Universidade Federal do ABC

Engenharia de Instrumentação, Automação e Robótica

Renato Fernandes Rodrigues

VISÃO COMPUTACIONAL APLICADA A SISTEMAS EMBARCADOS

Santo André - SP

2016

Renato Fernandes Rodrigues

VISÃO COMPUTACIONAL APLICADA A SISTEMAS EMBARCADOS

Monografia de conclusão de curso apresentada à Universidade Federal do ABC para obtenção do título de Engenheiro.

Orientador: Prof. Dr. Luiz Antônio Celiberto Junior

Santo André - SP

2016

Resumo

A crescente demanda de tecnologias embarcadas nos mais diversos dispositivos e sistemas, proporciona uma maior difusão de conteúdo e materiais cada vez mais acessíveis para a prototipação de novas ideias e conceitos. Através da aplicação de algoritmos de visão computacional em um sistema baseado na placa Raspberry Pi 2, foi desenvolvido um sistema embarcado capaz de detectar de gestos feitos pela mão do usuário, os quais são interpretados e classificados em comandos para serem enviados para um robô móvel através de uma rede Wi-Fi. Os resultados obtidos foram satisfatórios para esta aplicação, o sistema executa o processamento da imagem e envio do comando a uma frequência de 4.8 imagens por segundo, com um total de nove comandos designados para cada gesto descrito, mostrando a sua aplicabilidade em outros sistemas de mesma natureza.

Palavras-chave: Sistemas Embarcados, Visão Computacional, OpenCV, Raspberry Pi.

Abstract

The growing demand for embedded technologies in the most diverse devices and systems, provides greater dissemination of content and materials increasingly available for prototyping new ideas and concepts. By applying computer vision algorithms in a system based on Raspberry Pi 2 board, it was developed an embedded system able to detect gestures made by the user's hand, which are interpreted and classified into commands to be sent to a mobile robot through a Wi-Fi network. The results were satisfactory for this application, the system performs the image processing and sending the command at a rate of 4.8 frames per second with a total of nine commands assigned to each gesture described, showing its applicability in other systems that have the same purpose.

Keywords: Embedded Systems, Computer Vision, OpenCV, Raspberry Pi.

Lista de Ilustrações

Lista de Tabelas

Lista de Abreviaturas

APT – *Advanced Packaging Tool*

CSI – Camera Serial Interface

DSI – Display Serial Interface

GPIO – *General Purpose Input/output*

HDMI – *High-Definition Multimedia Interface*

HSL – *Hue Saturation Lightness*

I²C – *Inter-Integrated Circuit*

LED – *Light Emitting Diode*

PIP – *Python Package Index*

SCP – *Secure Copy Protocol*

SD – *Secure Digital*

SPI – *Serial Peripheral Interface*

SSH – *Secure Shell*

UART – *Universal Asynchronous Receiver/Transmitter*

USB – *Universal Serial Bus*

Wi-Fi – *Wireless Fidelity*

IP – *Internet Protocol*

Sumário

1. Introdução

Presentes nos mais diversos setores, os sistemas dotados de visão computacional estão atualmente em crescimento e expansão. Nas suas áreas de aplicação, como a industrial, entretenimento, médica e de segurança, por exemplo, podem ser observadas as contribuições destas técnicas em seus respectivos procedimentos.

Essa ascensão deve-se primordialmente à redução dos custos de produção. Além disso, o aumento de desempenho, a otimização dos tamanhos dos hardwares de processamento e de fotografia, proporcionou - e ainda proporciona - o desenvolvimento de soluções cada vez melhores nestes campos (COELHO e DELAI, 2010).

Na interação entre o homem e a máquina podem ser observados desafios significantes na execução de tarefas com as mãos. Portanto, atividades como, o manuseio de um mouse e de um teclado - para visualização de objetos em três dimensões - poderiam ser otimizadas com o controle por imagem de gestos feitos com as mãos (KÄMMERER e MAGGIONI, 1998).

De acordo com Kämmerer e Maggioni (1998), “a mão humana é um dispositivo de entrada muito habilidoso capaz de manusear diversos graus de liberdade ao mesmo tempo”. O emprego da mão humana na realização de funções é algo natural ao usuário e capaz de trazer resultados melhores em práticas mais complexas.

No intuito de contribuir com o desenvolvimento de sistemas embarcados, este trabalho procura expor a aplicação de algoritmos com visão computacional. Nesta perspectiva, a meta é interpretar as informações contidas nas imagens capturadas para geração de comandos de controle.

