UNIVERSIDADE DE CAXIAS DO SUL

FELIPE JOSÉ ZANGIROLAMI BERTEI

DESENVOLVIMENTO DE SISTEMA DE NAVEGAÇÃO AUTÔNOMO TRACK AND FOLLOW PARA CADEIRA DE RODAS MOTORIZADA

BENTO GONÇALVES
2018

FELIPE JOSÉ ZANGIROLAMI BERTEI

DESENVOLVIMENTO DE SISTEMA DE NAVEGAÇÃO AUTÔNOMO TRACK AND FOLLOW PARA CADEIRA DE RODAS MOTORIZADA

Trabalho de Conclusão de Curso apresentado como requisito para a obtenção do título de Bacharel em Engenharia Eletrônica pela Universidade de Caxias do Sul.

Orientador:
Prof. Me. Angelo Zerbetto Neto

BENTO GONÇALVES
2018

FELIPE JOSÉ ZANGIROLAMI BERTEI

DESENVOLVIMENTO DE SISTEMA DE NAVEGAÇÃO AUTÔNOMO TRACK AND FOLLOW PARA CADEIRA DE RODAS MOTORIZADA

Trabalho de Conclusão de Curso apresentado como requisito para a obtenção do título de Bacharel em Engenharia Eletrônica pela Universidade de Caxias do Sul.

Orientador:
Prof. Me. Angelo Zerbetto Neto

Aprovado em ____/____/_____

Banca Examinadora

_____
Prof. Me. Angelo Zerbetto Neto (orientador)
Universidade de Caxias do Sul

_____
Prof. Dr Alexandre Mesquita
Universidade de Caxias do Sul

_____
Prof. Me Felipe Augusto Tondo
Universidade de Caxias do Sul

"*Let the future tell the truth, and evaluate*
each one according to his work and accomplishments.
The present is theirs; the future,
for which I have really worked, is mine"
(Nikola Tesla)

RESUMO

Este trabalho apresenta o desenvolvimento de um sistema de navegação *Track and Follow* (Detectar e Seguir) de baixo custo aplicado em uma cadeira de rodas motorizada. Este trabalho utiliza visão computacional como ferramenta de detecção de um alvo que está sendo carregado por uma pessoa, realizando em seguida o processo de acompanhamento dos passos da mesma. Primeiramente é apresentado o conceito de cadeiras de rodas inteligentes e as características e equipamentos que as definem, sendo em seguida apresentado um estudo sobre as tecnologias utilizadas em sistemas *Track and Follow* aplicados em sistemas de tecnologia assistiva e também em outros campos de aplicação, como robôs e veículos não tripulados. Após são discutidas tecnologias para visão computacional, algoritmos e técnicas de detecção disponíveis, além de sensores utilizados em veículos autônomos e técnicas de controle da movimentação de veículos elétricos. Por fim se apresenta o desenvolvimento deste trabalho, demonstrando os blocos funcionais desenvolvidos e técnicas de controle utilizadas para o funcionamento do trabalho proposto, indicando ao final os resultados obtidos e problemas detectados durante o desenvolvimento, além de possíveis melhorias futuras e uma comparação dos custos financeiros estre este trabalho e outros similares.

Palavras-chave: *Track and Follow*. Tecnologia Assistiva. Cadeira de rodas motorizada. Visão Computacional.

ABSTRACT

This work presents the development of a low cost Track and Follow navigation system applied in a motorized wheelchair. This work uses computational vision as a tool for detecting a target being carried by a person, then performing the process of following the steps of the same. First, the concept of intelligent wheelchairs and the characteristics and equipment that define them are presented. A study is then presented on the technologies used in Track and Follow systems applied in assistive technology systems and also in other fields of application, such as robots and unmanned vehicles. Afterwards we discuss technologies for computer vision, algorithms and detection techniques available, as well as sensors used in autonomous vehicles and techniques to control the movement of electric vehicles. Finally, it is presented the development of this work, demonstrating the functional blocks developed and control techniques used for the proposed work, indicating at the end the obtained results and problems detected during the development, besides possible future improvements and a comparison of the financial costs and similar work.

Keywords: Track and Follow. Assistive Tecnology. Electric Wheelchair. Computer Vision.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE SIGLAS

ABNT	Associação Brasileira de Normas Técnicas
BCI	*Brain Control Interface*
EMG	Eletromiografia
IBGE	Instituto Brasileiro de Geografia e Estatística
LIDAR	*Light Detection And Ranging*
OMS	Organização Mundial de Saúde
PID	*Proportional–Integral–Derivative*
PTZ	*Pan-Tilt-Zoom*
RGB-D	*Red-Green-Blue-Depth*
SDK	*Software Development Kit*
SLAM	*Simultaneous Localization And Mapping*
SURF	*Speeded Up Robust Features*

SUMÁRIO

1 INTRODUÇÃO

O avanço tecnológico em sistemas inteligentes e de locomoção vem possibilitado o desenvolvimento de novas tecnologias que vem ao auxilio de pessoas com deficiência física ou com alguma incapacidade motora. Estimativas feitas pela Organização Mundial de Saúde indicam que 15,6% da população mundial com mais de 18 anos possui dificuldades severas para exercer tarefas básicas do dia a dia (OMS, 2011). No Brasil, segundo dados do Censo do IBGE (IBGE, 2010), mais de 13 milhões de brasileiros possuem algum tipo de deficiência motora. No Rio Grande do sul, esse número chega a 800 mil, sendo que somente na região da Serra Gaúcha existem mais de 50 mil pessoas com algum tipo de deficiência motora (IBGE, 2010).

Previsões, realizadas também pelo IBGE, indicam que até 2060 a população de pessoas com mais de 60 anos será de 30% da população total brasileira (IBGE, 2013), devido aos avanços na medicina e o aumento da longevidade populacional, sendo que, baseado em dados de 2003, 10% da população com mais de 65 anos necessita do auxilio de um cuidador(a) para exercer atividades básicas do dia a dia (KARSCH, 2003).

Para toda essa considerável parcela da população, melhorias tecnológicas na área de locomoção podem prover uma importante melhoria na qualidade de vida. Tecnologias assistivas são desenvolvidas com o propósito de auxiliar pessoas com necessidades especiais a exercer tarefas cotidianas por si só (ENCARNAÇÃO; COOK, 2017). Uma destas tecnologias são as cadeiras de rodas inteligentes, que utilizam dispositivos e sensores originalmente criados para robótica móvel, capazes de executar tarefas como localização, navegação e evitar colisões de forma autônoma (DESAI; MANTHA; PHALLE, 2017).

Além de propiciar uma melhoria na mobilidade de pessoas com deficiência motora, o desenvolvimento de tecnologias para cadeiras de rodas inteligentes se torna necessário para uma população que está cada vez mais longeva. Quanto maior o número de pessoas com necessidades para locomoção, maior será o numero de cuidadores necessários para suprir esta demanda sendo que, baseado nas projeções realizadas pelo IBGE acima citadas, manter o numero de cuidadores proporcional ao de idosos se torna insustentável. Tecnologias *Track and Follow*, que são técnicas utilizadas em robótica móvel e veículos autônomos com o objetivo de detectar e seguir uma pessoa ou objeto, ao serem implementadas em cadeiras de rodas motorizadas, podem vir a auxiliar à suprir esta demanda, pois permitem que o cuidador não necessite mais empurrar a cadeira, sendo que a mesma pode segui-lo. O desenvolvimento desta tecnologia pode vir à permitir ainda que várias cadeiras de rodas acompanhem o mesmo cuidador(MOTOKUCHO; ODA, 2014).

Baseando-se nessas informações, pode-se dizer que a criação de uma cadeira de rodas que possua a capacidade de acompanhar uma pessoa pode vir a agregar qualidade de vida para indivíduos com deficiência motora e também para pessoas com mais de 65 anos. Este tipo de tecnologia pode vir ainda a permitir melhor contato social.

1.1 OBJETIVO GERAL

O objetivo geral deste trabalho é o desenvolvimento de um sistema de navegação Track and Follow de baixo custo, aplicado em cadeira de rodas motorizada, capaz de acompanhar os passos de uma pessoa de forma autônoma.

1.2 OBJETIVOS ESPECÍFICOS

Os objetivos específicos deste trabalho são:

a) Adaptação de câmera e eletrônica embarcada em uma cadeira de rodas motorizada;

b) Desenvolvimento de um sistema de rastreamento baseado em imagens e medição de distância;

c) Melhoria do sistema de controle de movimento atualmente presente na cadeira de rodas;

d) Integração do sistema de rastreamento com o sistema de controle de movimentos da cadeira;

e) Validação do projeto por meio da análise de trajetória;

1.3 RESTRIÇÕES

As restrições aplicadas a este trabalho são:

a) O sistema somente irá detectar o acompanhante na frente da cadeira de rodas;

2 REVISÃO BIBLIOGRÁFICA

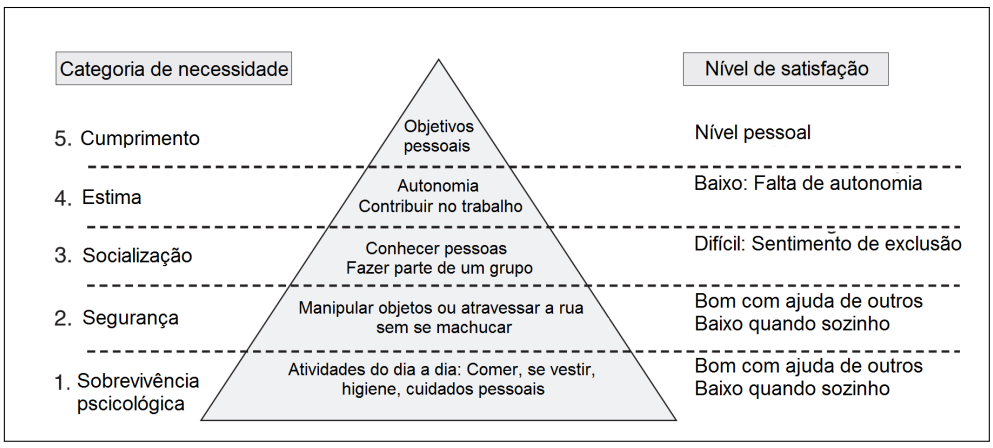
Com a finalidade de fundamentar a metodologia utilizada para a realização deste trabalho, serão apresentados conceitos teóricos e práticos para compreensão dos assuntos necessários para a elaboração do projeto.

2.1 TECNOLOGIA ASSISTIVA

Em um sociedade em rápido crescimento, se torna cada vez mais difícil dar o suporte necessário para o bem estar de cada pessoa, ainda mais quando o indivíduo possui necessidades especiais. Em muitas sociedades primitivas se considerava que possuir algum tipo de deficiência era um castigo, sendo que muitos bebês, nascidos com alguma deficiência, eram sacrificados por serem considerados inúteis para a sobrevivência do resto da população. Essa postura se modificou com o passar do tempo e, durante a Revolução Industrial no século XVIII, se criaram os primeiros dispositivos assistivos, como cadeiras de rodas, bengalas e próteses (SONZA et al., 2013).

Maslow criou, em seu trabalho *A Theory of Human Motivation* (1943), um modelo com cinco categorias principais de necessidades básicas de um ser humano. Esse modelo pode ser adaptado para indicar as necessidades de pessoas com algum tipo de deficiência, como mostra a Figura 1, onde são demonstrados, em forma de pirâmide, cada necessidade de uma pessoa, da menor para a maior (cima para baixo), juntamente com o nível de satisfação pessoal pelo cumprimento das mesmas (HELAL; MOKHTARI; ABDULRAZAK, 2008).

Figura 1: Modelo piramidal de Maslow adaptado para pessoas com necessidade especiais



Fonte: Adaptado de Helal, Mokhtari e Abdulrazak (2008, p. 4).

Tecnologias assistivas possuem várias definições, uma delas vem da OMS (2007) que considera dispositivos assistivos que auxiliam a movimentação como "equipamentos ou produtos adaptados ou especialmente projetados para ajudar pessoas a se moverem dentro e fora de edifícios". Já Helal et al (2008) considera tecnologias assistivas como sendo adaptações ou

personalizações de dispositivos com o objetivo de melhorar a condição física do usuário. Outra definição para tecnologia assistiva é dada por Hersh e Johnson (2008) que consideram sendo "tecnologias, equipamentos, dispositivos, aparelhos, serviços, sistemas, processos e modificações utilizadas por deficientes e/ou idosos para superar barreiras sociais, de infraestrutura ou à sua independência, permitindo realizar atividades com facilidade e segurança". Sobre os tipos de tecnologias assistivas existentes, Dubey et al (2014) categorizam as mesmas entre as seguintes áreas:

a) Médica: Leitores de tela, sinalizadores sonoros e *Closed Caption*[1];

b) Educação: Ferramentas de leitura e dispositivos apontadores;

c) Comunicação: Reconhecimento de voz e softwares para sintetização de texto em voz;

d) Atividades diárias: *Joysticks* e telas *touchscreen*;

e) Recreação e lazer: Técnicas para aumento de telas;

f) Movimentação: Cadeira de rodas, andadores, bengalas;

g) Computacional: conversores TTY[2]

2.2 VEÍCULOS AUTÔNOMOS

Qualquer tipo de veículo pode ser considerado autônomo quando possui algum sistema inteligente que seja capaz de realizar e controlar a sua movimentação sem instruções de comando da parte de um operador humano (COX; WILFONG, 1990). Nos últimos anos, tem-se visto na mídia uma corrida entre fabricantes de automóveis em busca da criação de seus respectivos veículos autônomos, sendo que todos devem possuir quatro módulos que caracterizam um sistema autônomo (GE; LEWIS, 2006):

a) Sensoriamento: Se refere aos tipos de sensores e técnicas de sensoriamento que podem ser implementados em veículos autônomos;

b) Controle: Responsável por comandar o movimento do veículo;

c) Planejamento de trajetória: Utiliza dados de localização, e de outros sensores, para estimar a trajetória que deve ser exercida pelo veículo;

[1]Closed Caption (CC) é um sistema de transmissão das legendas de programas televisivos, transmitidas juntamente com a informação de imagem, que pode ser visualizada em televisores que possuam esta função.

[2]Do inglês *Telephone Typewriters*, dispositivo para comunicação textual desenvolvido para pessoas com dificuldade de escuta e fala (POWER; POWER; HORSTMANSHOF, 2007).

d) Tomada de decisão: Responsável por decidir qual trajetória exercer, baseado em dados de localização e de sensores, e fornecer as informações necessárias para o módulo de controle, permitindo assim a execução do movimento.

O conceito autonomia pode ser aplicado a qualquer tipo de veículo, sendo ele terrestre, aéreo ou subaquático. Esses equipamentos podem ser desenvolvidos para a área da agricultura, onde se pode criar um dispositivo capaz de monitorar plantações e o solo (DURMUŞ et al., 2015), na área de segurança, como dispositivos para vigilância (MICHELONI et al., 2007) ou até em áreas médicas, ao se desenvolver um veículo que entrega comida e remédios dentro de um hospital (KRISHNAMURTHY; EVANS, 1992).

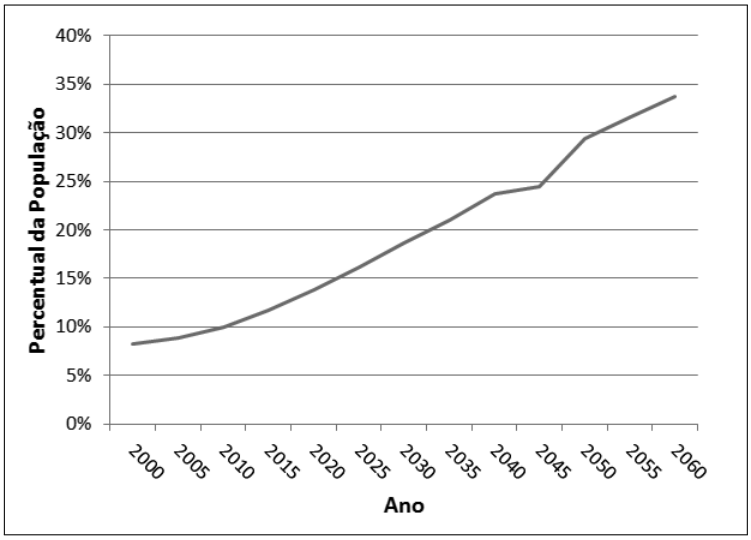
Na área assistiva também já se desenvolvem tecnologias autônomas, como cadeiras de rodas inteligentes, capazes de navegar em ambientes internos ou externos de forma autônoma com a necessidade de poucos comandos recebidos de um operador (LEAMAN; LA, 2015).

2.3 CADEIRA DE RODAS INTELIGENTE

O Censo de 2010 constatou que 45 milhões de brasileiros possuem algum tipo de deficiência física ou intelectual, sendo que mais de 13 milhões de pessoas possuem alguma deficiência motora, sendo que dessas, 700 mil são consideradas permanentemente incapazes, necessitando de auxilio para a realização de tarefas básicas como caminhar ou subir escada e 3,5 milhões possuem grande dificuldade para realização das mesmas tarefas anteriores mesmo com o uso de bengalas ou próteses (IBGE, 2010).

Outro dado importante é a taxa de crescimento da população de idosos, sendo que a projeção para 2060 é de que mais de 30% da população possua 60 anos ou mais, como mostra a Figura 2.

Figura 2: Previsão do crescimento do número de pessoas com mais de 60 anos



Fonte: Adaptado de IBGE (2010).

O numero elevado de pessoas com alguma deficiência física e o aumento da expectativa de vida da população cria uma demanda de novas tecnologias que possuam o objetivo de melhorar a vida e facilitar a locomoção, permitindo uma maior independência dos usuários. Uma destas tecnologias são as cadeiras de rodas inteligentes, que já são objeto de estudo há muitos anos (DESAI; MANTHA; PHALLE, 2017).

Cadeiras de rodas inteligentes são na maioria das vezes, segundo Desai et al (2017), cadeiras motorizadas na qual são adicionados sensores, para detecção do ambiente, e uma unidade de processamento, que é responsável por interpretar as informações vindas dos sensores, controlar os motores, entre outras funções. Uma das primeiras cadeiras de rodas inteligentes desenvolvidas foi feita por Madarasz em 1986, sendo que era capaz de navegar autonomamente utilizando câmera e sensores ultrassônicos (MADARASZ et al., 1986). A partir de então, varias instituições de ensino passaram a desenvolver seus protótipos, uma delas é a cadeira de rodas capaz de navegar em ambientes internos e evitar obstáculos autonomamente, além de poder receber comandos por uma interface cérebro-computador (WEE-KIAT; YING-WEI; GOH, 2014) ou ainda a iChair, mostrada na Figura 3, que é um veículo em desenvolvimento para pessoas quadriplégicas, capaz de navegar autonomamente em ambientes internos além de poder seguir objetos ou pessoas (LEAMAN; LA, 2015) e (LEAMAN; LA; NGUYEN, 2016).

Figura 3: Projeto da iChair.

Fonte: Adaptado de Leaman e La (2015).

Esses dispositivos podem ser classificados por cinco características, o tipo de interface com o usuário, a sua forma construtiva, seu nível de autonomia, funcionalidades disponíveis e sensores utilizados (DESAI; MANTHA; PHALLE, 2017).

A interface com o usuário é o dispositivo utilizado para comandar a cadeira de rodas. Atualmente existem diversas tecnologias utilizadas para esta função, como por exemplo, joystick (RABHI; MRABET; FNAIECH, 2015), comando de voz (WANG; YU, 2017), comando

cervical (NETO et al., 2015), EMG (ISHII; KONISHI, 2016) ou até por uma interface cerebral, também chamada de BCI (Brain Control Interface) (BASTOS-FILHO et al., 2014).

A característica da forma construtiva de uma cadeira de rodas inteligente é referente à maneira como ela foi criada, baseada em uma cadeira de rodas motorizada onde foram adicionados sensores e instrumentos, como da Figura 4, ou se foi adicionada uma cadeira a um veículo elétrico previamente desenvolvido (LEAMAN; LA, 2017).

Figura 4: Cadeira de rodas motorizada com dispositivos incorporados.

Fonte: (ZHANG et al., 2016).

Já o nível de autonomia indica se a cadeira de rodas somente auxilia o usuário a comandar os movimentos ou a evitar colisões com outros objetos, sendo assim considerada semi-autônoma, ou se ela é capaz de transportar o usuário entre pontos conhecidos por si só, sendo assim completamente autônoma (DESAI; MANTHA; PHALLE, 2017).

No item das funcionalidades consideram-se as ações que o dispositivo consegue executar, como desviar de obstáculos, localizar sua posição em ambientes internos ou externos, planejar sua trajetória, ou até acompanhar uma pessoa autonomamente (DESAI; MANTHA; PHALLE, 2017).

No item sensores estão presentes os inúmeros dispositivos que podem ser embarcados para adicionar funcionalidades à cadeiras de rodas inteligentes, os mais usuais são sensores ultrassônicos, sensores a laser Lidar, câmera, sensores inerciais, odômetros ou até scanners 3D (LEAMAN; LA, 2017).

2.4 SISTEMA TRACK AND FOLLOW

A tradução literal para *Track and Follow* significa *Localizar e Seguir*. Segundo Park e Kuipers (2013), a habilidade de acompanhar uma pessoa é uma função importante para uma cadeira de rodas inteligente. Atualmente existem varias tipologias de sistemas *Track and Follow* utilizadas em veículos elétricos, algumas são:

a) Segue o líder: Sistema onde o veículo está configurado para acompanhar um líder, que pode ser uma pessoa ou outro objeto qualquer, sistema implementado por Mutz et al (2017), por Gupta et al (2017) e por Ye et al (2016), entre outros;

b) Segue som: Sistema em que o veículo possui uma rede de sensores de ondas sonoras, permitindo assim que o veículo detecte a direção da onda sonora e que se direcione em direção ao mesmo, projeto implementado por Ha et al (2012) e por Luo, Huang e Lin (2010);

c) Segue onda eletromagnética: Utiliza antenas direcionais que detectam a origem de uma onda eletromagnética, assim indicando a direção em que o veículo deve se locomover, sistema implementado por Kim et al (2007) e por Zhao et al (2015).

O conceito de sistemas *Track and Follow* já foi muito abordado na literatura. Um módulo de localização *Track* pode ser segmentado em duas partes, pelos tipos de sensores utilizados para a localização e os algoritmos implementados para a realização do mesmo. Já o módulo para acompanhamento *Follow* é representado pelo tipo de sistema implementado para o acionamento e controle dos motores do veículo para executar a ação de seguir o objeto desejado. Esse tipo de sistema já é muito utilizado em robótica móvel, assim como em tecnologias assistivas ao ser embarcada em cadeiras de rodas motorizadas. Examinando-se diversas publicações científicas, apresenta-se a seguir os principais sensores e algoritmos de reconhecimento aplicados em sistemas Track and Follow.

Motokusho e Oda (2014) desenvolveram uma cadeira de rodas motorizada que utiliza sistema de visão estereoscópica para seguir uma pessoa, a mesma se posicionando a frente da cadeira. Utiliza dois computadores para a execução das tarefas, um para o controle da cadeira de rodas e outro para o processamento de imagem. O sistema detecta somente as pernas da pessoa a ser seguida e o movimento é estimado utilizando vetores de fluxo óptico e disparidade, obtendo bons resultados para baixas velocidades, porém com oscilações na movimentação para uma caminhada mais rápida.

Wu et al (2012) criaram uma cadeira de rodas robótica onde se utilizou uma câmera PTZ[3] para um reconhecimento inicial do acompanhante a ser seguido, através de uma placa com padrões de texto e descritores/detectores SURF[4] , e um sensor de medição a laser para

[3]Câmera PTZ (*Pan-Tilt-Zoom*) é uma câmera que consegue mudar a sua direção e aplicar zoom remotamente

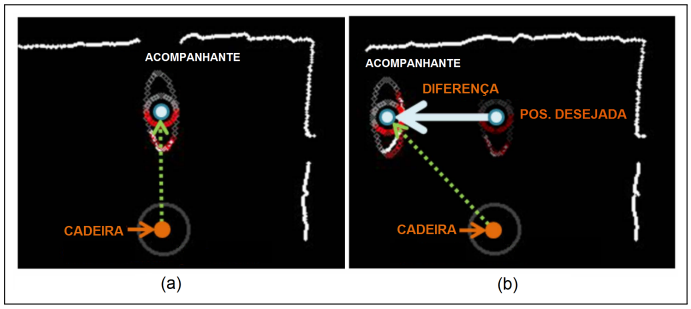
[4]Do inglês *Speeded Up Robust Features* é um recurso patenteado que serve de descritor e detector em visão computacional (BAY et al., 2008)

estimar a distância e posição do acompanhante. Em seguida, a cadeira de rodas utiliza um sistema de controle Fuzzy para controlar os motores e seguir o acompanhante, permitindo também desviar de obstáculos que possam vir a existir no percurso previamente calculado. Quando um obstáculo é detectado, o sistema Fuzzy muda suas regras de controle, dando prioridade para a ação de desvio. Se obtiveram resultados positivos para ambientes com uma ou mais pessoas.

Em um segundo trabalho, os mesmos autores substituíram os sensores anteriormente utilizados por um sensor RGB-D Kinect, da fabricante Microsoft®, permitindo adicionar ao projeto uma função de mapeamento e localização do tipo SLAM[5]. Para a detecção de candidatos a serem acompanhados, primeiramente são selecionados a partir da estimativa da largura dos ombros utilizando dados de profundidade, fornecidos pelo sensor Kinect e, após isso, o acompanhante é selecionado ao se detectar um padrão de texto que pode estar presente em uma placa ou na própria camiseta do acompanhante. Foram obtidos resultados semelhantes ao seu trabalho anterior, porém utilizando um sensor muito mais barato em comparação ao sensor de medição a laser (WU et al., 2013).

Kobayashi, Suzuki e Kuno (2012) desenvolveram uma cadeira de rodas capaz de seguir uma pessoa, ficando ao lado da mesma, permitindo assim uma melhor comunicação entre o cadeirante e o acompanhante. Se utilizou de um sensor de medição a laser para a estimativa da localização da pessoa e uma câmera omnidirecional para o reconhecimento da mesma, porém é necessário que o acompanhante forneça inicialmente sua posição através de um procedimento de calibração. Após a seleção do acompanhante, é criado um histograma de cores que armazena a aparência do mesmo, facilitando assim o acompanhamento em ambientes com mais de uma pessoa e, em caso de perda da visualização do acompanhante, o histograma criado é usado para a relocalização do mesmo. A cadeira possui ainda um *joystick* que permite ao usuário desativar o sistema autônomo e controlar a cadeira caso necessário. O controle da velocidade e direção da cadeira de rodas é realizado através de informações de posição e direção de movimento do acompanhante, sendo que a velocidade da cadeira é controlada através da diferença entre a posição atual da pessoa em relação à cadeira e a posição padrão, que é a posição desejada, método mostrado na Figura 5.

Figura 5: Acompanhante na: (a) Posição padrão (desejada) (b) Fora da posição



Fonte: Adaptado de Kobayashi, Suzuki e Kuno (2012).

[5]Do inglês *Simultaneous localization and mapping*

Existem trabalhos sobre técnicas *Track and Follow* também fora da área assistiva. Gupta et al (2017) desenvolveram um algoritmo de detecção para ser utilizado por um veículo robótico que acompanha uma pessoa, utilizando câmera e descritores/detectores para visão computacional SURF.

Também na parte de algoritmos de detecção, Hassan et al (2016) desenvolveram um algoritmo que detecta um quadro colorido que é posicionado junto a pessoa que se deseja seguir e o aplicaram em um robô móvel que, utilizando também sensores a laser e magnetômetros, executa a tarefa de seguir a mesma.

Ahmad et al (2015) desenvolveram um carro seguidor para auxiliar cadeirantes a carregar compras ou bagagem. Utiliza visão computacional para detectar um padrão de cores fixado na traseira da cadeira de rodas e sensores ultrassônicos para manter uma distância mínima da mesma, além de desviar de obstáculos.

Wunderlich, Schmölz e Kühnlenz (2017) desenvolveram um sistema *Track and Follow* onde um robô móvel, utilizando uma câmera e sensores de medição de distância a laser, consegue acompanhar uma pessoa ao detectar o movimento de uma das pernas. A detecção do acompanhante é realizada com o uso de histogramas.

Ye et al (2016) desenvolveram um veículo móvel do tipo pêndulo invertido capaz de seguir uma pessoa. Utilizando uma câmera especial modelo OptiTrack, um sensor Kinect e algoritmo de detecção próprio, o veículo é capaz de seguir uma pessoa ou outro veículo ao detectar um marcador reflexivo fixado no alvo desejado.

Ren et al (2016) utilizaram sensor de visão Kinect e algoritmos de rastreamento CamShift para detectar e acompanhar uma pessoa. A plataforma móvel inicia a rotina de acompanhamento quando detecta um movimento de mão pré definido.

Leigh et al (2015) utilizaram um veículo robô para avaliar um sistema *Track and Follow* que deve ser futuramente utilizado em uma cadeira de rodas inteligente. Esse sistema utiliza sensores de medição de distância a laser como principal ferramenta para detecção da pessoa a ser seguida, sendo que é avaliado a movimentação das pernas. Foi utilizado algoritmo próprio para o processamento dos dados do sensor e um controlador PID para o movimento do robô.

Bartak e Vyskovsky (2015) utilizaram uma abordagem do tipo rastreamento-aprendizagem-detecção (TLD) com o software de código aberto OpenCV e uma câmera. Esta abordagem foi implementada em um quadricóptero e o controle dos movimentos é realizado utilizando controlador PID.

Babaians et al (2015) desenvolveram um algoritmo que utiliza detecção de esqueleto a partir de um sensor Kinect juntamente com algoritmo OpenTLD para rastreamento visual. O projeto foi implementado em uma plataforma móvel.

Zhao et al (2015) desenvolveram um sistema para estimar a localização da origem de um sinal eletromagnético utilizando um emissor, que é posicionado junto à pessoa a ser acompanhada, e uma rede de receptores posicionados no robô móvel. A detecção é realizada utilizando algoritmos *Least-Square* e filtro de Kalman.

Já Han et al (2012) construíram uma rede de sensores de onda sonora com o objetivo de detectar a origem de um transmissor sonoro, assim possibilitando estimar a localização da pessoa que se deseja seguir, utilizando, como informação principal, a diferença entre os tempos de recepção do sinal pelos sensores.

A Tabela 1 mostra um resumo dos trabalhos acima citados e de outros já publicados.

