INSTITUTO FEDERAL DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA DE SANTA CATARINA
CAMPUS FLORIANÓPOLIS
DEPARTAMENTO ACADÊMICO DE ELETRÔNICA
CURSO SUPERIOR DE TECNOLOGIA EM ELETRÔNICA INDUSTRIAL

ASAPH MROSS BECKER

IDENTIFICAÇÃO DE ALVO POR VISÃO COMPUTACIONAL APLICADA A UM ROBÔ DA COMPETIÇÃO FIRST ROBOTICS COMPETITION

FLORIANÓPOLIS, OUTUBRO DE 2018.

INSTITUTO FEDERAL DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA DE SANTA CATARINA
CAMPUS FLORIANÓPOLIS
DEPARTAMENTO ACADÊMICO DE ELETRÔNICA
CURSO SUPERIOR DE TECNOLOGIA EM ELETRÔNICA INDUSTRIAL

ASAPH MROSS BECKER

IDENTIFICAÇÃO DE ALVO POR VISÃO COMPUTACIONAL APLICADA A UM ROBÔ DA COMPETIÇÃO FIRST ROBOTICS COMPETITION

Trabalho de Conclusão de Curso submetido ao Instituto Federal de Educação, Ciência e Tecnologia de Santa Catarina como parte dos requisitos para obtenção do título de Tecnólogo em Eletrônica Industrial.
Professor Orientador: Fernando S. Pacheco.

FLORIANÓPOLIS, OUTUBRO DE 2018.

Ficha de identificação da obra elaborada pelo autor.

Becker, Asaph
IDENTIFICAÇÃO DE ALVO POR VISÃO COMPUTACIONAL APLICADA A UM ROBÔ DA COMPETIÇÃO FIRST ROBOTICS COMPETITION / Asaph Becker ; orientação de Fernando Santana Pacheco.
– Florianópolis, SC, 2018.
64 p.

Trabalho de Conclusão de Curso (TCC) – Instituto Federal de Santa Catarina, Câmpus Florianópolis. CST em Eletrônica Industrial. Departamento Acadêmico de Eletrônica.
Inclui Referências.

1. Visão computacional. 2. Robô. 3. Raspberry Pi. 4. roboRIO. I. Santana Pacheco, Fernando. II. Instituto Federal de Santa Catarina. Departamento Acadêmico de Eletrônica. III. Título.

IDENTIFICAÇÃO DE ALVO POR VISÃO COMPUTACIONAL APLICADA A UM ROBÔ DA COMPETIÇÃO FIRST ROBOTICS COMPETITION

ASAPH MROSS BECKER

Este trabalho foi julgado adequado para obtenção do Título de Tecnólogo em Eletrônica Industrial e aprovado na sua forma final pela banca examinadora do Curso Superior de Tecnologia em Eletrônica Industrial do Instituto Federal de Educação, Ciência e Tecnologia de Santa Catarina.

Florianópolis, 10 de outubro, 2018.

BANCA EXAMINADORA:

FERNANDO SANTANA PACHECO, PROF. DR. ENG.

ALEXANDRE LEIZOR SZCZUPAK, PROF. DR. ENG.

REGINALDO STEINBACH, PROF. ME.

AGRADECIMENTOS

Agradeço a Deus por ter me guiado nesses anos e me abençoado.

À minha esposa Cristhina, por todo o apoio, incentivo e companheirismo nos momentos bons e ruins durante toda esta etapa da minha vida.

Aos meus pais, Carlos e Marcia que sempre me apoiaram e incentivaram na busca dos meus sonhos.

À minha irmã Natali, por todo o suporte, amizade e parceria desde a minha infância, estando disposta a me ajudar nas horas boas e nas horas ruins.

À minha sobrinha Isabella, por tornar meus finais de semana mais alegres.

Ao meu irmão Jeser e minha cunhada Daiane por todo o suporte e conselhos que me deram.

Aos meus sogros Altair e Eliana por todo o apoio que me deram.

Aos professores do IFSC, em especial ao meu professor e orientador Fernando Pacheco, pela dedicação, orientação e paciência na realização deste trabalho.

Aos meus colegas do curso, em especial ao Henrique, Adriano, Lauro, Vinicius e Cesar, por todos esses anos de amizade e parceria.

Ao Instituto Federal de Educação, Ciência e Tecnologia de Santa Catarina, pela estrutura e por proporcionar um ensino de qualidade.

À todos que de alguma formar contribuíram na minha formação profissional.

“Conte-me, e eu esquecerei; ensina-me, e eu poderei lembrar; envolva-me, e eu aprenderei.”

Benjamin Franklin

RESUMO

O presente trabalho apresenta o desenvolvimento de uma aplicação de visão computacional capaz de identificar um alvo em tempo real. Tal aplicação é empregada em um robô da competição First Robotics Competition para alinhá-lo a um alvo. Desenvolvemos algoritmos em linguagem C, Python e Java, utilizando funções da biblioteca de processamento de imagens OpenCV, para realizar a aquisição das imagens, limiarização e detecção de contornos. A informação da posição do alvo é transmitida para o módulo roboRIO, que comanda os motores que movimentam as rodas do robô. Neste trabalho, avaliamos a robustez dos algoritmos em diferentes condições de iluminação. Os testes realizados, tanto com imagens estáticas quanto com capturas de vídeo em tempo real, mostram o correto funcionamento do algoritmo de identificação do alvo e sua aplicação em tempo real para alinhamento do robô.

Palavras-Chaves: Visão computacional; Robô; *Raspberry Pi; roboRIO.*

ABSTRACT

This work presents the development of a computer vision application capable of identifying a target in real time. This application is embedded in a First Robotics Competition robot, for aligning it to a target. We developed algorithms in C, Python, and Java, using OpenCV, a computer vision library. These algorithms accomplish the image acquisition, binarization, and contour detection. Information about the target position is transmitted to the roboRIO module, that controls the motors attached to the robot wheels. In this work, we evaluated the robustness of the identification for different lighting conditions. Our tests, both with static images and real time video capture, show that the identification algorithm works correctly and the real time application for alignment is possible.

Keywords: Computer vison; Robot; *Raspberry Pi; roboRIO.*

LISTA DE SIGLAS

UDP: Protocolo de Datagrama de Usuário;

FRC: First Robotics Competition;

PWM: Pulse Width Modulation;

USB: Universal Serial Bus;

SPI: Serial Peripheral Interface;

FPGA: Field-programmable gate array;

RAM: Random-access memory;

CSI: Camera Serial Interface;

TCP: Transmission Control Protocol.

LISTA DE FIGURAS

SUMÁRIO

1. INTRODUÇÃO

Uma competição muito importante na área de robótica é chamada de *FIRST ROBOTICS COMPETITION* (FRC) (FIRST, 2018). Nesta competição, tarefas são definidas pela organização e devem ser cumpridas por robôs construídos pelas equipes participantes. Para cada edição, a organização define tarefas inéditas, e normalmente, os robôs devem manipular, lançar e entregar objetos em determinadas áreas e alvos. Há dois períodos onde estas tarefas são executadas:

- Autônomo, em que o robô deve executar as tarefas sem que haja nenhum tipo de interação com os participantes.
- Controlado, em que os participantes controlam o robô utilizando *joysticks*.

Espera-se que, no período autônomo, as equipes sejam capazes de programar os seus robôs para executarem as tarefas. Por se tratar de um período em que as tarefas são mais difíceis de serem executadas, a pontuação é maior no período autônomo, proporcionando uma grande vantagem para as equipes que conseguem executá-las com êxito. Portanto, para atingir esse objetivo, se faz necessário o uso de diversas tecnologias como sensores, controladores digitais e visão computacional, possibilitando a leitura, processamento e controle de dados referente à aceleração, velocidade, distância e posição.

Estas aplicações trazem também grandes vantagens para o período controlado, pois auxiliam o participante a controlar e alinhar o robô, garantindo que o objeto possa ser lançado ao alvo precisamente, substituindo o uso da visão humana e diminuindo as chances de erros.

O autor deste trabalho participa dessas competições desde 2005. Ao longo do tempo, se deparou com problemas em relação ao correto posicionamento do robô na execução das tarefas autônomas e controladas, onde foram necessárias muitas tentativas para a execução das tarefas. Este trabalho foi motivado pelas vantagens que o uso da visão computacional agrega ao controle da posição do robô.

1.1. Justificativa

Diante da necessidade do correto alinhamento de um robô FRC para lançar um objeto ao alvo com precisão e confiabilidade, é necessário o uso de diversas tecnologias, entre elas a visão computacional, tendo como uma de suas aplicações a identificação de objetos em imagens. O período autônomo consiste em apenas 15 segundos, tempo limite em que o robô deve pontuar, podendo haver diversas interferências em campo, como outros robôs ou componentes do jogo. Assim a aplicação da visão computacional é importante para que uma pontuação maior e mais consistente seja alcançada.

Portanto, devido a esta dificuldade no alinhamento do robô ao alvo, esta pesquisa se justifica como uma contribuição para um melhor ajuste de posição e o correto alinhamento do robô através do uso de visão computacional.

1.2. Objetivo geral

Utilizar visão computacional para identificar um alvo, auxiliando no ajuste da posição do robô para alinhá-lo com este alvo.

1.3. Objetivos específicos

Como objetivos específicos, que corroboram para atingir o objetivo geral, elencamos:

a) Utilizar as funções da biblioteca *Opencv* para detecção de contorno;

b) desenvolver um programa em C/C++ que detecte o centro do alvo;

c) implementar o programa de detecção em um sistema computacional;

d) avaliar o funcionamento do programa implementado no sistema computacional*;*

e) empregar o protocolo de comunicação UDP (*User Datagram Protocol*) no programa de detecção para envio das informações da *Raspberry Pi* para o r*oboRIO*;

f) analisar o uso das informações para ajustar o posicionamento do robô;

g) desenvolver um programa em Java que ajuste a posição do robô utilizando as informações do centro do alvo;

h) apresentar os resultados.

2. FUNDAMENTAÇÃO TEÓRICA

Para o desenvolvimento deste trabalho, foram necessários conhecimentos das áreas de visão computacional e instrumentação. As próximas seções têm como objetivo apresentar assuntos essenciais para a compreensão do desenvolvimento e funcionamento do projeto.

Iniciamos com uma apresentação breve da competição FRC. Em seguida, discutimos tópicos relacionados à visão computacional e uma das bibliotecas mais conhecidas da área, *OpenCV*. Apresentamos, também, algumas informações sobre o protocolo de comunicação UDP. Sobre o hardware, apresentamos as plataformas *Raspberry Pi*, *Roborio*, além de alguns sensores.

2.1. First Robotics Competition

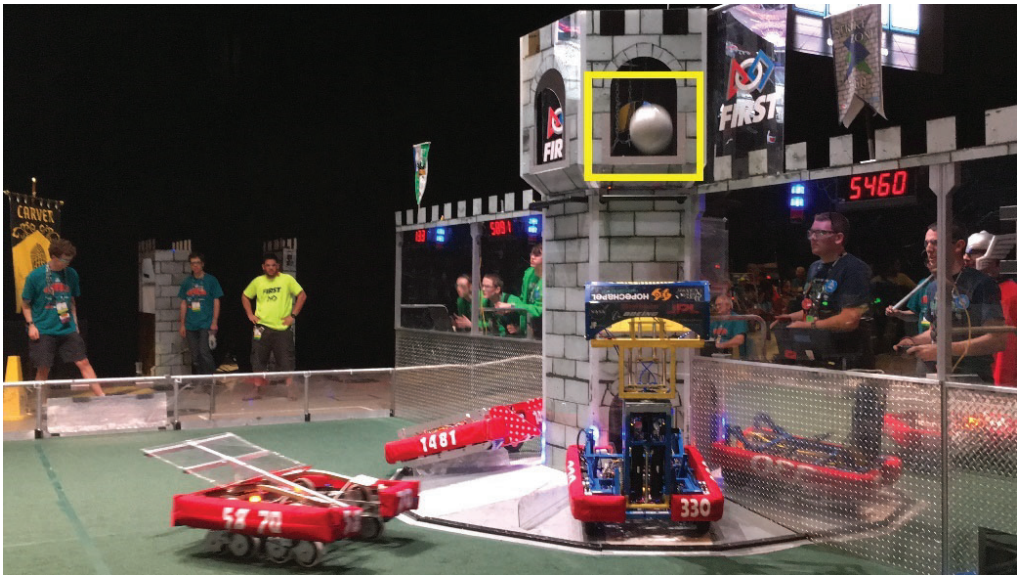
A *First Robotics Competition* (FRC) é uma competição de robótica que conta atualmente com mais de 3 mil equipes participantes de 27 diferentes nacionalidades (FIRST ROBOTICS COMPETITION, 2018). Ela foi fundada por Dean Kamen em 1992 com a intenção de incentivar alunos do ensino médio a ingressarem em áreas de tecnologia, colocando-os em contato com diferentes profissionais.

