado. Esta discretização consiste da divisão do sinal em um número finito de valores computacionalmente viáveis. Em sinais de imagem, este é o processo que define a resolução da imagem resultante, e portanto a quantidade e valor dos *pixels* na matriz.

A etapa seguinte é a quantização, que envolve a aproximação da faixa contínua de valores obtidos do sinal analógico para um conjunto, ou uma escala, de valores finitos. Esse processo determina as cores presentes na imagem digital

resultante, que é o valor individual de cada *pixel.* Existem diversos sistemas que regem a representação do valor do *pixels*, dois dos mais usados são o RGB (representação na escala das três cores primárias: vermelho, verde e azul) e o HSV, que divide o valor do *pixel* nas componentes: tonalidade (ou *hue*), saturação (*saturation*) e valor (*value*).

Quando uma câmera ou um dispositivo está gravando um vídeo, na verdade ele está fotografando uma grande variedade de imagens em sequência (*frames*). Quando se exibe as imagens em ordem e em uma certa velocidade, estas irão infringir a sensação de ser um vídeo que está se exibindo.

2.2.3. Processamento de Imagem

O método de processamento digital de imagens pode ser classificado, quanto ao grau de abstração, em três níveis distintos: baixo, médio e alto, como pode ser visto na Figura 2.4 (FALCON, 2005). Entre cada mudança de nível no sentido crescente de abstração, há uma redução progressiva da quantidade de informações manipuladas.

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 aos *pixels*.

No processamento de nível médio, este conjunto de valores produz como resultado uma lista de características.

No processamento de alto nível, o resultado, obtido pela lista de características, é uma interpretação do conteúdo da imagem.

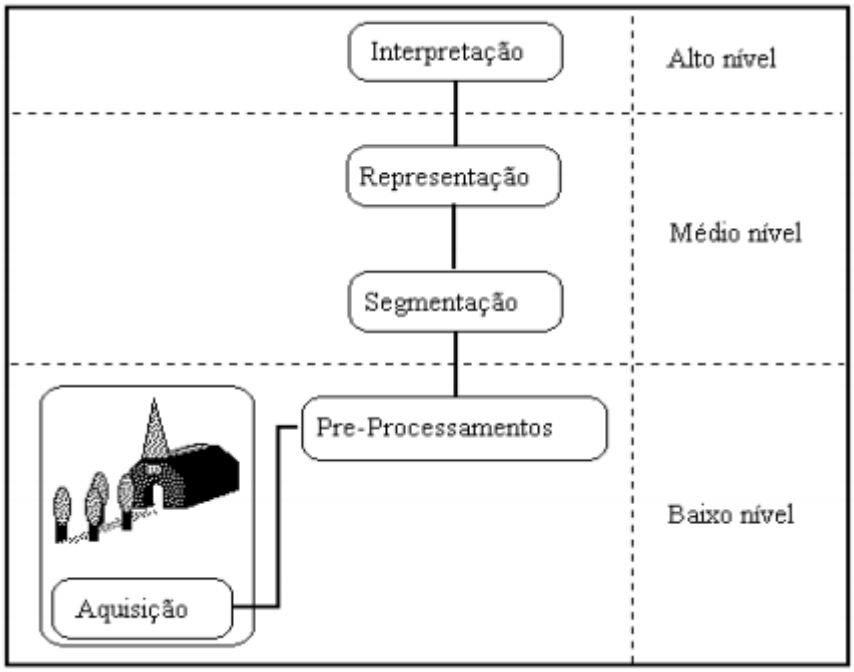


Figura 2.4 - Estrutura funcional de um sistema de visão artificial (FACON, 2005).

Aquisição: A imagem é obtida pelo dispositivo de captura de imagem, e transformada em uma matriz de valores discretos inteiros chamados *pixels*.

Pre-processamento: Etapa que tem o objetivo de corrigir defeitos e imperfeições provindos por características físicas do sistema, condições de iluminação, entre outros.

Segmentação: Consiste em dividir uma imagem em partes constitutivas, efetuada pela detecção de descontinuidades (contornos) ou áreas similares (regiões) na imagem.

Representação: Objetiva a elaboração de uma estrutura adequada, agrupando resultados das etapas precedentes e armazenando nos diversos padrões que contém o conhecimento a priori.

Interpretação: Permite obter a compreensão e a descrição final da operação com relação aos resultados obtidos nos níveis inferiores (aquisição, pre-processamento, segmentação e representação).

2.2.4. Segmentação de imagem por bordas

Bordas em uma imagem são áreas com forte contraste 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 (DE SALES, 2008).

A maioria das técnicas de detecção de bordas empregam operadores diferenciais de primeira ou de segunda ordem. Os operadores diferenciais ressaltam os contornos das bordas mas também amplificam o ruído da cena. Grande parte dos operadores de borda utilizam algum tipo de suavização da imagem antes da operação diferencial.

Operador de Canny

O operador de Canny, escolhido como método de detecção de bordas neste trabalho, é um método correspondente a um aperfeiçoamento do algoritmo de detecção de bordas feito por J. Canny. Ele é um operador gaussiano de primeira derivada que suaviza os ruídos e localiza as bordas.

O Operador de Canny trabalha com base em três critérios ótimos para localização de contornos: mínima probabilidade de detecção de múltiplas bordas; boa localização, isto é, mínima possibilidade de erro na detecção dos pontos pertencentes à borda verdadeira; e detecção de uma única borda, ou seja, se houver duas respostas uma delas é considerada falsa (SILVA E ALVES, 2008).

Para a utilização do operador de Canny, é preciso passar por quatro etapas importantes:

1. Suavização com o filtro gaussiano;
2. Computação do gradiente;
3. Não eliminação da máxima amplitude do gradiente;
4. Limiarização.

Na Equação 2.1 e na Equação 2.2 é mostrado, respectivamente, o operador gaussiano e sua respectiva derivada em uma dimensão:

$$G_\sigma(x) = \frac{1}{\sqrt{2\pi\sigma}} e^{x^2/2_{\sigma^2}} \quad (2.1)$$

$$G_\sigma^{'}(x) = \left(-\frac{x}{\sigma^2}\right) \frac{1}{\sqrt{2\pi\sigma}} e^{-x^2/2_{\sigma^2}} \quad (2.2)$$

Sendo x representando as coordenadas polares e σ representando o desvio padrão.

A suavização é realizada pela convolução do operador gaussiano com o sinal de entrada, conseguindo-se a redução ou atenuação das altas frequências existentes na imagem. O filtro gaussiano tem a vantagem de atuar nas frequências da imagem de uma forma suave.

Após a etapa de suavização, é feito uma diferenciação seguidamente com uma convolução do filtro gaussiano com a imagem de entrada, onde são determinados os gradientes de cada região e as respectivas bordas nas direções horizontal ou vertical mesmo na presença de ruídos.

Depois da etapa citada no parágrafo acima, é realizada a localização do gradiente de maior amplitude da imagem, com a finalidade de minimizar o número de indesejáveis bordas da região. Esse procedimento é feito por uma técnica chamada de “não máxima supressão”, que atua reduzindo a espessura dos contornos de um *pixel*.

Por último vem a etapa de limiarização. Os detectores de bordas, geralmente, agem utilizando apenas um limiar T qualquer, entretanto o operador de Canny utiliza dois limiares, um de alta e um de baixa. Caso o valor de uma borda ultrapasse este limite de alta este é imediatamente aceito, enquanto que com valor menor que o limiar de baixa o mesmo é rejeitado. Para os valores de bordas situados entre os dois limiares serão aceitos os *pixels* que estiverem conectados de forma a se obter uma boa resposta. A vantagem deste procedimento é que mais pontos são conectados facilitando a determinação de contornos e dando maior consistência aos resultados obtidos com a técnica (SILVA e ALVES, 2008).

Como ao segmentar uma imagem, a mesma fica dividida em regiões (R), com características comuns, as máscaras ao percorrerem os *pixels* destas regiões

operam com valores de uma e de outra. Nos pontos localizados no limite das regiões o valor de R é diferente de zero. Com base neste princípio, os contornos ou bordas ficam delineados pelo operador (SILVA e ALVES, 2008).

2.2.5. 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 do máximo local do acumulador, quais parâmetros são correspondentes à forma especificada (DE SALES, 2008).

Há várias parametrizações possíveis para o espaço de linhas. Hough usou a equação *declive-interpecte*, definida por y= ax+b, como a representação paramétrica de uma linha.

Para iniciar o processo é necessária uma matriz acumuladora que corresponde aos valores "quantizados" para "a" e "b". 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*.

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.

Tendo essa 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 os máximos valores 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.

2.2.6. Biblioteca OpenCV

A OpenCV (*Open Source Computer Vision*) é uma biblioteca de programação multiplataforma de código aberto. Ela foi desenvolvida pela Intel em 2000 e é totalmente livre ao uso acadêmico e comercial para desenvolvimento de aplicativos na área de Visão Computacional. A OpenCV possui mais de 350 algoritmos de Visão Computacional como: filtros de imagem, calibração de câmera, reconhecimento de objetos, análise estrutural e outros. O seu processamento é em tempo real de imagens. Esta biblioteca foi desenvolvida nas linguagens de programação C/C++.

Um dos objetivos da OpenCV é fornecer uma infra-estrutura simples de usar visão de computador, que ajuda as pessoas a construir aplicações de visão bastante sofisticados rapidamente (BRADSKI e KAEBLER, 2008).

2.2.7. Kit Arduino Uno

O Arduino é uma placa microcontroladora bem como uma plataforma de prototipagem eletrônica e está presente em diversos tipos de aplicações como: robótica, segurança, arte, entre outras.

O principal objetivo dessa placa é ser usada para o desenvolvimento de objetos interativos independentes, o Arduino em si não possui acesso a rede, porém no mercado existem diversos tipos de *Shields* que complementam o Arduino.

O Arduino Uno tem como base o chip ATmega328, possui 14 entradas/saídas digitais (das quais 6 podem ser usadas como saídas PWM), 6

entradas analógicas, um cristal oscilador de 16 MHz, conexão USB, uma entrada para fonte, soquetes para ICSP e um botão de reset.

O Arduino Uno pode ser programado com o *software* Arduino, no qual utiliza uma linguagem que se assemelha ao C/C++. O ATmega328 no Arduino Uno vem pré-gravado com um *bootloader* que permite ao usuário enviar código novo para ele sem a utilização de um programador de *hardware* externo. Ele se comunica utilizando o protocolo original STK500.

Capítulo 3: Desenvolvimento do projeto

Nesse capítulo será relatado a parte da metodologia desenvolvida para a realização do trabalho, desde a parte mecânica e eletrônica do projeto até a parte de *software* para a aquisição e tratamento da visão computacional, como também o envio do controle do carro autônomo.

3.1. Parte mecânica e eletrônica

A princípio, foi feita a montagem e fixação das rodas no chassi do carrinho, duas rodas na parte de trás e uma na parte da frente. Esse chassi é constituído por uma placa de alumínio com dimensões que serão mostradas na Figura 3.3. As duas rodas presentes na traseira do carro, estarão se comunicando diretamente por dois motores de corrente contínua com capacidade de no máximo 12V e de 30rpm. Neles estão presentes dois redutores que farão com que o carro consiga carregar toda a parte estrutural, eletrônica e de *software* (neste caso o *netbook* que estará embarcado no carrinho projetado). Os dados técnicos dos motores com redução estão presentes na Tabela 3.1:

Tabela 3.1 – Dados técnicos do motor CC com redução.