Tabela 1: Resumo de trabalhos publicados sobre sistemas *Track and Follow*

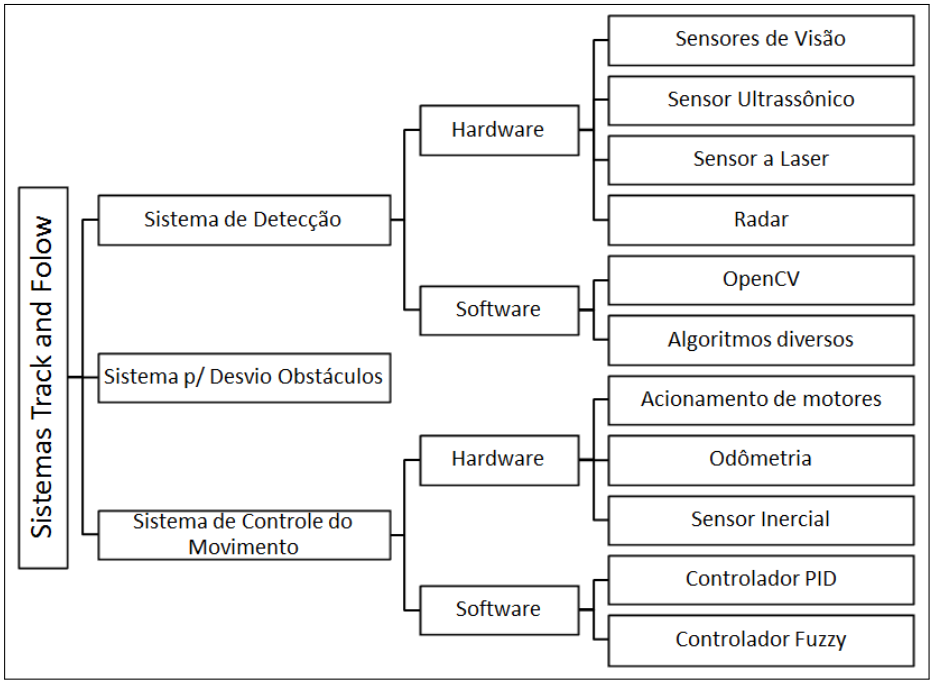
Autores	**Veículo**	**Sensores**	**Alg. Track**
Gupta et al (2017)	Veículo robô	Câmera	Algoritmo SURF classificador K-D
Wunderlich et al (2017)	Veículo robô	Câmera Sensor Laser	Histogramas
Hassan et al (2016)	Veículo robô	Câmera	Algoritmo próprio Padrão de cores
Ren et al (2016)	Veículo robô	Sensor Kinect	Kinect SDK Algoritmo CamShift
Ye et al (2016)	Veículo robô	Câmera Sensor Kinect	Fusão de sensores Algoritmo próprio
Barták e Vykovský (2015)	Drone	Câmera	Software OpenCV
Leigh et al (2015)	Veículo robô	Sensor Laser	Algoritmo proprio Filtro Kalman
Babaians et al (2015)	Veículo robô	Câmera	Software OpenTLD
Ahmad et al (2015)	Veículo robô	Câmera	Detecta padrão de cores
Zhao et al (2015)	Veículo robô	Sensor wireless	Algoritmo Genético
Markovic et al (2014)	Veículo robô	Câmera Ominidirecional	Distribuição Mises-Fisher

Tabela 1 continuação

Autores	Veículo	Sensores	Alg. Track
Motokucho e Oda (2014)	Cadeira motorizada	Câmera Estereoscópica	Vetores de fluxo óptico e Disparidade
Wu et al (2013)	Cadeira motorizada	Sensor Kinect	Algoritmo SURF
Pestana et al (2013)	Drone	Câmera	Software OpenTLD
Wu et al (2012)	Cadeira motorizada	Câmera Sensor Laser	Algoritmo SURF Padrão de texto
Kobayashi et al (2012)	Cadeira motorizada	Sensor a Laser Câmera Omnidirecional	Algoritmo próprio
Han et al (2012)	Veículo robô	Sensores sonoros	Tempo de recepção do sinal
Luo et al (2010)	Veículo robô	Sensores sonoros	Tempo de voo do sinal Filtro Kalman
Hu et al (2009)	Veículo robô	Câmera PTZ	Rosto, Cor da roupa Filtro de Partículas

Um breve resumo sobre esta tecnologia pode ser feito, baseado nos trabalhos avaliados. Primeiramente, os dispositivos de detecção mais utilizados são câmeras e sensores de visão, pelo fato da disponibilidade de tecnologias e algoritmos de processamento. Sensores a laser estão na segunda colocação em número de utilizações, sendo que são muito utilizados juntamente com sensores de visão. Uma tendência notada é a utilização do sensor Kinect, diminuindo muito o custo dos projetos. Sobre algoritmos de detecção, algoritmos SURF e softwares de código aberto são os mais utilizados para sensores de visão. Sobre o controle do movimento, poucos trabalhos forneceram informações sobre o mesmo, porém, baseado nos que disponibilizaram, verificou-se que o método mais utilizado é o sistema de controle Fuzzy, seguido pelo controlador PID. A Figura 6 mostra um diagrama resumido com os blocos construtivos básicos de um sistema *Track and Follow* com módulos, sensores e sistemas mais utilizados para detecção e controle de movimento.

Figura 6: Diagrama de blocos dos sistemas principais em tecnologias *Track and Follow*

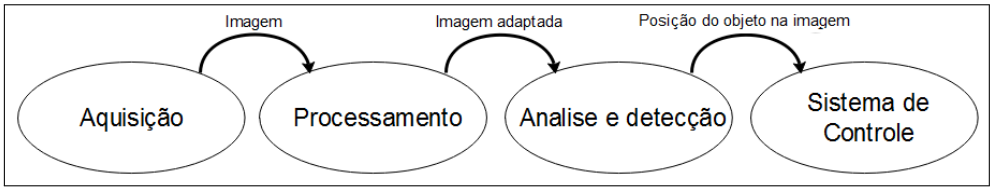


Fonte: Elaborado pelo Autor (2018).

2.5 SISTEMA DE DETECÇÃO POR IMAGEM

Em tecnologias *Track and Follow*, o sistema de detecção é o responsável por identificar um objeto ou a pessoa que se deseja acompanhar. Uma das técnicas existentes para a execução desta tarefa é por meio de análise de imagem. O diagrama mostrado na Figura 7 mostra um exemplo das etapas necessárias para se detectar um objeto em uma imagem. Primeiramente a mesma deve ser adquirida, através de uma câmera ou outro dispositivo. Em seguida a imagem deve ser processada, podendo sofrer transformações ou filtragens que adaptam a mesma com o objetivo de aumentar a eficiência da etapa seguinte, onde se realiza a análise da imagem e a detecção do objeto desejado através de alguma técnica de visão computacional. Por último a posição, em pixels, do objeto desejado pode ser enviado ao sistema de controle para que sejam executadas as devidas ações. Nessa seção serão discutidas algumas técnicas e algoritmos, existentes em visão computacional, para se detectar um objeto, e o próprio acompanhante, em uma imagem.

Figura 7: Etapas para detecção por imagem



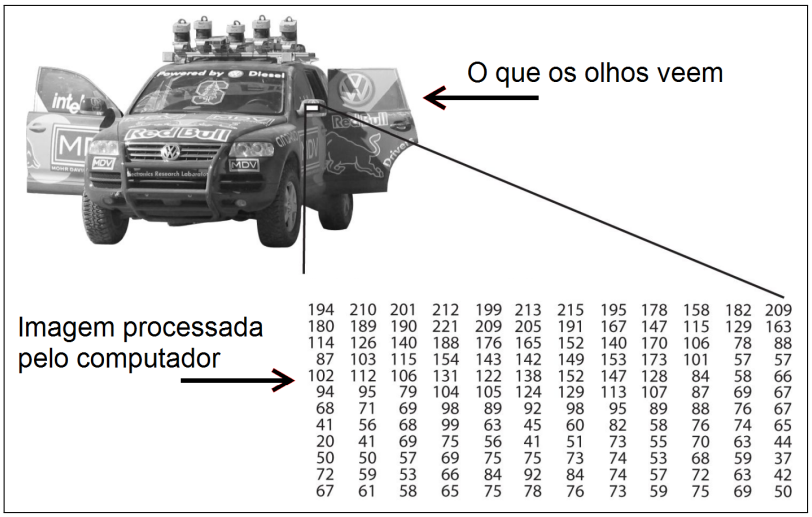
Fonte: Elaborado pelo Autor (2018).

2.5.1 Visão Computacional

Visão Computacional é uma tecnologia muito utilizada em sistemas *Track and Follow*. Esse é um campo de atuação vasto que está em rápido crescimento, seja pelo desenvolvimento de sensores de visão melhores e mais baratos ou pela criação de algoritmos de processamento de imagem mais rápidos e eficazes. Processos de visão computacional são transformações aplicadas em dados provenientes de imagens ou vídeo, com o objetivo de extrair características especificas dos mesmos, sendo a detecção de um dado objeto em diversas imagens ou o reconhecimento de vários itens em uma imagem (BRADSKI; KAEHLER, 2017).

A visão é um dos sentidos mais evoluídos nos seres humanos, logo pode parecer trivial a extração das informações acima citadas, porém, sistemas de visão computacional processam as informações visuais de maneira diferente, como mostrado na Figura 8, ou seja, computadores transformam informações visuais em números, onde cada *pixel* é representado por um valor numérico, neste caso em escala de cinza (BRADSKI; KAEHLER, 2017).

Figura 8: Uma imagem vista pelos olhos e pelo computador



Fonte: Adaptado de Bradski e Kaehler (2017, p. 4).

Alguns dos principais dispositivos utilizados em visão computacional são baseados em câmeras digitais comuns, como conjunto estereoscópicos e câmeras omnidirecionais.

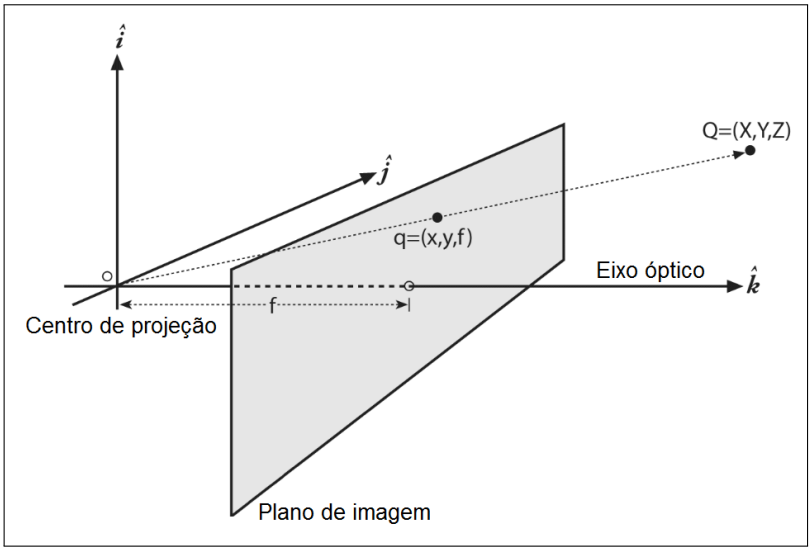
2.5.2 Modelo geométrico da câmera

O conhecimento do modelo de uma câmera auxilia na melhor compreensão do seu funcionamento. O principal modelo utilizado é o do tipo *pinhole*, mostrado na Figura 9. A imagem no ponto Q, com distância Z é representada no ponto q do plano de imagem, que pode ser considerado como o sensor de captura e que está distante de f do centro focal das lentes. O ponto central presente nesse plano é chamado de ponto principal. Esse modelo é ideal e, como

se sabe, a fabricação desses dispositivos necessita de procedimentos mecânicos que fazem com que esses valores não sejam precisos, fazendo com que o ponto principal esteja deslocado em relação ao ponto central do plano de imagem, causando distorções na informação capturada pelo sensor. Para a correção destas imperfeições se utilizam procedimentos de calibração, que consistem basicamente da captura de várias imagens de um padrão geométrico conhecido em orientações e posições diferentes, normalmente realizado com um tabuleiro de xadrez. A partir da análise das imagens é possível extrair a matriz intrínseca da câmera. Para se obter os pontos corrigidos (x,y) se deve realizar operações matriciais entre o ponto na imagem (X,Y,Z) e a matriz intrínseca da câmera M, operação demonstrada em 2.1. A matriz M fornece os valores das deformações geométricas presentes onde c_x e c_y são deslocamentos lineares nos dois eixos entre o centro do sensor e o focal e f_x e f_y são o produto entre a distância focal, em mm, e a unidade de pixels em cada eixo, em pixels/mm. As bibliotecas do OpenCV fornecem diversas funções para a realização destas tarefas, logo não se irá aprofundar neste trabalho toda a teoria matemática envolvida no processo, caso desejado o mesmo pode ser visto em Bradski e Kaehler (2017) e em Medioni e Kang (2004).

$$M_{3x1} = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix} >> \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \end{bmatrix} \tag{2.1}$$

Figura 9: Modelo para câmera tipo *pinhole*



Fonte: Adaptado de Bradski e Kaehler (2017).

2.5.3 Técnicas para Detecção de Objetos

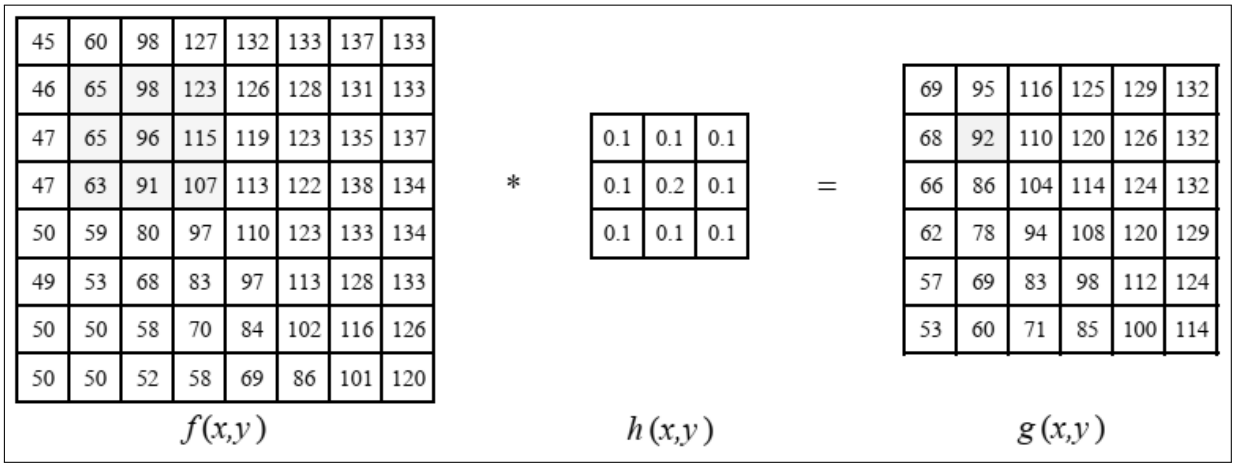
Detecção de objetos em imagens é uma tarefa importante para a área da visão computacional e também para tecnologias *Track and Follow*. Possui o objetivo de identificar, em

uma imagem qualquer, a presença de determinado objeto, forma geométrica ou característica específica (KHURANA; AWASTHI, 2013).

A primeira etapa em um sistema de detecção por imagem é o processamento da mesma. Algumas das principais técnicas disponíveis são a utilização de filtros e transformações morfológicas e de cor. Normalmente, em processos de filtragem, se cria uma pequena janela de tamanho fixo, também chamada de *Kernel*, para a execução de uma varredura por toda a imagem, onde se aplica alguma função matemática, sendo essa normalmente uma convolução ou correlação, entre os valores dos pixels da imagem presentes dentro do Kernel e os valores do mesmo. A operação de correlação não é nada mais que uma soma ponderada entre os valores anteriormente citados, mostrada matematicamente em 2.2, onde g é a imagem tratada, f representa a imagem original e h é o Kernel. A Figura 10 demonstra o processo de convolução entre os valores de pixel de uma imagem e um dado Kernel, onde o campo escurecido em f representa a sobreposição do Kernel sobre a imagem e o campo escurecido em g representa o valor do pixel resultante (SZELISKI, 2010).

$$g(x,y) = \sum_{k,l} f(x+k, y+l)h(k,l) \tag{2.2}$$

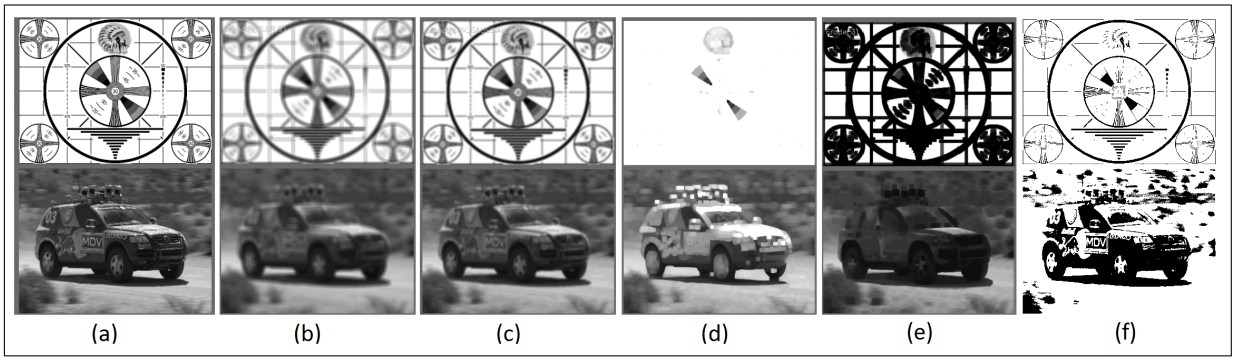
Figura 10: Operação de convolução com o uso de Kernel



Fonte: Adaptado de Szeliski (2010).

Vários filtros utilizam da técnica acima citada para executar o processamento de imagens. A Figura 11 mostra alguns exemplos destes filtros, como o filtro de suavização, o filtro gaussiano, o filtro de erosão e de dilatação. A figura também mostra um exemplo de binarização, que não utiliza de Kernel mas é muito comumente utilizada, onde é aplicado um valor de limiar em uma imagem em escala de cinza, e os valores dos pixels maiores que o valor de limiar são convertidos para a cor preta e, os com valor menor, em cor branca (BRADSKI; KAEHLER, 2017).

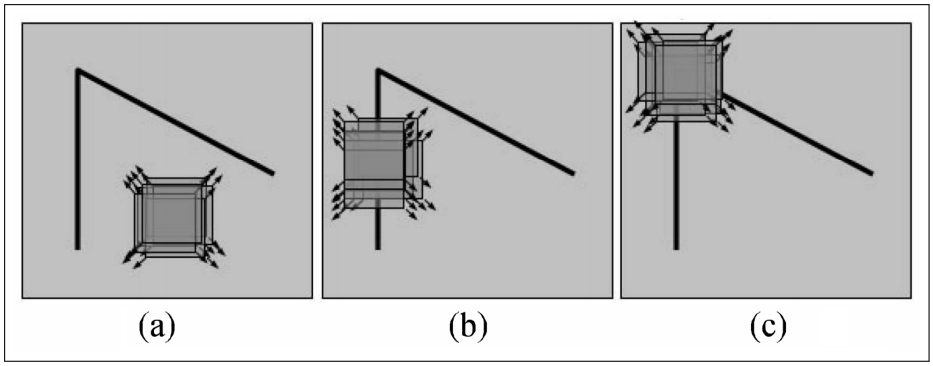
Figura 11: Operações em imagens: (a) Imagem Original (b) Filtro de Suavização (c) Filtro Gaussiano (d) Filtro de Erosão (e) Filtro de Dilatação (f) Binarização



Fonte: Adaptado de Bradski e Kaehler (2017).

Com a imagem filtrada e tratada, se podem aplicar algoritmos que possuem a função de detectar certas características da imagem. Uma destas características são cantos de imagem, ou *corners*. Cantos, ou cruzamento de linhas, são uma das características mais presentes em imagens, onde que a sua correta detecção é base para técnicas de calibração de câmeras, sendo que um dos principais algoritmos utilizados é o detector Harris (ZOU et al., 2008). Esse algoritmo utiliza uma janela que é movimentada por toda uma imagem em escala de cinza. A cada ponto é realizado o cálculo do gradiente da cor para todas as direções, caso o valor resultante seja maior que um limiar selecionado para pelo menos duas direções, isso indica um cruzamento no centro da janela (QIAO; TANG; LI, 2013). A Figura 12 mostra a execução deste algoritmo, onde em (a) não é presente nenhuma linha dentro da janela, assim a mesma é movimentada até localizar uma variação considerável do gradiente, como em (b). Nesse caso a janela é movimentada na direção do gradiente localizado e, caso se detecte outra variação, o ponto é armazenado como um cruzamento, mostrado em (c).

Figura 12: Detecção de cantos por algoritmo Harris

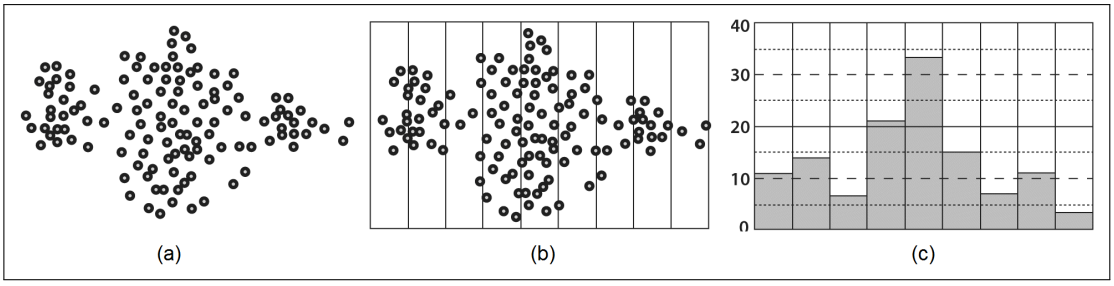


Fonte: (QIAO; TANG; LI, 2013)

Além de formas geométricas, também pode-se detectar características mais complexas em imagens, como um padrão de cor ou um objeto mais complexo. Uma das ferramentas disponíveis para a execução desta tarefa são os Histogramas. Servem para representar características

de imagens, como distribuição de cores, padrões de borda e pontos de interesse em forma de matrizes ou vetores de dados. A Figura 13 mostra a construção de um histograma a partir de um conjunto de pontos em uma imagem (BRADSKI; KAEHLER, 2017).

Figura 13: Construção de um histograma: (a) Conjunto de pontos (b) Malha para contagem (c) Histograma criado



Fonte: Adaptado de Bradski e Kaehler (2017).

Ao se possuir a representação, em forma de histograma, de dado objeto, o mesmo pode ser reconhecido em uma imagem ao se efetuar uma comparação, que pode ser realizada com imagens completas ou somente fragmentos. A partir dos histogramas H_1 e H_2, alguns dos métodos matemáticos disponíveis para comparação podem ser aplicados, como o de correlação, onde se utiliza a equação 2.3, o de intersecção, representado pela equação 2.4, e de distância de Bhattacharyya, através da equação 2.5 (BRADSKI; KAEHLER, 2017).

$$d_{correl}(H_1, H_2) = \frac{\sum_i H_1^T(i) \cdot H_2^T(i)}{\sqrt{\sum_i (H_1^T(i))^2 \cdot \sum_i (H_2^T(i))^2}} \tag{2.3}$$

$$d_{inters}(H_1, H_2) = \sum_i min(H_1(i), H_2(i)) \tag{2.4}$$

$$d_{Bhatt}(H_1, H_2) = \sqrt{1 - \frac{\sum_i H_1(i) \cdot H_2(i)}{\sqrt{\sum_i H_1(i) \cdot \sum_i H_2(i)}}} \tag{2.5}$$

Existem diversas outras técnicas de visão computacional que podem ser utilizadas para a detecção de um objeto, uma delas é o Detector de Borda de Canny, que foi desenvolvido para filtrar texturas. Como diz Canny (1986) processos de detecção de borda simplificam a analise de imagens, pois permitem que se mantenha somente a informação útil. Diz também que uma técnica de detecção de borda deve ser precisa ao estimar a presença de bordas, deve marcar os pontos detectados o mais próximo da borda verdadeira e fornecer somente uma detecção para cada borda. A Figura 14 mostra um exemplo da aplicação dessa transformação e o processo de identificação ocorre com os seguintes passos (XU et al., 2014):

- Remoção de ruído com um filtro Gaussiano;
- Estimativa da magnitude e direção dos gradientes Gx e Gy para cada pixel;

- Se a direção do gradiente for uma das oito possíveis (0°, 45°, 90°, 135°, 180°, 225°, 270°, 315°) ele é comparado com seus dois vizinhos na direção do gradiente e, se a magnitude do pixel for maior que dos seus vizinhos, a magnitude é preservada, senão é zerada;
- Aplica limiares máximos e mínimos para as magnitudes dos gradientes

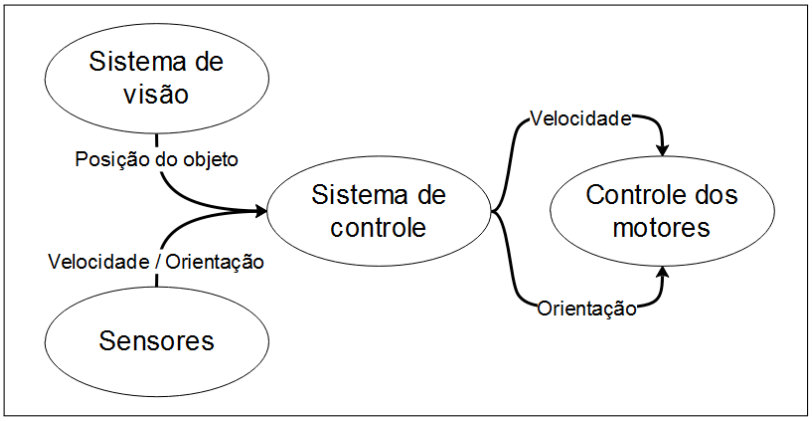
Figura 14: Aplicação do detector de borda Canny: (a) Imagem Original (b) Bordas por Canny

Fonte: Adaptado pelo Autor (2018).

2.6 SISTEMAS PARA CONTROLE DO MOVIMENTO

Após o objeto ou pessoa a ser acompanhada serem identificadas, o sistema de controle do veículo deve realizar os cálculos e movimentos necessários para executar a tarefa de acompanhamento. A Figura 15 mostra um sistema de controle aplicado a sistemas *Track and Follow*, onde o sistema de visão informa a posição, na imagem, do objeto, ou pessoa, que deve ser seguido e, juntamente com informações vindas de sensores, é executada a ação de controle dos motores para a execução da tarefa de acompanhamento. Nesta seção serão descritos alguns sistemas de controle e sensores utilizados para a execução desta tarefa.

Figura 15: Diagrama de um sistema de controle para tecnologias *Track and Follow*



Fonte: Elaborado pelo Autor (2018).

2.6.1 Sistemas de Controle

Atualmente existem diversas técnicas para controle do movimento de veículos elétricos autônomos, os principais e mais utilizados são o controlador PID e técnicas de controle baseadas em *Lógica Fuzzy*. A seguir será realizada a revisão bibliográfica sobre o primeiro desses.

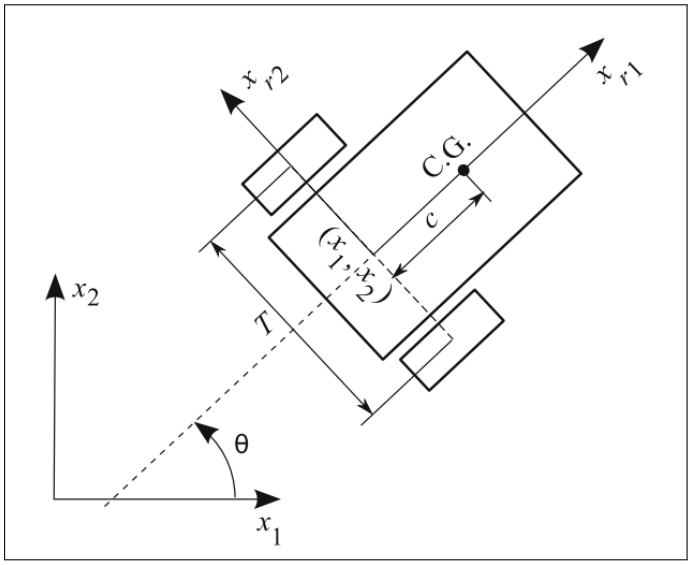
2.6.1.1 Modelo Cinemático

Segundo Fahimi (2008) existem dois tipos principais de veículos móveis terrestres, o tipo triciclo (*Hilary-type*) e o tipo carro (*car-like*), sendo que os tipo triciclo realizam o controle de direcionamento e tração pelas duas rodas principais e possuem uma roda castor somente para apoio.

Para a dedução das equações cinemáticas para um veículo tipo triciclo, Fahimi (2008) utilizou o modelo apresentado na Figura 16. Primeiramente considerou que o veículo exerce somente velocidades baixas e que não existe movimento lateral, ou seja, o veículo não desliza em nenhuma direção. Os eixos x_{r1} e x_{r2} representam os eixos cartesianos do veículo, sendo que o ponto (x_1, x_2) é o ponto de origem e a variável θ indica o ângulo entre o plano inercial e o plano do veículo. A partir destas informações pode-se representar a configuração geométrica do veículo em qualquer instante de tempo por (FAHIMI, 2008, p. 165):

$$q = \begin{bmatrix} x_1 \\ x_2 \\ \theta \end{bmatrix} \tag{2.6}$$

Figura 16: Modelo cinemático para veículo tipo triciclo



Fonte: (FAHIMI, 2008, p. 164).

Em seguida, assumindo que o veículo está se movimentando com velocidades linear v e angular w, Fahimi (2008) estima as componentes vetoriais do ponto (x_1, x_2) e a taxa de variação da orientação do veículo $\dot{\theta}$ como nas equações em 2.7:

$$\dot{x_1} = v\cos\theta \quad ; \quad \dot{x_2} = v\sin\theta \quad ; \quad \dot{\theta} = w \tag{2.7}$$

Ao final, Fahimi (2008) combina as equações em 2.7 e especifica as equações cinemáticas do movimento do veículo, na sua forma vetorial, como na equação 2.8:

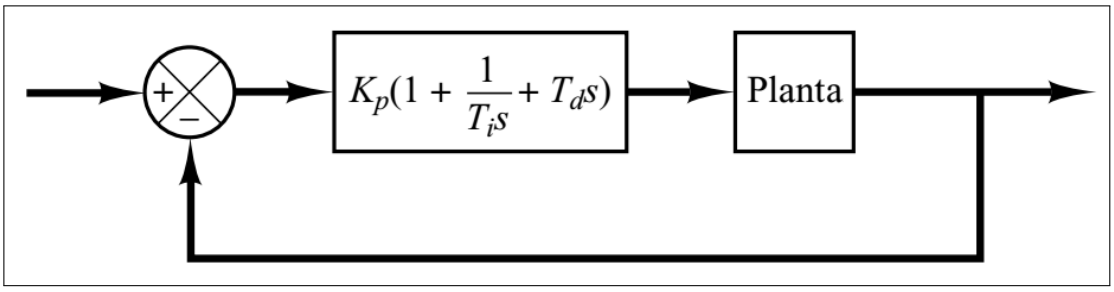
$$\dot{q}_{3x1} = \begin{bmatrix} \cos\theta & 0 \\ \sin\theta & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} v \\ w \end{bmatrix} \tag{2.8}$$

2.6.1.2 Controlador PID

O controlador Proporcional-Integral-Derivativo (PID) é um dos mais utilizados algoritmos de controle em veículos elétricos e até em ambientes industrias, sendo que mais da metade dos controladores industriais, utilizam esta técnica ou uma de suas variantes. (OGATA, 2009).