O grande desafio imposto é a construção em apenas seis semanas, de um robô, que seja apto a exercer determinadas tarefas que são reveladas pela competição no início de cada temporada. Neste período, o robô deve ser construído e programado pelos alunos, com auxílio de seus mentores. Mentores são profissionais de diversas áreas que participam de equipes com o intuito de compartilhar conhecimento e experiência que normalmente não seriam adquiridos em sala de aula (KALIN, 2015).

Após o período de construção do robô, as equipes então competem em etapas regionais, podendo se classificar para a etapa internacional. Cada partida é formada por um período autônomo de quinze segundos seguidos por um período controlado de dois minutos e quinze segundos. No período autônomo, o robô deve

ser pré-programado para executar movimentos e cumprir determinada tarefas. A fim de facilitar o uso da visão computacional, a competição utiliza fitas retrorrefletivas em alvos e objeto. Pode-se visualizar um exemplo na Figura 1. Deste modo simplifica-se processo de identificação de objetos sem a intervenção humana.

Figura 1 – Período controlado de uma partida da competição FRC de 2016



Fonte: LANDAU (2017).

Como assegura Melchior (2016), pode-se dizer que a competição é um grande incentivo para o ingresso de alunos na área de tecnologia. Neste contexto, fica claro que o número de alunos que ingressam em áreas de tecnologia nos Estados Unidos da América tem aumentado (MELCHIOR, COHEN, *et al.*, 2005). É importante, constatar que os alunos também desenvolvem outras habilidades como comunicação, trabalho em equipe, gerenciamento e resolução de problemas.

> A missão da competição é mostrar para estudantes de qualquer idade que ciência, tecnologia e solução de problemas não somente são interessantes e gratificantes, mas são caminhos comprovados para carreiras de sucesso e um futuro brilhante para todos nós (KAMEN, 2014, p. 1).

Portanto, através do desafio proposto pela competição os estudantes têm a oportunidade de ter um primeiro contato com tecnologias através de seu uso

prático, que normalmente, só seriam abordadas em um curso de graduação, como o uso da visão computacional. Com maior conhecimento sobre as áreas de engenharia, a escolha de suas carreiras profissionais pode ser melhor fundamentada.

2.2. Visão computacional

Segundo Dawson (2014), a visão computacional é a análise automática de imagens e vídeos por computadores, a fim de obter informações. Portanto, a "visão computacional é a transformação de dados de uma câmera estática ou de vídeo em uma decisão ou em uma nova representação" (BRADSKI e KAEHLER, 2008, p. 2).

Como bem nos asseguram Ballard e Brown (1982), a visão computacional é a área que estuda e desenvolve tecnologias para que as máquinas possam ter a percepção da visão, automatizando e integrando uma ampla gama de processos e representações.

Para Szeliski (2010), a visão computacional facilita replicar a visão humana usando software e hardware. Para este autor:

> A visão computacional permite descrever o mundo que vemos em uma ou mais imagens e reconstruir suas propriedades, como forma, iluminação e distribuição de cores (SZELISKI, 2010, p. 5).

A visão computacional vem sendo usada em uma ampla variedade de aplicações como em visão de máquina, onde permite o reconhecimento óptico de caracteres, inspeções de máquinas, imagens médicas, vigilância, reconhecimento de impressões digitais e biometria, captura de movimento entre outras aplicações (SZELISKI, 2010).

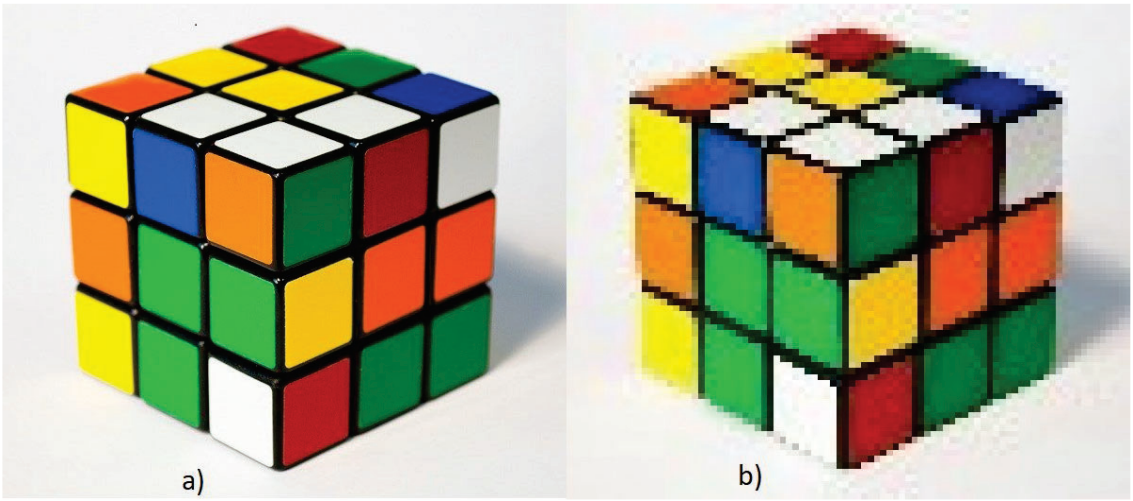
Fica evidente que a visão computacional contribui para o avanço da tecnologia pois possibilita que máquinas e aplicativos em diversas áreas tornem-se capazes de identificar objetos e extrair informações de um ambiente, permitindo que tarefas sejam automatizadas.

2.2.1. Imagem digital

Uma imagem digital é uma matriz de pixels, sendo um pixel o menor elemento manipulável de uma imagem. De forma simples, pode-se dizer que um pixel é um pequeno ponto preenchido por uma cor ou nível de intensidade de cinza e que a imagem digital é formada por milhares destes pequenos pontos (SMITH, 1999).

Fica evidente que quanto maior o número de pixels cobrindo uma mesma área, melhor o detalhamento da imagem. Por exemplo, na Figura 2(a), apresenta-se a imagem de um cubo mágico com 314.400 pixels. Na Figura 2(b), a mesma imagem é reproduzida, mas com 7.110 pixels. Devido à maior resolução, a primeira imagem reproduz mais detalhes e tem uma transição de cores mais suave do que a segunda.

Figura 2 – Imagens de cubos mágicos em diferentes resoluções (número de pixels)



Fonte: RACCOON (2018).

Em relação à cor, existem três tipos de imagens digitais sendo elas binária, imagens em escala de cinza e imagens coloridas. Nas imagens binárias, cada pixel pode ser representado por apenas dois valores, ou seja, um único bit, sendo assim um pixel pode assumir apenas as cores preta ou branca. Em imagens em escala de cinza de 8 bits, a imagem contém apenas um canal de 8 bits e cada pixel pode ser representado por diferentes tons de cinza, assumindo valores entre 0

e 255. Nas imagens coloridas cada pixel é representado pelo menos por três canais e a cor resultante é a união desses valores (MOESLUND, 2012).

2.2.2. O espaço de cor RGB

O espaço de cor RGB é um padrão de representação de cores no qual cada cor é formada pela combinação de três cores primárias: vermelho, verde e azul (GONZALES e WOODS, 2009). A sigla RGB é originada da letra inicial de cada uma dessas três cores na língua inglesa (*red, green and blue*).

As imagens em RBG, conhecidas como imagens de cores reais, podem ser representadas matematicamente por três matrizes bidimensionais distintas, uma para cada canal. Cada elemento dessas matrizes é um número inteiro, que representa a intensidade da cor de um pixel. (BARELLI, 2018).

Portanto, cada ponto contém três valores que representam a intensidade das cores vermelho, verde e azul que juntas geram a cor naquele determinado pixel. Geralmente, cada canal é representado por 8 bits podendo então representar um total de $(2^8)^3$ = 16.777.216 cores (GONZALES e WOODS, 2009).

2.3. Opencv

A OpenCV é uma biblioteca de visão computacional de código aberto, portanto pode ser utilizada e alterada sem a necessidade da aquisição uma licença. A OpenCV foi criada com o intuito de acelerar o uso da visão computacional em produtos comerciais fornecendo uma infraestrutura comum para aplicações em visão computacional (OPENCV, 2018).

A biblioteca contém mais de 2500 algoritmos otimizados, incluindo um conjunto abrangente de visão computacional e aprendizado de máquina. A OpenCV possui várias aplicações, entre elas a identificação de bordas, segmentação de imagem, reconhecimento de rostos e gestos, detecção de movimento entre outros (OPENCV, 2018).

> Desde seu lançamento em janeiro de 1999, a OpenCV tem sido usado em muitas aplicações, produtos e esforços de pesquisa. Estas aplicações incluem a união de imagens em mapas via satélite e web, alinhamento de escaneamento de imagens, redução de ruídos em imagens médicas, análise de objetos, segurança e sistemas de detecção de intrusão, monitoramento automático e sistemas de segurança, sistemas de inspeção de fabricação, calibração de câmeras, aplicações militares, veículos aéreos não tripulados, terrestres e submarinos (BRADSKI e KAEHLER, 2008, p. 2).

Sendo assim, a biblioteca OpenCV proporciona ao usuário a possibilidade de desenvolver projetos complexos de visão computacional de maneira mais rápida, pois compartilha uma ampla gama de ferramentas e algoritmos que podem ser adaptados para o uso específico de uma aplicação como no caso deste que trabalho será utilizada para a detecção de um objeto em uma imagem.

2.4. Protocolo UDP

Para que dois ou mais computadores possam trocar informações entre si, por exemplo, a localização de um alvo em uma imagem, é necessário que todos utilizem as mesmas regras para o envio e recebimento destas informações. Estes conjuntos de regras são chamados de protocolo (KUROSE e ROSS, 2013). Existem diversos protocolos de comunicação, porém, este trabalho aborda apenas o protocolo UDP.

O protocolo UDP (*User datagram protocol*) é um protocolo da camada de transporte do padrão TCP/IP. O UDP é um protocolo de comunicação leve e com um modelo de serviço simplificado. É um serviço que não tem orientação a conexão; portanto, não é estabelecida uma conexão antes do envio das informações e não há garantia de ordem e entrega dos dados (KUROSE e ROSS, 2013).

Conforme Coulouris, Dollimore *et al.* (2013), dependendo da aplicação é aceitável o uso de serviços que estão expostos a eventuais falhas de omissão e ordenamento. Os autores deixam claro que ocasionalmente o uso do UDP é uma

solução atraente por não sofrer com sobrecargas à garantia de entrega de dados como armazenamento de informações de estado na origem e no destino, a transmissão de mensagens extras e a latência do remetente.

Desta maneira, o UDP é utilizado para transmissão de dados que não necessitam de confirmação de recebimento, como chamadas de áudio e vídeo, onde pacotes de dados que não são entregue na ordem correta podem ser descartados por não serem mais relevantes.

O protocolo UDP tem maior capacidade de envio de dados do que outros protocolos pois dispõem de um menor cabeçalho de pacote, utilizando somente 8 bytes enquanto o TCP, que é um protocolo orientado a comunicação, utiliza 20 bytes (KUROSE e ROSS, 2013).