Dados Técnicos	
Tensão de funcionamento	**12 V**
Velocidade de rotação	**30 rpm**
Corrente máxima sem carga	**0,1 A**
Corrente máxima com carga	**0,5 A**
Torque avaliado	**3,5 kg/cm**
Razão de redução	**1/150**

Para fixar os motores no chassi, previamente foi feito o processo de furação do chassi e nas duas cantoneiras de alumínio utilizando uma furadeira de bancada com a broca de 2,5 mm. No chassi do carrinho foram feitos dois furos na parte de trás do carrinho no lado esquerdo e direito, e nas cantoneiras foram feitos quatro furos, sendo dois em cada aba. Na parte frontal do chassi foram feitos mais dois furos para fixação da roda não atuada de apoio. Após a realização deste processo, foi aberto mais um furo em cada cantoneira para poder passar o eixo do motor, sendo neste caso com uma broca de 8mm.

Feitas as furações, foram fixados os motores nas cantoneiras e as cantoneiras no chassi do veículo, utilizando parafusos de 2mm com suas respectivas porcas. Logo após foi feita a fixação das rodas nos eixos dos motores de corrente contínua. A montagem desta parte mecânica pode ser vista na Figura 3.1 e na Figura 3.2.

Figura 3.1 – Montagem mecânica da parte inferior do veículo (vista superior).

Figura 3.2 – Montagem mecânica da parte inferior do veículo (vista inferior).

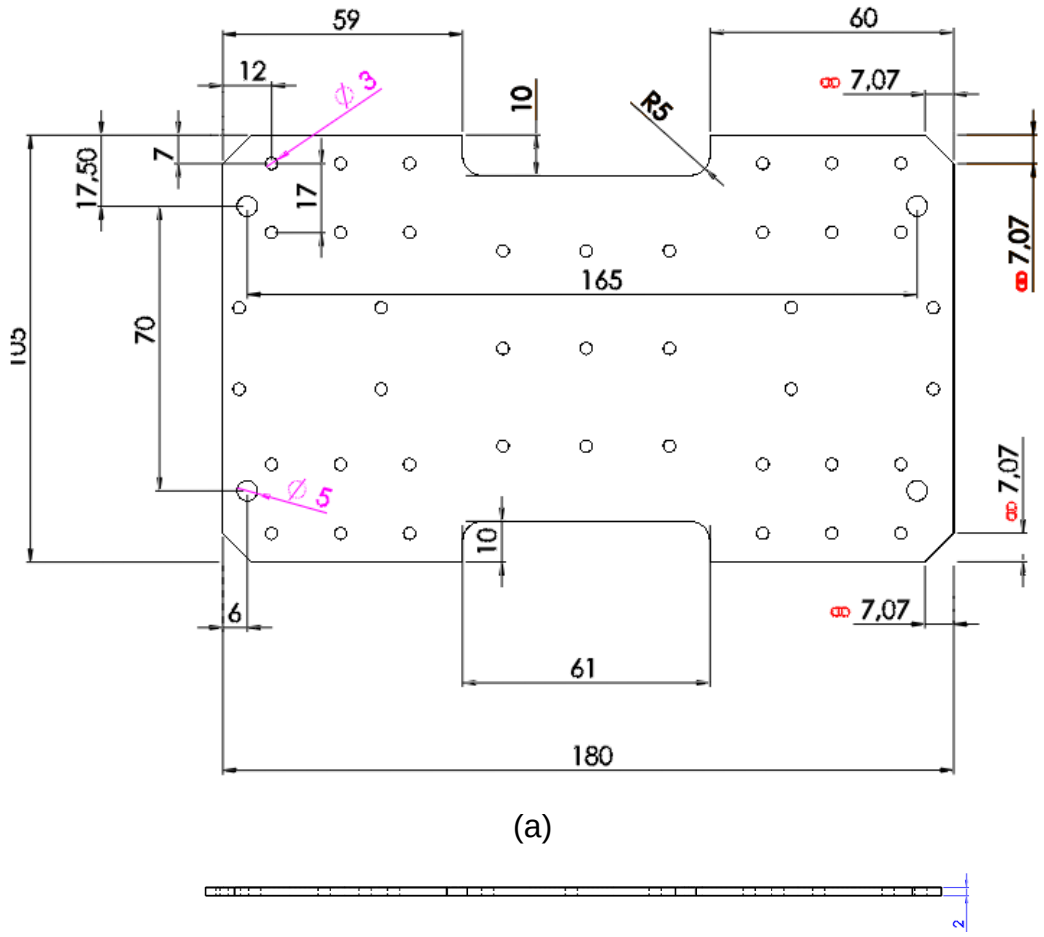


Figura 3.3 – Desenho técnico da placa de alumínio na parte inferior do veículo (vista superior (a) e frontal(b)) medidas dadas em milímetros.

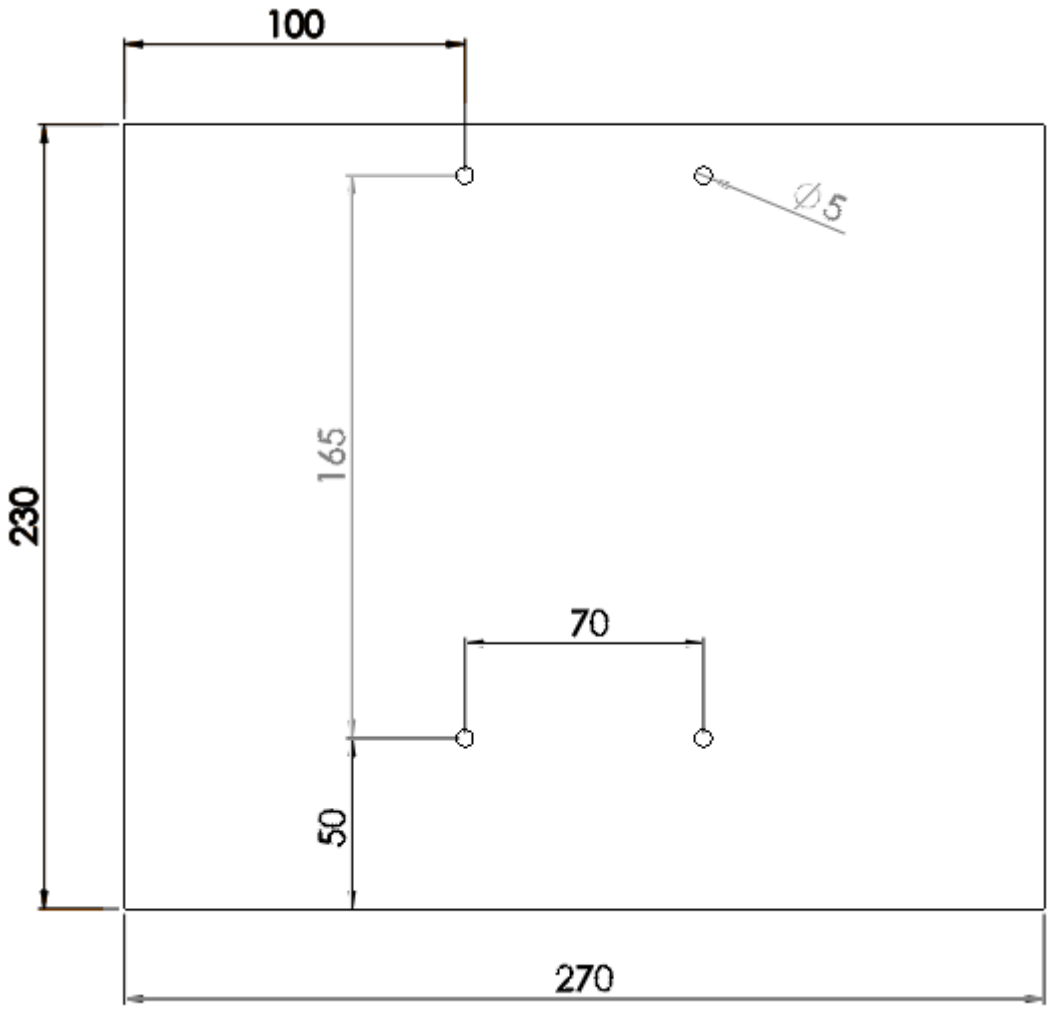
Esta parte da infraestrutura conterá toda a parte eletrônica do projeto (a placa Arduino, o *driver* de motor contendo ponte H, e a bateria de 12 V). A parte eletrônica, citada anteriormente, será explicada com mais detalhes posteriormente.

Para a parte superior do carrinho, mostrado na Figura 3.4, foi realizado o corte e a furação de uma placa de aço. Para realização deste processo, foi utilizada uma furadeira de bancada para o processo de furação, e para a realização do corte foi utilizada uma fresadora, ambas presentes no galpão de mecânica do Campus V do CEFET-MG. Nesta estrutura estará presente o *netbook* e a *webcam*, que farão toda a parte de processamento de imagens e comando do projeto.

Figura 3.4 – Montagem mecânica da parte superior do veículo.

Como proteção, a placa de aço foi pintada com uma tinta de esmalte sintético para metais na cor branco gelo, para impedir de que esta enferrujasse e também para melhorar a aparência do carrinho.

Para conectar a parte superior da armação do veículo com a parte inferior, utilizou-se quatro parafusos de 75 mm de altura e de 5 mm de diâmetro e 16 roscas para a fixação dos parafusos com as duas carcaças do carrinho. Entre as duas partes da estrutura foi deixado um espaço de 60mm para acomodar a parte eletrônica do projeto. Na Figura 3.5 irá mostrar o desenho técnico da parte superior do veículo.



(a)

(b)

Figura 3.5 – Desenho técnico da parte superior do veículo (vista superior (a) e frontal (b)) medidas dadas em milímetros.

Após realizados os procedimentos citados anteriormente, foi acomodado toda a parte eletrônica do projeto na parte inferior do carrinho. O *driver* dos motores que contém a ponte H, foi fixado com dois parafusos de 3 mm de diâmetro diretamente na placa. Já a bateria foi fixada utilizando braçaderias de nylon, isto porque caso necessite a troca da bateria ou a recarga dela, o usuário poderá retirá-la e colocá-la facilmente na parte inferior do veículo.

Entretanto, o Arduino Uno teve que ser fixado em uma cantoneira de alumínio com parafusos de 3 mm de diâmetro. Esta cantoneira foi anexada na placa por dois dos quatro parafusos que ligam a parte superior e inferior do carrinho. Os furos foram realizados por uma furadeira de bancada.

Como foi dito, na estrutura do veículo foi fixado uma bateria de 12 V (Figura 3.6) que tem a função de fornecer energia para os dois motores CC. Entretanto, eles não são ligados diretamente. A bateria primeiramente é conectada no *driver* contendo a ponte H, que faz então a comunicação da bateria com os dois motores CC. Como a placa funciona com uma tensão de operação na faixa de 4 a 35 V, esta se enquadra perfeitamente com a necessidade do projeto. Contudo, para que ela funcione com alimentação externa, no caso a bateria, o *jumper* presente na parte superior do carrinho (Figura 3.7) deverá estar presente. A Tabela 3.2 mostra os dados técnicos da bateria.

Tabela 3.2 – Dados técnico da bateria.

Dados técnicos	
Tensão fornecida	12 V
Duração	1,3 A.h
Marca	UP 1213
Fabricante	UNIPOWER

Figura 3.6 – Bateria UNIPOWER/12 V.