A Figura 17 mostra o diagrama de blocos de um controlador PID, onde *Kp* é o valor de controle proporcional, *Ti* é o tempo de controle integral e *Td* é o tempo de controle derivativo. Esses três valores são chamados de parâmetros do controlador. O bloco *Planta* representa o modelo matemático do sistema em que se irá utilizar o controlador, caso disponível (OGATA, 2009).

Figura 17: Diagrama de blocos de um controlador PID

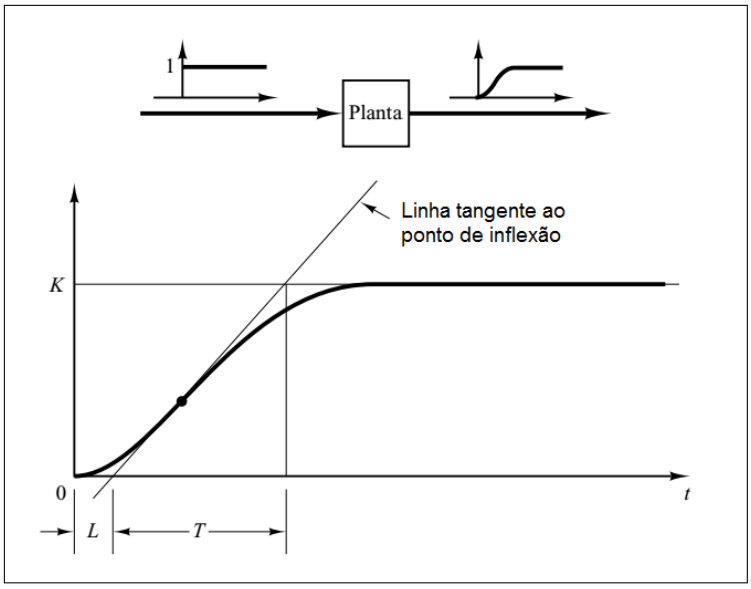


Fonte: (OGATA, 2009, p. 568).

Caso o modelo matemático não esteja disponível ou seja de difícil dedução, deve-se utilizar metodologias experimentais desenvolvidas para a realização do ajuste do controlador. Uma das técnicas mais utilizadas são os métodos de *Ziegler-Nichols*. No primeiro método, é aplicado um degrau de excitação no sistema em malha aberta, como mostrado na Figura 18 e, a partir da curva de resposta da saída, são determinados os valores dos parâmetros do controlador através da Tabela 2. Deve-se desenhar uma reta tangente ao ponto de inflexão da curva de resposta para que os valores de *L* e *T* sejam obtidos, que representam respectivamente, o atraso

de transporte e o tempo entre o cruzamento da reta nos pontos mínimo e máximo (OGATA, 2009).

Figura 18: Primeiro método de Ziegler-Nichols



Fonte: Adaptado de Ogata (2009, p. 569).

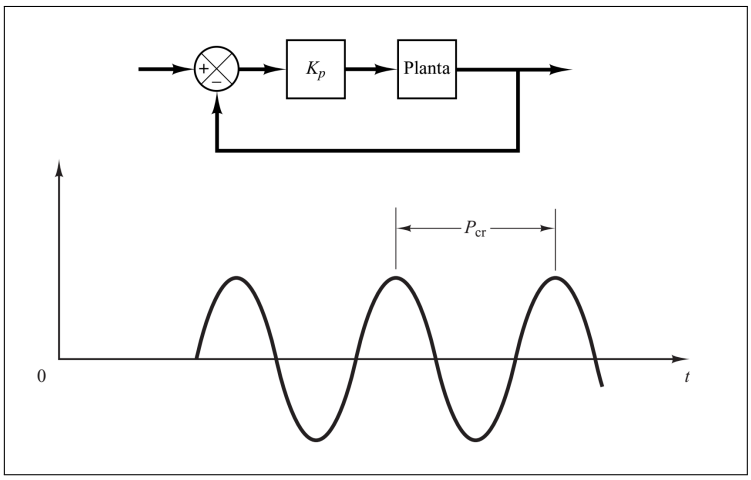
Tabela 2: Parâmetros para primeiro método Ziegler-Nichols

Tipo de Controlador	K_p	T_i	T_d
P	$\frac{T}{L}$	∞	0
PI	$0.9\frac{T}{L}$	$\frac{L}{0.3}$	0
PID	$1.2\frac{T}{L}$	$2L$	$0.5L$

Fonte: (OGATA, 2009, p. 570).

O segundo método é realizado ao se aplicar, em malha fechada, um controlador proporcional em série à planta, onde aumentando-se o ganho do controlador gradualmente, até que o sistema entre em um regime de oscilação sustentada, como mostrado na Figura 19. O valor do ganho do controlador nesse ponto é chamado de *ganho crítico* e é utilizado para a estimação dos valores dos parâmetros do controlador a partir da Tabela 3, juntamente com o valor P_{cr}, que é o período de oscilação da resposta do sistema (OGATA, 2009).

Figura 19: Segundo método de Ziegler-Nichols



Fonte: Adaptado de Ogata (2009, p. 570).

Tabela 3: Parâmetros para segundo método Ziegler-Nichols

Tipo de Controlador	K_p	T_i	T_d
P	$0.5K_{\text{cr}}$	∞	0
PI	$0.45K_{\text{cr}}$	$\frac{1}{1.2}P_{\text{cr}}$	0
PID	$0.6K_{\text{cr}}$	$0.5P_{\text{cr}}$	$0.125P_{\text{cr}}$

Fonte: (OGATA, 2009, p. 571).

Resta constar que estes métodos de ajuste não estimam os valores finais dos parâmetros do controlador, somente fornecem um ponto de início. Em seguida deve-se realizar um ajuste fino dos valores dos parâmetros até que se obtenha a resposta desejada (OGATA, 2009).

Para que se possa utilizar um sistema de controle PID em um ambiente digital, como em microcontroladores, deve-se realizar a *discretização* do mesmo, transferindo o controlador do meio contínuo para o discreto. Os principais métodos de aproximação utilizados são o *Forward*, o *Backward* e o método de *Tustin*, também chamada de Transformação Bilinear. Para ser realizada a aproximação por Tustin, deve-se ter a função de transferência na forma laplaciana, como da equação 2.9, onde *u(s)* é a resposta do sistema e *e(s)* é o erro da saída realimentado, e substituir os valores de *s* pela equação 2.10, onde T_s é o tempo de amostragem do sinal, convertendo assim o controlador para o domínio z, resultando nas parcelas do controlador PID, mostradas nas equações 2.11, 2.12 e 2.13 em formato de equações de diferença (FADALI, 2009).

$$u(s) = K_p(1 + \frac{1}{T_i s} + T_d s)e(s) \tag{2.9}$$

$$s = \frac{2}{T_s}\frac{z-1}{z+1} \tag{2.10}$$

$$P(k) = K_p e(k) \tag{2.11}$$

$$I(k) = I(k-1) + \frac{K_i T_s (e(k) - e(k-1))}{2} \tag{2.12}$$

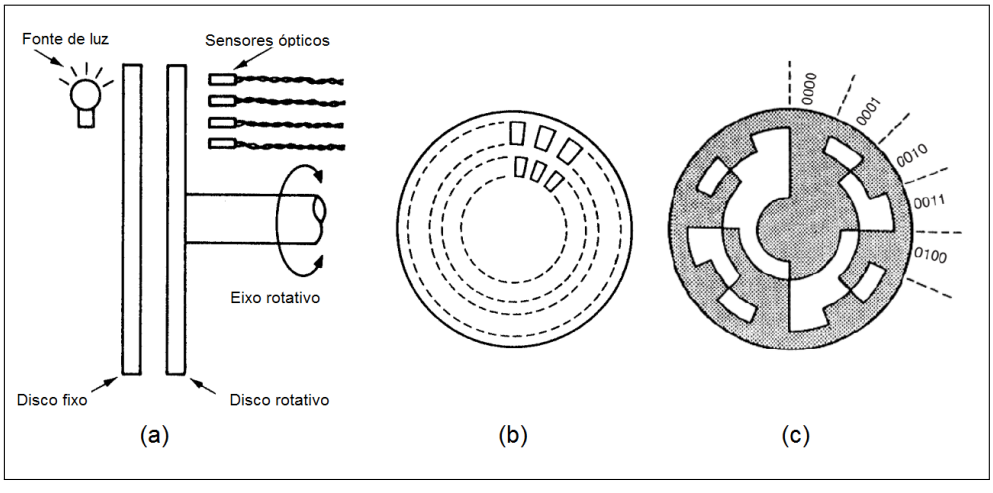
$$D(k) = \frac{2pK_d(e(k) - e(k-1)) + D(k-1)(2 - pT_s)}{2 + pT_s} \tag{2.13}$$

2.6.2 Odômetros

Odômetros são dispositivos utilizados para a estimação da velocidade de um veículo. Os principais sensores utilizados para este fim são os encoders, que são dispositivos ópticos em que um disco com várias aberturas é submetido a um feixe de luz e, ao se detectar a luz que passa por essas aberturas durante o movimento rotacional, é possível de se estimar a velocidade de rotação e a posição do eixo. Existem dois tipos principais de encoders, os incrementais e os absolutos, onde os incrementais utilizam discos com aberturas equidistantes, permitindo assim a detecção somente da velocidade do eixo. Já os absolutos possuem discos onde as aberturas são posicionadas de tal maneira a permitir que seja possível de se estimar também a posição absoluta do eixo (MORRIS, 2001).

A Figura 20(a) mostra um exemplo de construção de um encoder óptico, onde o disco fixo é utilizado para guiar os feixes de luz emitidos Também na Figura 20(b) e (c) são apresentados, respectivamente um disco incremental e um disco absoluto com codificação binária.

Figura 20: Estrutura de um *encoder* óptico: (a) Estrutura física de um *encoder* óptico (b) Disco de *encoder* incremental (c) Disco de *encoder* absoluto codificado

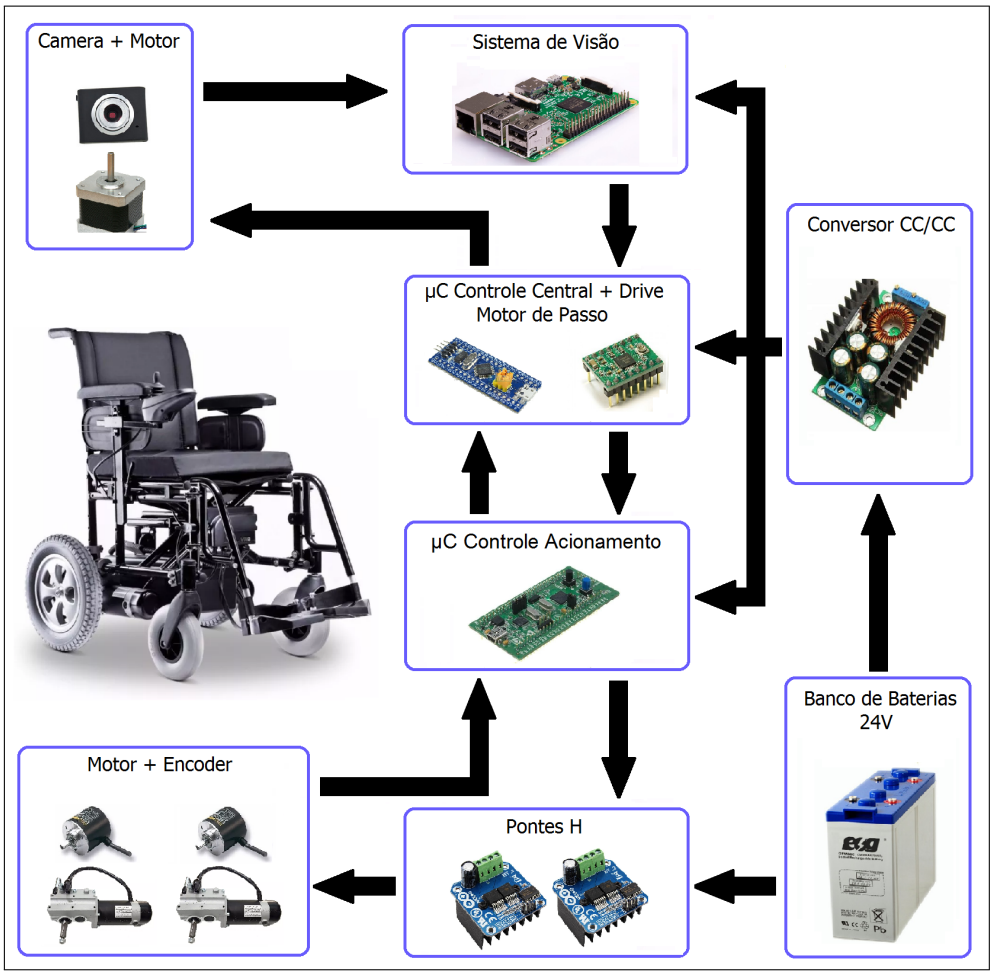


Fonte: Adaptado de Morris (2001).

3 DESENVOLVIMENTO DO PROJETO

Nesse capitulo serão descritos os procedimentos práticos desenvolvidos para a execução da proposta do trabalho. Primeiramente será apresentada uma visão geral do sistema proposto, indicando todos os componentes presentes e, em seguida, serão demonstrados em detalhes cada bloco presente no projeto. A estrutura básica do projeto implementado pode ser vista na Figura 21, sendo formado por três blocos principais, o bloco de visão, o bloco de controle principal e o bloco de controle de velocidade.

Figura 21: Projeto Desenvolvido



Fonte: Elaborado pelo Autor (2018).

O primeiro deles é formado por um microcomputador Raspberry Pi 3 e uma *Webcam*. Neste bloco foram implementadas as rotinas de processamento de imagem para a detecção do acompanhante. As imagens são capturadas pela *webcam* que está fixada em um motor de passo utilizado para a rotação da mesma, permitindo o enquadramento do alvo fixado no acompanhante no sistema de rastreamento. Os dados de posicionamento do acompanhante são enviados ao bloco seguinte através do barramento USB do microcomputador.

O segundo bloco é composto por um microcontrolador STM32, que é responsável pelo

calculo dos processos de controle da velocidade das rodas da cadeira, através de um controlador PI, e do ajuste do ângulo de rotação do motor em que a *webcam* está fixada, utilizando um controlador proporcional. Esse microcontrolador recebe os dados de posicionamento do acompanhante pelo sistema de visão através de um conversor USB/Serial e se comunica com o próximo bloco através de um barramento Serial, enviando para o mesmo a velocidade desejada para cada uma das rodas.

O terceiro bloco, também constituído principalmente de um microcontrolador STM32, recebe as informações de velocidade do bloco anterior e realiza a tarefa de controle da velocidade de rotação dos dois motores das rodas da cadeira através de um controlador PI. O acionamento dos motores é realizado através de duas pontes H comerciais, que por meio de um controle PWM fornece um nível de tensão DC usado na alimentação dos motores. A velocidade é controlada através da variação do *duty-cycle* do sinal PWM que impacta diretamente no valor médio da tensão de armadura fornecida ao motor. A velocidade das rodas são estimadas através do uso de 2 *encoders* óticos do tipo emissor-receptor infravermelho.

Para a alimentação de tensão de todo o sistema embarcado foram utilizadas duas baterias de 12V ligadas em série, sendo que a alimentação dos motores principais é feita diretamente pelas baterias e o sistema eletrônico é alimentado por um conversor CC-CC, que utiliza a tensão de 24V para fornecer um nível de tensão de saída de 5V.

3.1 Bloco de Visão

Existem diversos métodos para a detecção de um objeto em visão computacional. Neste trabalho foi utilizada, por simplicidade, a função *findchessboardcorners()*, disponível na biblioteca OpenCV. Esta função foi desenvolvida pelos criadores da biblioteca em questão para ser utilizada em procedimentos de calibração de câmeras. De forma simplificada, esta função executa uma varredura em uma imagem em busca de um padrão geométrico quadriculado em forma de jogo de xadrez e, caso esteja presente na imagem o dado padrão, fornece os pontos na imagem, em formato de coordenadas x e y, em que se ocorrem os cruzamentos, ou *corners*, entre os quadrados do padrão de xadrez. A forma completa da função, e a explanação de cada variável, podem ser vistas abaixo:

bool findChessboardCorners(InputArray image, Size patternSize, OutputArray corners, int flags);

- *image*: Variável do tipo Array de entrada em que se insere a imagem a ser analisada, deve ser do tipo escala de cinza em 8 bits;
- *patternSize*: Variável do tipo Size (especial do OpenCV) em que se indica o numero de cruzamentos, ou *corners*, entre os quadrados do padrão de xadrez;
- *corners*: Variável do tipo Array de saída onde o algoritmo fornece os pontos na imagem em que estão presentes os corners do padrão de xadrez;

- *flags*: Aqui se podem utilizar quatro tipos de flags, em que pode-se aumentar a velocidade de processamento ou melhorar a precisão de detecção;
- A função retorna os valores booleanos *TRUE* ou *FALSE* para informar da correta, ou não, detecção.

O algoritmo implementado com a utilização da função acima citada, em que o código em C é mostrado no Apêndice A, realiza os seguintes passos, demonstrados na Figura 22:

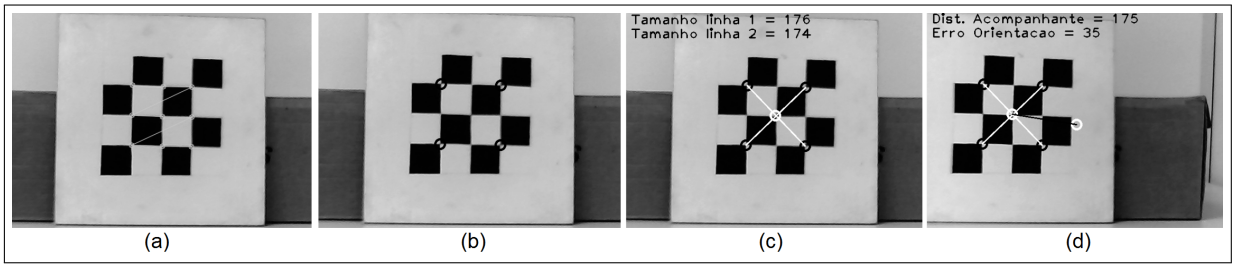
a) Detecção dos pontos de intersecções internas do padrão de xadrez, informação fornecida pela função *findchessboardcorners()* através do vetor de saída;

b) É analisado o posicionamento dos pontos fornecidos e são separados os presentes nos cantos externos;

c) Utilizando o Teorema de Pitágoras é estimado o tamanho das linhas, em X, formadas pela conexão entre os pontos externos. Esse valor é o numero de pixels entre os pontos. Também é estimado o ponto central através do calculo do valor médio entre os pontos externos nos eixos x e y;

d) Por fim são capturadas as informações que devem ser enviadas para o bloco de controle, que são a distância do acompanhante, representada pelo valor médio, em pixels, do tamanho das linhas acima citadas, e o valor, também em pixels, do erro de posicionamento na imagem, esse representado pela distancia entre o ponto central do padrão de xadrez e o centro da imagem no eixo x, pontos esses desenhados na Figura 22 (d).

Figura 22: Passos executados para a detecção



Fonte: Elaborado pelo Autor (2018).

Após o processo de detecção, é enviado para o bloco de controle, através de um conversor USB-Serial, as informações da distância do acompanhante e a diferença entre os centros do padrão de xadrez e da imagem, essas citadas acima. Esses dados serão utilizados como valores de referência para o cálculo da velocidade que se deve exercer, com a cadeira de rodas, para que se execute a tarefa de acompanhamento.

O algoritmo demonstrado na Figura 23 demonstra os passos realizados pela função *findchessboardcorners()* para a detecção do padrão de xadrez que são:

a) Primeiramente, se fornece ao algoritmo uma imagem em escala de cinza, com resolução de 320x240 *píxels*, e o número de cruzamentos internos dos quadrados, nesse caso de tamanho 3x3. O algoritmo então realiza uma binarização da imagem, utilizando um valor de limiar variável a partir da análise do gradiente do histograma da imagem em escala de cinza. Isso permite que se utilizem valores de limiares diferentes para pontos diversos da imagem, reduzindo a perda de informação de textura;

b) Se realiza uma dilatação na imagem binarizada. Esse processo executa um engrossamento nas linhas brancas da imagem com o uso de um Kernel com tamanho 3x3, onde o pixel da imagem presente no centro do Kernel é substituído pelo maior valor presente no interior da parte sobreposta pelo Kernel;

c) Em seguida se detectam os contornos da imagem com a utilização do detector Canny (CANNY, 1986) e se realiza a ligação de linhas para tentar formar quadrados. Essa conexão é realizada ao se alongar uma linha até que toque em outra linha;

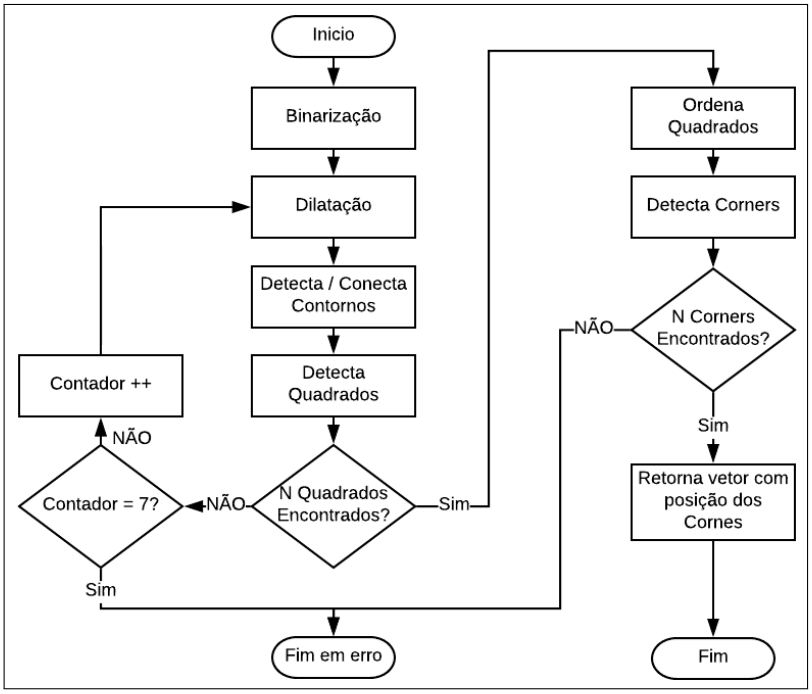
d) Após, por loops de comparação e de tentativa e erro, se buscam quadrados completos na imagem. Essa busca é realizada ao analisar se o ângulo entre as linhas, no ponto de contato, são próximos de $90°$. Após se realizam mais comparações para ver se as linhas se conectam com pelo menos duas outras, criando assim um dos lados do quadrado e, caso essas outras linhas se conectem em uma outra linha, define-se um quadrado. Por fim se filtram detecções dobradas ou incorretas e se criam vetores com os pontos, na imagem, em que os quadrados estão presentes;

e) Se forem corretamente encontrados o número mínimo para conseguir formar um padrão de xadrez do tamanho desejado, se vai adiante no algoritmo. Se não foram encontrados, se realiza a dilatação novamente, retornando ao ponto em b). Se após sete tentativas não se detectou os quadrados, a função é terminada em erro;

f) Se forem encontrados os quadrados, em seguida eles são organizados, ou seja, se cria um vetor com as suas respectivas posições na imagem de maneira ordenada, realizando uma varredura de cima para baixo e da esquerda para a direita na imagem;

g) Com os quadrados ordenados se analisa se os mesmos estão próximos, no caso se estão conectados ou muito próximos, e após se buscam os pontos de cruzamento, ou *corners*, do padrão de xadrez. Se o número de *corners* for igual ao informado ao algoritmo, se retorna um vetor com as posições na imagem dos pontos de cruzamento. Se não forem detectados os pontos desejados, se retorna em erro.

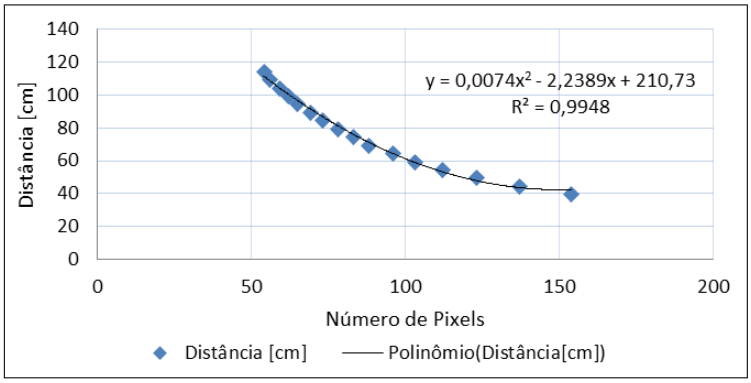
Figura 23: Algoritmo da função *findchessboardcorners()*



Fonte: Elaborado pelo Autor (2018).

Como ja dito anteriormente, o valor de distância do acompanhante é estimado através da media dos tamanhos das linhas desenhadas no padrão de xadrez, valor esse em *píxels*. Esse valor aumenta ao passo que o acompanhante se aproxima e diminui ao que o mesmo se afasta. A Figura 24 mostra a relação entre a distância real, em centímetros[cm], entre a câmera e o padrão de jogo de xadrez e o número de pixels capturado pelo bloco de visão para a dada distância. Essa captura de valores foi realizada em bancada, onde o padrão de xadrez foi fixado e a câmera foi movimentada. Pontos de medição foram realizados em intervalos de 5cm entre 40cm e 115cm, medição essa feita com a utilização de uma trena comercial da marca Starrett. A linha de tendência resultante foi do tipo polinomial, sendo que a equação de conversão resultante é $Dist = 0.0074p^2 - 2,2389p + 210,73$ com coeficiente de correlação de 0,9948.

Figura 24: Relação entre a distância real medida e o valor de pixels

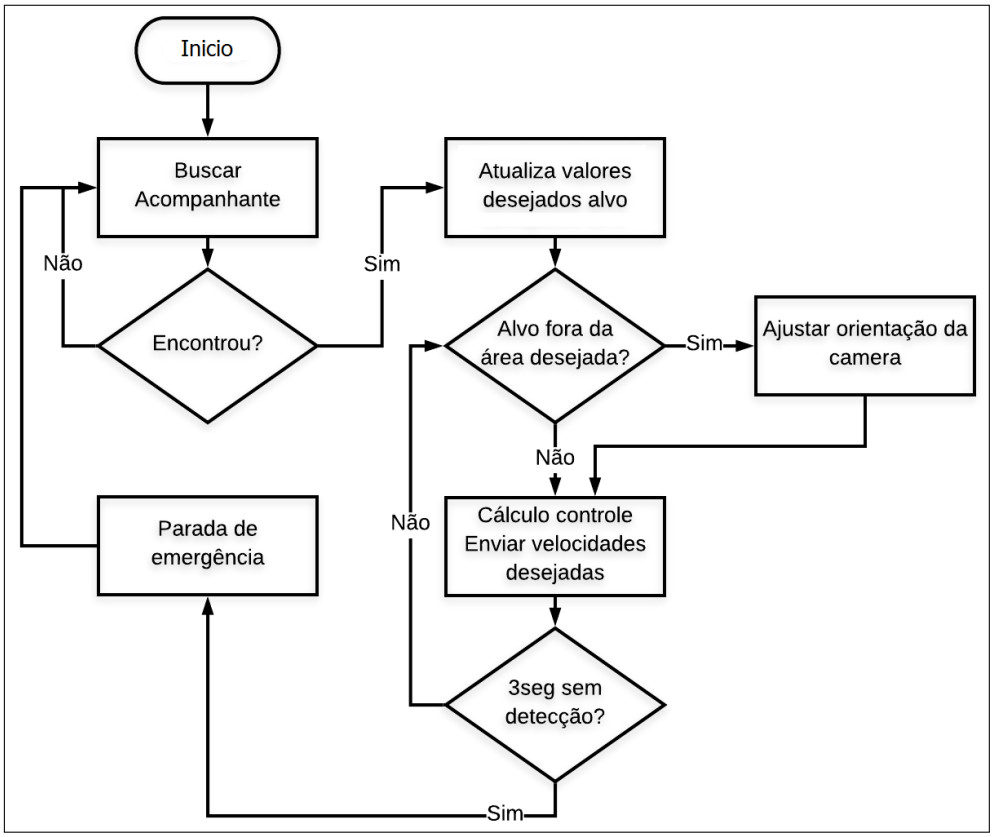


Fonte: Elaborado pelo Autor (2018).

3.2 BLOCO DE CONTROLE

Nesse bloco são realizadas as rotinas de controle principais, onde são executadas as tarefas de decisão da velocidade que a cadeira de rodas deve exercer, assim como o ajuste do campo de visão da câmera por meio de um motor de passo. A Figura 25 mostra um fluxograma simplificado do processo executado neste bloco, constituído pelas seguintes etapas:

Figura 25: Algoritmo de controle.



Fonte: Elaborado pelo Autor (2018).