Portanto, o uso deste protocolo foi uma boa alternativa para ser utilizada neste projeto em conjunto com a aplicação da visão computacional, pois as limitações desta tecnologia não comprometem o funcionamento do sistema.

2.5. Raspberry Pi

Raspberry Pi é o nome de uma série de *single-board computers* (computadores de placa única), de baixo custo, desenvolvidos por uma fundação do Reino Unido, com foco na área educacional (RASPBERRY, 2018). Dentre a série de computadores *Raspberry Pi* disponíveis atualmente, neste trabalho iremos utilizar a placa *Pi 3 Model B* (Figura 3). A *Raspberry Pi 3 model b* foi escolhida para o desenvolvimento deste projeto pois conta com uma porta *ethernet* e uma porta *CSI* para a câmera, permitindo o uso de uma câmera chamada *Camera module v2*. Além de dispor de 1GB de memória RAM, a placa tem um processador de 64 bits ARM Cortex-A53 com quadro núcleos de processamento. Levou-se em conta também o custo-benefício, pois outros computadores como o *Jetson TX1* que são mais robustos são muito mais caros do que a *Raspberry Pi 3 model b.*

Figura 3 – A *Raspberry Pi* 3 Model B

Fonte: HEALTH (2017).

2.6. RoboRIO

RoboRIO é um controlador avançado de robótica, que permite ser reconfigurado. Seu núcleo é formado por um processador *Cortex-A9* e um FPGA (*Field Programmable Gate Array*). É utilizado para controlar robôs através de diversos protocolos de comunicação que este equipamento suporta, tais como CAN, I2C, SPI, RS232, USB, *Ethernet* e PWM (National Instruments, 2018).

Figura 4 – O *roboRIO*

Fonte: NATIONAL INSTRUMENTS (2018).

Este computador foi desenvolvido pela empresa *National Instruments* e é utilizado pelas equipes participantes da competição *First Robotics Competition (FRC)* para executar os programas desenvolvidos para o controle da movimentação de robôs, além da execução dos movimentos dos subsistemas (FIRST, 2018).

2.7. Sensores

Para que um robô possa executar movimentos precisos é necessário que se tenha acesso a certas informações do robô como: posição, ângulo, velocidade, aceleração entre outros, permitindo o monitoramento e ajuste destes movimentos. Desta maneira se faz necessário o uso de sensores para que seja possível a leitura destes dados.

Para Fraden (2016), um sensor pode ser definido como um dispositivo que recebe e responde a um sinal ou estímulo. A finalidade de um sensor é responder a algum tipo de propriedade física de entrada e convertê-lo em um sinal elétrico compatível com um circuito eletrônico (FRADEN, 2016). “Esta propriedade pode ser algo simples como temperatura ou luminosidade ou uma medida um

pouco mais complexa como a rotação de um motor ou a distância de um carro até algum objeto” (PATSKO, 2016, p. 1).

2.7.1. Encoder magnético

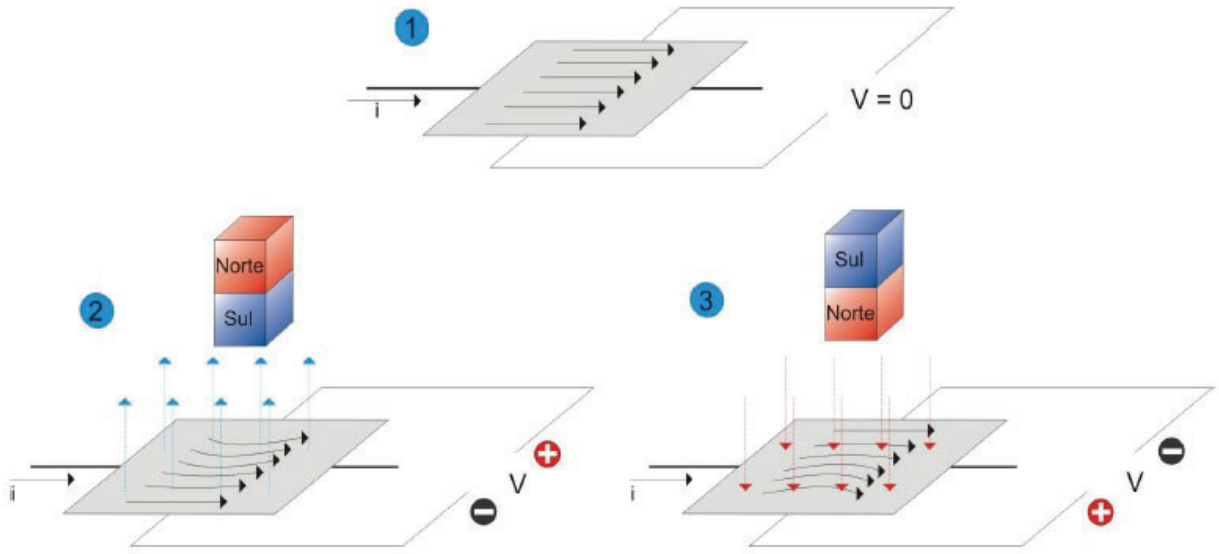
Os encoders são dispositivos que convertem movimentos lineares ou rotacionais em sinais elétricos, estes sinais podem ser lidos por computadores e utilizados para determinar posição, contagem, velocidade ou direção (ENCODER PRODUCTS COMPANY, 2016)

O encoder magnético utiliza propriedades magnéticas para determinar presença, força ou direção de um campo magnético, este sensor pode medir estas propriedades sem que exista contato físico (WILSON, 2005).

“Existem dois tipos principais de sensores de campo magnético: os magneto-resistivos e os que baseiam-se no efeito Hall” (PATSKO, 2016, p. 58). Neste trabalho abordamos apenas o sensor de efeito Hall.

O sensor de efeito Hall é baseado no princípio de interação entre campos magnéticos e cargas elétricas. Conforme ilustrado na Figura 5, uma corrente quando passa por um material condutor se distribui uniformemente sem que haja diferença de potencial nas laterais deste condutor. Porém, quando um ímã é aproximado, o campo magnético gera uma perturbação na distribuição da corrente ao longo deste material, fazendo com que haja um acúmulo de cargas negativas em um dos lados do condutor, gerando uma diferença de potencial em suas laterais. Se o campo magnético for invertido, a tensão também será invertida (PATSKO, 2016).

Figura 5 – Ilustração do efeito Hall



Fonte: PATSKO (2016).

Portanto, um sensor de efeito Hall, quando colocado no centro da rotação de um ímã, estará exposto ao mesmo campo magnético, e poderá detectar a rotação deste imã através da mudança do sentido deste campo (RAMSDEN, 2006)

2.7.1.1 SRX Mag Encoder

O encoder escolhido para este projeto foi o *SRX Mag Encoder*, ilustrado na Figura 6.

O *SRX Mag Encoder* é um sensor de rotação que pode ser utilizado para medir posição rotacional e velocidade. Este sensor detecta o campo magnético de um ímã polarizado diametralmente para determinar a posição de rotação com precisão de 12 bits (CTR ELECTRONICS, 2018).

Figura 6 – SRX Mag Encoder

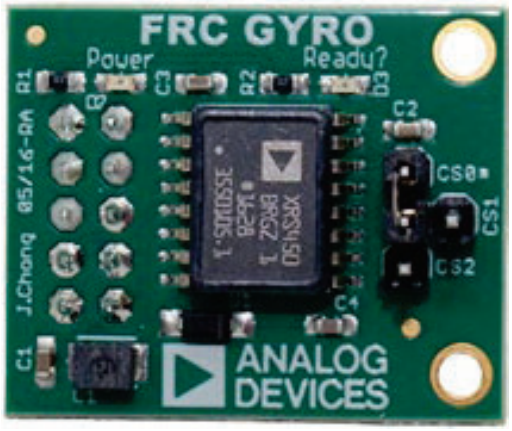
Fonte: CTR ELECTRONICS (2018).

2.7.2. Giroscópio

O giroscópio é provavelmente o sensor mais comum em sistemas de navegação e é utilizado para medir movimento rotacional. Tem sua operação baseada no princípio fundamental da conservação de momento angular, onde, em um sistema de partículas, o momento angular total do sistema em relação a qualquer ponto fixo no espaço permanece constante, desde que nenhuma força externa atue no sistema (FRADEN, 2004).

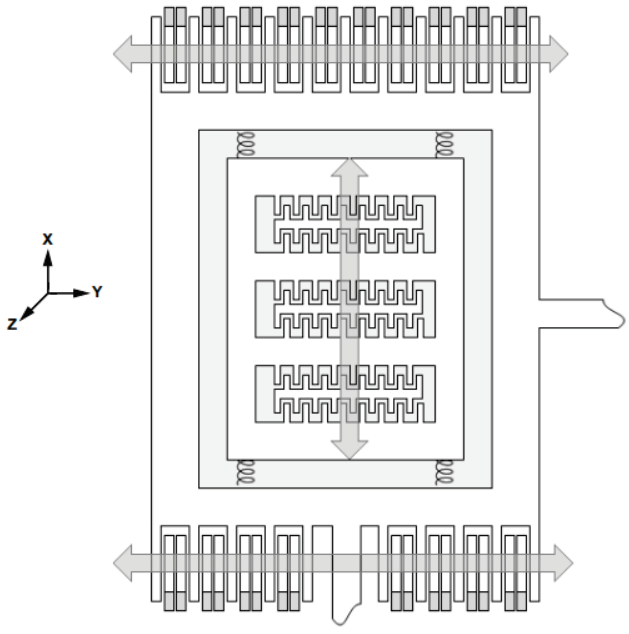
O sensor escolhido para ser utilizado neste projeto foi o *Gyro Board* (Figura 7) fabricado pela *Analog Devices.* Esta placa eletrônica contém o giroscópio ADXRS450, que opera pelo princípio de um giroscópio ressonador. Uma versão simplificada de uma das quatro estruturas de detecção pode ser observada na Figura 8. Cada estrutura de detecção contém um quadro de oscilação que é impulsionado eletricamente para ressonância, produzindo velocidade suficiente para gerar uma força de *Coriolis* ao movimentar-se angularmente, que possibilita a medição deste movimento.

Figura 7 – Sensor GYRO BOARD



Fonte: ANDYMARK (2018).

Figura 8 – Estrutura simplificada do giroscópio



Fonte: ANALOG DEVICES (2013).

2.7.3. Câmera

Uma câmera é um sensor que captura imagens baseada no mesmo princípio do olho humano, portanto, mede a quantidade de luz em diferentes

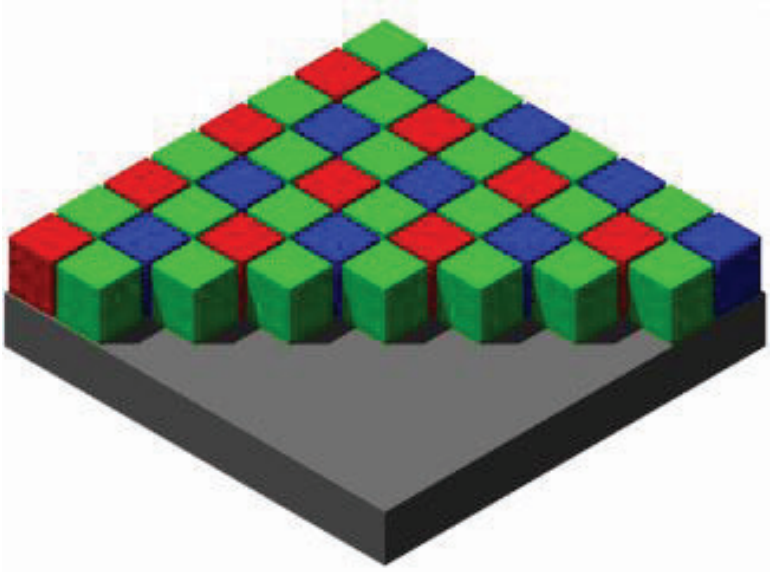
comprimentos de onda (sendo o mais comum vermelho, verde e azul). A maneira que isto é feito depende da quantidade de sensores na câmera, podendo haver três sensores ou apenas um sensor. No caso de três sensores, cada sensor mede a intensidade da sua respectiva cor, resultando em três matrizes bidimensionais que contêm o valor referente ao nível de vermelho, verde e azul. A imagem digital construída por essas matrizes pode ser observada na Figura 9, onde as intensidades de vermelho, verde e azul são mostrados em escala de cinza e a intensidade da cor de cada canal (MOESLUND, 2012).