Já para fazer a comunicação do *netbook* com o *driver* dos motores CC, utilizou-se o Arduino Uno, responsável pela tradução dos comandos enviados pelo programa, para então, passar estes comandos traduzidos para o *driver*, que acionará ou não os atuadores do carrinho.

Como o Arduino Uno contém *drivers* comerciais específicos de controle de motores CC, com características de fácil implementação, de necessitar de uma linguagem de programação padrão que se assemelhar C/C++, e ter custo baixo, por estas razões que foi escolhido este tipo de plataforma para o projeto. Os dados técnicos podem ser visualizados na Tabela 3.3. A ligação da parte elétrica do projeto pode ser observada na Figura 3.7, e na Figura 3.8 mostra o esquema elétrico do projeto.

Tabela 3.3 – Dados técnico do Arduino Uno.

Dados técnicos	
Micro controlador	ATmega328
Tensão de operação	5 V
Tensão de Entrada	7-12 V
Portas digitais	14 (6 podem ser usadas como PWM)

Os comandos são enviados do *netbook* para o Arduino pela comunicação USB. Já os comandos traduzidos pelo Arduino são transmitidos pelas saídas digitais 8, 9, 10 e 11; que são utilizados como saídas analógicas. Já o aterramento será utilizado pela saída GND do próprio Arduino interligado com o negativo da bateria de 12 V.

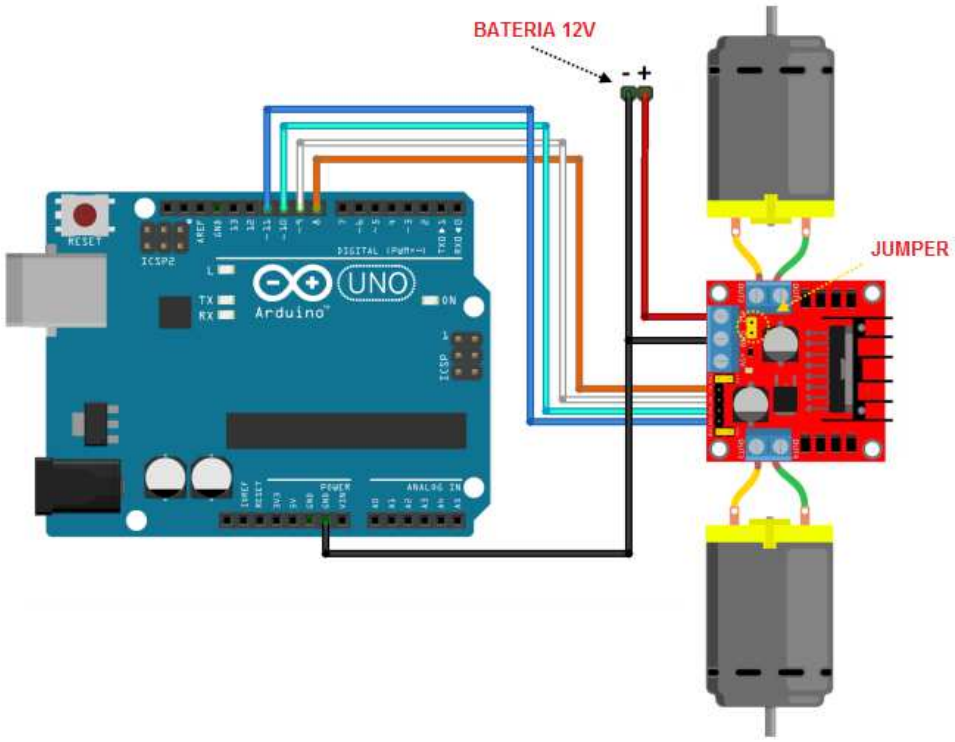


Figura 3.7 – Esquema das ligações elétricas do carrinho autônomo.

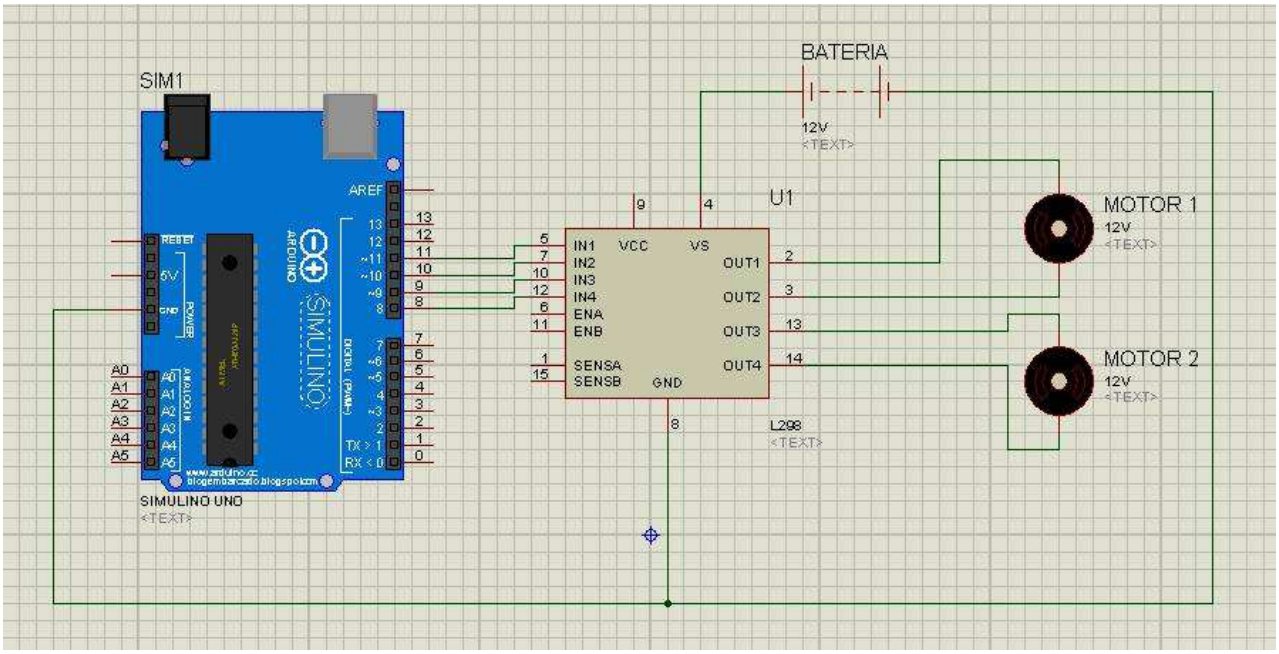


Figura 3.8 – Esquema elétrico do carrinho autônomo.

O *driver* utilizado para controlar os atuadores juntamente com o Arduino é um *shield* que tem como principio de funcionamento o L298N que faz todo o processo

de ponte H. Este *shield* pode controlar cargas indutivas como relés, solenóides, motores DC e motores de passo. Nele é possível controlar independentemente a velocidade e rotação de dois motores de corrente contínua ou um motor de passo. Entretanto neste trabalho o *shield* tem o único objetivo de controlar os dois motores de corrente contínua.

A Figura 3.9 mostra a montagem completa do veículo autônomo proposto, com o *netbook* e a câmera.

Figura 3.9 – Montagem completa do veículo.

3.2. Parte do *software*

Esta seção será dividida em duas partes, sendo que a primeira parte explicará a programação realizada no Arduino, e a segunda parte tratará da visão computacional do projeto.

3.2.1. Programação do Arduino

A programação do Arduino utiliza quatro saídas analógicas para mandar tensões que variam de 0 até 5 V, sendo dividido em uma escala de 0 até 255 (1 byte de escala). Entre essas quatro saídas, duas comandam a velocidade de rotação do motor da roda da direita e os outros dois comandam a velocidade de rotação do motor da roda da esquerda. A frequência utilizada para a comunicação serial entre o Arduino e o *netbook* foi 9600 Hz.

Resumidamente, o programa consiste em ler a entrada serial que receberá um byte de comando do *netbook*, sendo que este poderá conter as palavras: 1, 3, 4 e 5, todos estruturados na codificação ASCII. Após lido um byte, o programa irá identificar a palavra contida no byte e mandará, nas saídas digitais 8, 9, 10 e 11 do Arduino, comandos para o *driver* contendo a ponte H que fará o motor ligar ou desligar. A palavra "3" em ASCII no programa significa que o carrinho pode se movimentar. A palavra "4" em ASCII significa que o veículo deve parar. A palavra "1" e "5" em ASCII, todos previamente seguido pela palavra "3", fará o carrinho se movimentar para direita ou esquerda. Caso o Arduino tenha recebido a palavra "3" em ASCII e logo após não foi enviado a palavra "1" ou a palavra "5" em ASCII, o carrinho se movimentará para frente.

O programa completo pode ser visto no Apêndice 1 deste trabalho.

3.2.2. Programação do *software* da visão computacional

Para utilizar a programação da visão computacional, utilizou-se o *netbook* Dell Latitude 2120, que tem a configuração mostrada na Tabela 3.4.

Tabela 3.4 – Configuração do *netbook*.

Modelo	**Latitude 2120**
Fabricante	Dell
Processador	Intel Atom CPU N550 1,50 GHz
Memória	2 GB
Sistema Operacional	Windows 7 Professional 32 Bits

A configuração da câmera é mostrada na Tabela 3.5.

Tabela 3.5 – Configuração da câmera.

Modelo	**Easy WC301**
Fabricante	NewLink
Resolução	2.0 Mega*pixels*
Taxa de frames	15 fps
Distância focal	Manual
Tamanho da imagem gerada	640 x 480 *pixels*

Primeiramente, para poder utilizar os recursos de aquisição e de tratamento de imagem, foi empregado a biblioteca OpenCV, que trata especialmente de visão computacional e contém licença livre tanto para meios educacionais como para meios comerciais.

Para este programa foi utilizado as seguintes bibliotecas:

`cv.h` → Biblioteca responsável pela visão computacional (OpenCV), contém o básico de processamento de imagem e algoritmos de visão computacional;

`highgui.h` → Biblioteca responsável pela visão computacional (OpenCV), contém rotinas entrada e saída e funções para armazenar e carregar imagens e vídeos;

`stdio.h` → Biblioteca que cuida da parte de entrada e saída de dados;

`stdlib.h` → Biblioteca que emula o programa no *prompt* do sistema operacional que está sendo programado;

`math.h` → Biblioteca que cuida da parte de algorítmicos matemáticos do programa;

`windows.h` → Biblioteca que contêm todas as definições de janelas: criar, abrir, etc;

`cstdlib.h` → Biblioteca que contêm funções envolvendo alocação de memória, controle de processos, conversões e outras;

`string.h` → Biblioteca que contêm funções envolvendo manipulações de strings;

Após a definição das bibliotecas, foi feito a função principal do programa de identificação, de tratamento e de comando do carrinho autônomo.

A princípio foi feito a configuração da porta serial utilizada na comunicação do *netbook* embarcado com o Arduino, também a realização de testes com relação a abertura da porta serial e a associação desta porta com a estrutura DCB. Esta parte pode ser visualizada na Figura 3.10.

```
HANDLE hSerial;
char mens[100];
char *NomePorta = "COM1";
DWORD BytesEscritos = 0;

hSerial = CreateFile(NomePorta, GENERIC_READ|GENERIC_WRITE, 0, NULL, OPEN_EXISTING, 0, NULL);

if(hSerial == INVALID_HANDLE_VALUE)
{
    printf( "Porta nao inicializada \n");
    system("pause");
    return false;
} //Erro ao tentar abrir a porta

DCB dcb; //Estrutura DCB é utilizada para definir todos os parâmetros da comunicação.

if( !GetCommState(hSerial, &dcb)){

    printf( "Erro associando hSerial com DCB. \n"  );

    system("pause");

    return false;
} //// Erro na leitura de DCB.

dcb.BaudRate = CBR_9600;
dcb.ByteSize = 8;
dcb.Parity = NOPARITY;
dcb.StopBits = ONESTOPBIT;

//Define novo estado.
if( SetCommState(hSerial, &dcb) == 0 )
{
     printf( "Erro setando serial. \n"  );
     system("pause");
     return false; //Erro.
}
```

Figura 3.10 – Parte do programa que configura a comunicação da porta serial.