1) Primeiramente é feita uma varredura com a câmera em busca do alvo carregado pelo acompanhante, realizada com uma rotação da mesma utilizando um motor de passo. Ao se receber os dados do bloco de visão, indicando a detecção do alvo, os valores de distância do acompanhante, vindos do bloco de visão, e posição angular da câmera, adquirida através da contagem dos pulsos do motor de passo, no momento da detecção são armazenados na memória, sendo utilizados como pontos de referência (valores desejados) em que se deseja manter o acompanhante;

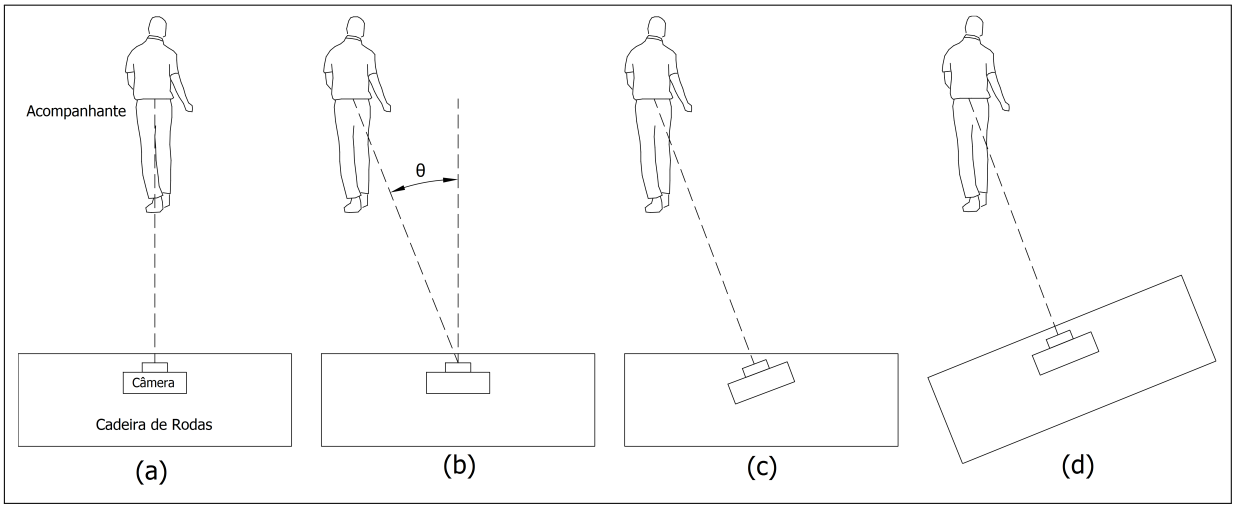
2) A cada nova recepção de dados, vindos do bloco de visão, caso o valor do erro de posicionamento na imagem, representado pela distancia entre o ponto central do padrão de xadrez e o centro da imagem no eixo x em pixels, é realizado um ajuste na posição angular da câmera ao se executar um número de pulsos no motor de passo, esse valor sendo

calculado com a utilização de um controlador proporcional em malha aberta, fazendo com que o alvo esteja sempre próximo do centro da imagem. A posição angular da câmera é então atualizada na memória;

3) Em seguida, baseando-se na diferença entre a distância desejada e a atual, informação essa vinda do bloco de visão, se acelera ou desacelera a cadeira de rodas e, caso exista uma diferença entre a posição angular atual da câmera e a detectada no momento da primeira detecção, são realizados pequenos ajustes na orientação da cadeira ao se acelerar uma roda e desacelerar a outra proporcionalmente, tentando-se manter o ângulo de rotação da câmera na posição desejada no momento da primeira detecção. Esse controle é realizado com o uso de um controlador PI. Esse procedimento de ajuste da orientação da cadeira de rodas pode ser vista na Figura 26;

4) Por fim, caso não se receba novos dados do bloco de visão dentro de três segundos, indicando a perda do acompanhante, é realizada uma parada de emergência da cadeira e se reinicia o processo de busca.

Figura 26: Ajuste de orientação: (a) Posição desejada (inicial) (b) Acompanhante deslocado (c) Ângulo da câmera ajustado (d) Orientação da cadeira acompanha o ângulo da câmera



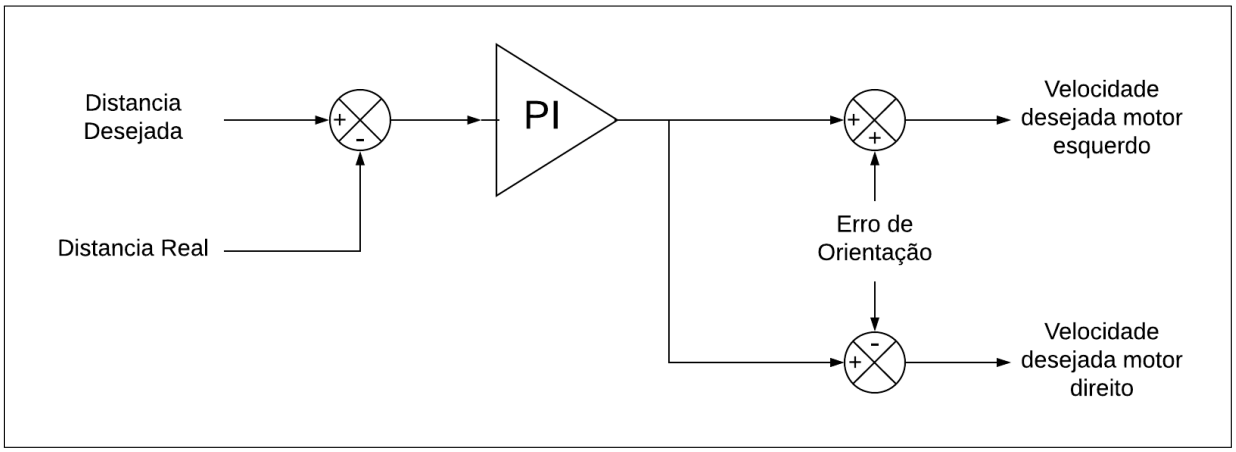
Fonte: Elaborado pelo Autor (2018).

3.2.1 Sistema de Controle Principal

A Figura 27 mostra o diagrama do sistema de controle principal implementado. Um controlador PI gera o sinal de controle utilizando como referência o valor da distância desejada do acompanhante, capturada durante o processo de busca do mesmo. O erro é gerado através da comparação entre distância desejada e a atual. A saída deste controlador é a velocidade desejada, em número de pulsos dos encoders, para as duas rodas tomando em consideração somente a correção da distância. Em seguida, se utiliza um valor proporcional ao erro de orientação da

câmera para o ajuste do sentido de movimentação caso esse valor de erro ultrapasse um dado limiar, sendo que este valor é adicionado para uma das velocidades e subtraído para a outra, conforme o sentido de rotação que se deve imprimir na cadeira. Essa configuração permite com que se consiga uma aceleração igual para as duas rodas e que se consiga exercer uma pequena rotação na orientação da cadeira de maneira suave, ou seja, se desacelera uma roda na mesma taxa em que se acelera a outra. Por fim os valores de velocidade desejada para as rodas são enviadas, pelo barramento serial, para o microcontrolador do bloco de acionamento dos motores, que irá realizar o ajuste da velocidade da cadeira de rodas.

Figura 27: Sistema de controle principal.



Fonte: Elaborado pelo Autor (2018).

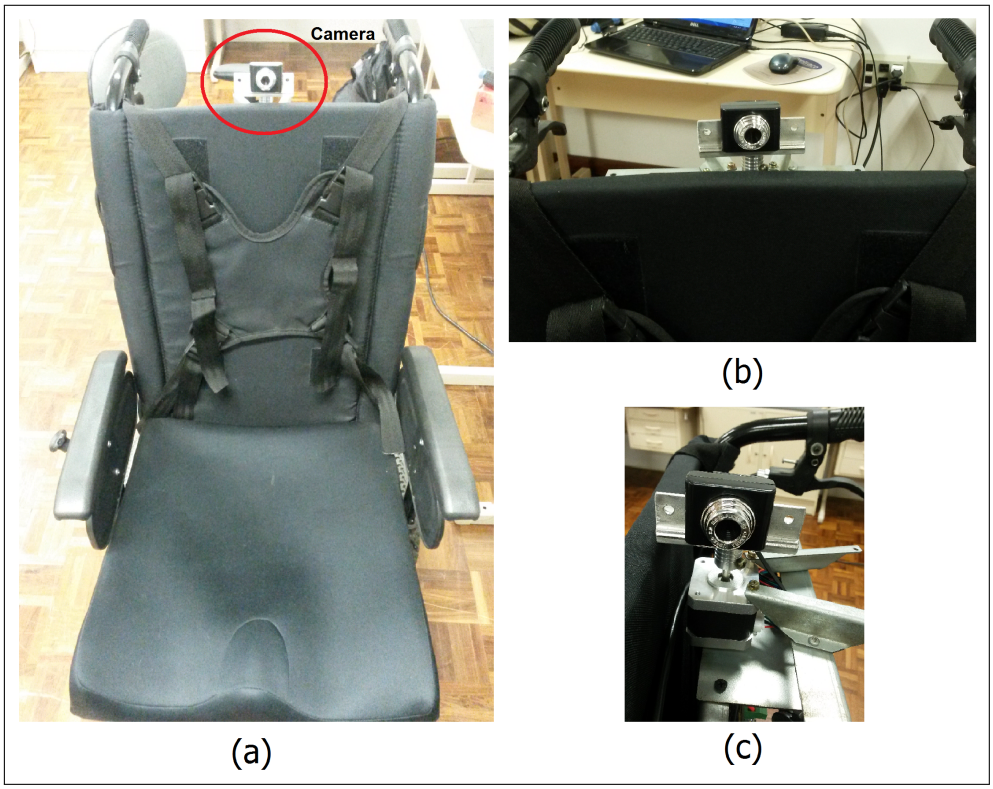
O Apêndice D mostra o circuito eletrônico do microcontrolador STM32F100C8 utilizado. Utilizou-se basicamente os barramentos de comunicação e os pinos de acionamento do circuito de drive do motor da câmera. Nesse microcontrolador é executado o cálculo do sistema de controle da velocidade desejada e também o controle do acionamento do motor de passo, utilizado para o ajuste da rotação da câmera. A rotina de controle da velocidade desejada é exercida a cada 50ms através da interrupção de um timer do microcontrolador, porém somente é calculada caso se receba uma atualização das variáveis do bloco de visão. Para se realizar o calculo da variável de controle da velocidade desejada com um valor de período de amostragem (Ts) correto, se realiza uma contagem de quantas interrupções do timer ocorreram sem a atualização dos dados do sistema de visão. Já o controle da rotação da câmera só é realizada no momento da recepção de dados vindos do bloco de visão. O código em C desenvolvido para este microcontrolador pode ser visto no Apêndice B. Por fim, os dados da velocidade desejada para as duas rodas são enviados através do barramento serial para o bloco de acionamento.

3.2.2 Sistema de Controle da Câmera

A câmera utilizada foi uma *webcam* usb comum com resolução máxima de 640x480 e taxa de atualização máxima de 30fps. Devido ao reduzido campo de visão da mesma, foi

necessário o desenvolvimento de uma forma de ajuste do enquadramento a medida que o acompanhante se movimenta. Para isso se fixou a câmera em um motor de passo do tipo NEMA17 modelo 42BYGHW609 (HETAI, 2014). Para o acionamento foi utilizado um driver de motor de passo comercial modelo A4988 (ALLEGRO, 2014). A Figura 28 mostra a câmera montada no motor e o seu posicionamento na cadeira de rodas, sendo que a câmera é fixada nas costas da cadeira de rodas. Para o funcionamento do driver são necessários dois sinais, um sinal que indica a direção de rotação do motor e outro que, ao receber um pulso, realiza um passo no motor. Esse motor possui ângulo de $1.8°$ para cada passo, porém, se utilizando da função microstepping do drive que permite dividir o passo original em ângulos menores, foi obtida uma resolução angular de $0.45°$. No momento da inicialização da cadeira posiciona-se a câmera, manualmente, apontando para a direita da cadeira de rodas (lado esquerdo na Figura 28 (a)), sendo esse considerado o ângulo inicial da câmera ($0°$). Por fim, é permitida uma excursão máxima de $180°$ na câmera, possibilitando a visualização de toda área frontal da cadeira de rodas. O circuito eletrônico do drive de acionamento pode ser visto no Apêndice D

Figura 28: Sistema câmera + motor: (a) Visão Frontal (b) Detalhe (c) Visão Lateral



Fonte: Elaborado pelo Autor (2018).

3.3 Bloco de Acionamento

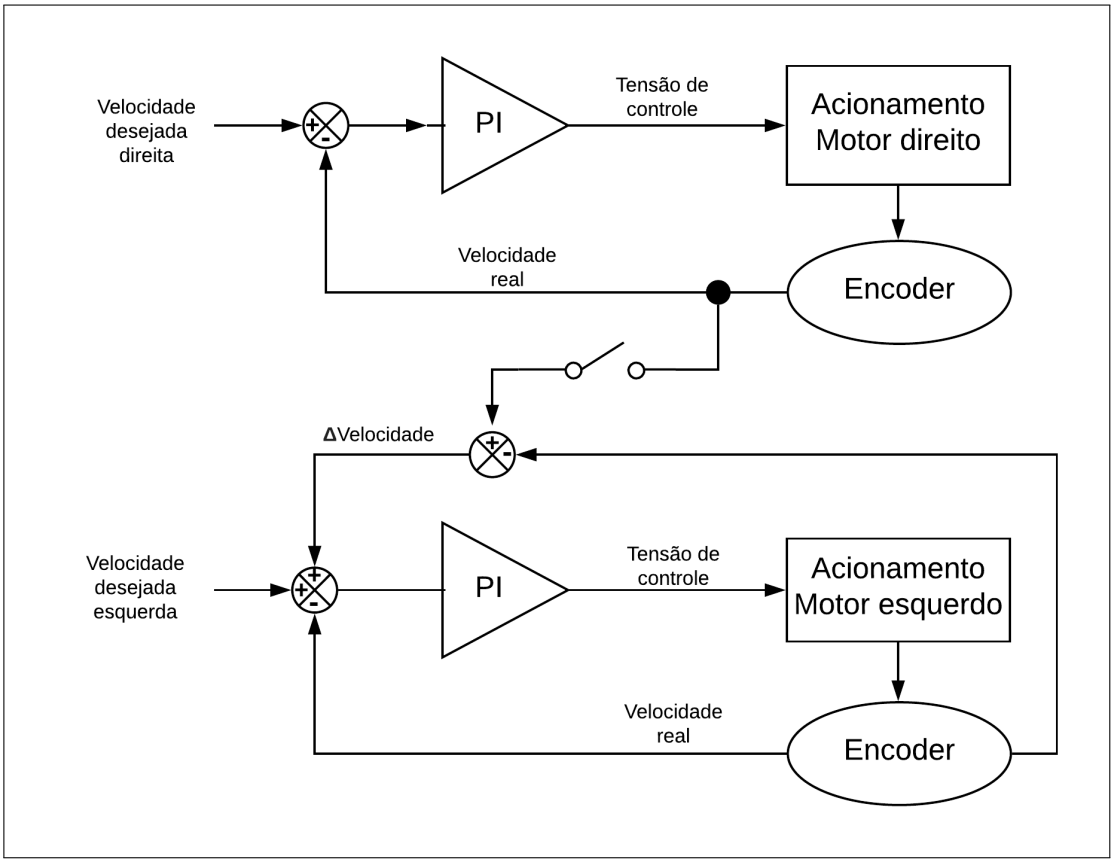
Esse bloco é constituído basicamente por um microcontrolador STM32VLDISCOVERY, um sistema de leitura de velocidade por meio de encoder ótico e dispositivos de acionamento

eletrônico dos motores. Tem a função de controlar a velocidade das rodas, a partir dos dados de velocidade desejada para cada roda, vindos através do barramento serial do bloco de controle principal, e realização da leitura da velocidade das rodas, através de encoder ótico.

3.3.1 Sistema de Controle

O sistema de controle de velocidade implementado nesse microcontrolador, que pode ser visto na Figura 29, é calculado com período de amostragem de 50ms e usa os valores de velocidade desejada como referência, as aplicando em um controlador PI para cada roda, onde os valores de saída dos controladores são utilizados como valor de tensão de controle para os acionadores eletrônicos. Em seguida a velocidade real é capturada, através dos encoders ópticos, e realimentada para a geração do sinal de erro. Utilizou-se uma conexão entre os dois controladores para que se garanta que as duas rodas acelerem juntas, logo caso a roda direita adquira uma velocidade maior, ou menor, do que a esquerda, se adiciona, ou se subtrai, ao sinal de erro da roda esquerda um valor proporcional a diferença entre as velocidades (Δ Velocidade).

Figura 29: Sistema de controle de velocidade das rodas



Fonte: Elaborado pelo Autor (2018).

O circuito eletrônico microcontrolado, que pode ser visto no Apêndice D, se comunica continuamente com o Bloco de Controle, através de um barramento de comunicação serial as-

síncrona USART, recebendo do mesmo os valores da velocidade desejada, em Hertz[Hz], para cada roda e retornando a velocidade real para que o bloco anterior possa realizar o monitoramento da velocidade real.

O programa, desenvolvido em linguagem C pode ser analisado no Apêndice C deste trabalho. Além do cálculo dos sinais de controle, o sistema desenvolvido prevê três estados diferentes de operação:

a) EMERGÊNCIA: Neste caso se executa uma parada de emergência, onde a velocidade da cadeira é reduzida a zero. Esse estado é acessado quando o acompanhante sai do campo de visão da câmera ou quando para bruscamente de caminhar. A ativação deste estado é controlada pelo bloco de controle principal;

b) ORIENTANDO: Este estado é detectado quando os valores de velocidade desejada, recebidos pelo barramento de comunicação, possuem valores diferentes entre as duas rodas. Neste caso, o fator de correção (Δ Velocidade) usado para que a roda esquerda siga o comportamento da velocidade da roda direita é desabilitado pois os motores deverão apresentar velocidades distintas

c) NORMAL: Estado normal de funcionamento, ocorrendo quando a velocidade desejada é a mesma para as duas rodas.

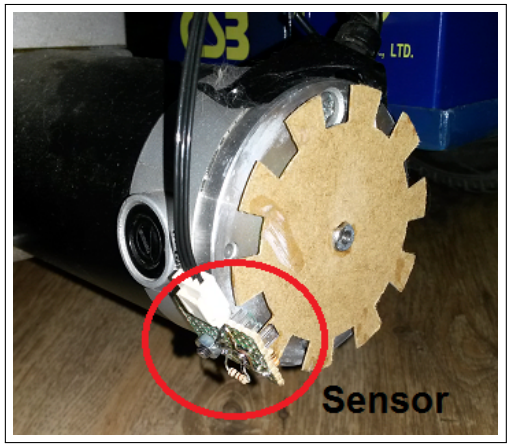
3.3.2 Sistema de Leitura da Velocidade

A velocidade das rodas é adquirida através de dois conjuntos de sensores ópticos do tipo emissor-receptor infravermelho, que detectam a rotação de um disco de encoder fixado na extremidade dos motores. A Figura 30 mostra uma imagem do conjunto sensor-encoder da roda esquerda, sendo igual para a roda direita. O disco de encoder construído possui 78mm de diâmetro e 10 dentes.

A medição da frequência dos pulsos é realizada fazendo-se uso de dois *Timers* do microcontrolador, configurados em modo de entrada PWM. Nesta configuração é possível medir o período do sinal de entrada e o seu *duty-cycle* sem a necessidade de execução de nenhuma interrupção no código. O microcontrolador analisa o sinal de entrada e armazena em um registro o valor de contagem do timer entre duas bordas de subida do sinal e, baseado na frequência de contagem e o prescaler do timer, se calcula a frequência do sinal utilizando a equação 3.1 (ST, 2018).

$$Freq = \frac{Clock_{Timer}}{(prescaler_{Timer} + 1) * periodo_{sinal}} \tag{3.1}$$

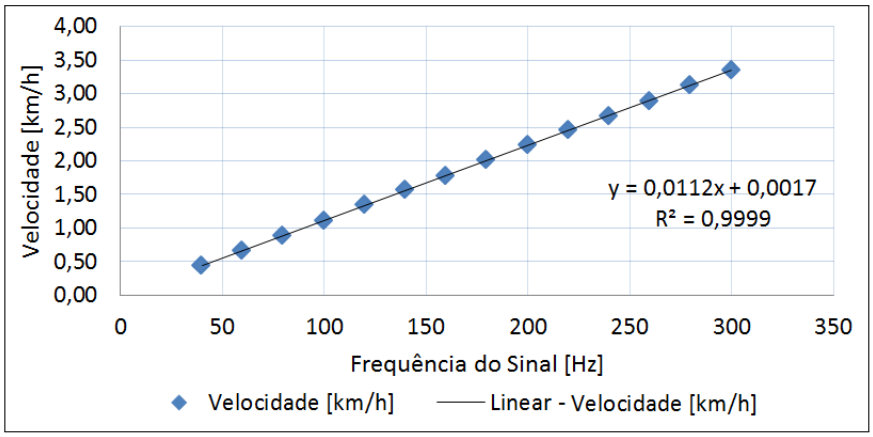
Figura 30: Sistema de leitura de velocidade por encoder ótico



Fonte: Elaborado pelo Autor (2018).

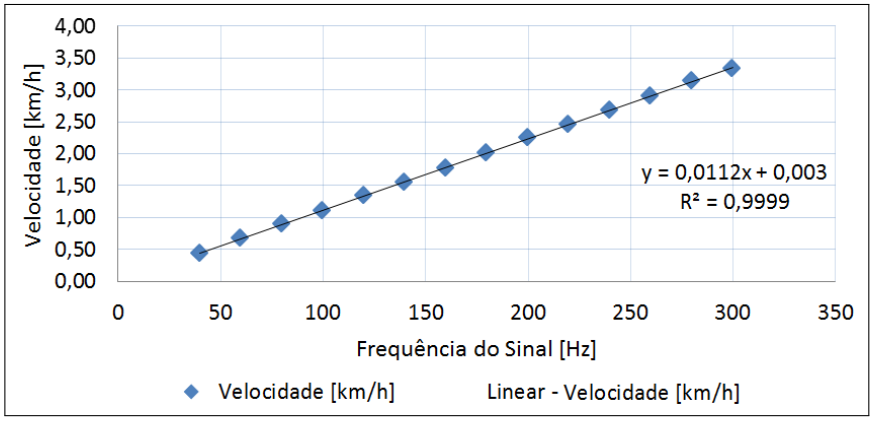
Para a validação do sistema de leitura, foi realizada a caracterização do sensor e se estimou a relação entre a frequência do sinal pulsado, gerado pelo encoder ótico, e a rotação real da roda. Para a realização deste levantamento foi utilizado um tacômetro ótico da marca Minipa, modelo MDT-2238A (MINIPA, 2015). Nessa caracterização fez-se o uso do sistema de controle de velocidade desenvolvido, sendo que foi enviado ao mesmo, através do barramento serial, o valor da frequência desejada para a realização da regulação. Foi realizada uma varredura da velocidade de rotação [RPM] das rodas em função da frequência dos pulsos gerados pelos encoders óticos desenvolvidos e, em seguida, se foi convertido para o valor de velocidade linear em km/h com a utilização da equação 3.2, sendo que $D_{roda} = 0,318m$. A Figura 31 mostra a caracterização realizada para a roda esquerda, onde a equação da linha de tendência linear resultante foi $Vel[km/h] = 0,0112Freq + 0,0017$ com coeficiente de correlação de 0,9999. Já a Figura 32 mostra a caracterização realizada para a roda direita, onde a equação da linha de tendência linear resultante foi $Vel[km/h] = 0,0112Freq + 0,0033$ com coeficiente de correlação também de 0,9999.

Figura 31: Caracterização sensor da roda esquerda



Fonte: Elaborado pelo Autor (2018).

Figura 32: Caracterização sensor da roda direita



Fonte: Elaborado pelo Autor (2018).

$$V_{km/h} = V_{Rpm}\frac{2\pi}{60} * 3,6\frac{D_{roda}}{2} \tag{3.2}$$

Por conveniência, e pela desnecessidade, não se converte para velocidade linear (km/h) os dados de velocidade para o calculo das variáveis de controle nos sistemas de controle principal e de acionamento. Logo, se utiliza os dados de frequência dos pulsos dos encoders como informação de velocidade linear da cadeira de rodas.

3.3.3 Sistema de Acionamento dos Motores

Para o acionamento dos motores foram utilizados os dos módulos ponte H já presentes na cadeira de rodas, do tipo IBT2 para motores de 5V até 27V e corrente máxima de 43A, sendo que este módulo utiliza dois circuitos integrados BTS7960 (INFINEON, 2004). Esse módulo de ponte H precisa de um sinal de PWM, um sinal de *ENABLE* e um sinal que indica o sentido de rotação. A tensão criada por esse módulo na armadura do motor é proporcional ao valor médio de tensão do sinal PWM, logo, o valor resultante do sistema de controle é utilizado para variar o duty-cycle dos sinais de PWM, modificando assim a tensão média detectada pela ponte H. Os motores utilizados são os disponibilizados pela fabricante da cadeira de rodas, em um conjunto motor-redução e são da marca HUMMEL modelo EC82M244630CL com tensão de alimentação de 24V. O circuito eletrônico utilizado pode ser visto no Apêndice D, onde se utilizam de portas lógicas e um transistor para selecionar em qual pino da ponte H irá ser enviado o sinal de PWM, controlando assim o sentido de rotação dos motores.

4 RESULTADOS

Neste item, serão apresentados os resultados do desenvolvimento deste projeto para cada bloco anteriormente discutido, assim como a validação do projeto proposto.

4.1 BLOCO DE VISÃO

No Bloco de visão se avaliou separadamente somente o tempo necessário para o processamento dos dados da imagem e a extração do ponto de referência.

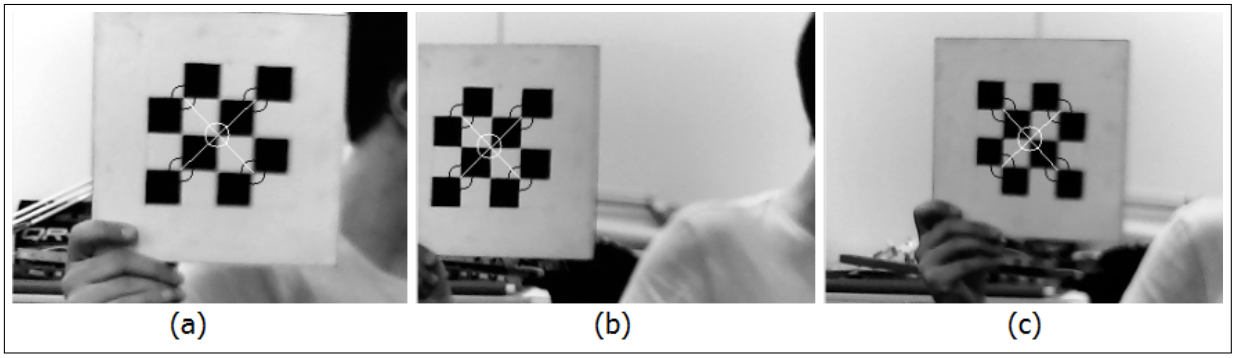
Para validar o tempo utilizado pelo algoritmo no processo de detecção do alvo, se utilizou da biblioteca *ctime*, disponível pelo compilador gcc/gnu em Linux. Essa biblioteca possui funções para medição do tempo de execução de um algoritmo. Os resultados foram de 130ms até 150ms para quando o padrão de jogo de xadrez não era presente na imagem e de 50ms até 100ms para quando o padrão estava presente na imagem. Essa diferença nos tempos se deve ao fato que, como já evidenciado anteriormente, o algoritmo realiza varias tentativas até a detecção do padrão, sendo finalizado quando o mesmo é detectado. Foi realizada uma tentativa de redução do tempo de processamento necessário pelo algoritmo de detecção, onde se executa uma redução de 50% no tamanho da imagem caso não se detecte o padrão de xadrez, retornando ao tamanho original após a detecção, sendo esse de 320x240 pixels. Essa ação reduziu para até 50ms o tempo da rotina do algoritmo de detecção nos momentos em que não está presente o padrão de xadrez na imagem.

4.2 BLOCO DE CONTROLE

Neste bloco pode-se avaliar separadamente somente o sistema de controle da rotação da câmera, sendo que se avaliou o ajuste da orientação da mesma através das imagens capturadas. O controle da velocidade e orientação se avalia juntamente com o bloco de acionamento dos motores.

Para a validação do sistema de ajuste de orientação da câmera se aplicou o controle de funcionamento normal e se movimentou o padrão de xadrez em várias posições diferentes e em ângulos de posicionamento da câmera diversos, avaliando em seguida o erro final em *píxels*. A Figura 33 mostra um exemplo do ajuste no ângulo de orientação da câmera e a Tabela 4 mostra os resultados obtidos, sendo que o erro representa a diferença entre a posição central do padrão de xadrez e o centro da imagem após o ajuste, onde um erro negativo representa diferença para a esquerda e positivo para a direita. Este teste foi executado dez vezes para cada faixa de valores angulares.

Figura 33: Processo de ajuste da câmera: (a) Posição inicial (b) Erro (c) Após centralização



Fonte: Elaborado pelo Autor (2018).

Tabela 4: Avaliação do erro obtido na orientação do ângulo da câmera

Faixa angular	Média	Desvio Padrão
13-17	-1	2
28-32	1	3
43-47	1	2
58-62	0	4
73-77	1	3
88-92	0	2
103-107	-1	3
118-122	1	2
133-137	0	3
148-152	0	4
163-167	1	3

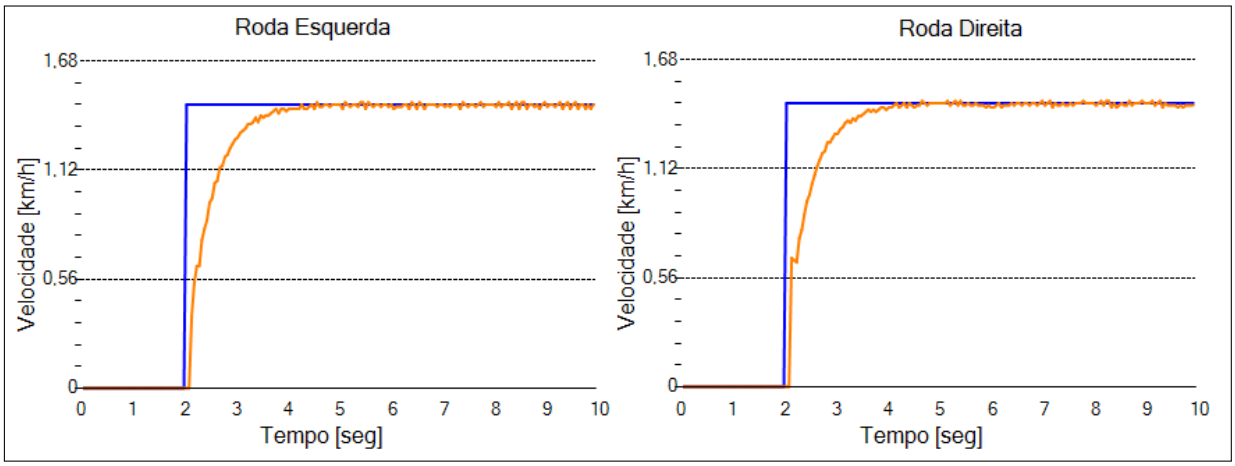
Fonte: Elaborado pelo Autor (2018).

Como a leitura do ângulo é realizada através da contagem dos pulsos executados pelo motor, somente será realizada uma leitura errada caso o motor, ou o drive de acionamento, falhem um pulso ou o apresentem algum defeito elétrico. Por garantia, foi executada uma rotina em que posicionou-se o motor na posição de ângulo zero e executou-se a rotação exercida durante a fase de busca do acompanhante, logo o motor exerce toda a rotação permitida e depois retorna ao ponto inicial, reiniciando o processo. Essa rotina foi executada cinquenta vezes continuamente e se analisou, visualmente, se ao final da rotina o motor era capaz de posicionar no ângulo zero corretamente. Não foi verificada uma diferença nas posições do ângulo zero e final entre o inicio e o fim do procedimento.