Figura 9 – Imagem colorida constituída de três planos de cor: vermelho, verde e azul

Fonte: MOESLUND (2012).

Como afirma o autor, uma alternativa mais barata é o uso de apenas um sensor, neste caso cada célula do sensor é sensível à uma das três cores, isto pode ser feito de diversas maneiras, uma delas é utilizando mosaico de filtro *Bayer* (Figura 10), onde 50% das células do sensor detectam verde e o restante das células detectam azul ou vermelho. A razão do sensor conter mais células que detectam verde é devido ao olho humano ser mais sensível ao verde (MOESLUND, 2012).

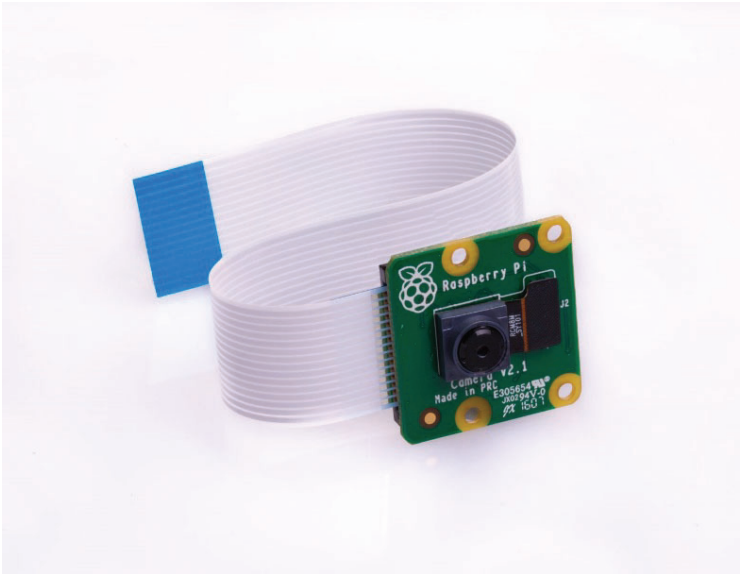
Figura 10 – Mosaico de filtro Bayer um sensor de imagem



Fonte: DXOMARK IMAGE LABS (2018).

A câmera utilizada neste projeto foi a *Camera Module V2* (Figura 11), que contém o sensor de imagem IMX219 da fabricante *SONY.*

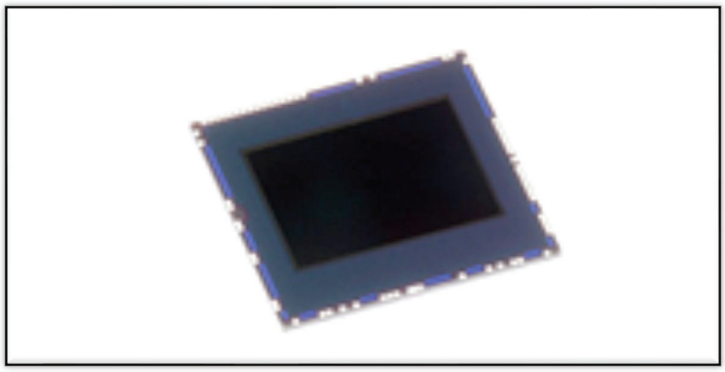
Figura 11 – Camera Module V2



Fonte: RASPBERRY (2018).

O IMX219 (Figura 12) é um sensor de imagem CMOS de pixel ativo com uma matriz de pixels quadrados, contém um total de 8.08M de pixels efetivos (SONY, 2018).

Figura 12 – Sensor IMX219



Fonte: SONY (2018).

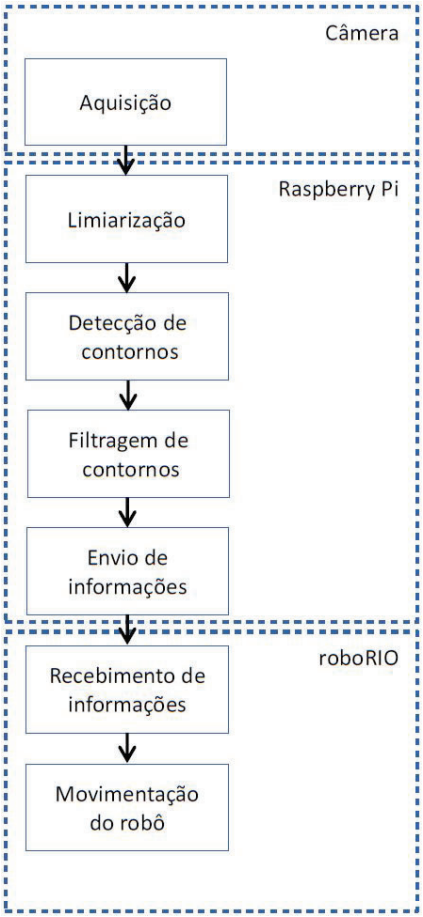
3. DESENVOLVIMENTO DO SISTEMA

Neste capítulo são apresentadas todas as etapas do desenvolvimento do projeto, com o propósito de desenvolver um sistema capaz de identificar a posição do alvo auxiliando no ajuste da posição do robô.

Para atingir o objetivo proposto, foram implementados algoritmos de segmentação de imagem com o uso de funções da biblioteca *OpenCV.* Foi implementado um algoritmo de filtragem de contornos, encarregado de identificar apenas o contorno de interesse. Além disso, foi desenvolvido um algoritmo para a comunicação com o robô.

A arquitetura geral do sistema desenvolvido é apresentada na Figura 13.

Figura 13 – Diagrama de blocos da arquitetura do sistema



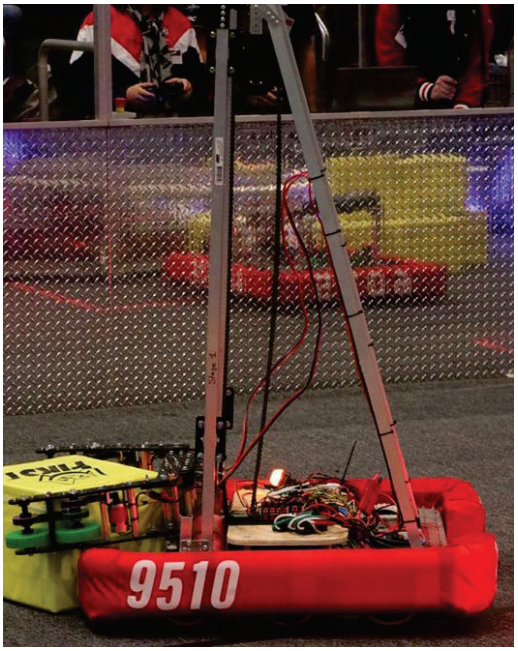
Fonte: Elaboração própria (2018).

A aquisição de imagem é feita por uma câmera e enviada para o computador *Raspberry Pi 3.* Este computador é responsável pela segmentação das imagens, etapa que inclui a limiarização, detecção dos contornos e filtragem de contornos, além do envio das informações da posição do alvo para o roboRIO. O roboRIO recebe as informações e realiza a movimentação do robô. O funcionamento de cada bloco será descrito nas seções seguintes.

3.1 Composição do sistema

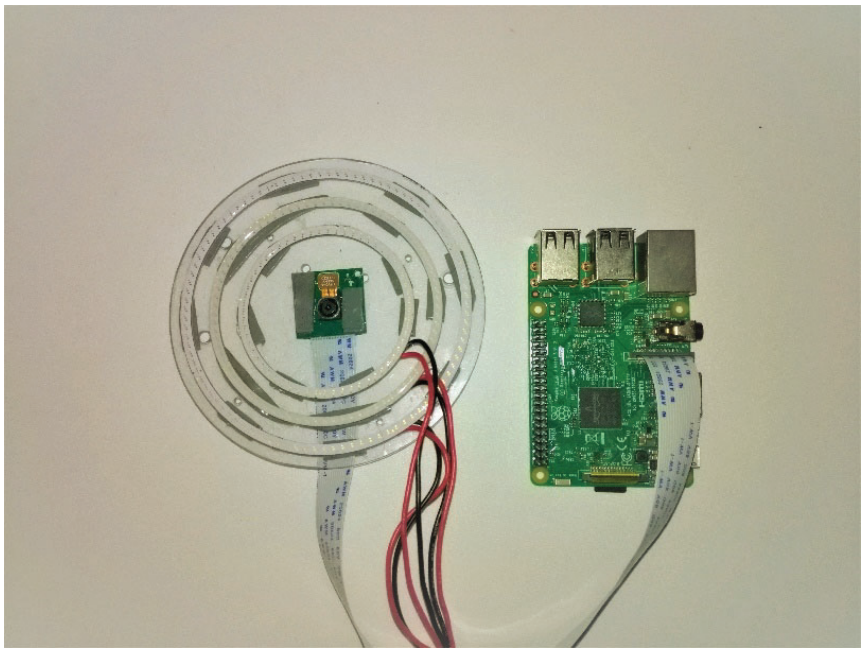
O sistema é composto por dois módulos: o encarregado pela visão computacional (Figura 15) e o módulo responsável pela movimentação do robô (Figura 16). Ambos módulos implementados no robô construído para a competição FRC (Figura 14). O primeiro é composto por uma câmera, um computador *Raspberry Pi 3 model B* e 3 argolas de LEDs. O segundo é composto por um controlador robótico *roboRIO,* um modem *wireless,* sensores, controladores de motores DC, painel de distribuição de energia e motores DC.

Figura 14 – Robô utilizado



Fonte: Elaboração própria (2018).

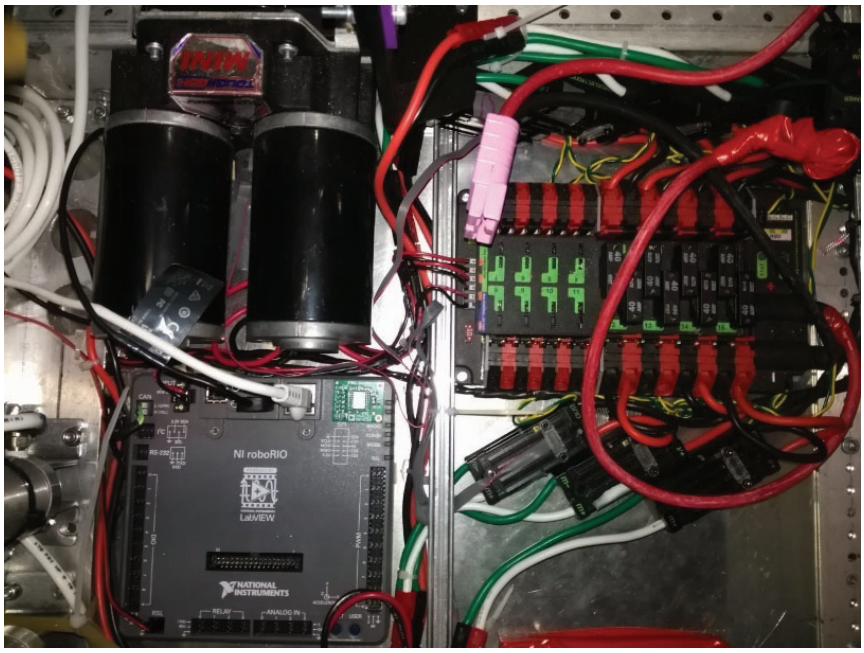
Figura 15 – Módulo responsável pela visão computacional



Fonte: Elaboração própria (2018).