Este trecho do programa mostra que a porta serial utilizada é a COM1 (caso a COM1 não seja a mesma utilizada pelo Arduino, deve-se mudar a configuração do Arduino para COM1), sendo sua configuração: frequência de comunicação 9600 Hz, tamanho da palavra enviada 8 bytes, não há paridade e com um bit de parada.

Caso os testes de setagem, de leitura e de iniciação da porta serial indique erro, o programa mostrará a mensagem falando o erro e esperará o usuário apertar alguma tecla para sair do programa.

Logo após essas instruções, é feito a busca e a identificação da câmera conectada no *netbook*, também realiza-se a criação da estrutura de referência dela e a testagem da realização da identificação da câmera. Isto pode ser verificado na Figura 3.11.

```
CvCapture *camera = cvCreateCameraCapture(0);
//Caso a criação falhe, motifica o usuário e depois sai do programa
if(!camera)
{
    printf("Nao foi encontrada nenhuma camera conectada. \n");
    system("pause");
    return false;
}
```

Figura 3.11 – Parte do programa que identifica e referencia a câmera no programa.

Em seguida o programa manda um comando para o Arduino para que ele faça os motores ligarem, fazendo com que o carrinho ande para frente.

Após o programa enviar o comando citado acima, ele entra em uma estrutura de *loop* que só terminará quando o usuário apertar a tecla ESC do teclado do *netbook*. Dentro da estrutura de *loop*, no início conterá alguns ponteiros que serão explicados abaixo:

`IplImage *imagem_pequena;` → ponteiro que conterá a imagem capturada na resolução de 320x240 *pixels* e posteriormente conterá a imagem contendo somente a área de interesse.

`IplImage *imagem_pequena_cinza;` → ponteiro que conterá a imagem captura na escala de cor cinza.

`IplImage *imagem_contorno;` → ponteiro que conterá a imagem capturada na escala cinza e com contorno.

`IplImage *imagem_contorno_dilatada;` → ponteiro que conterá a imagem capturada na escala cinza, com contorno e dilatada.

`IplImage *imagem;` e **`IplImage *frame;`** → ponteiro que conterá a imagem real vista pela câmera na resolução 640x480 *pixels*.

`CvMemStorage *storage;` e **`CvSeq *contours;`** → são estruturas utilizadas para a coleta de dados dinâmicos. **`CvMemStorage`** contém ponteiros para a memória alocada real, e **`CvSeq`** é usado para acessar esses dados. No programa, **`CvSeq`** irá realizar a lista dos contornos da imagem.

Após feito a definição dos ponteiros para as imagens e a realização de teste para verificação de uma nova captura de imagem, inicia-se o processo de captura da imagem. Para isso utilizou-se a função **`cvQueryFrame()`** que é contida na biblioteca OpenCV e que retém uma imagem e a guarda em um ponteiro. Neste programa ela é apontada por "*frame*". Depois se faz um outro teste para verificar se foi possível obter a imagem.

Seguidamente vem a parte do programa que faz o tratamento da imagem. Esse trecho pode ser visto na Figura 3.12.

```
//Faz uma copia da imagem capturada pela câmera
imagem = cvCloneImage(frame);

imagem_pequena = cvCreateImage(cvSize(IMG_WIDTH,IMG_HEIGHT),IPL_DEPTH_8U,3);

//Redimensiona a imagem para 320x480 pixels
cvResize(imagem, imagem_pequena, CV_INTER_LINEAR);

//Faz um recorte na imagem, obtendo somente a parte de interesse para o projeto
CvRect Area = cvRect(0 , 180 , 100, 60);
cvSetImageROI(imagem_pequena, Area);

//Converte a escala da imagem RGB para cinza.
imagem_pequena_cinza = cvCreateImage (cvGetSize(imagem_pequena), IPL_DEPTH_8U, 1);
cvCvtColor(imagem_pequena, imagem_pequena_cinza, CV_RGB2GRAY);

//Função que deixa a imagem contendo só os contornos dos objetos
imagem_contorno = cvCreateImage(cvGetSize(imagem_pequena), IPL_DEPTH_8U, 1);
cvCanny(imagem_pequena_cinza, imagem_contorno, 200, 200, 3);

//Função que dilata a imagem
imagem_contorno_dilatada = cvCreateImage(cvGetSize(imagem_pequena), IPL_DEPTH_8U, 1);
cvDilate(imagem_contorno, imagem_contorno_dilatada, 0, 1);

//Função que identifica as linhas contidas das imagens
contours = cvHoughLines2(imagem_contorno_dilatada, storage, CV_HOUGH_PROBABILISTIC, 1, CV_PI/180, 50, 50, 10);
CvPoint* ponto;
```

Figura 3.12 – Parte do programa que trata a imagem capturada pela webcam.

Este trecho é a parte que consome maior tempo de processamento do programa e por esta razão foi feito um grande esforço para otimizar as funções de tratamento de imagem.

A função **`cvCloneImage()`** presente no programa, tem como finalidade de fazer uma cópia de uma imagem, neste caso foi da imagem apontada pelo "frame".

A função imagem **cvCreateImage()** cria uma imagem, sendo esta tendo as características passadas como parâmetros. No caso do programa foram passados os parâmetros de tamanho, de profundidade **(IPL_DEPTH_8U)** e o número de canais por *pixel*. Os parâmetros de tamanho são obtidos pela função **cvSize(IMG_WIDTH,IMG_HEIGHT)** , sendo que **IMG_WIDTH** tem valor padrão de 320 e **IMG_HEIGHT** tem valor padrão de 240, ou pela função **cvGetSize()** que deve ser passado como parâmetro o ponteiro da imagem que se deve copiar o formato do tamanho. O número de canais variam, neste programa, em dois valores: 1 quando se refere a imagem na escala cinza e 3 quando se refere a escala RGB.

Como foi dito acima, para otimizar o processamento da imagem, foi realizado a diminuição da escala da imagem que será tratada e o recorte da região de interesse que conterá a faixa retilínea contida no caminho que o veículo deve percorrer. As funções que fazem este processo são:

cvRelize() → No programa, ele faz a interpolação linear da imagem apontada por "imagem" e guardada na memória onde é apontada por "imagem_pequena".

cvSetImageROI() → No programa, ele é realizado juntamente com a função **cvRect()**, que cria um retângulo com a origem no ponto (0,180) com altura e comprimento de 100 e 60 pixels. Feito a área do retângulo, este deve ser passado juntamente com o ponteiro da imagem que será recortado como parâmetro da função **cvSetImageROI().**

Com estas duas funções, o programa não tratará as regiões que não são interessantes para o projeto e também diminui a quantidade de informações contidas nas imagens.

Posteriormente, a imagem é convertida para a escala cinza, pois como será explicado mais na frente, a função que traça os contornos na imagem precisa que a imagem esteja em escala cinza. A função que faz essa conversão é a **cvCvtColor()**, que tem como parâmetros: o ponteiro da imagem que será convertido para cinza, o ponteiro da imagem onde será salva o resultado da conversão e o tipo de conversão. Neste caso, será o tipo **CV_RGB2GRAY** que é da escala vermelho, verde e azul para a escala cinza.

Com a conversão, o programa entra na parte de segmentação por detecção de bordas. Neste caso, utilizou-se a técnica de detecção de borda desenvolvida por Canny, que embora seja a mais complexa das técnicas de detecção, é a que tem melhor desempenho (MARENGONI e STRINGHINI, 2009). A função que executa essa técnica é a **`cvCanny()`**, onde deve ser passado o ponteiro da imagem original, o ponteiro da imagem de destino, *threshold* para o processo de histerese, tamanho da abertura para o operador **`Sobel()`.**

Para que os contornos fiquem mais nítidos para a detecção de linhas no programa, foi feito a dilatação da imagem. Para isso, utilizou-se a função **`cvDilate()`**, onde deve-se passar os seguintes parâmetros: ponteiro da imagem de origem, ponteiro da imagem de destino, estrutura que será usada para fazer a operação dilatar (neste caso, passou o parâmetro 0 que fará a função utilizar uma estrutura retângula 3x3), e o número de iterações que a operação será realizada.

Feito a parte de segmentação, o programa irá procurar as linhas na imagem. A função que aplica técnicas de detecção de linhas é a transformada de Hough. Entre os tipos de transformadas de Hough foi escolhido a da probabilística, pois esta não armazena todo o conteúdo de linhas individualmente, ela armazena somente um pedaço dessas linhas, diminuindo assim o tempo de processamento com relação ao número de cálculos realizados.

Os parâmetros necessários para que este tipo de transformada de Hough funcione são as seguintes: o ponteiro da imagem de origem, o ponteiro onde os resultados serão armazenados, o tipo de método usado, a resolução entre as linha, o valor que uma linha será retornada quando o valor do acumulador for maior que este, o tamanho mínino da linha e a distância mínima que pode existir entre um seguimento de linha para ser considerada uma linha única.

Às Figuras 3.13 à 3.18 mostram o desenvolvimento do programa com relação ao tratamento da imagem adquirida pela *webcam*.

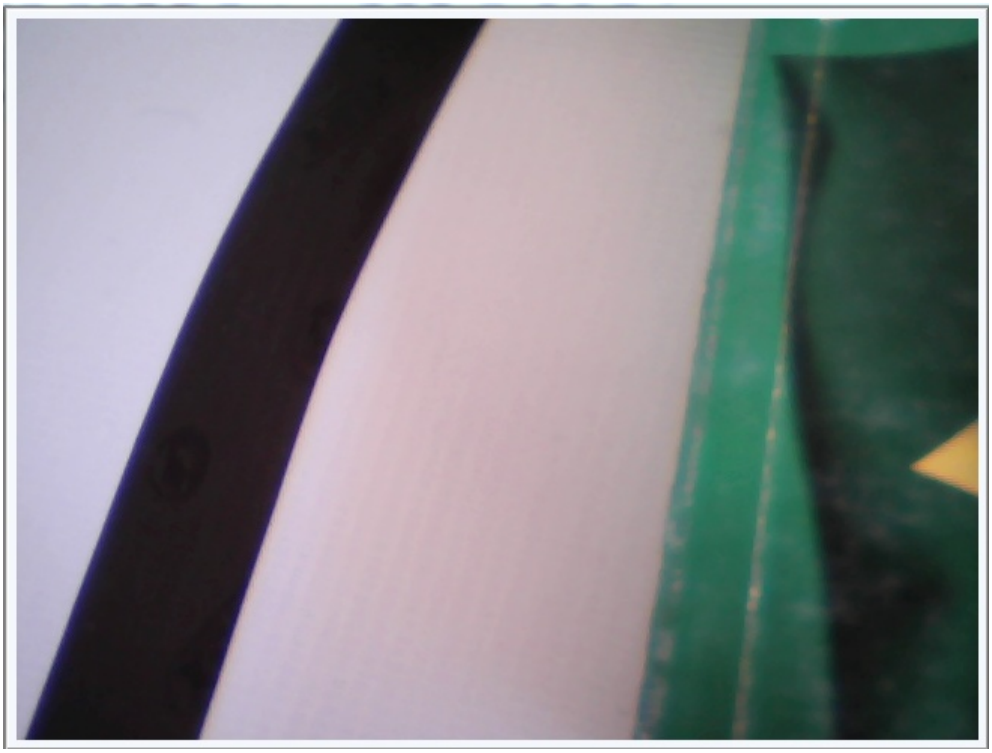


Figura 3.13 - Imagem do ponteiro "imagem".