4.3 BLOCO DE ACIONAMENTO

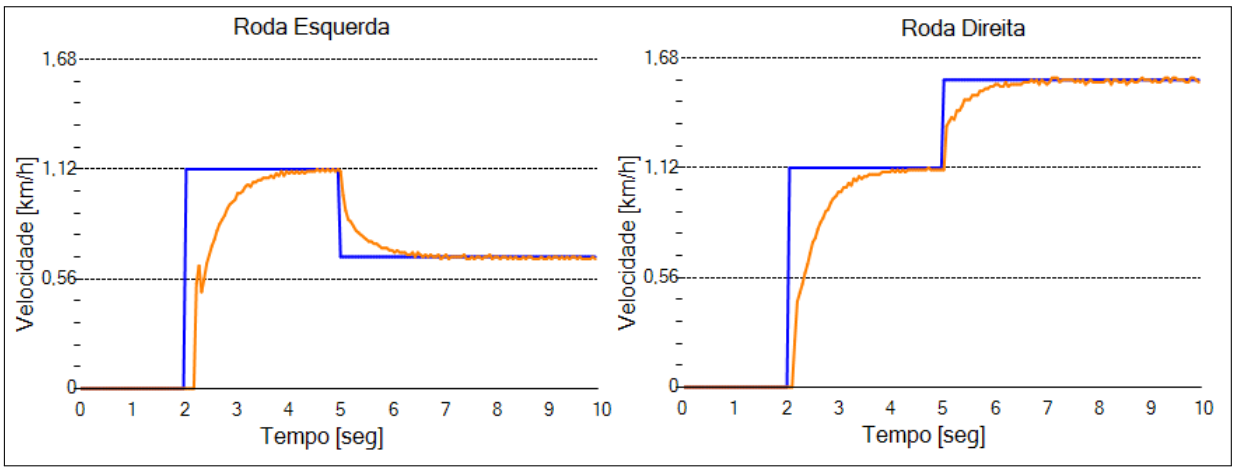
Neste bloco se avaliou o comportamento temporal de aceleração e desaceleração da cadeira de rodas com a utilização de um sistema supervisório desenvolvido em linguagem C#, que recebe os dados do microcontrolador através de um conversor USB/Serial. Nesta avaliação se pré configurou o microcontrolador de controle do acionamento com as velocidades de referência desejadas e se controlou visualmente a excursão do sinal de leitura da velocidade através do supervisório desenvolvido. As Figuras 34, 35 e 36 mostram, respectivamente, as velocidades das rodas esquerda e direita para execução de movimento frontal, rotação a esquerda e rotação a direita, onde as curvas em azul são os valores de referência e as em laranja são a velocidade real da cadeira de rodas.

Figura 34: Aceleração das rodas até 1,46 km/h



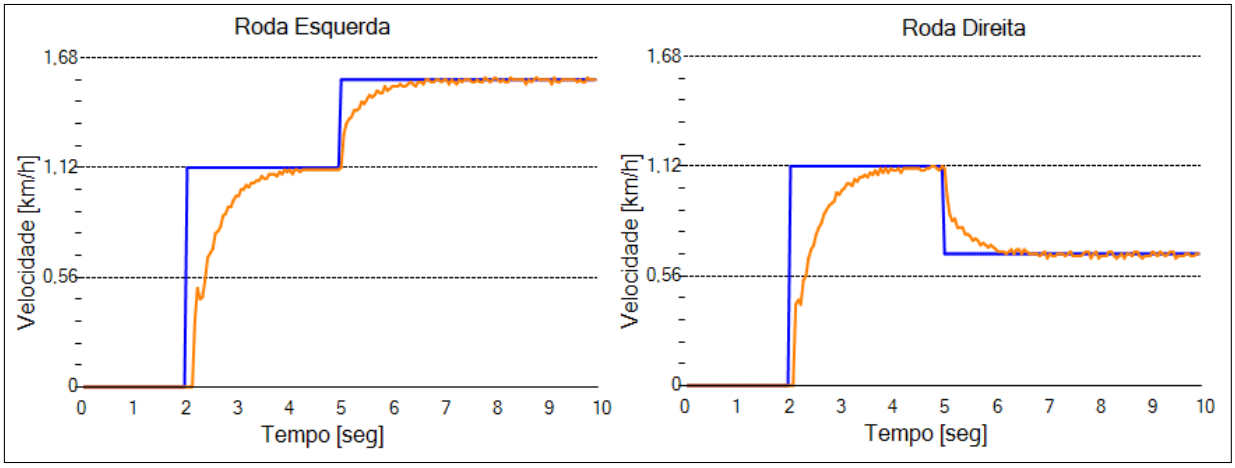
Fonte: Elaborado pelo Autor (2018).

Figura 35: Aceleração e troca de velocidade para rotação à esquerda



Fonte: Elaborado pelo Autor (2018).

Figura 36: Aceleração e troca de velocidade para rotação à direita



Fonte: Elaborado pelo Autor (2018).

4.4 Validação Final

Para a validação completa deste trabalho, se propôs uma trajetória fixa em uma parte do ambiente interno do bloco CD, no Campus da Universidade de Caxias do Sul em Bento Gonçalves. Neste procedimento, o autor fez o papel de acompanhante, segurando o padrão da xadrez na altura do peito e andou de costas para que se pudesse analisar a movimentação da cadeira de rodas. A Figura 37 mostra a trajetória em que se aplicou o processo *Track and Follow* proposto. O movimento é iniciado no ponto 1, indo em linha reta até o ponto 2. Em seguida é realizada uma oscilação na movimentação até o ponto 3 e novamente em linha reta até o ponto 4. A partir disso, foi realizada a rotação da cadeira acompanhando a curva do corredor até o ponto 5, onde foi realizada a trajetória de rotação de $180°$ na cadeira de rodas, finalizando a mesma no ponto 6. Essa rotina foi repetida por cinco vezes de maneira completa e mais cinco vezes entre os pontos marcados. Mesmo realizando poucas tentativas, pode-se perceber o comportamento do sistema completo e os pontos criticos. A seguir se apresentam os resultados para cada etapa:

1) Entre 1 e 2: Nenhum erro foi evidenciado nesta etapa;

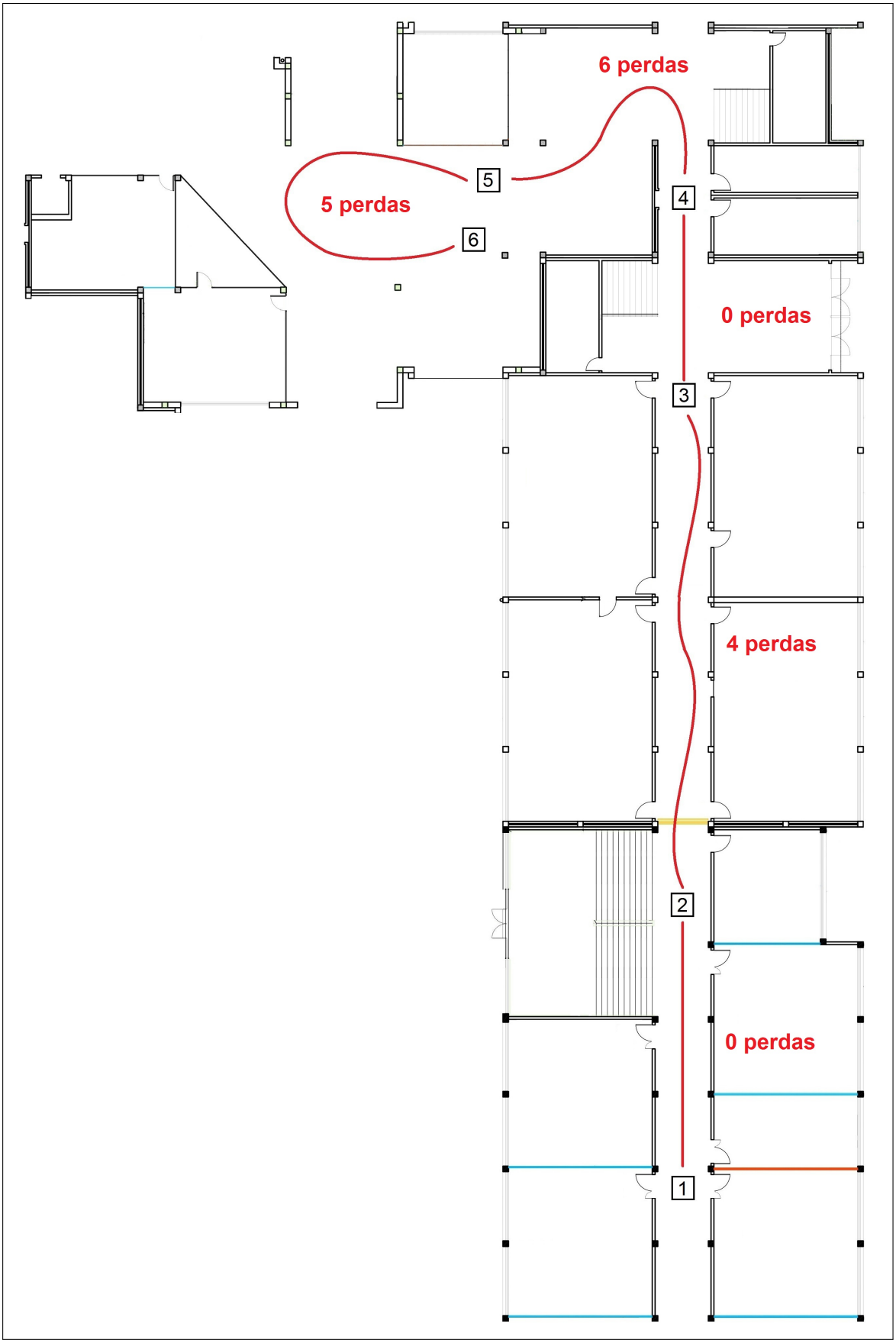
2) Entre 2 e 3: Nesta parte do trajeto constatou-se algumas perdas de detecção. Foi observado que os motivos do mesmo foram desníveis no piso. Das dez tentativas que foram realizadas, em quatro delas ouve perda da detecção.

3) Entre 3 e 4: Nenhum erro foi evidenciado nesta etapa;

4) Entre 4 e 5: Essa etapa foi uma das mais criticas, pois era necessária a realização de uma curva acentuada com a cadeira de rodas. Nesta etapa ouve perda de detecção em seis tentativas negativas entre as dez totais.

5) Entre 5 e 6: Nesta etapa ocorreram cinco perdas das dez tentativas totais.

Figura 37: Trajetória avaliada



Fonte: Elaborado pelo Autor (2018).

Sobre o custo financeiro deste trabalho, pode-se comparar com os trabalhos semelhantes ja apresentados. A Tabela 5 mostra uma comparação dos valores, em dólar, entre os equipamentos utilizados pelo autor deste trabalho e outros trabalhos ja apresentados anteriormente. Os equipamentos elencados na tabela são os informados nos artigos avaliados e os seu valores foram obtidos atraves de sites na internet dos fornecedores dos mesmos. Esta comparação compreende somente os equipamentos utilizados para detecção do acompanhante e processamento da informação, desconsiderando o valor da cadeira de rodas motorizada e os instrumentos de medição de velocidade e acionamento.

Tabela 5: Comparação de valores econômicos entre trabalhos apresentados

Trabalho	Detecção	Processamento	Total
O Autor	Webcam + Motor $25,00	Raspberry Pi 3B $35,00	$60,00
Motokucho e Oda (2014)	2 x Camera high-speed 2 x $225,00	2 x PC Linux Não Especificado 2 x $200,00	$850,00
Wu et al (2012)	Câmera PTZ $50,00 Laser RangeFinder URG-04LX $1080,00	1 x Pc Onboard Não Especificado $250,00	$1380,00
Wu et al (2013)	Sensor Kinect $50,00	1 x Pc Onboard Não especificado $250,00	$300,00
Kobayashi et al (2012)	Laser RangeFinder UTM-30LX $4700,00 3 x Laser Rangefinder URG-04LX $1080,00	1 x Notebook Linux Não especificado $450,00	$8390,00

Fonte: Elaborado pelo Autor (2018).

5 CONCLUSÕES E DISCUSSÕES

Tecnologias *Track and Follow* podem ser utilizadas em diversos outros campos de aplicação além de tecnologias assistivas, como em veículos autônomos ou robôs de auxílio.

Esse projeto possuiu, como objetivo principal, a adaptação, em uma cadeira de rodas motorizada já disponível, de instrumentos de medição e controle para que a mesma fosse capaz de realizar a tarefa de acompanhamento de uma pessoa de forma autônoma, levando em consideração a realização do mesmo utilizando equipamentos de baixo custo. Essa característica de baixo custo foi confirmada ao ser realizada a comparação deste trabalho com outros ja realizados.

Neste trabalho foram apresentados os componentes construtivos do projeto divididos em três blocos, o bloco de visão, responsável pela detecção do acompanhante utilizando visão computacional, o bloco de controle, responsável pelo cálculo da velocidade necessária que deve ser exercida pela cadeira de rodas para a correta realização da tarefa de acompanhamento, além do controle da orientação da câmera, e o bloco de acionamento, responsável pelo controle dos motores e a leitura da velocidade exercida pela cadeira de rodas.

O principal resultado positivo obtido foi o fato de que, de maneira geral, se obteve sucesso no funcionamento do projeto onde, mesmo com dificuldades técnicas e disponibilizando de tecnologias e recursos financeiros limitados, se obtiveram resultados favoráveis com o correto funcionamento do algoritmo de detecção e seguimento, dentro de um ambiente controlado e em velocidade reduzida.

Porém, podem-se elencar alguns defeitos notados durante os procedimento de validação final. No bloco de visão, o conjunto câmera + motor não forneceu os resultados necessários para o perfeito funcionamento do resto do projeto. Isso se deve à alguns fatores evidenciados, como a baixa taxa de atualização das informações de detecção, devido ao alto consumo de processamento utilizado pelo algoritmo de detecção e biblioteca de visão computacional utilizada, a alta vibração do conjunto que afeta a qualidade da imagem adquirida, devido a presença de três componentes móveis no sistema (acompanhante, cadeira de rodas e a câmera) e também devido ao baixo campo de visão da câmera. Todos esses fatores resultaram em um sistema de visão pouco confiável e que, pelo fato disso, não permitiu o correto funcionamento dos blocos seguintes. Também, durante os testes de validação final, se obtiveram perdas de detecção do alvo carregado pelo acompanhante em 43% das tentativas.

Algumas ações buscaram corrigir os defeitos evidenciados, como o uso da ação de redução da imagem em 50%, executada quando se perde a detecção do alvo, reduziu-se em até 160% (de 130ms para 50ms) o tempo de processamento para quando o alvo não está presente na imagem, ou quando a mesma apresenta muita vibração e ruídos. Também se evidencia o fato de que o controlador PI com realimentação do erro entre os motores, desenvolvido no bloco de controle central, foi capaz de corrigir corretamente diferenças elétricas e de velocidade evidenciados entre os dois motores de movimentação da cadeira de rodas. Por fim, para o

correto funcionamento do projeto, o acompanhante deve se manter a uma distância ideal de em torno de 1,2 metros e o mesmo deve executar movimentações suaves, sem realizar rotações ou movimentos bruscos.

Baseando-se nos resultados obtidos, pode-se notar que o projeto desenvolvido obteve funcionamento satisfatório dentro de um ambiente controlado, porém algumas melhorais podem ser realizadas em possíveis trabalhos futuros:

- Realização dos testes de funcionamento com a presença de uma pessoa que possua necessidades especiais para a validação das sensações fornecidas ao cadeirante durante a utilização da cadeira de rodas;
- Substituição da técnica de controle PI aqui implementada por um algoritmo de projeção e estimativa de trajetória utilizando técnicas avançadas de controle, como redes neurais ou *Machine Learning*;
- Utilização de pelo menos três câmeras alinhadas e posicionadas em modo que cubram toda a área angular a frente da cadeira de rodas e, em seguida, utilização de uma algoritmo para que se construa uma imagem panorâmica utilizando dados das três câmeras. Essa melhoria pode ser utilizada para substituir o conjunto câmera + motor, responsável por parte dos erros de funcionamento, mantendo o baixo custo financeiro do sistema de visão;
- Desenvolvimento de um algoritmo próprio para a realização da detecção e rastreamento do acompanhante, sem a utilização da biblioteca OpenCV. Essa ação pode vir a melhorar o tempo de processamento e fidelidade da detecção.
- Adição de outros dispositivos à cadeira de rodas, como sensores para detecção de obstáculos, evitando assim colisões com outros objetos ou estruturas prediais.

REFERÊNCIAS

AHMAD, M. F. et al. Visual based sensor cart follower for wheelchair by using microcontroller. In: INTERNATIONAL CONFERENCE ON CONTROL SYSTEM, COMPUTING AND ENGINEERING, 2015, George Town, Malasia. **Anais...** IEEE, 2015. p. 123–128.

ALLEGRO.
DMOS Microstepping Driver with Translator And Overcurrent Protection. Disponível em <https://www.pololu.com/file/0J450/a4988_DMOS_microstepping_driver_with_translator.pdf> Acesso em 20/10/2019, Datasheet.

BABAIANS, E. et al. Skeleton and visual tracking fusion for human following task of service robots. In: INTERNATIONAL CONFERENCE ON ROBOTICS AND MECHATRONICS, 2015, Teerã, Irã. **Anais...** IEEE, 2015. p. 761–766.

BARTÁK, R.; VYKOVSKY, A. Any object tracking and following by a flying drone. In: MEXICAN INTERNATIONAL CONFERENCE ON ARTIFICIAL INTELLIGENCE, 2015, Cuernavaca, México. **Anais...** IEEE, 2015. p. 35–41.

BASTOS-FILHO, T. F. et al. Towards a new modality-independent interface for a robotic wheelchair. **IEEE TRANSACTIONS ON NEURAL SYSTEMS AND REHABILITATION ENGINEERING**, [S.l.], v. 22, n. 3, p. 567–584, Maio 2014.

BAY, H. et al. Speeded-up robust features (SURF). **COMPUTER VISION AND IMAGE UNDERSTANDING**, [S.l.], v. 110, n. 3, p. 346 – 359, Jun. 2008.

BRADSKI, G. R.; KAEHLER, A. **Learning OpenCV3**: Computer Vision in C++ with the OpenCV library. 1ª. ed. [S.l.]: O'Reilly Media, 2017.

CANNY, J. A computational approach to edge detection. **IEEE TRANSACTIONS ON PATTERN ANALYSIS AND MACHINE INTELLIGENCE**, [S.l.], v. PAMI-8, n. 6, p. 679–698, Nov. 1986.

COX, I. J.; WILFONG, G. T. (Ed.). **Autonomous Robot Vehicles**. 1ª. ed. New York: Springer, 1990. v. 1.

DESAI, S.; MANTHA, S. S.; PHALLE, V. Advances in smart wheelchair technology. In: INTERNATIONAL CONFERENCE ON NASCENT TECHNOLOGIES IN THE ENGINEERING FIELD, 2017, India. **Anais...** IEEE, 2017.

DUBEY, A. K. et al. Challenges in design and deployment of assistive technology. In: INTERNATIONAL CONFERENCE ON SIGNAL PROPAGATION AND COMPUTER TECHNOLOGY, 2014, Ajmer, India. **Anais...** IEEE, 2014. p. 466–469.

DURMUŞ, H. et al. The design of general purpose autonomous agricultural mobile-robot: "AGROBOT". In: INTERNATIONAL CONFERENCE ON AGRO-GEOINFORMATICS, 2015, Turquia. **Anais...** IEEE, 2015.

ENCARNAÇÃO, P.; COOK, A. M. (Ed.). **Robotic Assistive Technologies Principles and Practice**. Boca Raton: CRC Press, 2017. 381 p.

FADALI, M. S. **Digital Control Engineering**: Analysis and Design. [S.l.]: Academic Press, 2009.

FAHIMI, F. **Autonomous Robots: Modeling, Path Planning and Control**. [S.l.]: Springer, 2008.

GE, S. S.; LEWIS, F. L. **Autonomous Mobile Robots: Sensing, Control, Decision Making and Applications**. Boca Raton: CRC Press, 2006.

GUPTA, M. et al. A novel vision-based tracking algorithm for a human-following mobile robot. **IEEE TRANSACTIONS ON SYSTEMS, MAN, AND CYBERNETICS: SYSTEMS**, [S.l.], v. 47, n. 7, p. 1415–1427, Jul. 2017.

HAN, J. et al. Tracking of a moving object by following the sound source. In: INTERNATIONAL CONFERENCE ON ADVANCED INTELLIGENT MECHATRONICS, 2012, Kachsiung, Taiwan. **Anais...** IEEE/ASME, 2012. p. 422–427.

HASSAN, M. S. et al. A computationally low cost vision based tracking algorithm for human following robot. In: INTERNATIONAL CONFERENCE ON CONTROL, AUTOMATION AND ROBOTICS, 2016, Hong Kong, China. **Anais...** IEEE, 2016. p. 62–65.

HELAL, A.; MOKHTARI, M.; ABDULRAZAK, B. (Ed.). **The Engineering Handbook of Smart Technology for Aging, Disability and Independence**. Nova Jersey: John Wiley and Sons INC, 2008.

HERSH, M.; JOHNSON, M. A. On modelling assistive technology systems - Part I: Modelling framework. **TECHNOLOGY AND DISABILITY**, [S.l.], v. 20, n. 3, p. 193–215, 01 2008.

HETAI. **Hybrid Stepper Motor–42BYGHW Series**. Disponível em <http://www.promoco-motors.com/products/StepperMotors/ 42BYGHW%20Series.pdf> Acesso em 04/10/2018, Datasheet.

IBGE. **Censo demográfico características gerais da população, religião, e pessoas com deficiência**. São Paulo, 2010.

IBGE. **Projeção da população do brasil por sexo e idade 2000-2060**. 2013. Disponível em:<http://www.ibge.gov.br/home/estatistica/populacao/projecao_da_populacao/2013/default.shtm> Acesso em 10/08/17.

INFINEON. **BTS 7960**. Disponível em <http://www.robotpower.com/downloads/BTS7960_v1.1_2004-12-07.pdf> Acesso em 07/10/2018, Datasheet.

ISHII, C.; KONISHI, R. A control of electric wheelchair using an emg based on degree of muscular activity. In: EUROMICRO CONFERENCE ON DIGITAL SYSTEM DESIGN, 2016, Limassol, Chipre. **Anais...** IEEE, 2016.

KARSCH, U. M. Idosos dependentes: famílias e cuidadores. **CARDERNO DE SAÚDE PÚBLICA**, [S.l.], v. 19, p. 861 – 866, Jun. 2003.

KHURANA, K.; AWASTHI, R. Techniques for object recognition in images and multi-object detection. **INTERNATIONAL JOURNAL OF ADVANCED RESEARCH IN COMPUTER ENGINEERING AND TECHNOLOGY**, [S.l.], v. 2, n. 4, p. 1383 – 1388, Abr 2013.

KIM, M. et al. RFID-enabled target tracking and following with a mobile robot using direction finding antennas. In: INTERNATIONAL CONFERENCE ON AUTOMATION SCIENCE AND ENGINEERING, 2007, Scottsdale, EUA. **Anais...** IEEE, 2007. p. 1014–1019.

KOBAYASHI, Y.; SUZUKI, R.; KUNO, Y. Robotic wheelchair with omni-directional vision for moving alongside a caregiver. In: CONFERENCE ON INDUSTRIAL ELECTRONICS SOCIETY, 2012, Montreal, Canada. **Anais...** IEEE, 2012. p. 4177–4182.

KRISHNAMURTHY, B.; EVANS, J. Helpmate: A robotic courier for hospital use. In: INTERNATIONAL CONFERENCE ON SYSTEMS, MAN AND CYBERNETICS, 1992, Chicago, EUA. **Anais...** IEEE, 1992.

LEAMAN, J.; LA, H. M. iChair: Intelligent powerchair for severely disabled people. In: INTERNATIONAL CONFERENCE ON MODELING OF COMPLEX SYSTEMS AND ENVIRONMENTS, 2015, Vietnã. **Anais...** ISSAT, 2015.

LEAMAN, J.; LA, H. M. A comprehensive review of smart wheelchairs: past, present, and future. **IEEE TRANSACTIONS ON HUMAN-MACHINE SYSTEMS**, [S.l.], v. 47, n. 4, p. 486–499, Ago. 2017.

LEAMAN, J.; LA, H. M.; NGUYEN, L. Development of a smart wheelchair for people with disabilities. In: INTERNATIONAL CONFERENCE ON MULTISENSOR FUSION AND INTEGRATION FOR INTELLIGENT SYSTEMS, 2016, Baden-Baden, Alemanha. **Anais...** IEEE, 2016.

LEIGH, A. et al. Person tracking and following with 2d laser scanners. In: INTERNATIONAL CONFERENCE ON ROBOTICS AND AUTOMATION (ICRA), 2015, Seattle, EUA. **Anais...** IEEE, 2015. p. 726–733.

LUO, R. C.; HUANG, C. H.; LIN, T. T. Human tracking and following using sound source localization for multisensor based mobile assistive companion robot. In: CONFERENCE ON IEEE INDUSTRIAL ELECTRONICS SOCIETY, 2010, Glendale, EUA. **Anais...** IEEE, 2010. p. 1552–1557.

MADARASZ, R. L. et al. The design of an autonomous vehicle for the disabled. **IEEE JOURNAL OF ROBOTICS AND AUTOMATION**, [S.l.], v. RA-2, n. 3, p. 117–126, Set. 1986.

MASLOW, A. H. A Theory of Human Motivation. **PSYCHOLOGICAL REVIEW**, [S.l.], v. 50, n. 4, p. 370–396, 1943.

MEDIONI, G.; KANG, S. B. **Emerging Topics in Computer Vision**. Upper Saddle River, NJ, USA: Prentice Hall PTR, 2004.

MICHELONI, C. et al. An autonomous vehicle for video surveillance of indoor environments. **IEEE TRANSACTIONS ON VEHICULAR TECHNOLOGY**, [S.l.], v. 52, n. 2, p. 487 – 498, Mar. 2007.

MINIPA. **Tacômetro Foto/Contato Digital MDT-2238A**. Disponível em <http://www.minipa.com.br/images/Manual/MDT-2238B-1100-BR-EN-ES.pdf> Acesso em 07/10/2018, Manual de Instruções.

MORRIS, A. S. **Measurement and Instrumentation Principles**. [S.l.]: Butterworth-Heinemann, 2001.

MOTOKUCHO, T.; ODA, N. Vision-based human-following control using optical flow field for power assisted wheelchair. In: INTERNATIONAL WORKSHOP ON ADVANCED MOTION CONTROL, 2014, Yokohama, Japão. **Anais...** IEEE, 2014. p. 266–271.

MUTZ, F. et al. Following the leader using a tracking system based on pre-trained deep neural networks. In: INTERNATIONAL JOINT CONFERENCE ON NEURAL NETWORKS, 2017, Anchorage, EUA. **Anais...** IEEE, 2017.

NETO, A. Z. et al. Prototype of a wheelchair controlled by cervical movements using inertial sensors operating in differential mode. **Network Modeling Analysis in Health Informatics and Bioinformatics**, [S.l.], v. 4, 12 2015.

OGATA, K. **Modern Control Engineering**. 5ª. ed. [S.l.]: PRENTICE HALL, 2009.

OMS. **International Classification of Functioning, Disability and Health: children and youth version** . [S.l.: s.n.], 2007. Disponível em <http://apps.who.int/iris/bitstream/10665/43737/1/9789241547321_eng.pdf>. Acesso em: 21/10/2017.

OMS. **Relatório mundial sobre a deficiência**. São Paulo: [s.n.], 2011.

PARK, J. J.; KUIPERS, B. Autonomous person pacing and following with model predictive equilibrium point control. In: INTERNATIONAL CONFERENCE ON ROBOTICS AND AUTOMATION, 2013, Karlsruhe, Alemanha. **Anais...** IEEE, 2013.

POWER, M. R.; POWER, D.; HORSTMANSHOF, L. Deaf people communicating via sms, tty, relay service, fax, and computers in australia. **THE JOURNAL OF DEAF STUDIES AND DEAF EDUCATION**, [S.l.], v. 12, n. 1, p. 80–92, 2007.

QIAO, Y.; TANG, Y.; LI, J. Improved harris sub-pixel corner detection algorithm for chessboard image. In: INTERNATIONAL CONFERENCE ON MEASUREMENT, INFORMATION AND CONTROL, 2013., 2013. **Proceedings...** [S.l.: s.n.], 2013. v. 02, p. 1408–1411.

RABHI, Y.; MRABET, M.; FNAIECH, F. Optimized joystick control interface for electric powered wheelchairs. In: INTERNATIONAL CONFERENCE ON SCIENCES AND TECHNIQUES OF AUTOMATIC CONTROL AND COMPUTER ENGINEERING, 2015, Tunisia. **Anais...** IEEE, 2015.

REN, Q. et al. Real-time target tracking system for person-following robot. In: CHINESE CONTROL CONFERENCE, 2016, Chengdu, China. **Anais...** IEEE, 2016. p. 6160–6165.

SONZA, A. P. et al. **Acessibilidade e Tecnologia Assistiva: Pensando a Inclusão Sociodigital de PNEs**. Bento Gonçalves, RS: IFRS, 2013. (Série Novos Autores da Educação Profissional e Tecnológica). 2013.

ST. **Stm32f10x reference manual**. Disponível em <https://www.st.com/content/ccc/resource/technical/document/reference_manual/59/b9/ba/7f/11/af/43/d5/CD00171190.pdf/files/CD00171190.pdf/jcr:content/translations/en.CD00171190.pdf> Acesso em 20/06/2018, Manual.

SZELISKI, R. **Computer Vision: Algorithms and Applications**. 1st. ed. New York, NY, USA: Springer-Verlag New York, 2010.

WANG, D.; YU, H. Development of the control system of a voice-operated wheelchair with multi-posture characteristics. In: ASIA-PACIFIC CONFERENCE ON INTELLIGENT ROBOT SYSTEMS, 2017, China. **Anais...** IEEE, 2017.

WEE-KIAT, N. D.; YING-WEI, S.; GOH, S.-Y. Development of an autonomous bci wheelchair. In: SYMPOSIUM ON COMPUTATIONAL INTELLIGENCE IN BRAIN COMPUTER INTERFACES, 2014, Orlando. **Anais...** IEEE, 2014.

WU, B. F. et al. Accompanist detection and following for wheelchair robots with fuzzy controller. In: THE INTERNATIONAL CONFERENCE ON ADVANCED MECHATRONIC SYSTEMS, 2012, Tóquio Japão. **Anais...** IEEE, 2012. p. 638–643.

WU, B. F. et al. RGB-D sensor based SLAM and human tracking with bayesian framework for wheelchair robots. In: INTERNATIONAL CONFERENCE ON ADVANCED ROBOTICS AND INTELLIGENT SYSTEMS, 2013, Tainan, Taiwan. **Anais...** IEEE, 2013. p. 110–115.

WUNDERLICH, S.; SCHMOLZ, J.; KUHNLENZ, K. Follow me: a simple approach for person identification and tracking. In: INTERNATIONAL SYMPOSIUM ON INDUSTRIAL ELECTRONICS, 2017, Edinburgo, Reino Unido. **Anais...** IEEE, 2017. p. 1609–1614.

XU, Q. et al. A Distributed Canny Edge Detector: Algorithm and FPGA Implementation. **IEEE TRANSACTIONS ON IMAGE PROCESSING**, [S.l.], v. 23, n. 7, p. 2944–2960, Jul. 2014.