1.1 Objetivos

Com o aumento da difusão e do acesso ao desenvolvimento de sistemas embarcados por meio de placas com capacidades computacionais cada vez melhores e de baixo custo, o objetivo central deste trabalho é, dentre as diversas formas de interação entre o homem e a máquina, produzir um sistema com capacidade de capturar comandos genéricos através da classificação

de gestos feitos com a mão e que esteja preparado para ser utilizados nas mais diversas aplicações, desde controlar maquinas até um simples tarefa de mouse.

Os objetivos específicos deste trabalho a serem alcançados são: projetar sistemas embarcados baseados em sistemas operacionais Linux; desenvolver programas na linguagem Python, no ambiente Linux e com as bibliotecas de visão computacional (OpenCV) para serem executados em uma placa Raspberry Pi 2; compreender os principais desafios envolvidos no processamento de imagem para desenvolver um sistema que seja capaz de ser executado tanto no computador, como na placa de desenvolvimento Raspberry Pi 2, atendendo as configurações desejadas.

2. Metodologia.

Neste capítulo estão apresentados os estudos e linhas de pesquisa que foram utilizados para a elaboração do protótipo e aplicação de algoritmos de visão computacional no sistema embarcado. Como o principal alvo deste estudo é a identificação de gesto feitos com as mãos, os métodos que serão apresentados a seguir são voltados para a binarização no espaço de cores e a segmentação. Para uma abordagem mais detalhada e ampla destes assuntos é recomendado as referências (UMBAUGH, 2005) e (OPENCV, 2015).

2.1 Sistemas Embarcados

Os sistemas embarcados possuem várias definições diferentes identificadas na literatura. Em geral, eles podem ser caraterizados como dispositivos desenvolvidos conjuntamente em hardware e software, voltados para a execução de funções dedicadas dentro de sistemas ou produtos maiores, através da constante interação com o meio em que está aplicado, utilizando sensores e atuadores integrados ao dispositivo. A confiabilidade, eficiência, especificidade, custos e a execução em tempo real são algumas das características essenciais de um sistema embarcado (SANTOS, 2006).

Estes sistemas usualmente estão presentes em diversos elementos cotidianos, como forno micro-ondas, televisores, celulares, automóveis, câmeras digitais, entre outros (BARROS e CAVALCANTE, [200-]).

2.2 Raspberry Pi

O Raspberry Pi é um computador de baixo custo e de tamanho reduzido, comparado as dimensões de um cartão de crédito, com diversas funcionalidades de um computador padrão (RASPBERRY PI FOUNDATION, [201-]).

Este dispositivo possui uma vantagem, que é a disponibilidade de pinos destinados como GPIO em sua própria placa, permitindo a integração com diferentes tipos de dispositivos externos. Dentre os GPIOs, alguns pinos possuem funcionalidades específicas que podem ser habilitadas via software, como os protocolos de baixo nível: UART, SPI e I²C (RASPBERRY PI FOUNDATION, [2015]).

Neste trabalho, a placa escolhida foi o Raspberry Pi 2, o qual foi lançado em 2015 no valor de 35 dólares, com capacidades substancialmente superiores aos modelos anteriores. As principais características desta placa apresentada por Raspberry Pi Foundation ([201-]) são:

- 4 portas USB.
- 40 pinos GPIO.
- Porta HDMI.
- Porta Ethernet.
- Saída de áudio e vídeo combinada em um conector de áudio de 3.5mm.
- Interface serial de câmera (CSI).
- Interface serial de display (DSI).
- Cartão micro SD.
- Processador gráfico VideoCore IV 3D.

O processador ARMv7 permite que o dispositivo opere com diversas distribuições ARM GNU-Linux e com o Windows 10. A placa possui interface gráfica, a qual contribui bastante para o desenvolvimento de projetos (RASPBERRY PI FOUNDATION, [201-]).

2.3 Biblioteca OpenCV

O OpenCV é uma biblioteca de visão computacional e aprendizado de máquina de código aberto, com uma comunidade de mais de 47 mil colaboradores e amplamente utilizado por grandes empresas, como Google, Intel, IBM e Microsoft, grupos de pesquisa e entidades governamentais.

A biblioteca possui mais de 2500 algoritmos otimizados os quais podem ser aplicados nas mais diferentes finalidades, como por exemplo, reconhecimento de objetos e cenários, detecção de movimentos e encontrar imagens parecidas em um banco de dados.

A documentação oficial do OpenCV possui descrições detalhadas de cada método, contendo exemplos e tutoriais o quais contribuíram para a compreensão da fundamentação teórica dos algoritmos e parâmetros utilizados no desenvolvimento do código (OPENCV, 2016).