Figura 3.14 - Imagem do ponteiro "imagem_pequena" antes do recorte.

Figura 3.15 - Imagem do ponteiro "imagem_pequena" depois do recorte.

Figura 3.16 –Imagem do ponteiro "imagem_pequena_cinza".

Figura 3.17 – Imagem do ponteiro "imagem_contorno".

Figura 3.18 – Imagem do ponteiro "imagem_contorno_dilatada".

Nesse ponto do projeto, encontraram-se duas soluções: uma seria de identificar e selecionar as linhas na imagem com relação ao ângulo, e o outro seria de identificar as linhas e utilizar a posição destas linhas na imagem.

No primeiro programa (Apêndice 2), utilizou-se a identificação de linhas pelo ângulo. Na estrutura desse trecho, ele captura 5 retas e passa as coordenadas dos pontos inicial e final, obtidos na função **`(CvPoint*) cvGetSeqElem(contours, 0)`**. Com estes dois pontos é feito o cálculo dos ângulos de cada reta com relação ao eixo x. Estes ângulos podem variar de -90º a 90º. E somente contará como uma reta válida, quando estes ângulos forem menores que -45º ou maiores que 45º.

Em seguida, é feita a comparação dos ângulos e, então, para cada tipo de ângulo é realizado um tipo de comando. Primeiro é tirado o ângulo médio entre os ângulos obtidos anteriormente. Para ângulos médios entre 0º e 85º, será enviado o comando de virar a direita, para ângulos médios entre 0º e -85º, será enviado o comando de virar a esquerda e para ângulos médios menores que -85º ou maiores que 85º, o comando enviado será de seguir em frente. Caso não se encontre linhas na imagem, o último comando é repassado até que ocorra a identificação de uma imagem contendo reta.

Para o envio dos comandos utilizou-se a função **`WriteFile()`**, sendo que os argumentos que devem ser passados são: a porta de comunicação (no programa é obtido pelo "hSerial"), a mensagem a ser passada (previamente deve ser utilizada a função **`strcpy`** que copia a mensagem para uma variável char), o tamanho da palavra (obtida pela função **`strlen()`**), a variável do tipo ponteiro longo onde esta função retorna a quantidade exata de bytes escritos e o ponteiro para uma estrutura "overlapped" que não será utilizada portanto será passado o parâmetro NULL.

O segundo programa (Apêndice 3) obtêm também as coordenadas do ponto inicial e final de somente uma reta. Depois é feita a análise do ponto que estiver mais abaixo da Figura. Após isso, é feito a comparação da posição do ponto da reta em relação ao eixo x com valores já predefinidos. Estes pontos foram analisados pelo usuário e foram postos por apresentar uma localização que o veículo começa a sair do caminho marcado para trafegar ou que se afasta muito desta marcação (faixa).

No programa mostrado no Apêndice 3, pode ser observado que quando a linha se situa nas regiões que apresenta coordenada x menores que 30, o programa envia para o Arduino o comando de virar à direita. Já quando o carrinho se localiza nas coordenadas do eixo x maiores que 55, o comando enviado será de virar a esquerda. Caso não satisfaça nenhuma das duas ocasiões, o comando enviado será de seguir para frente. Estes comandos são enviados pela mesma função apresentada no primeiro programa.

Já após estas partes dos dois programas, a estrutura se assemelha. Ocorre a visualização das imagens apontadas pelos ponteiros "imagem_pequena_cinza" e "imagem". Seguidamente, ocorre a limpeza da memória alocada (**`cvReleaseMemStorage()`**) e das imagens (**`cvReleaseImage()`**).

Feito isso, termina o laço infinito do programa e então é finalizado o programa com o envio do comando de parar o veículo, com o fechamento das janelas de transmissão das imagens (**`cvDestroyAllWindows()`**), com a limpeza da estrutura de captura da câmera (**`cvReleaseCapture()`**) e com fechamento da porta serial.

A Figura 3.19 e a Figura 3.20 mostram resumidamente os fluxogramas dos dois programas criados no decorrer do projeto.

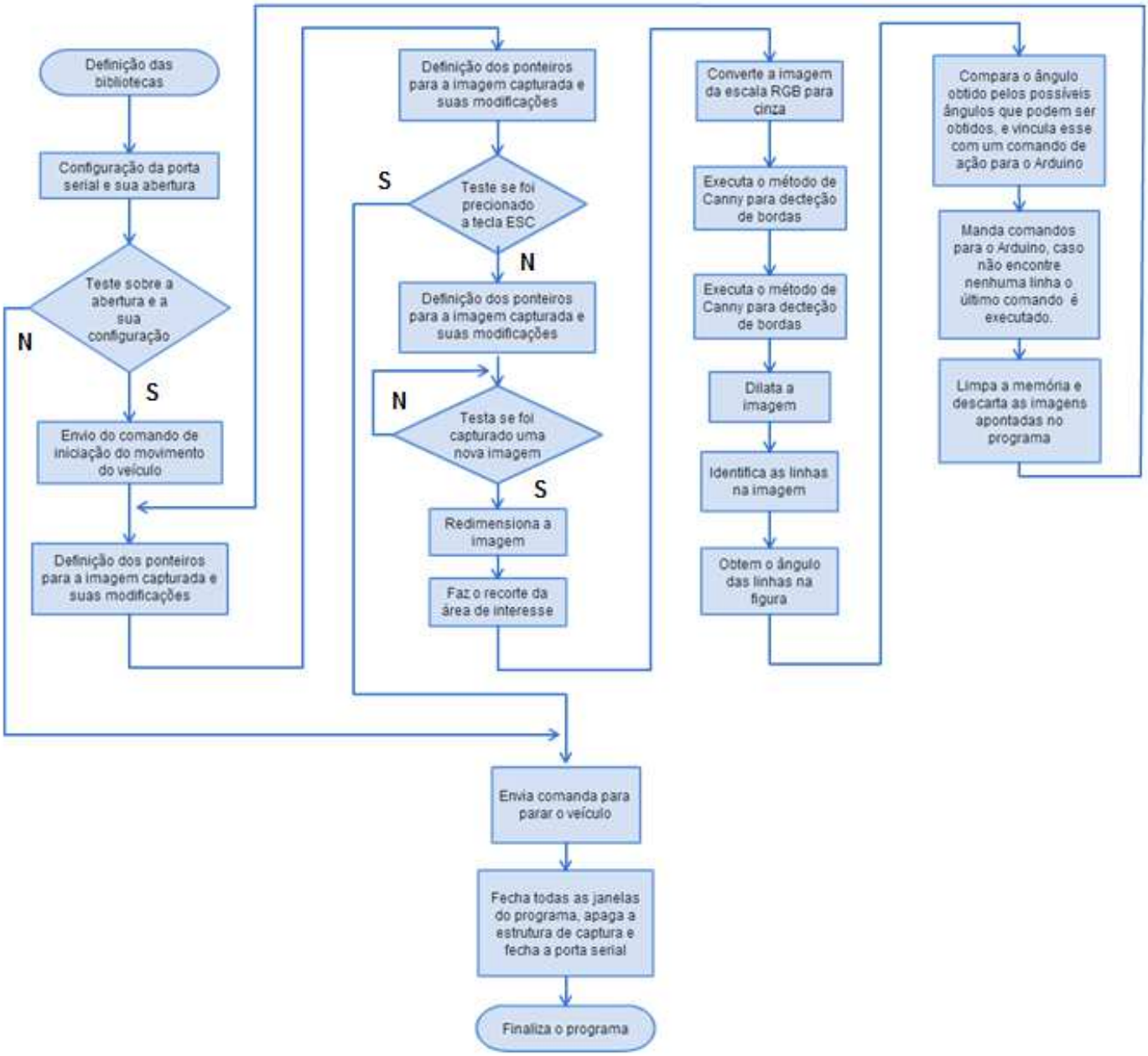


Figura 3.19 – Fluxograma do primeiro programa de visão computacional.

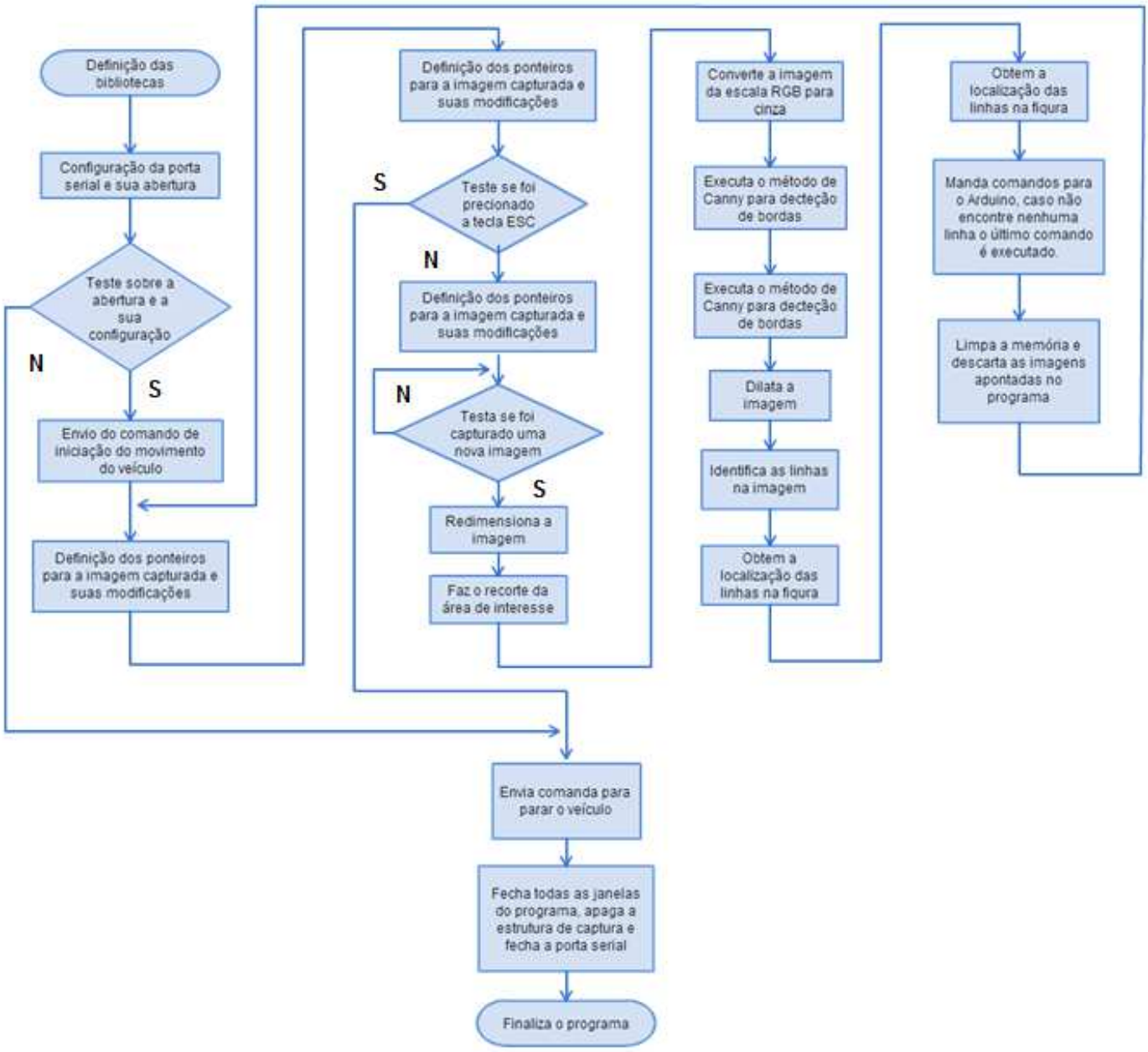


Figura 3.20 – Fluxograma do segundo programa de visão computacional.