No primeiro módulo, a câmera é responsável pela aquisição e envio de imagens para o computador *Raspberry Pi 3 model B* via protocolo CSI (do inglês *camera serial interface*). O computador por sua vez realiza o processamento de visão computacional com o objetivo de detectar a posição do centro do alvo. As argolas de LEDs exercem uma função importante, pois são responsáveis por incidir uma luz verde sobre as fitas retrorrefletivas coladas nos alvos. A câmera captura a luz refletida pelas fitas, destacando o alvo e facilitando o processo de detecção. A luz verde foi adotada pela competição para evitar interferências com objetos externos da quadra.

Figura 16 – Módulo responsável pela movimentação do robô



Fonte: Elaboração própria (2018).

No segundo módulo, o *roboRIO* é responsável por controlar os controladores de motores DC, receber as informações dos sensores e estabelecer comunicação com a *Raspberry Pi*.

3.2 Segmentação de imagem

A segmentação de imagem refere-se ao processo de separar a imagem digital em múltiplas áreas ou objetos, com o objetivo de mudar ou simplificar uma imagem facilitando sua análise (GONZALES e WOODS, 2009).

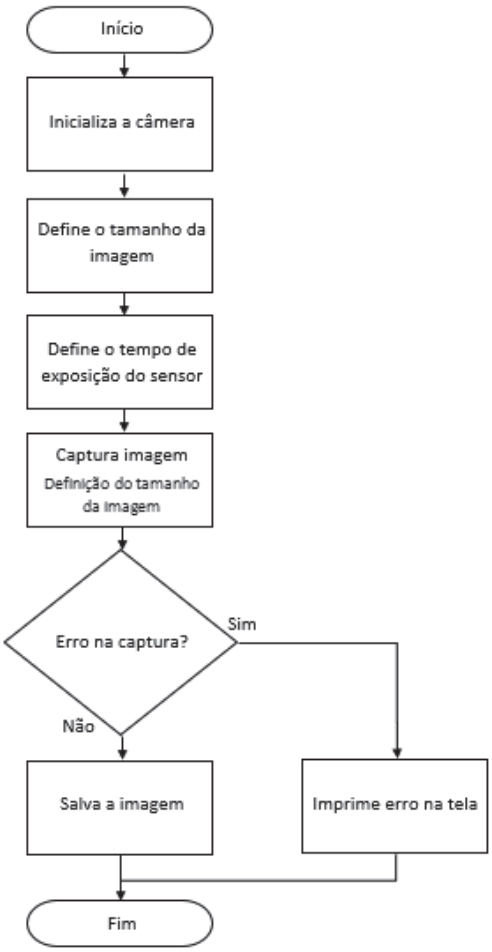
Utilizando os algoritmos da biblioteca *OpenCV,* foram aplicados os métodos de limiarização e detecção de contornos. Posteriormente foi implementado o código que identifica a localização do alvo.

Os testes foram executados, inicialmente, no *Codeblocks* pela praticidade e facilidade que o *software* dispõe para o uso da biblioteca *OpenCV*, permitindo que o pesquisador pudesse analisar os resultados de forma efetiva.

3.2.1 Aquisição da imagem

Foi desenvolvido um algoritmo para captura de imagem em linguagem *Python,* devido à facilidade de alterar as configurações da câmera, como tempo de exposição, dimensões e outras, através de uma biblioteca padrão que permite tais alterações. A aquisição da imagem foi feita utilizando a *Camera module v2.* O funcionamento deste algoritmo pode ser observado na Figura 17 através do seu fluxograma.

Figura 17 – Fluxograma do algoritmo de captura de imagens



Fonte: Elaboração própria (2018).

Notou-se que a luz refletida pela fita estava saturando, em vez de refletir uma luz verde, a fita estava refletindo algo muito próximo da cor branca, como pode ser observado nas Figuras 18 e 19. Na Figura 18, além dos LEDs, havia também a

iluminação por lâmpadas no ambiente. Na Figura 19, apenas os LEDs estavam ligados.

Figura 18 – Imagem saturada com iluminação do ambiente

Fonte: Elaboração própria (2018).

Figura 19 – Imagem saturada sem iluminação do ambiente

Fonte: Elaboração própria (2018).

Como solução, foi reduzido o nível de iluminação, com o desligamento das duas argolas de LEDs maiores. O resultado pode ser visto na Figura 20.

3.2.2 Limiarização da imagem

Limiarização é uma técnica de segmentação de imagem que destaca objetos ou áreas de interesse em imagens, gerando uma imagem binária (DAVIES, 2012).

Nesta etapa foi utilizado a função *inRange()* da biblioteca *OpenCV,* esta função tem por objetivo gerar uma imagem binária detectando apenas as cores de interesse da imagem referentes ao objeto a ser detectado. A imagem exemplo utilizada nesta etapa foi a Figura 20, onde o alvo se encontra em um ambiente que também contém verde, para que possa ser testada a correta limiarização da imagem.

Figura 20 – Imagem a ser limiarizada

Fonte: Elaboração própria (2018)

Primeiramente a função recebe o valor que especifica o limiar, todos os pixels da imagem são então comparados com esse valor. Caso o valor esteja dentro dessa delimitação, o pixel é então preenchido com branco, do contrário é preenchido com preto, gerando então uma imagem binária.

Foram identificados empiricamente os valores mínimos e máximos RGB mostrados na Tabela 1, e passados como parâmetros para a função *inRange()* através da função *Scalar(),* função que representa um vetor de 3 elementos.

Tabela 1 – Parâmetros utilizados na limiarização

Canal	R	G	B
Valor Máximo	118	255	159
Valor Mínimo	0	177	0

Fonte: Elaboração própria (2018)

Deste modo, o uso da função gerou uma imagem binária, conforme Figura 21.

Figura 21 – Imagem binária gerada

Fonte: Elaboração própria (2018)

Nota-se a correta limiarização da imagem, pois apenas a região referente ao alvo foi destacada. Porém, para garantir que a etapa de limiarização funcionasse de maneira correta em ambientes com diferentes iluminações, efetuaram-se testes em diferentes locais para avaliar o comportamento do sistema. Primeiramente, foi efetuado um teste com imagens adquiridas em uma sacada apenas com iluminação natural além dos LEDs. O resultado do teste pode ser observado na Figura 22 e Figura 23.

Figura 22 – Alvo com iluminação natural

Fonte: Elaboração própria (2018).

Figura 23 – Limiar do Alvo com iluminação natural

Fonte: Elaboração própria (2018).

É possível visualizar que mesmo com a iluminação natural o sistema consegue limiarizar a imagem, destacando o alvo. Foram então executados mais testes, com o intuito de testar o comportamento em ambientes mais escuros com e sem iluminação artificial. Na Figura 24, o ambiente está sendo iluminado apenas pela luz natural. A correta limiarização pode ser observada na Figura 25.

Figura 24 – Alvo em um ambiente escuro

Fonte: Elaboração própria (2018).

Figura 25 – Limiar do alvo em um ambiente escuro

Fonte: Elaboração própria (2018).

Foi executado mais um teste no mesmo ambiente do teste anterior, porém, agora contendo iluminação artificial amarela. O resultado pode ser observado nas Figura 26 e Figura 27.

Figura 26 – Alvo em um ambiente com iluminação artificial

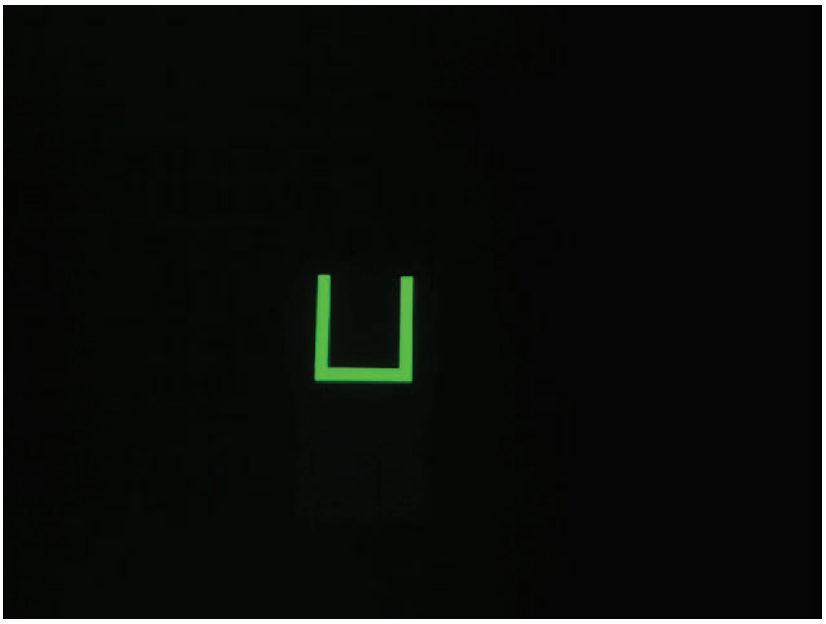
Fonte: Elaboração própria (2018)

Figura 27 – Limiar do alvo em um ambiente com iluminação artificial

Fonte: Elaboração própria (2018).

Com o objetivo de testar o funcionamento do sistema em ambiente com iluminação extremamente precária, foi executado um teste em uma garagem situada no subsolo de um prédio, sem a presença de luz natural, cuja imagem pode ser vista na Figura 28. Devido ao tempo de exposição do sensor ter sido configurado para um valor baixo desde o início dos testes, a Figura 28 aparenta estar em um ambiente totalmente sem iluminação, porém o teste foi executado em um ambiente onde a iluminação artificial era escassa. A correta limiarização é ilustrada na Figura 29.

Figura 28 – Alvo em um ambiente com iluminação precária



Fonte: Elaboração própria (2018).

Figura 29 – Limiar do alvo em um ambiente com iluminação precária

Fonte: Elaboração própria (2018).

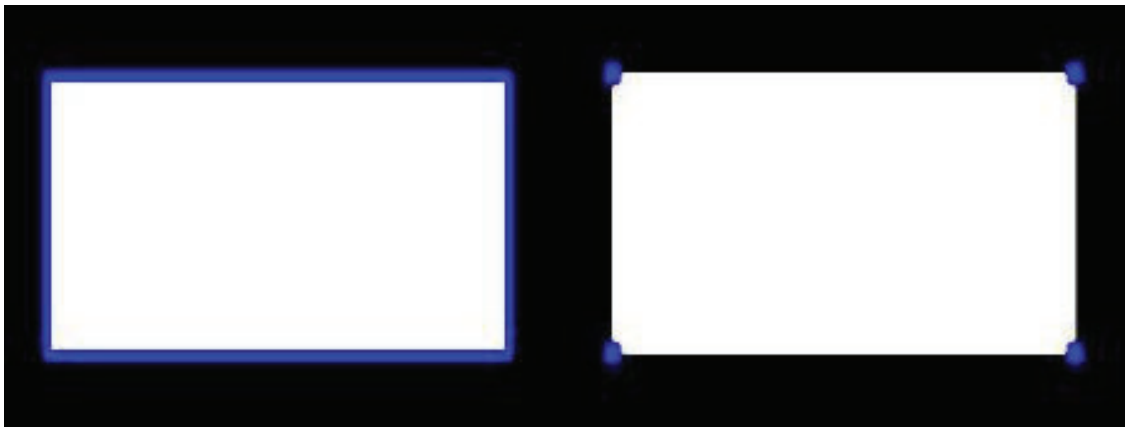
Embora outros espaços de cores, como HSV pudessem ser usados (GUPTA, 2017), verificamos que a combinação de iluminação pelos LEDs e o uso de fitas retrorrefletivas garantiu robustez ao processo. Assim, como o uso da função obteve os resultados esperados em ambientes com diferentes iluminações, foi dado continuidade e avançado para a etapa de detecção de contornos.