YE, W. et al. Vision-based human tracking control of a wheeled inverted pendulum robot. **IEEE TRANSACTIONS ON CYBERNETICS**, [S.l.], v. 46, n. 11, p. 2423–2434, Nov. 2016.

ZHANG, R. et al. Control of a wheelchair in an indoor environment based on a brain–computer interface and automated navigation. **IEEE TRANSACTIONS ON NEURAL SYSTEMS AND REHABILITATION ENGINEERING**, [S.l.], v. 24, n. 1, p. 128–139, Jan. 2016.

ZHAO, W. et al. Person localization and tracking for a leader following vehicle by wireless sensors. In: INTERNATIONAL CONFERENCE ON MECHATRONICS AND AUTOMATION, 2015, Beijing, China. **Anais...** IEEE, 2015. p. 2615–2620.

ZOU, L. et al. The comparison of two typical corner detection algorithms. In: SECOND INTERNATIONAL SYMPOSIUM ON INTELLIGENT INFORMATION TECHNOLOGY APPLICATION, 2008., 2008. **Anais...** [S.l.: s.n.], 2008. v. 2, p. 211–215.

APÊNDICE A – CÓDIGO FONTE DO BLOCO DE VISÃO

```
#include "opencv2/core.hpp"
#include "opencv2/calib3d.hpp"
#include <opencv2/highgui.hpp>
#include <opencv2/imgproc.hpp>
#include "opencv2/imgcodecs.hpp"
#include "opencv2/core/utility.hpp"
#include <stdio.h>
#include <iostream>
#include <string>
#include <math.h>
#include <chrono>
#include <ctime>
#include <fcntl.h>
#include <termios.h>
#include <unistd.h>
#include <errno.h>

using namespace cv;
using namespace std;

//variaveis para o calculo do tempo de processamento
chrono::time_point<chrono::system_clock> inicio, fim;

int main(int argc, char* argv[])
{
  //File Descriptor da serial
  int fd;
  //abre a porta serial
  fd = open("/dev/ttyUSB0",O_RDWR | O_NOCTTY | O_NDELAY);

  //configuracao da porta serial
  struct termios SerialPortSettings;
  tcgetattr(fd, &SerialPortSettings);
  cfsetispeed(&SerialPortSettings,B9600);
  cfsetospeed(&SerialPortSettings,B9600);
  SerialPortSettings.c_cflag &= ~CSIZE;
  SerialPortSettings.c_cflag |=  CS8;
  SerialPortSettings.c_cflag &= ~CRTSCTS;
  SerialPortSettings.c_cflag |= CREAD | CLOCAL;
  SerialPortSettings.c_iflag &= ~(IXON | IXOFF | IXANY );
  SerialPortSettings.c_iflag &= ~(ICANON | ECHO | ECHOE | ISIG);
  SerialPortSettings.c_oflag &= ~ONLRET;
  SerialPortSettings.c_cc[VMIN] = 10;
  SerialPortSettings.c_cc[VTIME] = 0;

  //testa a porta serial
  if((tcsetattr(fd,TCSANOW,&SerialPortSettings)) != 0)
  { printf("\n␣␣ERROR␣!␣in␣Setting␣attributes");  }

  //numero de cantos
  int board_w = 3;
  int board_h = 3;

  //variaveis para confirmacao de deteccao
  bool found = false;
```

```
bool found_ant = false;

//variaveis para algoritmo de deteccao
Size board_sz = Size(board_w, board_h);
vector<Point2f> corners, pontos;
Point2f central;

//variaveis gerais
int escape = 0, i = 0, k = 0;

//variaveis dos dados enviados
int distancia = 0, erro = 0;

//cria o leitor de video
VideoCapture capture = VideoCapture(0);
capture.set(CV_CAP_PROP_FRAME_WIDTH, 320);
capture.set(CV_CAP_PROP_FRAME_HEIGHT, 240);

//variaveis das imagens pro opencv
Mat img, gray;

//captura a primeira imagem
capture >> img;

//loop principal
while(1){
  //mede o tempo de captura de uma imagem e mostra no terminal
  inicio = chrono::system_clock::now();
  capture >> img;
  fim = chrono::system_clock::now();
  int elapsed_seconds = chrono::duration_cast<chrono::microseconds>(fim-inicio).count();
  cout << "Tempo␣captura␣imagem␣=␣" << elapsed_seconds << endl;

  //se no quadro anterior nao foi detectado o xadrez, reduz a imagem
  if(found_ant == false)
  {  resize(img, img, Size(), 0.5, 0.5, INTER_LINEAR);   }

  //converte imagem para cinza
  cvtColor(img, gray, CV_BGR2GRAY);

  //executa a funcao d deteccao e mede o tempo
  inicio = chrono::system_clock::now();
  found = findChessboardCorners(gray, board_sz, corners,  CV_CALIB_CB_ADAPTIVE_THRESH |
      CV_CALIB_CB_FILTER_QUADS );
  fim = chrono::system_clock::now();
  elapsed_seconds = chrono::duration_cast<chrono::microseconds>(fim-inicio).count();
  cout << "Tempo␣deteccao␣=␣" << elapsed_seconds << endl << endl;

  //se foi detectado o acompanhante
  if(found == 1){
    //melhora o posicionamento dos pontos do xadrez
    cornerSubPix(gray, corners, Size(11, 11), Size(-1, -1), TermCriteria(CV_TERMCRIT_EPS |
        CV_TERMCRIT_ITER, 30, 0.1));

    //insere os pontos no vetor desejado
    pontos.clear();
    int k = 0;
    for(k = 0 ; k < 9 ; k++)
    {  pontos.push_back(Point2f(corners.at(k).x, corners.at(k).y));  }
```

```
//medicao da distancia com teorema de pitagoras
int distancia1 = 0, distancia2 = 0;

distancia1 = (pow (((int)pontos.at(0).x - pontos.at(8).x),2) + pow(((int)pontos.at(0).y
    - pontos.at(8).y),2));
distancia1 = sqrt(distancia1);

distancia2 = (pow (((int)pontos.at(2).x - pontos.at(6).x),2) + pow(((int)pontos.at(2).y
    - pontos.at(6).y),2));
distancia2 = sqrt(distancia2);

int centrox1  = 0, centroy1 = 0, centrox2 = 0, centroy2 = 0, centrox = 0, centroy = 0;

//dois primeiros pontos
if((int)pontos.at(0).x < (int)pontos.at(8).x)
{  centrox1 = (int)pontos.at(0).x + (((int)pontos.at(8).x - (int)pontos.at(0).x)/2);  }
else
{  centrox1 = (int)pontos.at(8).x + (((int)pontos.at(0).x - (int)pontos.at(8).x)/2);  }

if((int)pontos.at(0).y < (int)pontos.at(8).y)
{  centroy1 = (int)pontos.at(0).y + (((int)pontos.at(8).y - (int)pontos.at(0).y)/2);  }
else
{  centroy1 = (int)pontos.at(8).y + (((int)pontos.at(0).y - (int)pontos.at(8).y)/2);  }

//dois segundos pontos
if((int)pontos.at(2).x < (int)pontos.at(6).x)
{  centrox2 = (int)pontos.at(2).x + (((int)pontos.at(6).x - (int)pontos.at(2).x)/2);  }
else
{  centrox2 = (int)pontos.at(6).x + (((int)pontos.at(2).x - (int)pontos.at(6).x)/2);  }

if((int)pontos.at(2).y < (int)pontos.at(6).y)
{  centroy2 = (int)pontos.at(2).y + (((int)pontos.at(6).y - (int)pontos.at(2).y)/2);  }
else
{  centroy2 = (int)pontos.at(6).y + (((int)pontos.at(2).y - (int)pontos.at(6).y)/2);  }

//ponto central
if(centrox1 < centrox2)
{  centrox = centrox1 + ((centrox2 - centrox1)/2);  }
else
{  centrox = centrox2 + ((centrox1 - centrox2)/2);  }

if(centroy1 < centroy2)
{  centroy = centroy1 + ((centroy2 - centroy1)/2);  }
else
{  centroy = centroy2 + ((centroy1 - centroy2)/2);  }

//agora estimar a distancia do acompanhante atravez da distancia entre os cantos
int centro_x = 160;
erro = centro_x - centrox;

//limitacao do valor de erro
if(erro > 100)
{  erro = 99;  }
else if(erro < -100)
{  erro = -99; }

//media das duas distancias
distancia = (distancia1 + distancia2)/2;
```

```
      //desenho dos pontos na imagem
      central = Point2f(centrox,centroy);
      Scalar color = Scalar( 0, 255, 255 );
      Scalar color1 = Scalar( 255, 255, 255 );

      circle(gray, pontos[0], 10, color, 1, 8, 0);
      circle(gray, pontos[2], 10, color, 1, 8, 0);
      circle(gray, pontos[6], 10, color, 1, 8, 0);
      circle(gray, pontos[8], 10, color, 1, 8, 0);
      circle(gray, central, 10, color1, 1, 8, 0);
      line(gray, pontos[0], pontos[8], color1, 1, 8, 0 );
      line(gray, pontos[2], pontos[6], color1, 1, 8, 0 );

      //indica os valores no terminal
      cout << "distancia = " << distancia << endl;
      cout << "erro = " << erro << endl;

      //string para enviar os dados
      char write_buffer[18];
      //adiciona os valores na string
      if(erro>0){
        //pra conseguir botar o + na frente
        sprintf(write_buffer, "000X+%0.2d;%0.3dY000\n", erro, distancia);
      }else if(erro<0){
        //o sprintf ja bota o menos na frente
        sprintf(write_buffer, "000X%0.2d;%0.3dY000\n", erro, distancia);
      }else{
        sprintf(write_buffer, "000X%0.3d;%0.3dY000\n", erro, distancia);
      }

      //se a string anterior obteve deteccao correta, se envia a string
      if(found_ant == 1){
        cout << "String enviada = " << write_buffer << endl;
        write(fd,write_buffer,sizeof(write_buffer));
      }
      else{
      //string de lixo quando troca o tamanho da imagem e pra nao dar o overflow do timer
      sprintf(write_buffer, "000XZZZ:ZZZZY000\n");
      write(fd,write_buffer,sizeof(write_buffer));
      }
    }

    //mostra a imagem
    imshow("Deteccao", gray);
    found_ant = found;
    escape = waitKey(5);
    //se eh apertado o ESC, termina o codigo
    if(escape==27){
      break;
    }
  }
  //libera a serial e a camera
  capture.release();
  close(fd);
}
```

APÊNDICE B – CÓDIGO FONTE DO BLOCO DE CONTROLE

```
#include "main.h"
#include "stm32f1xx_hal.h"
#include "stdlib.h"
#include "string.h"
#include "math.h"

//estados possiveis de movimentacao
#define NORMAL          1
#define ORIENTANDO      2
#define EMERGENCIA      9

//variaveis privadas
TIM_HandleTypeDef htim1;
TIM_HandleTypeDef htim2;
TIM_HandleTypeDef htim4;
UART_HandleTypeDef huart1;
UART_HandleTypeDef huart2;
UART_HandleTypeDef huart3;

//variaveis para comunicacao
static char string_envia_mov[50];
static char string_envia_sup[50];
char Rx_indx_rasp, Rx_data_rasp[2], Rx_Buffer_rasp[100];
char Rx_indx_mov, Rx_data_mov[2], Rx_Buffer_mov[100];
char Rx_indx_sup, Rx_data_sup[2], Rx_Buffer_sup[100];

//variaveis de controle enviadas pelo software da camera
int erro_angulo = 0;
int distancia = 0;
float angulo_camera = 0;

//variaveis para controle da camera
uint32_t max_pulsos_camera = 410;
uint32_t contador_pulsos_camera = 0;
float posicao_camera = 0.0;
int pulsos = 0;

//variavel que eh setada quando se busca o acompanhante e resetada quando se acha ele
short estado_busca = 1;
//posicao desejada do acompanhante
float angulo_camera_ref = 0.0;
//valor de distancia desejada do acompanhante
int distancia_camera_ref = 0.0;
//variavel usada para indicar que o processo de localizacao foi concluido
short localizado = 0;
//variavel que vai indicar que se esta centralizando o alvo depois da localizacao inicial
short centralizando = 0;

//controlador da camera
float Erro_camera = 0;
float Proporcional_camera = 0;
float kp_camera = 0.1;
int variavel_de_controle_camera = 0;
int aux_contador = 0;
//variavel para armazenar o estado desejado de movimentacao da cadeira
```

```
int estado = NORMAL;

//variaveis para controle do movimento
char sentido_giro_esq = '+';  //sentido de giro motor esquerdo, + = frente
char sentido_giro_dir = '+';  //sentido de giro motor direito, + = frente

float velocidade_desejada_esq = 0.0;
float velocidade_desejada_dir = 0.0;
float velocidade_dir_real = 0.0;
float velocidade_esq_real = 0.0;
float Erro_orientacao = 0.0;
float Erro_distancia = 0.0;
float Erro_controle_esq = 0.0;
float Erro_controle_esq_ant = 0.0;
float Erro_controle_dir = 0.0;
float Erro_controle_dir_ant = 0.0;
float Proporcional_esq = 0.0;
float Integral_esq = 0.0;
float Integral_esq_ant = 0.0;
float Proporcional_dir = 0.0;
float Integral_dir = 0.0;
float Integral_dir_ant = 0.0;
float kp = 0.15;
float ki = 0.8;

//variavel que eh usada pra calcular o controle soh quando receber dados da camera
short libera_controle = 0;

//variavel para contar quantas interrupcoes foram realizadas desde a ultima recepcao da imagem
//usada para ajusta o periodo de amostragem do controlador PI
int contador_interupcoes = 0;
int k = 0;

//funcoes privadas
void SystemClock_Config(void);
static void MX_GPIO_Init(void);
static void MX_TIM1_Init(void);
static void MX_TIM4_Init(void);
static void MX_USART1_UART_Init(void);
static void MX_USART2_UART_Init(void);
static void MX_USART3_UART_Init(void);
static void MX_TIM2_Init(void);

//funcoes prototipo
void HAL_UART_RxCpltCallback(UART_HandleTypeDef *huart);
void HAL_TIM_PeriodElapsedCallback(TIM_HandleTypeDef *htim);
int limpa_quadro_camera(void);
void limpa_quadro_velocidade(void);
void calcula_controle_camera(void);
void localiza_acompanhante(void);
void calcula_controle_movimento(void);
void Serial_Transmit(UART_HandleTypeDef *huart, uint8_t *s, int num);
void inicia_cadeira(void);
void envia_velocidade_desejada(void);
void parada_de_emergencia(void);

int main(void)
{
  HAL_Init();  //inicia bibliotecas
```

```
  SystemClock_Config();  //configura clock

  //inicia perifericos
  MX_GPIO_Init();
  MX_TIM1_Init();
  MX_TIM4_Init();
  MX_USART1_UART_Init();
  MX_USART2_UART_Init();
  MX_USART3_UART_Init();
  MX_TIM2_Init();

  //inicia interrupcao serial da Raspberry
  __HAL_UART_ENABLE_IT(&huart1, UART_IT_RXNE);
  HAL_UART_Receive_IT(&huart1, Rx_data_rasp, 1);

  //configuracao da interrupcao na serial do bloco de acionamento
  __HAL_UART_ENABLE_IT(&huart3, UART_IT_RXNE);
  HAL_UART_Receive_IT(&huart3, Rx_data_mov, 1);

  inicia_cadeira();
  localiza_acompanhante();

  //zera os contadores dos timers
  TIM4->CNT = 0;
  TIM1->CNT = 0;

  HAL_TIM_Base_Start_IT(&htim1);  //interrupcao para calculo do controle
  HAL_TIM_Base_Start_IT(&htim4);  //interrupcao timeout do bloco de visao

  while (1)
  {  }
}

void SystemClock_Config(void)  //inicializacao do clock
{
  RCC_OscInitTypeDef RCC_OscInitStruct;
  RCC_ClkInitTypeDef RCC_ClkInitStruct;

  RCC_OscInitStruct.OscillatorType = RCC_OSCILLATORTYPE_HSE;
  RCC_OscInitStruct.HSEState = RCC_HSE_ON;
  RCC_OscInitStruct.HSEPredivValue = RCC_HSE_PREDIV_DIV1;
  RCC_OscInitStruct.HSIState = RCC_HSI_ON;
  RCC_OscInitStruct.PLL.PLLState = RCC_PLL_ON;
  RCC_OscInitStruct.PLL.PLLSource = RCC_PLLSOURCE_HSE;
  RCC_OscInitStruct.PLL.PLLMUL = RCC_PLL_MUL9;
  if (HAL_RCC_OscConfig(&RCC_OscInitStruct) != HAL_OK)
  {  _Error_Handler(__FILE__, __LINE__);  }

  RCC_ClkInitStruct.ClockType = RCC_CLOCKTYPE_HCLK|RCC_CLOCKTYPE_SYSCLK | RCC_CLOCKTYPE_PCLK1|
      RCC_CLOCKTYPE_PCLK2;
  RCC_ClkInitStruct.SYSCLKSource = RCC_SYSCLKSOURCE_PLLCLK;
  RCC_ClkInitStruct.AHBCLKDivider = RCC_SYSCLK_DIV1;
  RCC_ClkInitStruct.APB1CLKDivider = RCC_HCLK_DIV2;
  RCC_ClkInitStruct.APB2CLKDivider = RCC_HCLK_DIV1;
  if (HAL_RCC_ClockConfig(&RCC_ClkInitStruct, FLASH_LATENCY_2) != HAL_OK)
  {  _Error_Handler(__FILE__, __LINE__);  }

  HAL_SYSTICK_Config(HAL_RCC_GetHCLKFreq()/1000);
  HAL_SYSTICK_CLKSourceConfig(SYSTICK_CLKSOURCE_HCLK);
```

```
  HAL_NVIC_SetPriority(SysTick_IRQn, 0, 0);
}

static void MX_TIM1_Init(void)  //Inicia TIM1
{
  TIM_ClockConfigTypeDef sClockSourceConfig;
  TIM_MasterConfigTypeDef sMasterConfig;

  htim1.Instance = TIM1;
  htim1.Init.Prescaler = 500;
  htim1.Init.CounterMode = TIM_COUNTERMODE_UP;
  htim1.Init.Period = 7200;
  htim1.Init.ClockDivision = TIM_CLOCKDIVISION_DIV1;
  htim1.Init.RepetitionCounter = 0;
  htim1.Init.AutoReloadPreload = TIM_AUTORELOAD_PRELOAD_DISABLE;
  if (HAL_TIM_Base_Init(&htim1) != HAL_OK)
  {  _Error_Handler(__FILE__, __LINE__);  }

  sClockSourceConfig.ClockSource = TIM_CLOCKSOURCE_INTERNAL;
  if (HAL_TIM_ConfigClockSource(&htim1, &sClockSourceConfig) != HAL_OK)
  {  _Error_Handler(__FILE__, __LINE__);  }

  sMasterConfig.MasterOutputTrigger = TIM_TRGO_RESET;
  sMasterConfig.MasterSlaveMode = TIM_MASTERSLAVEMODE_DISABLE;
  if (HAL_TIMEx_MasterConfigSynchronization(&htim1, &sMasterConfig) != HAL_OK)
  {  _Error_Handler(__FILE__, __LINE__);  }
}

static void MX_TIM2_Init(void)  //Inicia TIM2
{
  TIM_ClockConfigTypeDef sClockSourceConfig;
  TIM_MasterConfigTypeDef sMasterConfig;

  htim2.Instance = TIM2;
  htim2.Init.Prescaler = 6000;
  htim2.Init.CounterMode = TIM_COUNTERMODE_UP;
  htim2.Init.Period = 60000;
  htim2.Init.ClockDivision = TIM_CLOCKDIVISION_DIV1;
  htim2.Init.AutoReloadPreload = TIM_AUTORELOAD_PRELOAD_DISABLE;
  if (HAL_TIM_Base_Init(&htim2) != HAL_OK)
  {  _Error_Handler(__FILE__, __LINE__);  }

  sClockSourceConfig.ClockSource = TIM_CLOCKSOURCE_INTERNAL;
  if (HAL_TIM_ConfigClockSource(&htim2, &sClockSourceConfig) != HAL_OK)
  {  _Error_Handler(__FILE__, __LINE__);  }

  sMasterConfig.MasterOutputTrigger = TIM_TRGO_RESET;
  sMasterConfig.MasterSlaveMode = TIM_MASTERSLAVEMODE_DISABLE;
  if (HAL_TIMEx_MasterConfigSynchronization(&htim2, &sMasterConfig) != HAL_OK)
  {  _Error_Handler(__FILE__, __LINE__);  }
}

static void MX_TIM4_Init(void)  //Inicia TIM4
{
  TIM_ClockConfigTypeDef sClockSourceConfig;
  TIM_MasterConfigTypeDef sMasterConfig;

  htim4.Instance = TIM4;
  htim4.Init.Prescaler = 23000;
```

```
  htim4.Init.CounterMode = TIM_COUNTERMODE_UP;
  htim4.Init.Period = 10000;
  htim4.Init.ClockDivision = TIM_CLOCKDIVISION_DIV1;
  htim4.Init.AutoReloadPreload = TIM_AUTORELOAD_PRELOAD_DISABLE;
  if (HAL_TIM_Base_Init(&htim4) != HAL_OK)
  {  _Error_Handler(__FILE__, __LINE__);  }

  sClockSourceConfig.ClockSource = TIM_CLOCKSOURCE_INTERNAL;
  if (HAL_TIM_ConfigClockSource(&htim4, &sClockSourceConfig) != HAL_OK)
  {  _Error_Handler(__FILE__, __LINE__);  }

  sMasterConfig.MasterOutputTrigger = TIM_TRGO_RESET;
  sMasterConfig.MasterSlaveMode = TIM_MASTERSLAVEMODE_DISABLE;
  if (HAL_TIMEx_MasterConfigSynchronization(&htim4, &sMasterConfig) != HAL_OK)
  {  _Error_Handler(__FILE__, __LINE__);  }
}

static void MX_USART1_UART_Init(void)  //Inicia USART1 - Raspberry
{
  huart1.Instance = USART1;
  huart1.Init.BaudRate = 9600;
  huart1.Init.WordLength = UART_WORDLENGTH_8B;
  huart1.Init.StopBits = UART_STOPBITS_1;
  huart1.Init.Parity = UART_PARITY_NONE;
  huart1.Init.Mode = UART_MODE_TX_RX;
  huart1.Init.HwFlowCtl = UART_HWCONTROL_NONE;
  huart1.Init.OverSampling = UART_OVERSAMPLING_16;
  if (HAL_UART_Init(&huart1) != HAL_OK)
  {  _Error_Handler(__FILE__, __LINE__);  }
}

static void MX_USART2_UART_Init(void)  //Inicia USART2 - supervisorio
{
  huart2.Instance = USART2;
  huart2.Init.BaudRate = 115200;
  huart2.Init.WordLength = UART_WORDLENGTH_8B;
  huart2.Init.StopBits = UART_STOPBITS_1;
  huart2.Init.Parity = UART_PARITY_NONE;
  huart2.Init.Mode = UART_MODE_TX_RX;
  huart2.Init.HwFlowCtl = UART_HWCONTROL_NONE;
  huart2.Init.OverSampling = UART_OVERSAMPLING_16;
  if (HAL_UART_Init(&huart2) != HAL_OK)
  {  _Error_Handler(__FILE__, __LINE__);  }
}

static void MX_USART3_UART_Init(void)  //Inicia USART3 - bloco acionamento
{
  huart3.Instance = USART3;
  huart3.Init.BaudRate = 115200;
  huart3.Init.WordLength = UART_WORDLENGTH_8B;
  huart3.Init.StopBits = UART_STOPBITS_1;
  huart3.Init.Parity = UART_PARITY_NONE;
  huart3.Init.Mode = UART_MODE_TX_RX;
  huart3.Init.HwFlowCtl = UART_HWCONTROL_NONE;
  huart3.Init.OverSampling = UART_OVERSAMPLING_16;
  if (HAL_UART_Init(&huart3) != HAL_OK)
  {  _Error_Handler(__FILE__, __LINE__);  }
}
```

```
static void MX_GPIO_Init(void)  //Inicia pinos
{
  GPIO_InitTypeDef GPIO_InitStruct;

  __HAL_RCC_GPIOC_CLK_ENABLE();
  __HAL_RCC_GPIOD_CLK_ENABLE();
  __HAL_RCC_GPIOA_CLK_ENABLE();
  __HAL_RCC_GPIOB_CLK_ENABLE();

  HAL_GPIO_WritePin(GPIOC, Led_Pin|M_Camera_Dir_Pin|M_Camera_Step_Pin, GPIO_PIN_RESET);

  GPIO_InitStruct.Pin = Led_Pin|M_Camera_Dir_Pin|M_Camera_Step_Pin;
  GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
  GPIO_InitStruct.Pull = GPIO_NOPULL;
  GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
  HAL_GPIO_Init(GPIOC, &GPIO_InitStruct);
}

void calcula_controle_camera(void)  //Calculo do controle de rotacao da camera e ajuste
{
  Erro_camera = erro_angulo;

  //muda o ganho do controlador a depender do valor do erro
  if((erro_angulo > 50) || (erro_angulo < -50))
  {  kp_camera = 0.02;  }
  else if((erro_angulo > 30) || (erro_angulo < -30))
  {  kp_camera = 0.05;  }
  else if((erro_angulo <= 10) || (erro_angulo >= -10))
  {  kp_camera = 0.1;   }

  //calcula o controle
  Proporcional_camera = kp_camera * Erro_camera;
  variavel_de_controle_camera = (int) (Proporcional_camera);

  //o resultado do controlador eh a quantidade de pulsos para ajustar a camera
  pulsos = variavel_de_controle_camera;

  //variavel que guarda qual vai ser a quantidade de pulsos total apos o ajuste
  //pelo numero de pulsos executados, a partir do ponto zero, se calcula o angulo da camera
  aux_contador = contador_pulsos_camera + pulsos;

  if(pulsos<0)  //nao se pode ter pulsos negativos
  {  pulsos = -1*pulsos;  }

  float posicao_camera_ant = posicao_camera;  //armazena a posicao em outra variavel

  //calculo da nova posicao desejada da camera
  posicao_camera = 0.44*aux_contador;

  //se a posicao depois do ajuste nao extrapolar os limites
  if((posicao_camera < 180.0) && (posicao_camera > 0))
  {
    contador_pulsos_camera = aux_contador;
    k = 0;
    if(Erro_camera>0)  //selecionar o sentido de rotacao da camera baseado no sentido do erro
    {  HAL_GPIO_WritePin(M_Camera_Dir_GPIO_Port, M_Camera_Dir_Pin, GPIO_PIN_SET);  }
    if(Erro_camera<0)
    {  HAL_GPIO_WritePin(M_Camera_Dir_GPIO_Port, M_Camera_Dir_Pin, GPIO_PIN_RESET);  }
```

```
    //realiza os pulsos
    int aux = 0;
    for(k=0;k<pulsos;k++)
    {
      HAL_GPIO_WritePin(M_Camera_Step_GPIO_Port, M_Camera_Step_Pin, GPIO_PIN_SET);
      for(aux=0;aux<50000;aux++){       }
      HAL_GPIO_WritePin(M_Camera_Step_GPIO_Port, M_Camera_Step_Pin, GPIO_PIN_RESET);
      for(aux=0;aux<50000;aux++){       }
    }
  }
  else //nao muda se passar dos limites angulares
  {  posicao_camera = posicao_camera_ant;  }
  angulo_camera = posicao_camera; //atualiza a variavel que armazena o angulo atual da camera
}

void HAL_UART_RxCpltCallback(UART_HandleTypeDef *huart) //funcao da interrupcao nas seriais
{
  int i;
  if (huart->Instance == USART1) //interrupcao na serial da raspberry
  {
    //zera o contador do timeout pra nao ficar interrompendo toda vez
    TIM4->CNT = 0;

    if (Rx_indx_rasp==0) //limpa a string na primeira recepcao
    { for (i=0;i<50;i++){  Rx_Buffer_rasp[i]=0;  }  }

    if (Rx_data_rasp[0]!=13) //caractere que indica finalizacao = /r
    {  Rx_Buffer_rasp[Rx_indx_rasp++]=Rx_data_rasp[0];  }
    else
    {
      Rx_indx_rasp=0;
      if(limpa_quadro_camera()==1)  //confere a validade do quadro
      {
        estado_busca = 0;       //indica que o alvo foi encontrado
        libera_controle = 1;  //libera o calculo do controle
      }
      //se ja esta buscando centralizacao e ja esta dentro do angulo desejado
      int aux_angulo = abs(erro_angulo);
      if((centralizando == 1) && (aux_angulo < 2)){
        //usa o angulo atual da camera e a distancia atual como referencia
        angulo_camera_ref = angulo_camera;
        distancia_camera_ref = distancia;
        centralizando = 2;
      }

      //as vezes o controle nao pega o valor da distancia, aqui forca isso
      if((centralizando == 2) && (distancia_camera_ref == 0))
      {  distancia_camera_ref = distancia;  }

      if(distancia > 100)  //se esta muito perto, para a cadeira
      {
        estado = EMERGENCIA;
        velocidade_desejada_dir = 0;
        velocidade_desejada_esq = 0;
      }
      else
      {  calcula_controle_camera();  }
    }
    HAL_UART_Receive_IT(&huart1, Rx_data_rasp, 1);   //reinicia a cada recepcao
```

```
  }

  if (huart->Instance == USART3)  //interrupcao na serial do bloco de acionamento
  {
    if (Rx_indx_mov==0)  //limpa a string na primeira recepcao
    {  for (i=0;i<50;i++){  Rx_Buffer_mov[i]= '␣';  }  }

    if (Rx_data_mov[0]!=10) //caractere que indica finalizacao = /n
    {  Rx_Buffer_mov[Rx_indx_mov++]=Rx_data_mov[0];  }
    else
    {
      Rx_indx_mov=0;
      limpa_quadro_velocidade();
    }
    HAL_UART_Receive_IT(&huart3, Rx_data_mov, 1);   //reinicia a cada recepcao
  }
}

void HAL_TIM_PeriodElapsedCallback(TIM_HandleTypeDef *htim) //interrupcoes dos timers
{
  if (htim->Instance == TIM1)  //TIMER1
  {
    if(libera_controle == 1)
    {
      libera_controle = 0;
      if(centralizando == 2)  //se ja centralizou a camera, permite o movimento
      {
        calcula_controle_movimento();  //calcula o controle
        envia_velocidade_desejada();  //envia as velocidades desejadas
      }
    }
    else
    {  contador_interupcoes++;  }
  }

  if (htim->Instance == TIM4)  //TIMER4
  {  parada_de_emergencia();  }
}

void calcula_controle_movimento(void)  //calculo do controle de distancia e orientacao
{
  kp = 0.3;
  ki = 0.8;
  int velocidade_max = 120;     //velocidade maxima admitida pelas rodas, em Hz

  int Erro_distancia_ant = Erro_distancia;
  Erro_distancia = distancia_camera_ref - distancia;
  Erro_orientacao = angulo_camera_ref - angulo_camera;

  if(distancia > 70)  //forca parada caso muito perto
  {  Erro_distancia = -300;  }
  if((Erro_orientacao < 3) && (Erro_orientacao > -3)) //janela de histerese do erro da camera
  {  Erro_orientacao = 0;  }

  float Ts;
  if(contador_interupcoes == 0) //Ts depende de quantas interrupcoes tiveram
  {  Ts = 0.05;  }
  else
  {  Ts = 0.05*contador_interupcoes;  }
```

```
contador_interupcoes = 0;

//calculo da velocidade desejada roda esquerda
Erro_controle_esq_ant = Erro_controle_esq;
Integral_esq_ant = Integral_esq;
Erro_controle_esq = Erro_distancia;
Proporcional_esq = (float) kp * Erro_controle_esq;
Integral_esq = (float) (((ki * Ts) / 2) * (Erro_controle_esq  + Erro_controle_esq_ant)) +
    Integral_esq_ant;

velocidade_desejada_esq = Proporcional_esq + Integral_esq;

//calculo da velocidade desejada roda direita
Erro_controle_dir_ant = Erro_controle_dir;
Integral_dir_ant = Integral_dir;
Erro_controle_dir = Erro_distancia;
Proporcional_dir = (float) kp * Erro_controle_dir;
Integral_dir = (float) (((ki * Ts) / 2) * (Erro_controle_dir  + Erro_controle_dir_ant)) +
    Integral_dir_ant;

velocidade_desejada_dir = Proporcional_dir + Integral_dir;

//mexe na orientacao so proporcional ao erro do angulo
if((Erro_orientacao < 10) && (Erro_orientacao > -10)){
  velocidade_desejada_dir = velocidade_desejada_dir - Erro_orientacao*3.0;
  velocidade_desejada_esq = velocidade_desejada_esq + Erro_orientacao*3.0;
}else if((Erro_orientacao < 30) && (Erro_orientacao > -30)){
  velocidade_desejada_dir = velocidade_desejada_dir - Erro_orientacao*2.5;
  velocidade_desejada_esq = velocidade_desejada_esq + Erro_orientacao*2.5;
}else if((Erro_orientacao < 60) && (Erro_orientacao > -60)){
  velocidade_desejada_dir = velocidade_desejada_dir - Erro_orientacao*1.5;
  velocidade_desejada_esq = velocidade_desejada_esq + Erro_orientacao*1.5;
}

if(velocidade_desejada_esq > velocidade_max) //variaveis maximas e antiwindup
{
  velocidade_desejada_esq = velocidade_max;
  Integral_esq = velocidade_desejada_esq;
}
if(velocidade_desejada_dir > velocidade_max)
{
  velocidade_desejada_dir = velocidade_max;
  Integral_dir = velocidade_desejada_dir;
}
if(velocidade_desejada_esq<0)
{
  velocidade_desejada_esq = 0;
  Integral_esq = Integral_esq_ant;
}
if(velocidade_desejada_dir<0)
{
  velocidade_desejada_dir = 0;
  Integral_dir = Integral_dir_ant;
}

//se zerar o erro da orientacao, pega a media das velocidades e bota nas duas rodas
if( (estado == ORIENTANDO) && (Erro_orientacao == 0) )
{
  int media = (velocidade_desejada_dir + velocidade_desejada_esq) / 2;
```

```
    velocidade_desejada_dir = media;
    velocidade_desejada_esq = media;
    Integral_dir = velocidade_desejada_dir;
    Integral_esq = velocidade_desejada_esq;
    Integral_dir_ant = 0;
    Integral_esq_ant = 0;
    estado = NORMAL;
  }

  if(Erro_orientacao != 0)  //se e erro eh diferente de zero, bota em modo orientacao
  {  estado = ORIENTANDO;  }
}

void localiza_acompanhante(void)  //funcao para localizar o acompanhante
{
  //interrompe a interrupcao do controle e do timeout
  HAL_TIM_Base_Stop_IT(&htim1);
  HAL_TIM_Base_Stop_IT(&htim4);

  //zera os contadores
  TIM4->CNT = 0;
  TIM1->CNT = 0;

  //zera a variavel indicando que nao se tem o alvo localizado e centralizando
  localizado = 0;
  centralizando = 0;

  //primeira acao eh voltar para o ponto zero da camera
  int k = 0;
  HAL_GPIO_WritePin(M_Camera_Dir_GPIO_Port, M_Camera_Dir_Pin, GPIO_PIN_RESET); //giro horario

  int aux = 0;
  //desliga interrupcao da serial enquanto a camera retorna
  __HAL_UART_DISABLE_IT(&huart1, UART_IT_RXNE);

  //pulsos pra voltar a camera
  for(k=0;k<contador_pulsos_camera;k++)
  {
    HAL_GPIO_WritePin(M_Camera_Step_GPIO_Port, M_Camera_Step_Pin, GPIO_PIN_SET);
    for(aux=0;aux<30000;aux++){ }
    HAL_GPIO_WritePin(M_Camera_Step_GPIO_Port, M_Camera_Step_Pin, GPIO_PIN_RESET);
    for(aux=0;aux<30000;aux++){ }
  }

  //reabilita a serial para receber a interrupcao da camera
  __HAL_UART_ENABLE_IT(&huart1, UART_IT_RXNE);
  HAL_UART_Receive_IT(&huart1, Rx_data_rasp, 1);

  //vai ate os pulso maximo em busca do alvo
  HAL_GPIO_WritePin(M_Camera_Dir_GPIO_Port, M_Camera_Dir_Pin, GPIO_PIN_SET);
  for(k=0;k<max_pulsos_camera;k++)
  {
    HAL_GPIO_WritePin(M_Camera_Step_GPIO_Port, M_Camera_Step_Pin, GPIO_PIN_SET);
    for(aux=0;aux<130000;aux++){        }
    HAL_GPIO_WritePin(M_Camera_Step_GPIO_Port, M_Camera_Step_Pin, GPIO_PIN_RESET);
    for(aux=0;aux<130000;aux++){        }

    if(estado_busca==0)  //se achou
    {
```

```
      contador_pulsos_camera = k;  //pega os pulsos atuais para o calculo do controle
      localizado = 1;  //indica que se localizou
      centralizando = 1;  //indica que tem que se realizar uma centralizacao da camera

      //zera os contadores
      TIM4->CNT = 0;
      TIM1->CNT = 0;

      //limpa as flags
      __HAL_TIM_CLEAR_IT(&htim1, TIM_IT_UPDATE);
      __HAL_TIM_CLEAR_IT(&htim4, TIM_IT_UPDATE);

      //reinicia os timers
      HAL_TIM_Base_Start_IT(&htim1);
      HAL_TIM_Base_Start_IT(&htim4);

      break;    //quebra o loop
    }
  }

  if(estado_busca==1)  //se nao achou
  {
    contador_pulsos_camera = max_pulsos_camera;
    localiza_acompanhante();  //reinicia a funcao
  }
}

int limpa_quadro_camera(void)  //funcao para retirar parte desejada da string recebida
{
  int escape = 0, i = 0, j = 0;

  while (escape == 0) //loop para ver onde esta o identificador inicial = X
  {
    if (Rx_Buffer_rasp [i] == 'X')
    {
      for (j = 0; j < 100; j++)
      {
        if (Rx_Buffer_rasp [i] == 'Y') //quando achar o Y termina o loop
        {
          escape = 1;
          break;
        }
        else
        {
          Rx_Buffer_rasp [j] = Rx_Buffer_rasp [i + 1];
          i++;
        }
      }
    }
    else
    {  i++;  }
  }

  //quebra a funcao quando se detectar a string lixo da camera
  //string usada soh pra indicar que esta se captando o alvo mas os valores ainda estao ruins
  if(Rx_Buffer_rasp[0] == 'Z')
  {  return 0;  }
  i = 0;
  char texto [3];
```

```
for (i = 0; i < 3; i++) //rotina para pegar a parte do erro
{  texto [i] = '␣';  }
texto[0] = '0';
for (i = 0; i < 2; i++)
{  texto [i + 1] = Rx_Buffer_rasp [i + 1];  }
erro_angulo = 0;

int numero = 0;
for (i = 0; i < 4; i++)
{
  switch (texto [i])
  {
    case '1':
     numero = 1;
     break;
    case '2':
     numero = 2;
     break;
    case '3':
     numero = 3;
     break;
    case '4':
     numero = 4;
     break;
    case '5':
     numero = 5;
     break;
    case '6':
     numero = 6;
     break;
    case '7':
     numero = 7;
     break;
    case '8':
     numero = 8;
     break;
    case '9':
     numero = 9;
     break;
    case '0':
     numero = 0;
     break;
    default:
     break;
  }
  if (i == 0)
  {  erro_angulo = erro_angulo + (100 * numero);  }
  else if (i == 1)
  {  erro_angulo = erro_angulo + (10 * numero);  }
  else if (i == 2)
  {  erro_angulo = erro_angulo + (1 * numero);  }
}

if(Rx_Buffer_rasp[0] == '-')
{  erro_angulo = -1*erro_angulo;  }

numero = 0;
```

```
  for (i = 0; i < 3; i++) //rotina para pegar a parte da distancia
  {  texto [i] = '␣';  }
  for (i = 0; i < 3; i++)
  {  texto [i] = Rx_Buffer_rasp [i + 4];  }

  distancia = 0;
  for (i = 0; i < 4; i++)
  {
    switch (texto [i])
    {
      case '1':
       numero = 1;
       break;
      case '2':
       numero = 2;
       break;
      case '3':
       numero = 3;
       break;
      case '4':
       numero = 4;
       break;
      case '5':
       numero = 5;
       break;
      case '6':
       numero = 6;
       break;
      case '7':
       numero = 7;
       break;
      case '8':
       numero = 8;
       break;
      case '9':
       numero = 9;
       break;
      case '0':
       numero = 0;
       break;
      default:
       break;
    }
    if (i == 0)
    {  distancia = distancia + (100 * numero);  }
    else if (i == 1)
    {  distancia = distancia + (10 * numero);  }
    else if (i == 2)
    {  distancia = distancia + (1 * numero);  }
  }
  return 1;
}

void limpa_quadro_velocidade(void)  //funcao que separa parte util da string recebida
{
  //string recebida = 000Xabbbb;cdddd;fffffY000\n
  //a, c = sentido de giro rodas esq e dir esq ; b, d = vel esq e dir , f = erro do quadro
  int escape = 0, i = 0, j = 0;
  float vel_esq = 0.0, vel_dir = 0.0;
```

```
while (escape == 0)  //loop para ver onde esta o identificador inicial = X
{
  if (Rx_Buffer_mov [i] == 'X')
  {
    for (j = 0; j < 100; j++)
    {
      if (Rx_Buffer_mov [i] == 'Y')  //quando achar o Y termina o loop
      {
        escape = 1;
        break;
      }
      else
      {
        Rx_Buffer_mov [j] = Rx_Buffer_mov [i + 1];
        i++;
      }
    }
  }
  else
  {  i++;  }
}

i = 0;
char texto [4];

for(i=0;i<50;i++)
{  Rx_Buffer_mov [i] = '␣';  }
for (i = 0; i < 4; i++)  //rotina para pegar a parte da roda esquerda
{  texto [i] = '␣';  }
for (i = 0; i < 4; i++)
{  texto [i] = Rx_Buffer_mov [i+1];  }

if(Rx_Buffer_mov[0] == '+')
{  sentido_giro_esq = '+';  }
else if(Rx_Buffer_mov[0] == '-')
{  sentido_giro_esq = '-';  }

int numero = 0;
for (i = 0; i < 5; i++)
{
  switch (texto [i])
  {
    case '1':
     numero = 1;
     break;
    case '2':
     numero = 2;
     break;
    case '3':
     numero = 3;
     break;
    case '4':
     numero = 4;
     break;
    case '5':
     numero = 5;
     break;
    case '6':
```

```
      numero = 6;
      break;
    case '7':
      numero = 7;
      break;
    case '8':
      numero = 8;
      break;
    case '9':
      numero = 9;
      break;
    case '0':
      numero = 0;
      break;
    default:
      numero = 0;
      break;
  }
  if (i == 0)
  {  vel_esq = vel_esq + (1000 * numero);  }
  else if (i == 1)
  {  vel_esq = vel_esq + (100 * numero);  }
  else if (i == 2)
  {  vel_esq = vel_esq + (10 * numero);  }
  else if (i == 3)
  {  vel_esq = vel_esq + (1 * numero);  }
}

for (i = 0; i < 4; i++)  //rotina para pegar a parte da roda direita
{  texto [i] = ' ';  }
for (i = 0; i < 4; i++)
{  texto [i] = Rx_Buffer_mov [i + 7];  }

if(Rx_Buffer_mov[6] == '+')
{  sentido_giro_dir = '+';  }
else if(Rx_Buffer_mov[6] == '-')
{  sentido_giro_dir = '-';  }

int numero = 0;
for (i = 0; i < 5; i++)
{
  switch (texto [i])
  {
    case '1':
      numero = 1;
      break;
    case '2':
      numero = 2;
      break;
    case '3':
      numero = 3;
      break;
    case '4':
      numero = 4;
      break;
    case '5':
      numero = 5;
      break;
    case '6':
```

```
      numero = 6;
      break;
    case '7':
      numero = 7;
      break;
    case '8':
      numero = 8;
      break;
    case '9':
      numero = 9;
      break;
    case '0':
      numero = 0;
      break;
    default:
      numero = 0;
      break;
  }
  if (i == 0)
  {  vel_dir = vel_dir + (1000 * numero);  }
  else if (i == 1)
  {  vel_dir = vel_dir + (100 * numero);  }
  else if (i == 2)
  {  vel_dir = vel_dir + (10 * numero);  }
  else if (i == 3)
  {  vel_dir = vel_dir + (1 * numero);  }
}

numero = 0;
for (i = 0; i < 6; i++)  //rotina para pegar a parte da deteccao de erro no quadro
{  texto [i] = '␣'; }
for (i = 0; i < 5; i++)
{  texto [i] = Rx_Buffer_mov [i + 12];  }

int erro_quadro = 0;
for (i = 0; i < 5; i++)
{
  switch (texto [i])
  {
    case '1':
      numero = 1;
      break;
    case '2':
      numero = 2;
      break;
    case '3':
      numero = 3;
      break;
    case '4':
      numero = 4;
      break;
    case '5':
      numero = 5;
      break;
    case '6':
      numero = 6;
      break;
    case '7':
      numero = 7;
```

```
        break;
       case '8':
        numero = 8;
        break;
       case '9':
        numero = 9;
        break;
       case '0':
        numero = 0;
        break;
       default:
        break;
     }
     if (i == 0)
     {  erro_quadro = erro_quadro + (10000 * numero);  }
     else if (i == 1)
     {  erro_quadro = erro_quadro + (1000 * numero);  }
     else if (i == 2)
     {  erro_quadro = erro_quadro + (100 * numero);  }
     else if (i == 3)
     {  erro_quadro = erro_quadro + (10 * numero);  }
     else if (i == 4)
     {  erro_quadro = erro_quadro + (1 * numero);  }
   }

   if(erro_quadro == (int)(vel_dir + vel_esq))  //ve se o quadro esta ok
   {
     velocidade_dir_real = vel_dir;
     velocidade_esq_real = vel_esq;
   }
}

void envia_velocidade_desejada(void) //envia velocidades desejadas para bloco de acionamento
{
   int vel_dir = (int) velocidade_desejada_dir;
   int vel_esq = (int) velocidade_desejada_esq;
   int erro_quadro = vel_dir + vel_esq;
   sprintf(string_envia_mov, "000X%d;%c%4d;%c%4d;%5dY000\n", estado, sentido_giro_esq, vel_esq,
        sentido_giro_dir, vel_dir, erro_quadro);
   Serial_Transmit(&huart3, string_envia_mov,50);
}

void inicia_cadeira(void)  //funcao para iniciar a cadeira
{
   sprintf(string_envia_mov, "000X1;+0000;+0000;00000Y00\n");
   for(int i = 0; i<3; i++)  //envia 3 vezes por garantia
   {  Serial_Transmit(&huart3, string_envia_mov,50);  }
}

void Serial_Transmit(UART_HandleTypeDef *huart, uint8_t *s, int num)  //envia dados serial
{
   int cont = 0;
   while (cont<num) {
     while (!(huart->Instance->SR & 0x00000040));     //espera ate o registro de saida estar
         vazio
     huart->Instance->DR = (*s & (uint16_t)0x01FF);     //envia um char por vez
     *s++;
     cont++;     //aumenta valor do ponteiro
   }
```

```
}

void parada_de_emergencia(void)  //funcao para parar a cadeira em emergencia
{
  //reinicia todas as variaveis
  Erro_camera = 0;
  erro_angulo = 0;
  distancia = 0;
  estado_busca = 1;
  angulo_camera = 0;
  angulo_camera_ref = 0.0;
  distancia_camera_ref = 0.0;
  localizado = 0;
  centralizando = 0;
  Proporcional_camera = 0;
  kp_camera = 0.1;
  variavel_de_controle_camera = 0;
  sentido_giro_esq = '+';
  sentido_giro_dir = '+';
  estado = NORMAL;
  velocidade_desejada_esq = 0.0;
  velocidade_desejada_dir = 0.0;
  velocidade_dir_real = 0.0;
  velocidade_esq_real = 0.0;
  Erro_orientacao = 0.0;
  Erro_distancia = 0.0;
  Erro_controle_esq = 0.0;
  Erro_controle_esq_ant = 0.0;
  Erro_controle_dir = 0.0;
  Erro_controle_dir_ant = 0.0;
  Proporcional_esq = 0.0;
  Integral_esq = 0.0;
  Integral_esq_ant = 0.0;
  Proporcional_dir = 0.0;
  Integral_dir = 0.0;
  Integral_dir_ant = 0.0;
  kp = 0.15;
  ki = 0.8;
  libera_controle = 0;
  contador_interupcoes = 0;

  estado = EMERGENCIA;  //para a cadeira
  envia_velocidade_desejada();

  estado = NORMAL;  //ja volta pro estado normal pra nao bloquear

  localiza_acompanhante();
}
```