Capítulo 4: Resultados e Discussões

Neste capítulo serão mostrados os resultados encontrados durante o processo de realização deste projeto e juntamente também serão discutidos esses resultados.

Foram realizados dois testes, sendo que cada teste foi feito em dois lugares diferentes. Esses lugares podem ser observados na Figura 4.1 e na Figura 4.2. Nestes testes foram observados a estrutura do carrinho, a capacidade dos motores em relação ao peso do conjunto do veículo, a comunicação do *netbook* com o Arduino e o desempenho dos programas (do Arduino e da visão computacional).

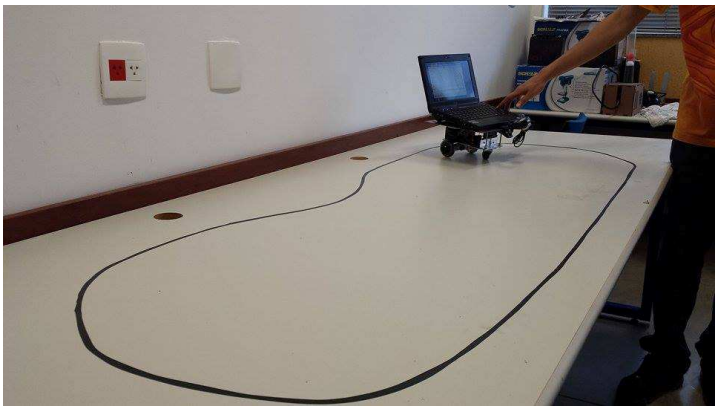


Figura 4.1 – Primeira pista que o carrinho foi testado.

Figura 4.2 – Segunda pista que o carrinho foi testado.

Com relação à parte da estrutura do veículo autônomo diferencial de pequeno porte, notou-se que ficou bem rígida e compacta. Ela não apresentou nenhuma anomalia durante os testes realizados.

Com relação à capacidade do carrinho, foi observado que nos testes os dois motores de corrente contínua responsáveis pela locomoção do veículo conseguiram satisfazer a sua função. Entretanto, quando foi feito o acréscimo de peso no veículo (substituição do *netbook*, presente na parte superior do carrinho, por um notebook), os dois motores começaram a apresentar uma sobrecarga. Com isso, eles diminuíram bastante a sua eficiência com relação à velocidade adquirida no trajeto das duas pistas. Com relação à eficiência do carrinho em trafegar lugares com inclinação, foram feitos testes em um outro local, com inclinação de aproximadamente 30 graus, e nestes testes o veículo conseguiu se movimentar, porém com muita dificuldade, causando uma grande sobrecarga nos motores.

De acordo com a comunicação do *software* de visão computacional com o Arduino Uno, obteve-se resultados satisfatórios. Nas duas pistas onde foram realizados os testes, os comandos enviados pelo *netbook* foram recebidos com sucesso e com rapidez. Isso se deve ao bom relacionamento entre o drive da ponte H com o Arduino Uno. Outra razão foi a simplicidade do processo de comunicação, que só tem a função de mandar comandos para os motores ligarem ou desligarem, não apresentando nenhuma outra opção específica.

No que diz respeito ao desempenho do programa de visão computacional tiveram os seguintes resultados:

- A parte de captura de imagem e tratamento em tempo real foi bastante eficiente, pois o processamento das imagens foram bem ágeis e eficazes.
- A parte de tratamento das imagens, o programa conseguiu modificar a escala de cor e redimensionar as imagens vindas da *webcam* com facilidade e agilidade.
- A segmentação das imagens e a identificação das linhas tiveram resultados satisfatórios, quando há uma boa iluminação no local onde são realizados os experimentos. Isso pode ficar bem claro ao anoitecer, pois as imagens obtidas pela *webcam* ficavam com tons mais escuros, acarretando assim uma dificuldade

em relação à detecção dos contornos das linhas. Já com a presença de sombras nas imagens, também ocorre a dificuldade de detecção das linhas, porém essa dificuldade apresentou-se menor do que na situação descrita antes.

Com relação aos dois programas desenvolvidos referentes à visão computacional, o segundo programa (utilizando a localização da linha nas imagens como referência para o controle do carrinho) teve resultados satisfatórios. Nele o controle do veículo foi eficiente, o veículo conseguiu se movimentar e se orientar de forma correta como era pretendido pelo trabalho. Entretanto, o programa que utilizava o ângulo da linha para controlar o carrinho ao longo do percurso não foi satisfatório. Quando ele realizava curvas, em um certo ponto do trajeto, a webcam não conseguia captar mais a linha do trajeto, fazendo o carrinho vagar pela pista sem destino. Isso não ocorre no segundo programa por ele apresentar o controle relacionado diretamente com a posição da reta nas imagens.

Um outro fator interessante a ser abordado foi o gasto de memória do programa de visão computacional. A princípio, o *netbook*, em que era executado o programa, sofria erros de falta de memória, alegando que o programa estava sobrecarregando-o. Isso foi corrigido utilizando os comandos de **`cvReleaseMemStorage`** e **`cvReleaseImage`** dentro da rotina de repetição do programa, que antes ficava na parte final do programa (executando somente quando era finalizado o programa).

Capítulo 5: Conclusões e Perspectivas

5.1. Conclusões

A partir dos resultados mostrados na análise e discussões, pode-se tirar as seguintes conclusões sobre o projeto apresentado:

1. O projeto desenvolvido conseguiu atender todos os objetivos propostos no trabalho.

2. A parte mecânica atendeu os requisitos necessários para a constituição da estrutura do veículo autônomo de pequeno porte.

3. A parte eletrônica também atendeu as expectativas desejadas pelo projeto, caracterizando como eficiente.

4. A comunicação entre o notebook com os motores de corrente contínua foi ágil e eficiente.

5. A parte de visão computacional também teve sucesso. Mesmo que o primeiro não tenha sucesso na parte de controle, os dois programas de visão computacional conseguiram identificar as linhas nos caminhos testados.

5.2. Proposta para trabalhos futuros

Com base nos resultados e nas discussões apresentados no Capítulo 4 deste trabalho, sugerem-se como trabalhos de continuidade deste projeto as seguintes propostas:

- Utilizar um material mais leve do que foi utilizado (chapa de aço) na parte superior da estrutura do carrinho;
- Desenvolver um controlador mais eficiente neste projeto, podendo assim controlar o carrinho utilizando a velocidade de rotação dos motores, que são disponíveis na placa do *driver* de ponte H utilizado no trabalho.
- Adaptar esse tipo de controle em um veículo de médio ou de grande porte, para participações em eventos nacionais ou internacionais.
- Desenvolver um sistema de detecção de objetos nas estradas, como também na identificação de sinalizações no caminho onde é percorrido o veículo (placas e semáforos).
- Desenvolver o programa de visão computacional para outros hardwares, como celulares.
- Utilizar outros sensores para auxiliar o programa de visão computacional, como sensores de luz, ultrassons e acelerômetros.
- Adaptar esse sistema de visão computacional para que funcione paralelamente ao sistema GPS, podendo assim traçar caminhos longos automaticamente.

Referências Bibliográficas

ARGO. **Argo.** Disponível em: http://www.argo.ce.unipr.it/argo/english/index.html. Acessado em 29/12/2014.

BRADSKI, Gary, & KAEBLER, Adrian. "***O'REILLY. Learning OpenCV Computer Vision with the OpenCV Library***". Estados Unidos da América, 2008.

BROGGI, A., BERTOZZI, A., FASCIOLI, A., e GUARINO, C. "***The argo autonomous vehicles vision and control systems***". International Journal on Intelligent Control Systems, 1999.

CAYRES, Flávia. "**Laboratório de Robótica Móvel desenvolve tecnologia para carro autônomo** ". Assessoria de comunicação da INCT-SEC, 2013. Disponível em: http://www.inct-sec.org/br/noticias/386-laboratorio-de-robotica-movel-desenvolve-tecnologia-para-carro-autonomo. Acessado em 14/04/2014.

DA SILVA, Luciano Rottava. "**Análise e Programação de Robôs Móveis Autônomos da Plataforma Eyebot**". Dissertação de Mestrado, Programa de Pós-Graduação em Engenharia Elétrica. Universidade do Estado de Santa Catarina – UFSC. Florianópolis – SC, 2003.

DE OLIVEIRA, Marcos. "**Carro sem motorista: Projetos brasileiros de veículos autônomos trazem contribuições para o futuro da mobilidade urbana**". Revista Pesquisa - FAPESP, 2013, p. 59-64.

DE SALES, Uilian Gonçalves. "**Criação de um *software*, baseado em um sistema de visão, para reconhecimento da posição do veículo na rodoviária".** UniCEUB - Centro Universitário de Brasília. Brasília, 2008.

DETRAN-MA. “**Ação educativa do Detran nos retornos da capital chamam a atenção de motoristas**”. Departamento de Transito do Maranhão, 2013. Disponível em: http://www.detran.ma.gov.br/noticias2/?noticia=421&titulo=A%E7%E3o%20educativa%20do%20Detran%20nos%20retornos%20da%20capital%20chamam%20a%20aten%E7%E3o%20de%20motoristas. Acessado em 13/04/2014.

DIAS, Jullierme Emiliano Alves. “**Modelagem longitudinal e controle de velocidade de um carro autônomo**”. Dissertação de Mestrado em Engenharia Elétrica. Universidade Federal De Minas Gerais - UFMG. Belo Horizonte, Minas Gerais, 2013.

DICKMANNS, E. D. “***Dynamic vision-based Intelligence***”. AI Magazine. Meno Park, 2004, p. 10-30.

DPRF. "**Número de acidentes por tipo e gravidade"**. Departamento de Polícia Rodoviária Federal, 2011. Disponível em: http://www.dnit.gov.br/rodovias/operacoes-rodoviarias/estatisticas-de-acidentes/quadro-0102-numero-de-acidentes-por-tipo-e-gravidade-ano-de-2011.pdf. Acessado em 29/12/2014.

DUDEK, G.; JENKIN, M. “***Computational Principles of Mobile Robotics***”. New York: Cambridge University Press, 2000.

ELETROMANÍACOS. **Eletromaníacos - Portal de eletrônica**. Disponível em: http://www.eletromaniacos.com/modules.php?name=News&file=article&sid=41. Acessado em 29/12/2014.

FACON, Jacques. **"Processamento e Análise de Imagens"**. Apostila do curso em Mestrado em Informática Aplicada. Pontifícia Universidade Católica do Paraná, Curitiba, 2005.