3.2.3 Identificação de contornos

Esta etapa tem por objetivo detectar os contornos da imagem, desenhando-os em uma nova imagem. Os contornos podem ser explicados como uma linha que une todos os pontos contínuos do limite de uma região na imagem (GONZALES e WOODS, 2009).

Foi utilizada a função *findContours()* da OpenCV para encontrar os contornos da imagem. Essa função identifica os contornos salvando-os em uma matriz de coordenadas (x,y). Foi usado como parâmetro *CHAIN_APPROX_SIMPLE,* que identifica o uso de um dos métodos de aproximação de contorno. Este método foi escolhido pois elimina os pontos redundantes da representação do contorno, comprimindo-o posteriormente e economizando memória (OpenCV,2018). A Figura 30 mostra o funcionamento desta técnica o retângulo da esquerda em azul representa o contorno do retângulo branco, que contém 734 pontos, os pontos azuis no retângulo da direita representam a informação armazenada sobre o contorno. Percebe-se que o contorno é modelado por apenas 4 pontos nas bordas (OpenCV,2018). Sendo assim, apenas as informações do início e do fim de uma linha são armazenadas. Portanto, esta técnica é apenas aplicada em linhas retas.

Figura 30 – Método de aproximação de contorno



Fonte: OpenCV (2018).

Após a detecção de contornos, foi criada uma matriz de três canais e cada elemento preenchido com zero, gerando uma imagem preta recebendo os contornos posteriormente. Com o uso da função *drawContours(),* os contornos foram desenhados nesta imagem, como pode ser verificado na Figura 31.

Figura 31 – Contorno detectado

Fonte: Elaboração própria (2018).

Conforme ilustrado na Figura 31, observa-se o correto funcionamento da função, detectando e destacando o contorno na imagem.

3.2.4 Filtragem de contornos

A filtragem de contornos é um processo muito importante, pois tem a função de detectar apenas o contorno do alvo, abstraindo outros contornos identificados anteriormente, além da aquisição das informações sobre o centro do alvo.

Para esta etapa foi implementada uma função que recebe parâmetros referente aos valores máximos e mínimos de área, largura, altura, solidez, vértices, proporção da altura e largura em pixels. Nessa etapa os valores desejados são comparados com os valores de cada contorno. Consequentemente, apenas o contorno que contém as características desejadas é detectado. Os valores foram encontrados empiricamente e podem ser vistos na Tabela 2.

Tabela 2 – Parâmetros utilizados na filtragem de contornos

Parâmetro	Área	Largura	Altura	Vértice	Proporção
Valor Máximo	-	200	300	300	1.5
Valor Mínimo	1000	0	0	8	0

Fonte: Elaboração própria (2018).

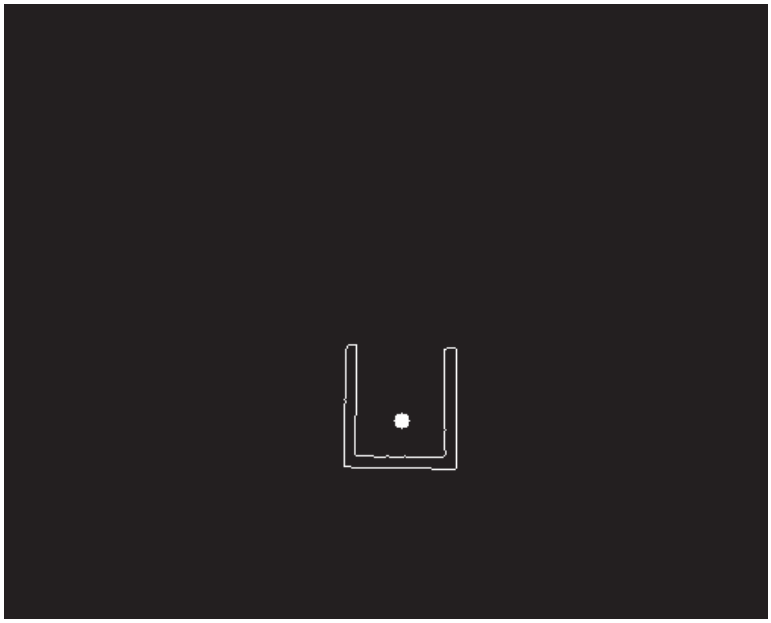
Após detectado, são extraídas as informações sobre a posição do centro do contorno do alvo nos eixos X e Y. Estas informações são enviadas posteriormente para o *roboRIO* permitindo que o robô possa ajustar a sua posição mantendo-se alinhado com o alvo.

Para facilitar a análise do funcionamento desta função, um círculo é desenhado no centro do alvo utilizando a função *circle()* da biblioteca *OpenCV*, esta

função recebe como parâmetros a imagem onde o círculo será desenhado, a posição do centro do círculo, raio, cor, espessura do contorno e tipo da borda.

Como exemplo apresenta-se aqui a aplicação da função à imagem mostrada na Figura 31. O resultado pode ser visto na Figura 32.

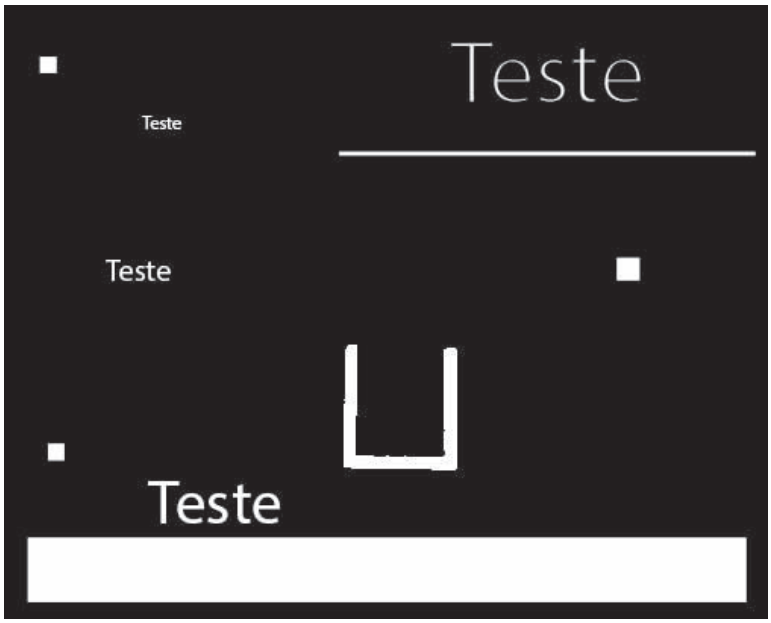
Figura 32 – Alvo detectado



Fonte: Elaboração própria (2018).

Como na etapa de filtragem não foram detectados outros contornos além do alvo, para verificar a robustez da detecção, foram feitas algumas alterações na imagem com o auxílio do *software Adobe Ilustrador* adicionando outros elementos na imagem binária, como mostrado na Figura 33.

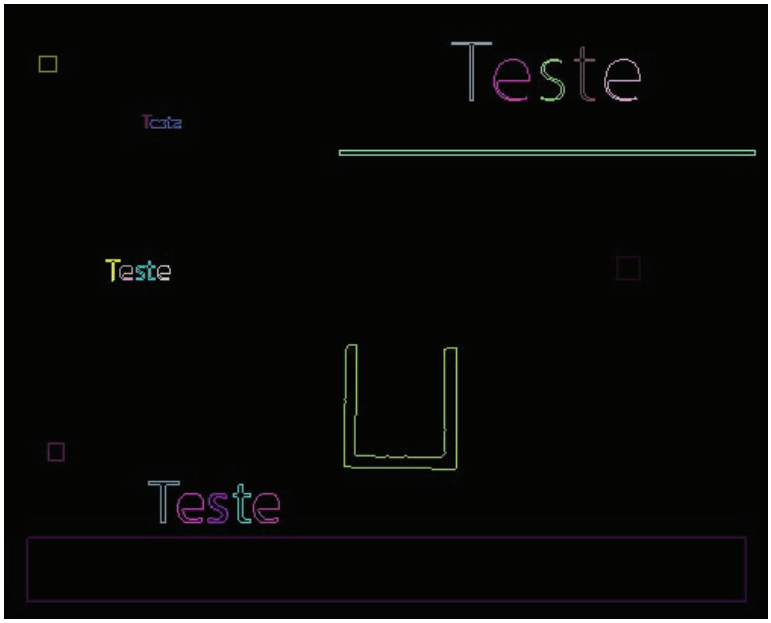
Figura 33 – Imagem binária alterada



Fonte: Elaboração própria (2018).

Após passar pelo mesmo processo de detecção de contorno da Seção 3.2.3, foram detectados 23 contornos destacados na Figura 34.

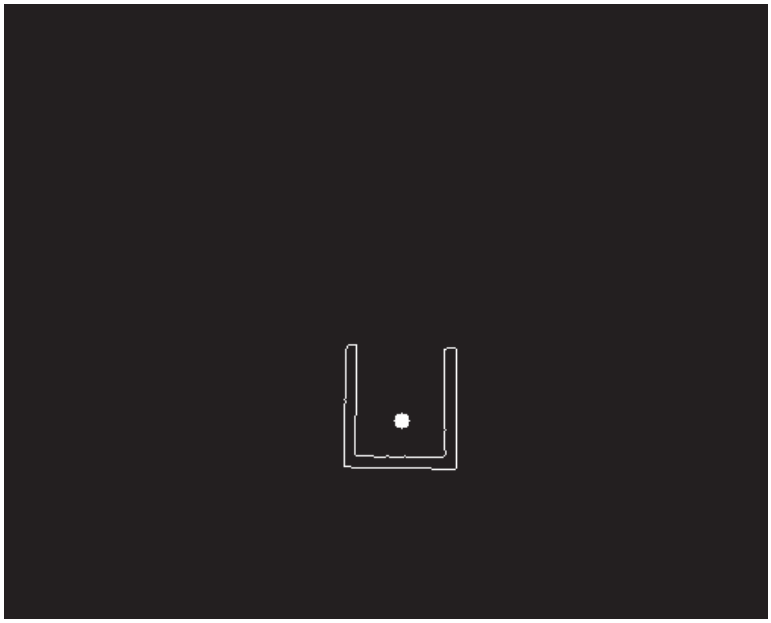
Figura 34 – Contornos detectados



Fonte: Elaboração própria (2018).

A partir da imagem, a função de filtragem de contorno foi executada recebendo a Figura 34 como argumento, tendo como resultado a Figura 35.

Figura 35 – Alvo detectado



Fonte: Elaboração própria (2018).

Foi possível notar que apenas o alvo foi identificado após a etapa de filtragem, portanto constatou-se a adequada operação da função.

3.3 Implementação na Raspberry Pi

Após o correto funcionamento do projeto em um computador pessoal, foi então implementado o código na *Raspberry Pi.* Nesta etapa foram necessárias algumas alterações e adaptações no código.

Inicialmente foram instalados o sistema operacional da *Raspberry Pi* e a biblioteca *OpenCV.* Após a execução de alguns testes para validar o funcionamento da biblioteca foi dado início à implementação do algoritmo de detecção do centro do alvo, que contém todas as etapas apresentadas nas seções anteriores.

Para que as informações do alvo fossem enviadas para o robô, foi necessário o uso de um protocolo de comunicação, portanto foi implementado um cliente UDP, responsável pelo envio das informações do centro do alvo.

Por último, os dois algoritmos foram integrados, resultando na última etapa do projeto executado na *Raspberry Pi,* referente à identificação do alvo e envio das informações

3.3.1 Implementação do algoritmo de detecção do centro do alvo

Inicialmente, foram executados testes com as imagens estáticas adquiridas anteriormente. Após o correto funcionamento dos algoritmos foi dado início ao uso de vídeos adquiridos em tempo real utilizando a *Camera Module* V2. Foi detectado um problema no uso da câmera, pois o tempo de exposição e dimensões do vídeo não poderiam ser alterados em linguagem C.

Desta maneira, foi necessário o uso da biblioteca *RaspiCam* disponível na internet, que permite a alteração de tais configurações. Testes executados mostraram os mesmos resultados obtidos da seção 3.2.