APÊNDICE C – CÓDIGO FONTE DO BLOCO DE ACIONAMENTO

```
#include "main.h"
#include "stm32f1xx_hal.h"

#define PWM_RIGHT       TIM3->CCR3      // PWM MOTOR 1 - DIREITA (Vista Traseira) (PINO PB.0)
#define PWM_LEFT        TIM3->CCR4      // PWM MOTOR 2 - ESQUERDA (Vista Traseira)(PINO PB.1)

//estados possiveis de movimentacao
#define NORMAL          1
#define ORIENTANDO  2
#define EMERGENCIA  9

//variaveis privadas
TIM_HandleTypeDef htim1;
TIM_HandleTypeDef htim2;
TIM_HandleTypeDef htim3;
TIM_HandleTypeDef htim4;
UART_HandleTypeDef huart2;

//strings para comunicacao
char Rx_indx, Rx_data[2], string_recebida_controle[50];
char string_envia_controle [50];

//variavel usada para indicar o estado de movimentacao da cadeira
short estado = NORMAL;

//variaveis utilizadas para a leitura da velocidade
uint32_t periodo_esq = 0;
uint32_t periodo_dir = 0;

//variaveis utilizadas para o controle do sinal de tensao inserido nos motores
short sentido_giro_motor_esquerdo = 1;          //frente
short sentido_giro_motor_direito = 1;           //frente
float tensao_motor_direito = 0.0;
float tensao_motor_esquerdo = 0.0;

//variaveis de referencia de velocidade para as rodas
float velocidade_desejada_esq = 0.0;
float velocidade_desejada_dir = 0.0;
float velocidade_dir_real = 0.0;
float velocidade_esq_real = 0.0;

//variaveis pro controle de velocidade
float Erro_velocidade_esq = 0.0;
float Erro_velocidade_dir = 0.0;
float Erro_velocidade_esq_ant = 0.0;
float Erro_velocidade_dir_ant = 0.0;

float Proporcional_velocidade_esq = 0.0;
float Integral_velocidade_esq = 0.0;
float Integral_velocidade_esq_ant = 0.0;
float Proporcional_velocidade_dir = 0.0;
float Integral_velocidade_dir = 0.0;
float Integral_velocidade_dir_ant = 0.0;
float kp_velocidade = 0.02;
float ki_velocidade = 0.1;
```

```
float variavel_controle_velocidade_esq = 0.0;
float variavel_controle_velocidade_dir = 0.0;

//funcao usada para soh voltar a andar depois de uma confirmacao do controle
short aguarda_desbloqueio_emergencia = 0;
//funcao que libera o controle soh se o quadro recebido for correto
short libera_calculo_controle = 0;

//funcoes prototipo privadas
void SystemClock_Config(void);
static void MX_GPIO_Init(void);
static void MX_TIM1_Init(void);
static void MX_TIM2_Init(void);
static void MX_TIM3_Init(void);
static void MX_TIM4_Init(void);
static void MX_USART2_UART_Init(void);
void HAL_TIM_MspPostInit(TIM_HandleTypeDef *htim);

//funcoes prototipo
void HAL_UART_RxCpltCallback(UART_HandleTypeDef *huart);
void calcula_controle_velocidade(void);
void HAL_TIM_PeriodElapsedCallback(TIM_HandleTypeDef *htim);
void parada_de_emergencia(void);
void sentido_giro_motores(short esquerdo, short direito);
void le_velocidade_rodas(void);
void Serial_Transmit(UART_HandleTypeDef *huart, uint8_t *s, int num);
int limpa_quadro(void);
void envia_velocidades(void);

int main(void)
{
  HAL_Init();  //inicia bibliotecas
  SystemClock_Config();  //inicia clock

  //inicia perifericos
  MX_GPIO_Init();
  MX_TIM1_Init();
  MX_TIM2_Init();
  MX_TIM3_Init();
  MX_TIM4_Init();
  MX_USART2_UART_Init();

  //configuracao da serial e iniciar interrupcao
  __HAL_UART_ENABLE_IT(&huart2, UART_IT_RXNE);
  HAL_UART_Receive_IT(&huart2, Rx_data, 1);

  //configura os pinos para sentido de rotacao inicial = frente
  sentido_giro_motores(sentido_giro_motor_esquerdo, sentido_giro_motor_direito);

  //Habilita as pontes H
  HAL_GPIO_WritePin(BRIDGE_MOTOR_LEFT_ENABLE_GPIO_Port, BRIDGE_MOTOR_LEFT_ENABLE_Pin,
      GPIO_PIN_SET);
  HAL_GPIO_WritePin(BRIDGE_MOTOR_RIGHT_ENABLE_GPIO_Port, BRIDGE_MOTOR_RIGHT_ENABLE_Pin,
      GPIO_PIN_SET);

  //valor inicial dos pwm
  PWM_RIGHT = (uint16_t) ((0.0 / 24.0) * 1200);
  PWM_LEFT = (uint16_t) ((0.0 / 24.0) * 1200);
```

```
  //inicia os pwm's de acionamento dos motores
  HAL_TIM_PWM_Start(&htim3, TIM_CHANNEL_3);
  HAL_TIM_PWM_Start(&htim3, TIM_CHANNEL_4);

  //inicia o timer da base de tempo do controlador
  HAL_TIM_Base_Start_IT(&htim1);

  //inicia os timers para captura do sinal PWM de entrada dos encoders
  HAL_TIM_IC_Start(&htim2, TIM_CHANNEL_1);  //roda esquerda
  HAL_TIM_IC_Start(&htim2, TIM_CHANNEL_2);
  HAL_TIM_IC_Start(&htim4, TIM_CHANNEL_1);  //roda direita
  HAL_TIM_IC_Start(&htim4, TIM_CHANNEL_2);

  while (1)
  {  }
}

void SystemClock_Config(void)  //Inicia o clock
{
  RCC_OscInitTypeDef RCC_OscInitStruct;
  RCC_ClkInitTypeDef RCC_ClkInitStruct;

  RCC_OscInitStruct.OscillatorType = RCC_OSCILLATORTYPE_HSE;
  RCC_OscInitStruct.HSEState = RCC_HSE_ON;
  RCC_OscInitStruct.HSEPredivValue = RCC_HSE_PREDIV_DIV1;
  RCC_OscInitStruct.PLL.PLLState = RCC_PLL_ON;
  RCC_OscInitStruct.PLL.PLLSource = RCC_PLLSOURCE_HSE;
  RCC_OscInitStruct.PLL.PLLMUL = RCC_PLL_MUL3;
  if (HAL_RCC_OscConfig(&RCC_OscInitStruct) != HAL_OK)
  {  _Error_Handler(__FILE__, __LINE__);  }

  RCC_ClkInitStruct.ClockType = RCC_CLOCKTYPE_HCLK|RCC_CLOCKTYPE_SYSCLK |RCC_CLOCKTYPE_PCLK1|
      RCC_CLOCKTYPE_PCLK2;
  RCC_ClkInitStruct.SYSCLKSource = RCC_SYSCLKSOURCE_PLLCLK;
  RCC_ClkInitStruct.AHBCLKDivider = RCC_SYSCLK_DIV1;
  RCC_ClkInitStruct.APB1CLKDivider = RCC_HCLK_DIV1;
  RCC_ClkInitStruct.APB2CLKDivider = RCC_HCLK_DIV1;
  if (HAL_RCC_ClockConfig(&RCC_ClkInitStruct, FLASH_LATENCY_0) != HAL_OK)
  {  _Error_Handler(__FILE__, __LINE__);  }
  HAL_SYSTICK_Config(HAL_RCC_GetHCLKFreq()/1000);
  HAL_SYSTICK_CLKSourceConfig(SYSTICK_CLKSOURCE_HCLK);
  HAL_NVIC_SetPriority(SysTick_IRQn, 0, 0);
}

static void MX_TIM1_Init(void)  //Inicia TIMER1
{
  TIM_ClockConfigTypeDef sClockSourceConfig;
  TIM_MasterConfigTypeDef sMasterConfig;

  htim1.Instance = TIM1;
  htim1.Init.Prescaler = 500;
  htim1.Init.CounterMode = TIM_COUNTERMODE_UP;
  htim1.Init.Period = 2400;
  htim1.Init.ClockDivision = TIM_CLOCKDIVISION_DIV1;
  htim1.Init.RepetitionCounter = 0;
  htim1.Init.AutoReloadPreload = TIM_AUTORELOAD_PRELOAD_DISABLE;
  if (HAL_TIM_Base_Init(&htim1) != HAL_OK)
  {  _Error_Handler(__FILE__, __LINE__);  }
```

```
  sClockSourceConfig.ClockSource = TIM_CLOCKSOURCE_INTERNAL;
  if (HAL_TIM_ConfigClockSource(&htim1, &sClockSourceConfig) != HAL_OK)
  {  _Error_Handler(__FILE__, __LINE__);  }

  sMasterConfig.MasterOutputTrigger = TIM_TRGO_RESET;
  sMasterConfig.MasterSlaveMode = TIM_MASTERSLAVEMODE_DISABLE;
  if (HAL_TIMEx_MasterConfigSynchronization(&htim1, &sMasterConfig) != HAL_OK)
  {  _Error_Handler(__FILE__, __LINE__);  }
}

static void MX_TIM2_Init(void)  //Inicia TIMER2
{
  TIM_ClockConfigTypeDef sClockSourceConfig;
  TIM_SlaveConfigTypeDef sSlaveConfig;
  TIM_IC_InitTypeDef sConfigIC;
  TIM_MasterConfigTypeDef sMasterConfig;

  htim2.Instance = TIM2;
  htim2.Init.Prescaler = 1000;
  htim2.Init.CounterMode = TIM_COUNTERMODE_UP;
  htim2.Init.Period = 65000;
  htim2.Init.ClockDivision = TIM_CLOCKDIVISION_DIV1;
  htim2.Init.AutoReloadPreload = TIM_AUTORELOAD_PRELOAD_DISABLE;
  if (HAL_TIM_Base_Init(&htim2) != HAL_OK)
  {  _Error_Handler(__FILE__, __LINE__);  }

  sClockSourceConfig.ClockSource = TIM_CLOCKSOURCE_INTERNAL;
  if (HAL_TIM_ConfigClockSource(&htim2, &sClockSourceConfig) != HAL_OK)
  {  _Error_Handler(__FILE__, __LINE__);  }
  if (HAL_TIM_IC_Init(&htim2) != HAL_OK)
  {  _Error_Handler(__FILE__, __LINE__);  }

  sSlaveConfig.SlaveMode = TIM_SLAVEMODE_RESET;
  sSlaveConfig.InputTrigger = TIM_TS_TI1FP1;
  sSlaveConfig.TriggerPolarity = TIM_INPUTCHANNELPOLARITY_RISING;
  sSlaveConfig.TriggerPrescaler = TIM_ICPSC_DIV1;
  sSlaveConfig.TriggerFilter = 15;
  if (HAL_TIM_SlaveConfigSynchronization(&htim2, &sSlaveConfig) != HAL_OK)
  {  _Error_Handler(__FILE__, __LINE__);  }

  sConfigIC.ICPolarity = TIM_INPUTCHANNELPOLARITY_RISING;
  sConfigIC.ICSelection = TIM_ICSELECTION_DIRECTTI;
  sConfigIC.ICPrescaler = TIM_ICPSC_DIV1;
  sConfigIC.ICFilter = 15;
  if (HAL_TIM_IC_ConfigChannel(&htim2, &sConfigIC, TIM_CHANNEL_1) != HAL_OK)
  {  _Error_Handler(__FILE__, __LINE__);  }

  sConfigIC.ICPolarity = TIM_INPUTCHANNELPOLARITY_FALLING;
  sConfigIC.ICSelection = TIM_ICSELECTION_INDIRECTTI;
  if (HAL_TIM_IC_ConfigChannel(&htim2, &sConfigIC, TIM_CHANNEL_2) != HAL_OK)
  {  _Error_Handler(__FILE__, __LINE__);  }

  sMasterConfig.MasterOutputTrigger = TIM_TRGO_RESET;
  sMasterConfig.MasterSlaveMode = TIM_MASTERSLAVEMODE_DISABLE;
  if (HAL_TIMEx_MasterConfigSynchronization(&htim2, &sMasterConfig) != HAL_OK)
  {  _Error_Handler(__FILE__, __LINE__);  }
}
```

```
static void MX_TIM3_Init(void)  //Inicia TIMER3
{
  TIM_ClockConfigTypeDef sClockSourceConfig;
  TIM_MasterConfigTypeDef sMasterConfig;
  TIM_OC_InitTypeDef sConfigOC;

  htim3.Instance = TIM3;
  htim3.Init.Prescaler = 0;
  htim3.Init.CounterMode = TIM_COUNTERMODE_UP;
  htim3.Init.Period = 1200;
  htim3.Init.ClockDivision = TIM_CLOCKDIVISION_DIV1;
  htim3.Init.AutoReloadPreload = TIM_AUTORELOAD_PRELOAD_DISABLE;
  if (HAL_TIM_Base_Init(&htim3) != HAL_OK)
  { _Error_Handler(__FILE__, __LINE__);  }

  sClockSourceConfig.ClockSource = TIM_CLOCKSOURCE_INTERNAL;
  if (HAL_TIM_ConfigClockSource(&htim3, &sClockSourceConfig) != HAL_OK)
  {  _Error_Handler(__FILE__, __LINE__);  }
  if (HAL_TIM_PWM_Init(&htim3) != HAL_OK)
  {  _Error_Handler(__FILE__, __LINE__);  }

  sMasterConfig.MasterOutputTrigger = TIM_TRGO_RESET;
  sMasterConfig.MasterSlaveMode = TIM_MASTERSLAVEMODE_DISABLE;
  if (HAL_TIMEx_MasterConfigSynchronization(&htim3, &sMasterConfig) != HAL_OK)
  {  _Error_Handler(__FILE__, __LINE__);  }

  sConfigOC.OCMode = TIM_OCMODE_PWM1;
  sConfigOC.Pulse = 0;
  sConfigOC.OCPolarity = TIM_OCPOLARITY_HIGH;
  sConfigOC.OCFastMode = TIM_OCFAST_DISABLE;
  if (HAL_TIM_PWM_ConfigChannel(&htim3, &sConfigOC, TIM_CHANNEL_3) != HAL_OK)
  {  _Error_Handler(__FILE__, __LINE__);  }
  if (HAL_TIM_PWM_ConfigChannel(&htim3, &sConfigOC, TIM_CHANNEL_4) != HAL_OK)
  {  _Error_Handler(__FILE__, __LINE__);  }
  HAL_TIM_MspPostInit(&htim3);
}

static void MX_TIM4_Init(void)  //Inicia TIMER4
{
  TIM_ClockConfigTypeDef sClockSourceConfig;
  TIM_SlaveConfigTypeDef sSlaveConfig;
  TIM_IC_InitTypeDef sConfigIC;
  TIM_MasterConfigTypeDef sMasterConfig;

  htim4.Instance = TIM4;
  htim4.Init.Prescaler = 1000;
  htim4.Init.CounterMode = TIM_COUNTERMODE_UP;
  htim4.Init.Period = 65000;
  htim4.Init.ClockDivision = TIM_CLOCKDIVISION_DIV1;
  htim4.Init.AutoReloadPreload = TIM_AUTORELOAD_PRELOAD_DISABLE;
  if (HAL_TIM_Base_Init(&htim4) != HAL_OK)
  {  _Error_Handler(__FILE__, __LINE__);  }

  sClockSourceConfig.ClockSource = TIM_CLOCKSOURCE_INTERNAL;
  if (HAL_TIM_ConfigClockSource(&htim4, &sClockSourceConfig) != HAL_OK)
  {  _Error_Handler(__FILE__, __LINE__);  }
  if (HAL_TIM_IC_Init(&htim4) != HAL_OK)
  {  _Error_Handler(__FILE__, __LINE__);  }
```

```
  sSlaveConfig.SlaveMode = TIM_SLAVEMODE_RESET;
  sSlaveConfig.InputTrigger = TIM_TS_TI1FP1;
  sSlaveConfig.TriggerPolarity = TIM_INPUTCHANNELPOLARITY_RISING;
  sSlaveConfig.TriggerPrescaler = TIM_ICPSC_DIV1;
  sSlaveConfig.TriggerFilter = 15;
  if (HAL_TIM_SlaveConfigSynchronization(&htim4, &sSlaveConfig) != HAL_OK)
  {  _Error_Handler(__FILE__, __LINE__);  }

  sConfigIC.ICPolarity = TIM_INPUTCHANNELPOLARITY_RISING;
  sConfigIC.ICSelection = TIM_ICSELECTION_DIRECTTI;
  sConfigIC.ICPrescaler = TIM_ICPSC_DIV1;
  sConfigIC.ICFilter = 15;
  if (HAL_TIM_IC_ConfigChannel(&htim4, &sConfigIC, TIM_CHANNEL_1) != HAL_OK)
  {  _Error_Handler(__FILE__, __LINE__);  }

  sConfigIC.ICPolarity = TIM_INPUTCHANNELPOLARITY_FALLING;
  sConfigIC.ICSelection = TIM_ICSELECTION_INDIRECTTI;
  if (HAL_TIM_IC_ConfigChannel(&htim4, &sConfigIC, TIM_CHANNEL_2) != HAL_OK)
  {  _Error_Handler(__FILE__, __LINE__);  }

  sMasterConfig.MasterOutputTrigger = TIM_TRGO_RESET;
  sMasterConfig.MasterSlaveMode = TIM_MASTERSLAVEMODE_DISABLE;
  if (HAL_TIMEx_MasterConfigSynchronization(&htim4, &sMasterConfig) != HAL_OK)
  {  _Error_Handler(__FILE__, __LINE__);  }
}

static void MX_USART2_UART_Init(void)  //Inicia USART2
{
  huart2.Instance = USART2;
  huart2.Init.BaudRate = 115200;
  huart2.Init.WordLength = UART_WORDLENGTH_8B;
  huart2.Init.StopBits = UART_STOPBITS_1;
  huart2.Init.Parity = UART_PARITY_NONE;
  huart2.Init.Mode = UART_MODE_TX_RX;
  huart2.Init.HwFlowCtl = UART_HWCONTROL_NONE;
  huart2.Init.OverSampling = UART_OVERSAMPLING_16;
  if (HAL_UART_Init(&huart2) != HAL_OK)
  {  _Error_Handler(__FILE__, __LINE__); }
}

static void MX_GPIO_Init(void)  //Inicia Pinos
{
  GPIO_InitTypeDef GPIO_InitStruct;

  __HAL_RCC_GPIOC_CLK_ENABLE();
  __HAL_RCC_GPIOD_CLK_ENABLE();
  __HAL_RCC_GPIOA_CLK_ENABLE();
  __HAL_RCC_GPIOB_CLK_ENABLE();

  HAL_GPIO_WritePin(GPIOC, RIGHT_MOTOR_DIRECTION_Pin|LEFT_MOTOR_DIRECTION_Pin|
      BRIDGE_MOTOR_RIGHT_ENABLE_Pin|BRIDGE_MOTOR_LEFT_ENABLE_Pin|LD4_Pin|LD3_Pin,
      GPIO_PIN_RESET);

  GPIO_InitStruct.Pin = RIGHT_MOTOR_DIRECTION_Pin|LEFT_MOTOR_DIRECTION_Pin|
      BRIDGE_MOTOR_RIGHT_ENABLE_Pin|BRIDGE_MOTOR_LEFT_ENABLE_Pin |LD4_Pin|LD3_Pin;
  GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
  GPIO_InitStruct.Pull = GPIO_NOPULL;
  GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
  HAL_GPIO_Init(GPIOC, &GPIO_InitStruct);
```

```
}

void HAL_UART_RxCpltCallback(UART_HandleTypeDef *huart)  //interrupcao da serial
{
  //string recebida vai ser do tipo abaixo
  //000Xa;bccc;deee;ffffY000\n onde
  //a = informacao de estado, onde:
  //b = sentido de giro roda esquerda, onde:
  //    + = frente
  //    - = tras
  //c = velocidade desejada da roda esquerda
  //d = sentido de giro roda direita
  //e = velocidade desejada roda direita
  //f = Parte do quadro para deteccao de erro

  uint8_t i;
  if (huart->Instance == USART2)  //se a recepcao vem do micro de controle
  {
    if (Rx_indx==0)  //limpa a string na primeira recepcao
    {  for (i=0;i<100;i++) string_recebida_controle[i]=0;  }

    if (Rx_data[0]!=10)         //ultimo caractere = /n
    {  string_recebida_controle[Rx_indx++]=Rx_data[0];  }
    else
    {
      Rx_indx=0;
      if(limpa_quadro()==1)  //se o quadro recebido estiver ok, libera o resto do procedimento
      {
        envia_velocidades();  //envia as velocidades reais para o controle
        libera_calculo_controle = 1;
        if( aguarda_desbloqueio_emergencia == 1 )
        {
          estado = NORMAL;
          aguarda_desbloqueio_emergencia = 0;
        }
      }
      else
      {  libera_calculo_controle = 0;  }
    }
    HAL_UART_Receive_IT(&huart2, Rx_data, 1);   //reinicia a cada recepcao
  }
}

void calcula_controle_velocidade(void)  //Funcao para calculo do controle de velocidade
{
  //Tensao maxima nos motores (escolhida)
  float tensao_max_motores = 8.0;
  float max_vel = 160;

  float Ts = 0.05;  //frequencia de amostragem

  le_velocidade_rodas();  //le as velocidades

  //se as velocidades forem diferentes, bota no estado orientando
  if(velocidade_desejada_dir != velocidade_desejada_esq)
  {  estado = ORIENTANDO;  }
  else
  {  estado = NORMAL;  }
```

```
  //Calculo do controle para a roda direita
  Erro_velocidade_dir_ant = Erro_velocidade_dir;
  Integral_velocidade_dir_ant = Integral_velocidade_dir;
  Erro_velocidade_dir = velocidade_desejada_dir - velocidade_dir_real;
  Proporcional_velocidade_dir = (float) kp_velocidade * Erro_velocidade_dir;
  Integral_velocidade_dir = (float) (((ki_velocidade * Ts) / 2) * (Erro_velocidade_dir  +
      Erro_velocidade_dir_ant)) + Integral_velocidade_dir_ant;

  variavel_controle_velocidade_dir = (float) Proporcional_velocidade_dir +
      Integral_velocidade_dir;

  //calcula o delta de velocidade
  float delta_velocidade = velocidade_dir_real - velocidade_esq_real;
  if(estado == ORIENTANDO)  //se orientando, tira fora o delta para a roda esquerda
  {  Erro_velocidade_esq = velocidade_desejada_esq - velocidade_esq_real;  }
  else
  {  Erro_velocidade_esq = velocidade_desejada_esq + delta_velocidade - velocidade_esq_real;}

  //Calculo do controle para a roda esquerda
  Erro_velocidade_esq_ant = Erro_velocidade_esq;
  Integral_velocidade_esq_ant = Integral_velocidade_esq;
  Proporcional_velocidade_esq = (float) kp_velocidade * Erro_velocidade_esq;
  Integral_velocidade_esq = (float) (((ki_velocidade * Ts) / 2) * (Erro_velocidade_esq  +
      Erro_velocidade_esq_ant)) + Integral_velocidade_esq_ant;

  variavel_controle_velocidade_esq = (float) Proporcional_velocidade_esq +
      Integral_velocidade_esq;

  tensao_motor_esquerdo = variavel_controle_velocidade_esq;
  tensao_motor_direito = variavel_controle_velocidade_dir;

  if(tensao_motor_esquerdo > tensao_max_motores){
    tensao_motor_esquerdo = tensao_max_motores;
    Integral_velocidade_esq = Integral_velocidade_esq_ant;      //anti-windup
  }else if(tensao_motor_esquerdo < 0.0)
  {
    tensao_motor_esquerdo = 0.0;
    Integral_velocidade_esq = 0.0;      //anti-windup
  }

  if(tensao_motor_direito > tensao_max_motores){
    tensao_motor_direito = tensao_max_motores;
    Integral_velocidade_dir = Integral_velocidade_dir_ant;      //anti-windup
  }else if(tensao_motor_direito < 0.0){
    tensao_motor_direito = 0.0;
    Integral_velocidade_dir = 0.0;      //anti-windup
  }

  //muda o valor da tensao nos motores
  PWM_RIGHT = (uint16_t) ((tensao_motor_direito / 24.0) * 1200);
  PWM_LEFT = (uint16_t) ((tensao_motor_esquerdo / 24.0) * 1200);
}

int limpa_quadro(void) //separa os valores numericos da string recebida do bloco de controle
{
  int escape = 0, i = 0, j = 0;

  while (escape == 0)  //loop para ver onde esta o identificador inicial = X
  {
```

```
  if (string_recebida_controle [i] == 'X')
  {
    for (j = 0; j < 100; j++)
    {
      if (string_recebida_controle [i] == 'Y')   //quando achar o Y termina o loop
      {
        escape = 1;
        break;
      }
      else
      {
        string_recebida_controle [j] = string_recebida_controle [i + 1];
        i++;
      }
    }
  }
  else
  {  i++;  }
}

//seleciona o estado a depender do estado recebido
if(string_recebida_controle[0] == '9')
{  estado = EMERGENCIA;  }
else if(string_recebida_controle[0] == '1')
{  estado = NORMAL;  }
else if(string_recebida_controle[0] == '2')
{  estado = ORIENTANDO;  }


if (string_recebida_controle [2] == '+')  //le os valores de sentido de giro
{  sentido_giro_motor_esquerdo = 1;  }
else if (string_recebida_controle [2] == '-')
{  sentido_giro_motor_esquerdo = 0;  }

if (string_recebida_controle [8] == '+')
{  sentido_giro_motor_direito = 1;  }
else if (string_recebida_controle [8] == '-')
{  sentido_giro_motor_direito = 0;  }

//configura o sentido de giro
sentido_giro_motores(sentido_giro_motor_esquerdo, sentido_giro_motor_direito);

//rotina para pegar a parte do motor esquerdo
char texto [6];
for (i = 0; i < 6; i++)
{  texto [i] = '␣';  }
for (i = 0; i < 4; i++)
{  texto [i] = string_recebida_controle [i+3];  }

int numero = 0;
velocidade_desejada_esq = 0;
for (i = 0; i < 4; i++)
{
  switch (texto [i])
  {
    case '1':
      numero = 1;
      break;
    case '2':
```

```
      numero = 2;
      break;
    case '3':
      numero = 3;
      break;
    case '4':
      numero = 4;
      break;
    case '5':
      numero = 5;
      break;
    case '6':
      numero = 6;
      break;
    case '7':
      numero = 7;
      break;
    case '8':
      numero = 8;
      break;
    case '9':
      numero = 9;
      break;
    case '0':
      numero = 0;
      break;
    default:
      break;
    }
    if (i == 0)
    {  velocidade_desejada_esq = velocidade_desejada_esq + (1000 * numero);  }
    else if (i == 1)
    {  velocidade_desejada_esq = velocidade_desejada_esq + (100 * numero);  }
    else if (i == 2)
    {  velocidade_desejada_esq = velocidade_desejada_esq + (10 * numero);  }
    else if (i == 3)
    {  velocidade_desejada_esq = velocidade_desejada_esq + (1 * numero);  }
}

numero = 0;
//rotina para pegar a parte da roda direita
for (i = 0; i < 6; i++)
{  texto [i] = '␣';  }
for (i = 0; i < 4; i++)
{  texto [i] = string_recebida_controle [i + 9];  }

velocidade_desejada_dir = 0;
for (i = 0; i < 4; i++)
{
  switch (texto [i])
  {
    case '1':
      numero = 1;
      break;
    case '2':
      numero = 2;
      break;
    case '3':
      numero = 3;
```

```
      break;
    case '4':
      numero = 4;
      break;
    case '5':
      numero = 5;
      break;
    case '6':
      numero = 6;
      break;
    case '7':
      numero = 7;
      break;
    case '8':
      numero = 8;
      break;
    case '9':
      numero = 9;
      break;
    case '0':
      numero = 0;
      break;
    default:
      break;
    }
    if (i == 0)
    {  velocidade_desejada_dir = velocidade_desejada_dir + (1000 * numero);  }
    else if (i == 1)
    {  velocidade_desejada_dir = velocidade_desejada_dir + (100 * numero);  }
    else if (i == 2)
    {  velocidade_desejada_dir = velocidade_desejada_dir + (10 * numero);  }
    else if (i == 3)
    {  velocidade_desejada_dir = velocidade_desejada_dir + (1 * numero);  }
}

numero = 0;
//rotina para pegar a parte da deteccao de erro no quadro
for (i = 0; i < 6; i++)
{  texto [i] = '␣';  }
for (i = 0; i < 5; i++)
{  texto [i] = string_recebida_controle [i + 14];  }

int erro_quadro = 0;
for (i = 0; i < 5; i++)
{
  switch (texto [i])
  {
    case '1':
     numero = 1;
     break;
    case '2':
     numero = 2;
     break;
    case '3':
     numero = 3;
     break;
    case '4':
     numero = 4;
     break;
```

```
      case '5':
       numero = 5;
       break;
      case '6':
       numero = 6;
       break;
      case '7':
       numero = 7;
       break;
      case '8':
       numero = 8;
       break;
      case '9':
       numero = 9;
       break;
      case '0':
       numero = 0;
       break;
      default:
       break;
      }
      if (i == 0)
      {  erro_quadro = erro_quadro + (10000 * numero);  }
      else if (i == 1)
      {  erro_quadro = erro_quadro + (1000 * numero);  }
      else if (i == 2)
      {  erro_quadro = erro_quadro + (100 * numero);  }
      else if (i == 3)
      {  erro_quadro = erro_quadro + (10 * numero);  }
      else if (i == 4)
      {  erro_quadro = erro_quadro + (1 * numero);  }
  }

  short retorno = 0;
  //se o quadsro for ok
  if(erro_quadro == (velocidade_desejada_dir + velocidade_desejada_esq))
  {  retorno = 1;  }
  else
  {  retorno = 0;  }
  return retorno;
}

void envia_velocidades(void)  //envia as velocidades das rodas para o bloco de controle
{
  char sent_esq;
  char sent_dir;

  int erro_quadro = 0;
  if(sentido_giro_motor_esquerdo == 1)
  {  sent_esq = '+';  }
  else if(sentido_giro_motor_esquerdo == 0)
  {  sent_esq = '-';  }

  if(sentido_giro_motor_direito == 1)
  { sent_dir = '+';  }
  else if(sentido_giro_motor_direito == 0)
  {  sent_dir = '-';  }

  int v_esq = (int) velocidade_esq_real;
```