FERREIRA, José Júlio Areal. “**Demonstrador de Condução Autónoma**”. Dissertação de Mestrado em Engenharia Eletrotécnica e de Computadores. Faculdade de Engenharia da Universidade do Porto, 2012.

FERNANDES, L., SOUZA, J., SHINZATO, P., PESSIN, G., MENDES, C., OSORIO, F., e WOLF, D. “***Intelligent robotic car for autonomous navigation: Platform and system architecture. In Proceedings of the Second Brazilian Conference on Critical Embedded Systems***”. Universidade de São Paulo - USP, São Carlos - Brasil, 2012.

FORSYTH, D. A.; PONCE, J. “***Computer Vision: A Modern Approach***”. Prentice Hall, 2003.

GINGICHASHVILI, Sarah. *"**DARPA's Urban Challenge 2007**"*. *The Future of Things*, 2007. Disponível em: http://thefutureofthings.com/3019-darpas-urban-challenge-2007/. Acessado em 29/12/2014.

GOMES, Pedro Miguel Amorim. “**FEUPCar: Condução Autônoma no Festival Nacional de Robótica**”. Dissertação de Mestrado em Engenharia Eletrotécnica e de Computadores. Faculdade de Engenharia da Universidade do Porto, 2010.

GONZALEZ, R. C.; WOODS, R. E. “***Digital Image Processing, 3rd Ed.***”. Prentice Hall, 2008.

HAYKIN, S.; VAN VENN, B. “***Signal and Systems, 2nd Ed.***”. Jonh Wiley & Sons, 2002.

NETO, Arthur de Miranda. “**Navegação de Robôs Autônomos baseada em Monovisão**. ” Campinas, São Paulo, 2007.

PISSARDINI, R. S. ; WEI, D. C. M. ; FONSECA JR., E. S. “**Veículos Autônomos: Conceitos, Histórico e Estado-da-Arte.** ” Anais do XXVII Congresso de Pesquisa e Ensino em Transportes, 2013.

SHAKEY. **Shakey**. 1968 . Disponível em: http://www.ai.sri.com/shakey. Acessado em 29/12/2014.

SILVA, F.J.V. E ALVES, C.H.F. "**Aplicação de técnicas de processamento de imagens digitais em imagens geradas por ultra-som**". 8° Encontro Regional de Matemática Aplicada e Computacional. Universidade Federal do Rio Grande do Norte – Natal/RN, 2008.

TANIMOTO, S., & KLINGER, A. “**Visão computacional estruturado: a percepção da máquina através de estruturas hierárquicas de computação**”. New York: McGraw-Hill, 1980.

TECHTUDO. "**CADU, o veículo autônomo mineiro".** Techtudo, 2012. Disponível em: http://www.techtudo.com.br/artigos/noticia/2012/11/conheca-cadu-e-carina-os-proejtos-de-veículos-autonomos-no-brasil.html. Acessado em 29/12/2014.

Apêndice

Apêndice 1 - Programa utilizado no Arduino.

```
//Motor A
int Saida1 = 8; //pino 8 do Arduino
int Saida2 = 9; //pino 9 do Arduino

//Motor B
int Saida3 = 10; //pino 10 do Arduino
int Saida4 = 11; //pino 11 do Arduino

//Variavel auxiliar
int tecla = 0;
int aux = 0;

void setup()
{
   //Configuração dos pinos do Arduino.
   pinMode(Saida1, OUTPUT);
   pinMode(Saida2, OUTPUT);
   pinMode(Saida3, OUTPUT);
   pinMode(Saida4, OUTPUT);
   //Frequencia para comunicação serial.
   Serial.begin(9600);
}
void loop()
{
   // Se conseguir pegar um byte, será lido entrada analógica e
armazenado na variável tecla:
```

```
if (Serial.available() > 0)
{
        tecla = Serial.read();
        //Quando lido o numero 3 em ASCII na porta, faz o
        //carrinho começar a andar.
        if (tecla == '3')
        {
             aux = 1;
        }
        //Quando lido o numero 4 em ASCII na porta serial, faz o
        //carrinho parar.
        if (tecla == '4')
        {
             aux = 0;
        }
        // Entra no if quando lido o numero 3 em ASCII na porta
        //serial.
        if( aux == 1 )
        {
             //Quando lido o numero 1 em ASCII na porta serial,
             //faz o carrinho virar a direita.
             if (tecla == '1')
             {
                  analogWrite(Saida1, 0);
                  analogWrite(Saida2, 127);
                  analogWrite(Saida3, 200);
                  analogWrite(Saida4, 0);
             }
             //Quando lido o numero 5 em ASCII na porta serial,
             //faz o carrinho virar a esquerda.
             if(tecla == '5')
```

```
            {
                  analogWrite(Saida1, 0);
                  analogWrite(Saida2, 200);
                  analogWrite(Saida3, 127);
                  analogWrite(Saida4, 0);
            }
            // Quando não for lido a palavra 1 ou 5 em ASCII na
            //porta, faz o carrinho andar para frente.
            if((tecla != '1') && (tecla != '5'))
            {
                  analogWrite(Saida1, 0);
                  analogWrite(Saida2, 255);
                  analogWrite(Saida3, 255);
                  analogWrite(Saida4, 0);
            }

      }
      // Faz o carrinho parar quando lido o byte contendo 4 em ASCII
      //na porta serial.
      if (aux == 0)
      {
            analogWrite(Saida1, 0);
            analogWrite(Saida2, 0);
            analogWrite(Saida3, 0);
            analogWrite(Saida4, 0);
      }

   }
}
```

Apêndice 2 - Primeiro programa desenvolvido utilizando visão computacional.

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

using namespace std;

//Define o tamanho que a imagem capturada deve estar.

#define IMG_WIDTH 320  //Comprimento da imagem
#define IMG_HEIGHT 240 //Altura da imagem

int main (int argc, const char * argv[])
{
    // Configuração da porta serial e como também realização de teste com
    //relação a abertura da porta serial e a acossiação desta porta com a
    //estrutura DCB.

    HANDLE hSerial;
    char mens[100];
    char *NomePorta = "COM1";
    DWORD BytesEscritos = 0;

    hSerial = CreateFile(NomePorta, GENERIC_READ|GENERIC_WRITE, 0, NULL,
                         OPEN_EXISTING, 0, NULL);

    if(hSerial == INVALID_HANDLE_VALUE)
    {
```

```
    printf( "Porta nao inicializada \n");
    system("pause");
    return false;
} //Erro ao tentar abrir a porta

DCB dcb; //Estrutura DCB é utilizada para definir todos os parâmetros
//da comunicação.

if( !GetCommState(hSerial, &dcb))
{
    printf( "Erro associando hSerial com DCB. \n"  );
    system("pause");
    return false;
} //Erro na leitura de DCB.

dcb.BaudRate = CBR_9600;
dcb.ByteSize = 8;
dcb.Parity = NOPARITY;
dcb.StopBits = ONESTOPBIT;

//Define novo estado.
if( SetCommState(hSerial, &dcb) == 0 )
{
     printf( "Erro setando serial. \n"  );
     system("pause");
     return false; //Erro.
}

//Variáveis auxiliares
char quit = 0;
char grab_frame = 1;

//Função que identifica a primeira câmera conectada e cria a sua
//estrutura de referência.
```

```
CvCapture *camera = cvCreateCameraCapture(0);

//Caso a criação falhe, motifica o usuário e depois sai do programa
if(!camera)
{
        printf("Nao foi encontrada nenhuma camera conectada. \n");
        system("pause");
        return false;
}

//Inicia a movimentação do carrinho
strcpy( mens , "3" );
int tam = strlen(mens);
WriteFile( hSerial, mens, tam, &BytesEscritos, NULL );

//Loop de captura e identificação das linhas
while(!quit)
{
      // CvMemStorage e CvSeq são estruturas utilizadas para a coleta
      //de dados dinâmico. CvMemStorage contém ponteiros para
      //a memória alocada real, mas CvSeq é usado para acessar esses
      //dados. Aqui, ele irá realizar a lista dos contornos da imagem.
      CvMemStorage *storage;
      storage = cvCreateMemStorage(0);
      CvSeq *contours = 0;
      //Estes são os ponteiros para as imagens IPL
      IplImage *imagem_pequena ;
      IplImage *imagem_pequena_cinza ;
      IplImage *imagem_contorno ;
      IplImage *imagem_contorno_dilatada ;
      IplImage *imagem;
      IplImage *frame; //Este ponteiro irá apontar para a imagem IPL
      //que será recuperar a partir da câmara.
```

```
int c = cvWaitKey(30); //Espera 30 ms para o usuário aperte um
//tecla.

//Resposta para a tecla prescionada.
switch(c)
{
        case 27: //Esc: Sai do loop do programa quando precionado
        //a tecla ESC do teclado.
        quit = 1;
        break;
};
//Captura um quadro da câmera.
frame = cvQueryFrame(camera);

if(!frame)continue; //Não foi possível obter uma imagem, tentar
//novamente.

//Faz uma copia da imagem capturada pela câmera
imagem = cvCloneImage(frame);

imagem_pequena = cvCreateImage (cvSize
                  (IMG_WIDTH,IMG_HEIGHT),IPL_DEPTH_8U,3);

 //Redimensiona a imagem para 480x320 pixels
 cvResize(imagem, imagem_pequena, CV_INTER_LINEAR);

 //Faz um recorte na imagem, obtendo somente a parte de interesse
 //para o projeto
 CvRect Area = cvRect(0 , 180 , 100, 60);
 cvSetImageROI(imagem_pequena, Area);

 //Converte a escala da imagem RGB para cinza.
 imagem_pequena_cinza = cvCreateImage (cvGetSize(imagem_pequena),
                        IPL_DEPTH_8U, 1);
```

```
 cvCvtColor(imagem_pequena, imagem_pequena_cinza, CV_RGB2GRAY);

 //Função que deixa a imagem contendo só os contornos dos objetos
 imagem_contorno = cvCreateImage(cvGetSize(imagem_pequena),
                                 IPL_DEPTH_8U, 1);
 cvCanny(imagem_pequena_cinza, imagem_contorno, 200, 200, 3);

 //Função que dilata a imagem
 imagem_contorno_dilatada = cvCreateImage(cvGetSize
                           (imagem_pequena), IPL_DEPTH_8U, 1);
cvDilate(imagem_contorno, imagem_contorno_dilatada, 0, 1);

//Função que identifica as linhas contidas das imagens
 contours = cvHoughLines2(imagem_contorno_dilatada, storage,
           CV_HOUGH_PROBABILISTIC, 1, CV_PI/180, 50, 50, 10);
for( int i=0; i < MIN(contours->total, 5); i++)
{
        CvPoint* ponto;
        double x = (ponto[1].x - ponto[0].x);
        double y = (ponto[0].y - ponto[1].y);
        double t = y/x;
        float anguloMedido = atan(t);
        anguloMedido = 180*anguloMedido/CV_PI;
        if((anguloMedido < 45) && (anguloMedido > (-45)))
        {}
        else
        {
              angulo[i]= anguloMedido;
                    linhasEncontradas++;
        }
  }
  int aux;
  if(linhasEncontradas > 0)
  {
```

```
        for(int j=0;j<5;j++)
        {
              anguloMedio = (angulo[j] + anguloMedio);
        }
              anguloMedio = (anguloMedio/5);
              if( (anguloMedio < 0) && (anguloMedio > (-85)) )
              {
                          tam = strlen(mens);
                          strcpy( mens , "1" );