Foram efetuados testes, utilizando imagens adquiridas em tempo real, com taxas de quadros de 20 FPS, 24 FPS e 30 FPS, onde o sistema se comportou de maneira esperada, identificando o centro do alvo.

3.3.2 Implementação do protocolo de comunicação UDP

Com base na fundamentação teórica, foi implementado um algoritmo em linguagem C na Raspberry Pi de um cliente UDP para o envio das informações, e um algoritmo de servidor em linguagem Java para o recebimento das informações no *roboRIO* utilizando o ambiente de desenvolvimento *Eclipse*.

No primeiro teste, entre a *Raspberry Pi* e o *roboRIO,* foi enviada uma informação constante para facilitar a identificação de erros.

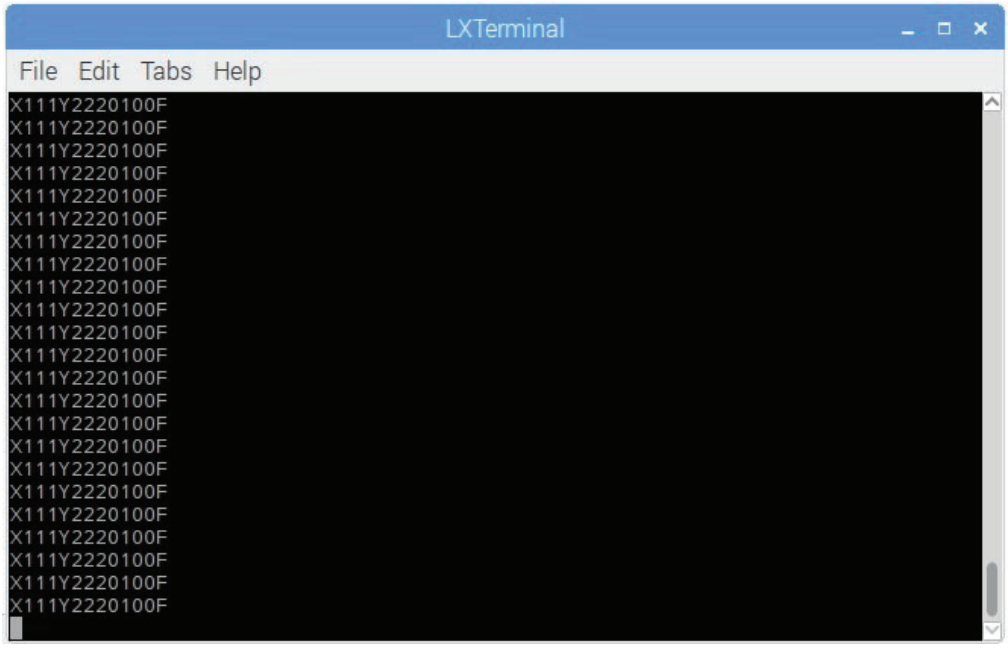
O pacote de dados continha informações sobre a localização do centro do alvo nos eixos x e y da imagem, além da ordem de envio. A ordem de envio é uma informação necessária pois auxilia a identificar qual pacote é o mais recente. Caso um pacote seja recebido em ordem errada, ele deve ser descartado por se tratar de uma informação atrasada. Por exemplo, se o robô recebe um pacote de dados informando a posição do alvo onde a ordem é menor do que a ordem recebida anteriormente isso significa que essa informação está atrasada e não é mais útil para o robô, sendo assim, é ignorada.

A informação enviada para executar o teste foi uma *string* contendo: “X111Y222O100F”, a letra “X” indica que os próximos três caracteres são referentes à posição do centro do alvo no eixo x e a letra “Y” indica que os próximos três caracteres são referentes à posição do centro do alvo no eixo y, a letra “O” indica a ordem do pacote de dados, desta maneira cada vez que um pacote de dados é enviado este valor é incrementado com 1. Quando este valor ultrapassa três dígitos ele é colocado em zero. A letra “F” representa o fim do pacote de dados.

Para que seja possível o envio dos dados, foi necessário configurar o cliente e o servidor para utilizarem a mesma porta de comunicação além de informar o endereço IP. A porta utilizada foi a “5800” e o endereço de IP foi “10.95.10.28”.

O envio dos dados pode ser visualizado através do terminal do *Linux* conforme ilustrado na Figura 36.

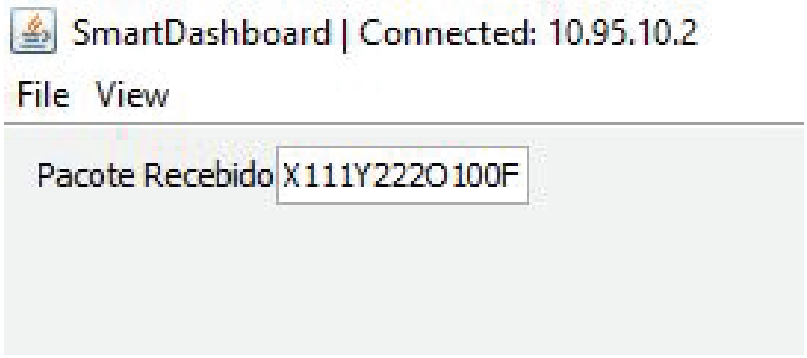
Figura 36 – Pacote de dados enviado



Fonte: Elaboração própria (2018).

Para verificar que o robô estava recebendo o pacote de dados foi utilizado uma interface disponível pela competição chamada *SmartDashboard,* esta interface permite que o usuário possa apresentar valores de variáveis na tela do computador. O pacote recebido pode ser visto na Figura 37.

Figura 37 – Pacote de dados recebido



Fonte: Elaboração própria (2018).

3.3.3 Integração dos algoritmos

Nesta etapa foi feita a integração dos algoritmos de detecção do centro do alvo e comunicação, sendo necessários alguns ajustes e implementação de outros algoritmos para a manipulação do pacote de dados a ser enviado.

Em razão de que *strings* são enviadas como pacote de dados através do protocolo UDP, não é possível o envio dos valores inteiros da posição do centro do alvo nos eixos X e Y adquiridos pela função de detecção do centro do alvo diretamente, além da ordem de envio. Para tornar isso possível foi implementado uma função que converte valores inteiros em *string.*

Após a conversão, foi utilizado a função *strcat*() para concatenar as *strings*. Esta função recebe duas *strings* como parâmetros, referente à *string* fonte e destino. Esta função adiciona a *string* fonte no final da *string* destino.

Deste modo os valores inteiros são convertidos em *string* e adicionados à *string* a ser enviada, além dos caracteres que identificam o início e o fim do pacote e das posições X e Y, conforme a Tabela 3.

Tabela 3 - Concatenação da string a ser enviada

Etapa	Conteúdo da *string*
Adição do caractere “X”	“X”
Adição da posição em x	“X111”
Adição do caractere “Y”	“X111Y”
Adição da posição em y	“X111Y222”
Adição do caractere “O”	“X111Y222O”
Adição da ordem do pacote	“X111Y222O333”
Adição do caractere “F”	“X111Y222O333F”

Fonte: Elaboração própria (2018).

3.4 Implementação do código do robô

Utilizando a biblioteca WPILIB fornecida pela competição foi desenvolvido o algoritmo do robô em linguagem *Java*. Desta maneira, o *roboRIO* recebe os dados via protocolo UDP, decodificando-os e executando a movimentação dos motores para que o robô possa continuar alinhado ao alvo.

3.4.1 Implementação do algoritmo de movimentação do robô

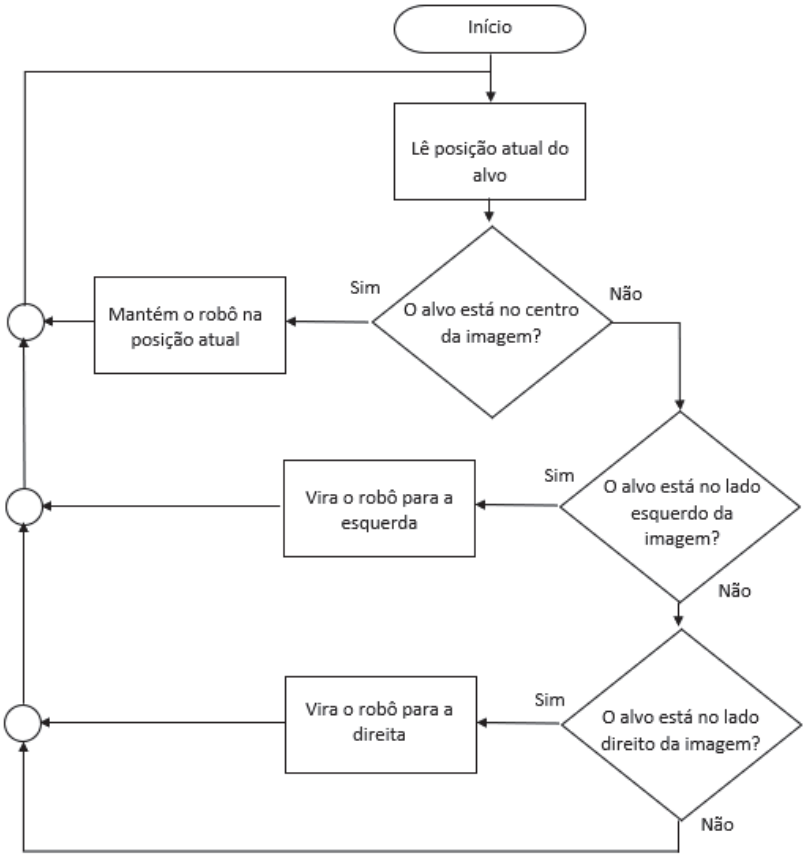
Inicialmente foi criada a classe *DriveTrain* do subsistema responsável pela movimentação do robô e o método *drive()* que executa esta movimentação. Este recebe dois valores referentes à movimentação de cada lado do chassi do robô, entre -1.0 e +1.0. Valores positivos movem o robô para a frente e valores negativos movem-no para trás. Para que o robô fique sem movimento, a função deve receber o valor zero. Este sistema funciona como um tanque de guerra, portanto, para virar o robô, os lados do chassi devem receber valores opostos.

Para o recebimento dos pacotes de dados foi implementado e adaptado um algoritmo de um servidor UDP, disponibilizado na internet por uma outra equipe participante (FRC TEAM 1736 ROBOT CASSEROLE, 2017), em linguagem Java, que foi executado no *roboRIO*, recebendo os dados e decodificando as informações da posição do centro do alvo, salvando-as em variáveis. Esta decodificação foi necessária pois os valores são recebidos pelo protocolo UDP como *string,* portanto foi utilizado o método *Character.getNumericValue(),* para converter cada caractere em número inteiro.

Foi implementado o comando *AutoTrackTarget,* responsável por manter o robô corretamente alinhado ao alvo de maneira autônoma. Este comando compara os dados da posição atual do centro do alvo no eixo x com o valor do centro da imagem. Caso o alvo esteja na esquerda ou na direta da imagem o comando executa a função *drive()* de modo que o robô vire para o lado que o alvo se encontra; ao detectar o alvo no centro da imagem o comando executa novamente o método *drive()* para que os motores sejam parados. Desta maneira,

independente da posição do alvo, o robô busca detectá-lo, alinhando-se a ele. O fluxograma deste algoritmo pode ser observado na Figura 38.

Figura 38 – Fluxograma do algoritmo de movimentação do robô



Fonte: Elaboração própria (2018).

Nesta etapa foram executados testes utilizando apenas imagem em tempo real.

4. RESULTADOS

Para validar o projeto desenvolvido, foram efetuados testes com imagens estáticas, vídeos e finalmente imagens em tempo real. Com o intuito de comprovar a robustez da aplicação, foram executados testes com imagens contendo outros objetos além do alvo desejado, a fim de testar a filtragem e detecção de alvo. Além disso foram usadas imagens e vídeos em ambientes com diferentes iluminações, como apresentado nas sessões anteriores. Estes testes apresentaram resultados positivos, pois, em todos, o alvo foi identificado corretamente sem apresentar falhas.