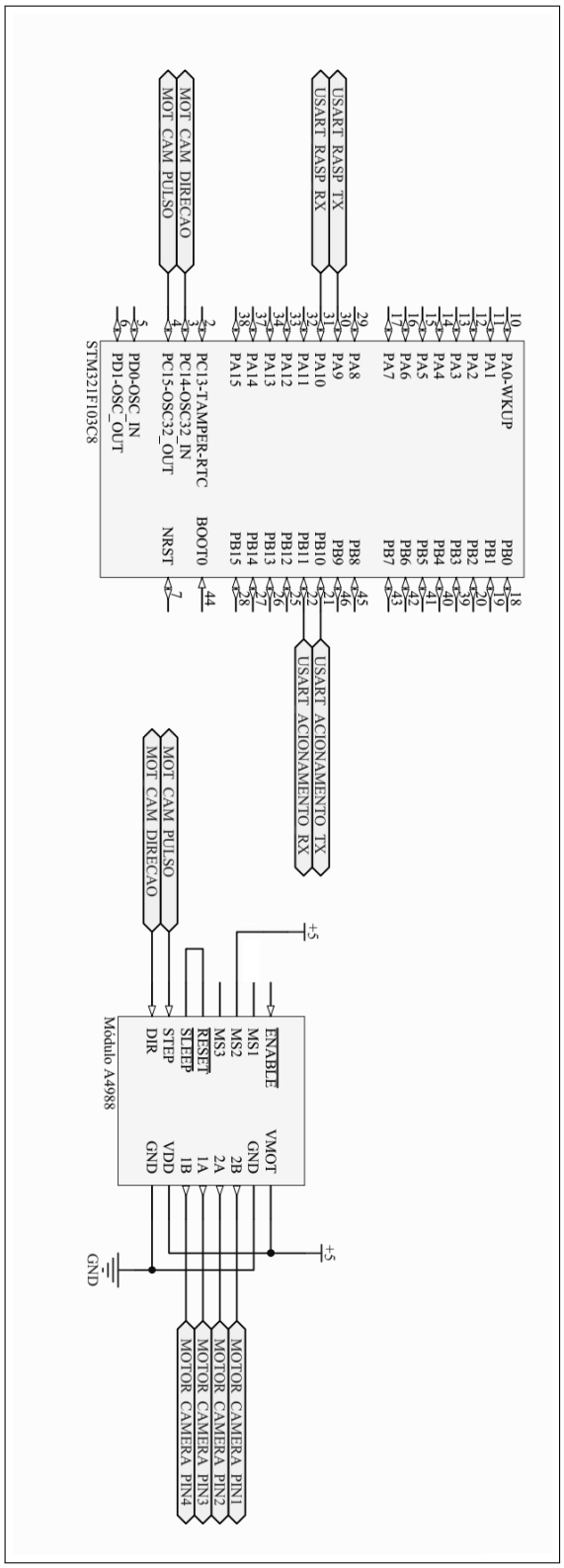
```
  int v_dir = (int) velocidade_dir_real;

  erro_quadro = v_esq + v_dir;

  //cria a string e envia
  sprintf(string_envia_controle, "000X%c%4d;%c%4d;%5dY000\n", sent_esq, v_esq, sent_dir, v_dir
      , erro_quadro);
  Serial_Transmit(&huart2, string_envia_controle, 30);
}

void HAL_TIM_PeriodElapsedCallback(TIM_HandleTypeDef *htim)  //interrupcao dos timers
{
  if(htim->Instance == TIM1)  //se a interrupcao for do timer 1
  {
    if(libera_calculo_controle == 1)  //soh faz o calculo se o ultimo quadro recebido for ok
    {
      HAL_GPIO_TogglePin(LD3_GPIO_Port, LD3_Pin);   //pisca o led

      //se tiver emergencia e nao esta aguardando comando vai parar
      //depois que parar vai esperar outro comando da serial para liberar o calculo
      if( (estado == EMERGENCIA) && (aguarda_desbloqueio_emergencia == 0) )
      {  parada_de_emergencia();  }
      else if (aguarda_desbloqueio_emergencia == 0)
      {  calcula_controle_velocidade();  }
    }
  }
}

void parada_de_emergencia(void)  //funcao da parada de emergencia
{
  //zera as variaveis interessadas
  velocidade_desejada_dir = 0;
  velocidade_desejada_esq = 0;
  Integral_velocidade_dir = 0.0;
  Integral_velocidade_dir_ant = 0.0;
  Integral_velocidade_esq = 0.0;
  Integral_velocidade_esq_ant = 0.0;

  //faz um loop que decrementa a tensao nos motores
  int i = 0;
  for (i=0; i<40; i++)
  {
    tensao_motor_direito = tensao_motor_direito - 0.25;
    tensao_motor_esquerdo = tensao_motor_esquerdo - 0.25;

    if(tensao_motor_direito<0)
    {  tensao_motor_direito = 0;  }
    if(tensao_motor_esquerdo<0)
    {  tensao_motor_esquerdo = 0;  }

    HAL_Delay(20);
    PWM_RIGHT = (uint16_t) ((tensao_motor_direito / 24.0) * 1200);
    PWM_LEFT = (uint16_t) ((tensao_motor_esquerdo / 24.0) * 1200);

    if( (tensao_motor_direito == 0) && (tensao_motor_esquerdo == 0) )
    {
      //tensao final
      tensao_motor_direito = 0;
      tensao_motor_esquerdo = 0;
```

```
      PWM_RIGHT = (uint16_t) ((tensao_motor_direito / 24.0) * 1200);
      PWM_LEFT = (uint16_t) ((tensao_motor_esquerdo / 24.0) * 1200);

      //atualiza as velocidades reais
      velocidade_dir_real = 0;
      velocidade_esq_real = 0;

      break;
    }
  }
  //seta a variavel indicando que vai esperar um novo comando para andar
  aguarda_desbloqueio_emergencia = 1;
}

void le_velocidade_rodas(void)  //Funcao que realiza a leitura de frequencia dos encoders
{
  //variaveis usadas na leitura
  periodo_esq = 0;
  periodo_dir = 0;

  int i = 0;
  short cont_esq = 0;
  short cont_dir = 0;

  //faz 6 leituras e tira a media
  for(i=0;i<1000;i++)
  {
    //pega o valor do periodo do sinal pra roda esquerda
    if( (__HAL_TIM_GET_FLAG(&htim2, TIM_FLAG_CC1) == 1) && (cont_esq < 6) )
    {
      periodo_esq = periodo_esq + HAL_TIM_ReadCapturedValue(&htim2, TIM_CHANNEL_1);
      cont_esq++;
      __HAL_TIM_CLEAR_FLAG(&htim2, TIM_FLAG_CC1);
    }

    //pega o valor do periodo do sinal pra roda direita
    if( (__HAL_TIM_GET_FLAG(&htim4, TIM_FLAG_CC1) == 1) && (cont_dir < 6) )
    {
      periodo_dir = periodo_dir + HAL_TIM_ReadCapturedValue(&htim4, TIM_CHANNEL_1);
      cont_dir++;
      __HAL_TIM_CLEAR_FLAG(&htim4, TIM_FLAG_CC1);
    }

    //se achar 6 valores pros dois termina
    if( (cont_esq == 6) && (cont_dir == 6) )
    {  break;  }
  }

  //calcula as medias
  periodo_esq = periodo_esq / cont_esq;
  periodo_dir = periodo_dir / cont_dir;

  //calcula a velocidade real da roda esquerda
  if(periodo_esq !=0)
  {
    velocidade_esq_real = HAL_RCC_GetPCLK1Freq()/ (htim2.Instance->PSC + 1);
    velocidade_esq_real = (float) (velocidade_esq_real / periodo_esq);
  }
  else
```

```
  {  velocidade_esq_real = 0.0;  }

  //calcula a velocidade real da roda direita
  if(periodo_dir !=0)
  {
    velocidade_dir_real = HAL_RCC_GetPCLK1Freq() / (htim4.Instance->PSC + 1);
    velocidade_dir_real = (float) (velocidade_dir_real / periodo_dir);
  }
  else
  {  velocidade_dir_real = 0.0;  }
}


void sentido_giro_motores(short esquerdo, short direito)  //configura pinos pontes h
{
  //1 = Frente
  //2 = Tras

  if (esquerdo == 1)
  {  HAL_GPIO_WritePin(LEFT_MOTOR_DIRECTION_GPIO_Port, LEFT_MOTOR_DIRECTION_Pin,
      GPIO_PIN_RESET);  }
  else if (esquerdo == 0)
  {  HAL_GPIO_WritePin(LEFT_MOTOR_DIRECTION_GPIO_Port, LEFT_MOTOR_DIRECTION_Pin, GPIO_PIN_SET)
      ;  }

  if (direito == 1)
  {  HAL_GPIO_WritePin(RIGHT_MOTOR_DIRECTION_GPIO_Port, RIGHT_MOTOR_DIRECTION_Pin,
      GPIO_PIN_RESET);  }
  else if (direito == 0)
  {  HAL_GPIO_WritePin(RIGHT_MOTOR_DIRECTION_GPIO_Port, RIGHT_MOTOR_DIRECTION_Pin,
      GPIO_PIN_SET);  }
}

void Serial_Transmit(UART_HandleTypeDef *huart, uint8_t *s, int num) //envia dados serial
{
  int cont = 0;
  while (cont<num)
  {
    while (!(huart->Instance->SR & 0x00000040));
    huart->Instance->DR = (*s & (uint16_t)0x01FF);     //envia um char por vez
    *s++;
    cont++;     //aumenta valor do ponteiro
  }
}
```

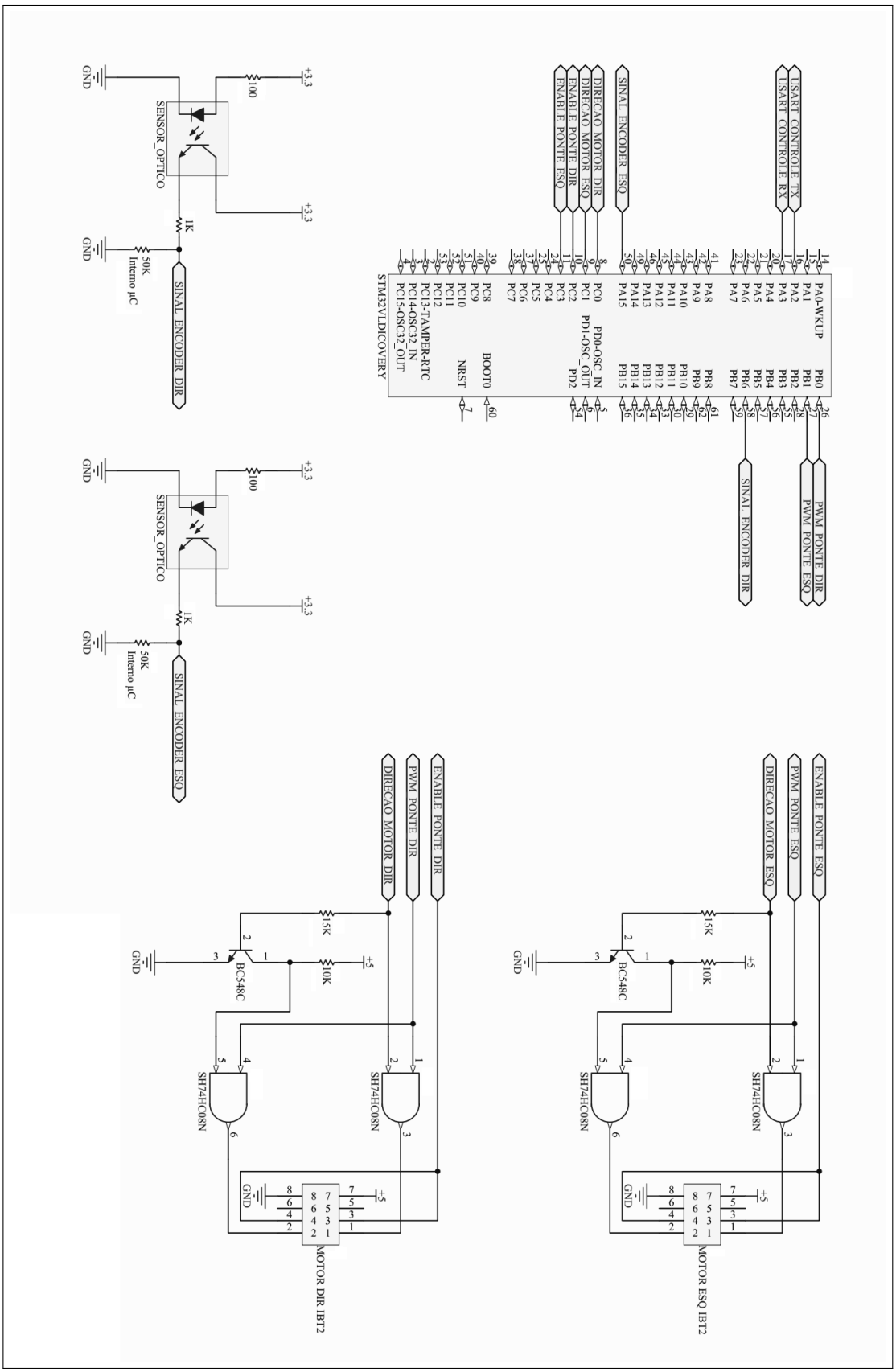
APÊNDICE D – ESQUEMAS ELETRÔNICOS

Figura 38: Circuito Eletrônico do Bloco de Controle



Fonte: Elaborado pelo Autor (2018).

Figura 39: Circuito Eletrônico do Bloco de Acionamento



Fonte: Elaborado pelo Autor (2018).