                          WriteFile( hSerial, mens, tam,
                          &BytesEscritos, NULL );
                          aux= 1;
                          printf("\n Virar Direita \n");
              }
              if( (anguloMedio > (0)) && (anguloMedio < (85)) )
              {
                          strcpy( mens , "5" );
                          tam = strlen(mens);
                          WriteFile( hSerial, mens, tam,
                                      &BytesEscritos, NULL );
                          aux = 5;
                          printf("\n Virar Esquerda \n");
              }
              if( (anguloMedio >= 85) || (anguloMedio <= (-85)))
              {
                          strcpy( mens , "3" );
                          tam = strlen(mens);
                          WriteFile( hSerial, mens, tam,
                                      &BytesEscritos, NULL );
                          aux =3;
                          printf("\n Seguir Reto \n");
              }
}
```

```
        else
        {
                printf("\n Nao encontrou linha \n");
                if(aux==1)
                {
                        strcpy( mens , "1" );
                        tam = strlen(mens);
                        WriteFile( hSerial, mens, tam,
                                &BytesEscritos, NULL );
                        aux= 1;
                        printf("\n N acho linha - Virar Direita
                                \n");
                }
                if(aux==5)
        {
                strcpy( mens , "5" );
                tam = strlen(mens);
                WriteFile( hSerial, mens, tam, &BytesEscritos, NULL
                        );
                aux = 5;
                printf("\n N acho linha - Virar Esquerda \n");
        }
        if(aux==3)
        {
                strcpy( mens , "3" );
                tam = strlen(mens);
                WriteFile( hSerial, mens, tam, &BytesEscritos, NULL
                        );
                aux =3;
                printf("\n N acho linha - Seguir reto \n");
        }
}
anguloMedio=0;
linhasEncontradas = 0;
```

```
      // Mostra as imagens do ponteiro imagem_contorno_dilatada e imagem
      cvShowImage ("imagem_contorno_dilatada", imagem_contorno_dilatada);
      cvShowImage("Imagem", imagem);

      // Limpa a memória e as imagens
      cvReleaseImage(&imagem_contorno_dilatada;
      cvReleaseImage(&imagem_pequena_cinza);
      cvReleaseImage(&imagem_contorno);
      cvReleaseImage(&imagem);
      cvReleaseImage(&imagem_pequena);
      cvReleaseMemStorage(&storage);

      printf("Passou um frame.\n\n");
   }

   strcpy( mens , "4" );
   printf("\n Parando\n\n");
   tam = strlen(mens);
   WriteFile( hSerial, mens, tam, &BytesEscritos, NULL );

   //Fecha todas as janelas abertas pelo programa
   cvDestroyAllWindows();

   //Apaga a estrutura de captura da câmera
   cvReleaseCapture(&camera);

   //Fecha a porta serial aberta pelo programa
   CloseHandle( hSerial );
   system("pause");

   //Finaliza o programa
   return 0;
}
```

Apêndice 3 - Segundo programa desenvolvido utilizando visão computacional.

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

using namespace std;

//Define o tamanho que a imagem capturada deve estar.

#define IMG_WIDTH 320  //Comprimento da imagem
#define IMG_HEIGHT 240 //Altura da imagem

int main (int argc, const char * argv[])
{
    // Configuração da porta serial e como também realização de teste com
    //relação a abertura da porta serial e a acossiação desta porta com a
    //estrutura DCB.

    HANDLE hSerial;
    char mens[100];
    char *NomePorta = "COM1";
    DWORD BytesEscritos = 0;

    hSerial = CreateFile(NomePorta, GENERIC_READ|GENERIC_WRITE, 0,
                         NULL, OPEN_EXISTING, 0, NULL);

    if(hSerial == INVALID_HANDLE_VALUE)
```

```
{
    printf( "Porta nao inicializada \n");
    system("pause");
    return false;
} //Erro ao tentar abrir a porta

DCB dcb; //Estrutura DCB é utilizada para definir todos os parâmetros
         //da comunicação.

if( !GetCommState(hSerial, &dcb))
{
    printf( "Erro associando hSerial com DCB. \n"  );
    system("pause");
    return false;
} // Erro na leitura de DCB.

dcb.BaudRate = CBR_9600;
dcb.ByteSize = 8;
dcb.Parity = NOPARITY;
dcb.StopBits = ONESTOPBIT;

//Define novo estado.
if( SetCommState(hSerial, &dcb) == 0 )
{
     printf( "Erro setando serial. \n"  );
     system("pause");
     return false; //Erro.
}

//Variáveis auxiliares
char quit = 0;
char grab_frame = 1;
```

```
//Função que identifica a primeira câmera conectada e cria a sua
//estrutura de referência.
CvCapture *camera = cvCreateCameraCapture(0);

 //Caso a criação falhe, motifica o usuário e depois sai do programa
 if(!camera)
 {
        printf("Nao foi encontrada nenhuma camera conectada. \n");
        system("pause");
        return false;
 }

//Inicia a movimentação do carrinho
strcpy( mens , "3" );
int tam = strlen(mens);
WriteFile( hSerial, mens, tam, &BytesEscritos, NULL );

//Loop de captura e identificação das linhas
while(!quit)
{
      // CvMemStorage e CvSeq são estruturas utilizadas para a coleta
      //de dados dinâmico. CvMemStorage contém ponteiros para
      //a memória alocada real, mas CvSeq é usado para acessar esses
      //dados. Aqui, ele irá realizar a lista dos contornos da imagem.
      CvMemStorage *storage;
      storage = cvCreateMemStorage(0);
      CvSeq *contours = 0;
      //Estes são os ponteiros para as imagens IPL
      IplImage *imagem_pequena ;
      IplImage *imagem_pequena_cinza ;
      IplImage *imagem_contorno ;
      IplImage *imagem_contorno_dilatada ;
      IplImage *imagem;
```

```
IplImage *frame; //Este ponteiro irá apontar para a imagem IPL
que será recuperar a partir da câmara.

int c = cvWaitKey(30); //Espera 30 ms para o usuário aperte um
tecla.

//Resposta para a tecla prescionada.
switch(c)
{
        case 27: //Esc: Sai do loop do programa quando precionado
        //a tecla ESC do teclado.
        quit = 1;
        break;
};
//Captura um quadro da câmera.
frame = cvQueryFrame(camera);

if(!frame)continue; //Não foi possível obter uma imagem, tentar
//novamente.

//Faz uma copia da imagem capturada pela câmera
imagem = cvCloneImage(frame);

imagem_pequena = cvCreateImage
                (cvSize(IMG_WIDTH,IMG_HEIGHT),IPL_DEPTH_8U,3);

 //Redimensiona a imagem para 320x480 pixels
 cvResize(imagem, imagem_pequena, CV_INTER_LINEAR);

 //Faz um recorte na imagem, obtendo somente a parte de interesse
 //para o projeto
 CvRect Area = cvRect(0 , 180 , 100, 60);
 cvSetImageROI(imagem_pequena, Area);

 //Converte a escala da imagem RGB para cinza.
```

```
imagem_pequena_cinza = cvCreateImage (cvGetSize(imagem_pequena),
                                      IPL_DEPTH_8U, 1);
cvCvtColor(imagem_pequena, imagem_pequena_cinza,
            CV_RGB2GRAY);

//Função que deixa a imagem contendo só os contornos dos objetos
imagem_contorno = cvCreateImage(cvGetSize(imagem_pequena),
                                IPL_DEPTH_8U, 1);
cvCanny(imagem_pequena_cinza, imagem_contorno, 200, 200, 3);

//Função que dilata a imagem
imagem_contorno_dilatada = cvCreateImage(cvGetSize(
                           imagem_pequena), IPL_DEPTH_8U, 1);
cvDilate(imagem_contorno, imagem_contorno_dilatada, 0, 1);

//Função que identifica as linhas contidas das imagens
contours = cvHoughLines2(imagem_contorno_dilatada, storage,
CV_HOUGH_PROBABILISTIC, 1, CV_PI/180, 50, 50, 10);
CvPoint* ponto;
for( int i=0; i < MIN(contours->total, 1); i++)
{
        ponto = (CvPoint*) cvGetSeqElem(contours, 0);
}
int yaux;
int comando;
int aux = 0;

//Comandos para os motores com relação a localização da reta na
imagem
printf("ponto[1].x - %d /" , ponto[1].x); printf("ponto[1].y - %d
/", ponto[1].y);                    printf("ponto[0].x    -    %d    /",
ponto[0].x); printf("ponto[0].y - %d \n ", ponto[0].y);

 if((ponto[1].y) > (ponto[0].y))
 {
```

```
     yaux = 1;
}
if((ponto[0].y) < (ponto[1].y))
{
     yaux = 0;
}

if( (ponto[1].x) <= (30))
{
     printf("\n Virar Direita \n");
     strcpy( mens , "1" );
     tam = strlen(mens);
     WriteFile( hSerial, mens, tam, &BytesEscritos, NULL );
     comando = 1;
     aux = 1;
}
if ( (ponto[1].x) >= (55))
{
     printf("\n Virar Esquerda \n");
     strcpy( mens , "5" );
     tam = strlen(mens);
     WriteFile( hSerial, mens, tam, &BytesEscritos, NULL );
     comando = 5;
     aux = 1;
}
if( ((ponto[1].x) > (30)) && ((ponto[1].x) < (55)) )
{
     printf("\n Seguir reto \n");
     strcpy( mens , "3" );
     tam = strlen(mens);
     WriteFile( hSerial, mens, tam, &BytesEscritos, NULL );
     comando = 3;
     aux = 1;
}
```

```
 if( aux == 0)
 {
      if( comando == 1)
      {
                  printf("\n N acho linha - Virar Direita \n");
                  strcpy( mens , "1" );
                  tam = strlen(mens);
                  WriteFile( hSerial, mens, tam, &BytesEscritos,
                              NULL );
      }
      if( comando == 5)
      {
                  printf("\n N acho linha - Virar Esquerda \n");
                  strcpy( mens , "5" );
                  tam = strlen(mens);
                  WriteFile( hSerial, mens, tam, &BytesEscritos,
                              NULL );
      }
      if( comando == 3)
      {
                  printf("\n N acho linha - Seguir reto \n");
                  strcpy( mens , "3" );
                  tam = strlen(mens);
                  WriteFile( hSerial, mens, tam, &BytesEscritos,
                  NULL );
      }
}

// Mostra as imagens do ponteiro imagem_contorno_dilatada e
//imagem
cvShowImage ("imagem_contorno_dilatada",
             imagem_contorno_dilatada);
cvShowImage("Imagem", imagem);
```

```
            // Limpa a memória e as imagens
            cvReleaseImage(&imagem_contorno_dilatada;
            cvReleaseImage(&imagem_pequena_cinza);
            cvReleaseImage(&imagem_contorno);
            cvReleaseImage(&imagem);
            cvReleaseImage(&imagem_pequena);
            cvReleaseMemStorage(&storage);

            printf("Passou um frame.\n\n");
        }

        strcpy( mens , "4" );
        printf("\n Parando\n\n");
        tam = strlen(mens);
        WriteFile( hSerial, mens, tam, &BytesEscritos, NULL );

        //Fecha todas as janelas abertas pelo programa
        cvDestroyAllWindows();

         //Apaga a estrutura de captura da câmera
         cvReleaseCapture(&camera);

         //Fecha a porta serial aberta pelo programa
         CloseHandle( hSerial );
         system("pause");

         //Finaliza o programa
         return 0;
    }
```