Por último, a aplicação foi executada no robô, utilizando imagens em tempo real. O algoritmo de detecção coleta as informações referentes ao centro do alvo e executa a movimentação do robô para mantê-lo alinhado ao alvo. O ambiente em que o teste foi executado pode ser observado na Figura 39.

Para avaliar o comportamento da movimentação do robô, o alvo foi movimentado de um lado para o outro da sala, como pode ser observado nas figuras 39 e 40.

Figura 39 – Ambiente de execução do teste

Fonte: Elaboração própria (2018).

Figura 40 – Alteração da posição do alvo durante o teste

Fonte: Elaboração própria (2018).

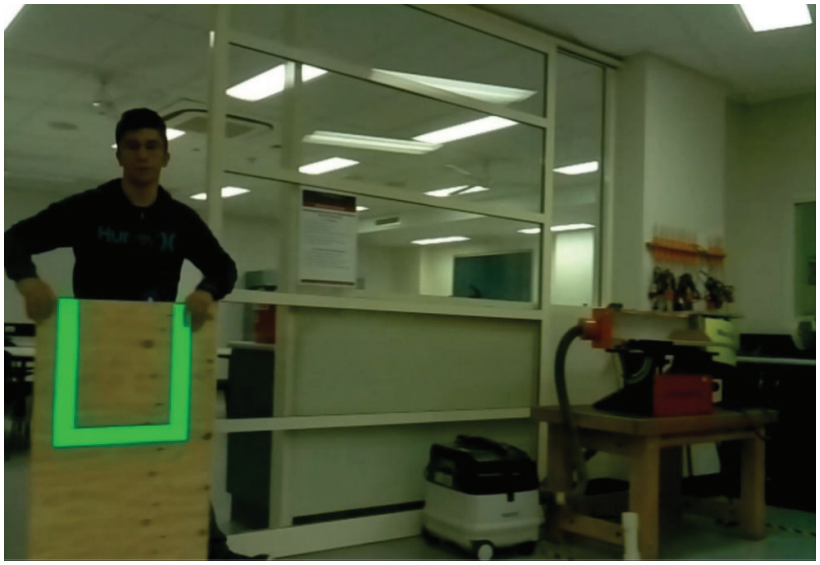
Alguns quadros deste teste foram coletados utilizando a câmera do robô e podem ser observados nas figuras 41, 42 e 43.

Na figura 41, o robô encontrava-se alinhado ao alvo, porém, após a movimentação do alvo para a esquerda (Figura 42), é identificado que o robô não se encontra mais alinhado ao alvo. Assim, é necessária a movimentação para a esquerda.

Figura 41 – Robô alinhado ao alvo

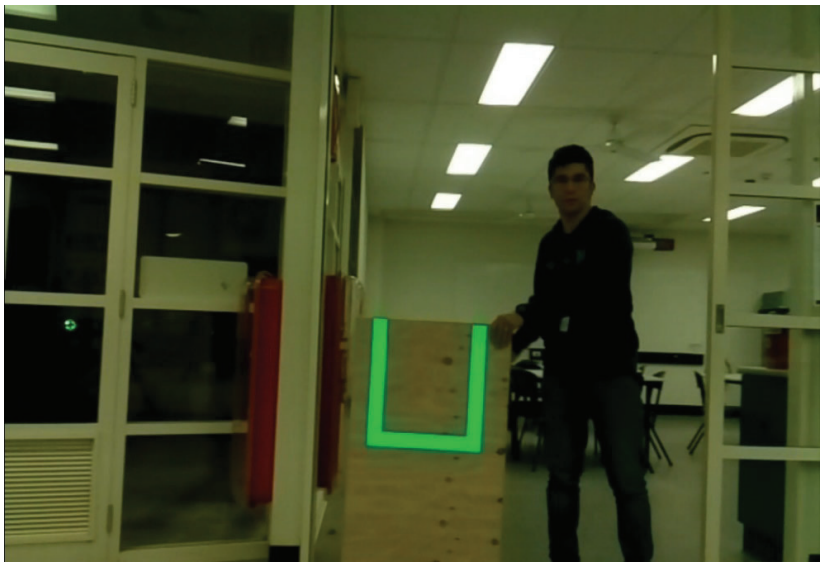
Fonte: Elaboração própria (2018).

Figura 42 – Alvo em movimento durante o teste



Fonte: Elaboração própria (2018).

Figura 43 – Robô realinhado ao alvo



Fonte: Elaboração própria (2018).

Este teste apresentou resultados satisfatórios, e com o seu correto funcionamento pode-se concluir que o uso da visão computacional auxilia no alinhamento do robô ao alvo. Porém, é evidente a necessidade do uso de um controlador digital mais elaborado para que a aplicação tenha uma resposta sem oscilação, que foi efeito detectado nos testes em tempo real.

5. CONCLUSÃO

O projeto tinha como objetivo utilizar visão computacional para auxiliar no alinhamento do robô com um alvo, isto posto, pode-se afirmar que o objetivo geral assim como os objetivos específicos do uso da biblioteca *OpenCV* em conjunto com o computador *Raspberry Pi* foram alcançados conforme os resultados apresentados neste trabalho.

Desta maneira, o sistema desenvolvido se mostrou robusto, sendo capaz de obter resultados satisfatórios ao ser utilizado em diversos ambientes com diferentes iluminações.

O programa desenvolvido em *Java* para ajustar a posição do robô proposto inicialmente obteve os resultados esperados, porém, este ajuste pode ser mais preciso.

Portanto, para trabalhos futuros é sugerido o uso de um controlador digital PID ou similar para auxiliar no controle da posição robô, permitindo com que o robô se alinhe de maneira precisa utilizando os sensores de posição e ângulo. Além do uso das informações referentes ao tamanho do alvo na imagem possibilitando obter uma estimativa do ângulo e distância que o robô se encontra do alvo.

REFERÊNCIAS

WILSON, J. S. **Sensor Technology Handbook**. Oxford: [s.n.], 2005.

BALLARD, D. H.; BROWN, C. M. **Computer Vision**. New Jersey: Prentice-Hall, 1982.

BARELLI,. **Introdução à Visão Computacional:** Uma abordagem prática com Python e OpenCV. [S.l.]: Casa do Código, 2018.

BRADSKI, G.; KAEHLER, A. **Learning OpenCV**. Sebastopol: [s.n.], 2008.

COULOURIS, G. et al. **Sistemas Distribuídos:** conceitos e projetos. 5. ed. Porto Alegre: BOOKMAN EDITORA LTDA, 2013.

DAVIES, E. R. **Computer and Machine Vision:** Theory, Algorithms, Praticalities. 4a. ed. Oxford: Elsevier, 2012.

FRADEN, J. **Handbook of Modern Sensors:** Physics, Designs and Applications. 3a. ed. New York: [s.n.], 2004.

FRADEN, J. **Handbook of Modern Sensors:** Physics, Designs, and Applications. 5. ed. San Diego: [s.n.], 2016. 758 p.

GONZALES, R. C.; WOODS, R. E. **Processamento Digital de Imagens, 3a edição**. São Paulo: Person Education do Brasil LTDA, 2009.

KUROSE, J.; ROSS, K. **Redes de computadores e a internet:** Uma abordagem top-down. 6. ed. São Paulo: [s.n.], 2013.

MOESLUND, T. B. **Introduction to Video and Image Processing:** Building Real Systems and Applications. New York: Springer - Verlag London Limited, 2012.

RAMSDEN, E. **Hall-Effect Sensors:** Theory and Application. 2a. ed. Oxford: [s.n.], 2006.

KALIN, N. HUFFPOST. **www.huffingtonpost.com**, 2015. Disponivel em: <https://www.huffingtonpost.com/natalie-kalin/5-reasons-teenagers-shoul_b_6981368.html>. Acesso em: 27 april 2018.

SZELISKI, R. **Computer Vision:** Algorithms and Applications. New York: Springer, 2010.

STALLINGS, W. **COMPUTER ORGANIZATION AND ARCHITECTURE:** Designing for Performance. 10a. ed. Hoboken: Pearson, 2016.

PATSKO, L. F. **TUTORIAL:** Aplicações, Funcionamento e Utilização de Sensores. [S.l.]: [s.n.], 2016.

SMITH, S. W. **The Scientist and Engineer's Guide to Digital Signal Processing**. 2a. ed. San Diego: [s.n.], 1999.

ANALOG DEVICES. HIGH PERFORMANCE, DIGITAL OUTPUT GYROSCOPE**.**

ANALOG DEVICES. NORWOOD, P. 28. 2013.

ANDYMARK. **AndyMark**, 2018. Disponivel em: <https://www.andymark.com/product-p/am-3555.htm>. Acesso em: 09 nov. 2018.

CTR ELECTRONICS, 2018. Disponivel em: <http://www.ctr-electronics.com/sensors/srx-magnetic-encoder.html#product_tabs_technical_resources>. Acesso em: 26 jul. 2018.

DXOMARK IMAGE LABS. dxomark, 2018. Disponivel em: <https://www.dxomark.com/glossary/color-depth/>. Acesso em: 06 set. 2018.

ENCODER PRODUCTS COMPANY. **What is an encoder?**, 2016. Disponivel em: <http://encoder.com/blog/company-news/what-is-an-encoder/>. Acesso em: 25 jul. 2018.

FIRST , 2018. Disponivel em: <https://wpilib.screenstepslive.com/s/currentCS/m/getting_started/l/599672-frc-control-system-hardware-overview>. Acesso em: 3 set. 2018.

FIRST ROBOTICS COMPETITION. 2018 SEASON FACTS. **First Inspires**, 2018. Disponivel em: <https://www.firstinspires.org/sites/default/files/uploads/resource_library/frc/game-and-season-info/competition-manual/2018/first-frc18-seasonfacts-fr023-v3.pdf>. Acesso em: 4 Junho 2018.

FRC TEAM 1736 ROBOT CASSEROLE. GitHub, 2017. Disponivel em: <https://github.com/RobotCasserole1736>. Acesso em: 01 jul. 2018.

GUPTA, V. Learn OpenCV, 2017. Disponivel em: <https://www.learnopencv.com/color-spaces-in-opencv-cpp-python/>. Acesso em: 26 set. 2018.

HEALTH, N. ZDNet, 2017. Disponivel em: <https://www.zdnet.com/article/what-is-the-raspberry-pi-3-everything-you-need-to-know-about-the-tiny-low-cost-computer/>. Acesso em: 23 ago. 2018.

KAMEN, D. First Inspires, 2014. Disponivel em: <https://www.firstinspires.org/about/vision-and-mission>. Acesso em: 26 set. 2018.

LANDAU,. Nasa. **nasa.gov**, 2017. ISSN v. Disponivel em: <https://www.nasa.gov/feature/jpl/nasa-mentored-teams-win-robotics-championship>. Acesso em: 01 ago. 2018.

MELCHIOR, A. et al. Techfire225, 2005. Disponivel em: <http://www.techfire225.com/uploads/6/3/7/1/6371896/first_study.pdf>. Acesso em: 10 maio 2018.

NATIONAL Instruments, 2018. Disponivel em: <http://www.ni.com/en-au/support/model.roborio.html>. Acesso em: 3 set. 2018.

OPENCV, 2018. Disponivel em: <https://opencv.org/>. Acesso em: 26 set. 2018.

RASPBERRY. **RASPBERRY PI FOUNDATION**, 2018. Disponivel em: <https://www.raspberrypi.org/products/camera-module-v2/>. Acesso em: 06 set. 2018.

RASPBERRY. Raspberry Pi Foundation, 2018. Disponivel em: <https://www.raspberrypi.org/about/>. Acesso em: 26 set. 2018.

SONY. **Sony Semiconductors Solutions Corporation**, 2018. Disponivel em: <https://www.sony-semicon.co.jp/products_en/new_pro/april_2014/imx219_e.html>. Acesso em: 06 set. 2018.