2.4 Representação de imagem digital

As imagens digitais podem ser descritas como matrizes multidimensionais, nas quais possuem, na maioria dos casos, duas dimensões espaciais referente à posição de cada pixel na imagem e uma ou mais dimensões para representar as informações individuais de cores e intensidades de cada pixel.

De acordo com Umbaugh (2005), “as imagens binárias são a forma mais simples de representação, são constituídas por apenas dois valores para seus pixels, tipicamente denominado como preto e branco”. Este tipo de imagem é largamente utilizado nos processos que utilizam informações de formas e contornos.

As informações individuais dos pixels de uma imagem podem ser representadas também por níveis de intensidade, as quais são classificadas como escala de cinza, no caso em que a informação do pixel ser apenas um nível de intensidade, e como de cores, no caso em quem os pixels contém mais de um canal de níveis de intensidades.

As imagens coloridas são usualmente capturadas no modelo de cores RGB e transformadas matematicamente em outros espaços de cores que separam as informações de uma forma mais intuitiva, através dos níveis de brilho e cores.

O espaço de cores HSL permite que a imagem seja descrita de forma mais natural para se compreender as informações das cores, baseado na percepção humana. A componente de cor é representada pelo canal *Hue*, o nível de branco pelo canal *Saturation* e o seu brilho através do canal *Lightness* ou *Intensity* (UMBAUGH, 2005). A Figura 1 demonstra a representação do espaço de cores HSL.

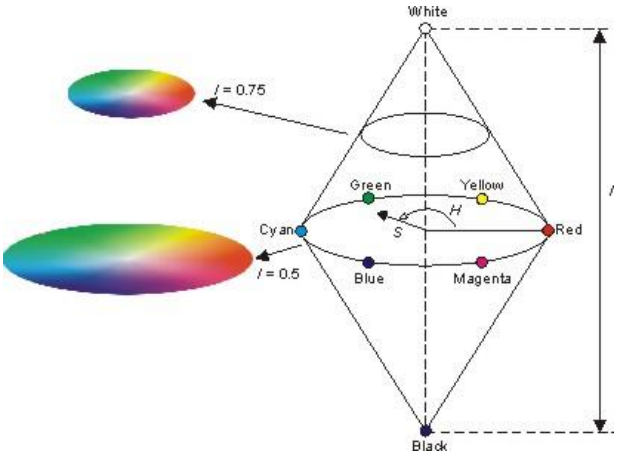

Figura 1: Representação do espaço de cores HSL (BLACKICE).

2.5 Segmentação

O processo de segmentação consiste em separar a imagem em regiões que contenham informações relevantes que se deseja avaliar. O método deve procurar as informações de contraste e homogeneidade entre as propriedades de cores dos pixels para definir regiões de interesse. Uma das formas mais simples de preparar a imagem para segmentação é através da binarização por histograma.

Os histogramas são representações da imagem de forma a descrever a quantidade de pixels de cada tipo de informação possível contida em cada um de seus canais do espaço de cores. Esta análise é invariante a translação, rotação e variações de escala (SHAPIRO e STOCKMAN, 2001). A figura 2 mostra o exemplo de um histograma da imagem convertida para escala de cinza.

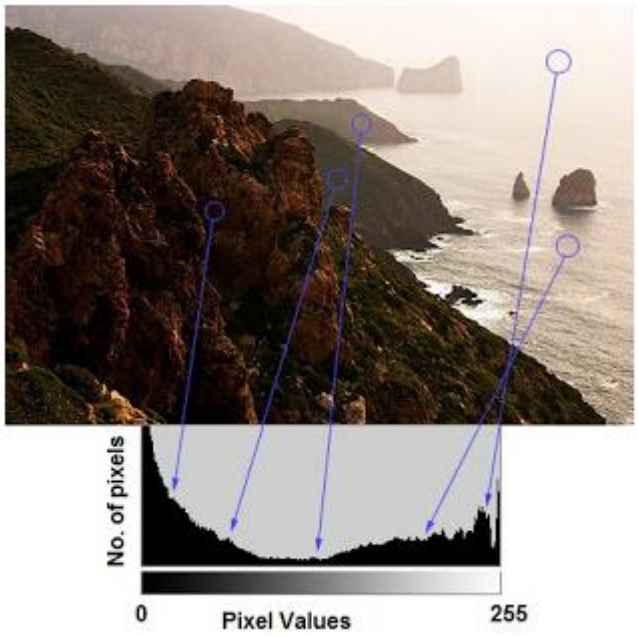

Figura 2: Exemplo de um histograma (OPENCV).

Uma simplificação dos métodos de binarização desenvolvido neste trabalho utiliza-se de uma pequena região no centro da imagem para verificar a cor da mão do usuário. Neste caso, pressupõe-se que esta pequena área contenha apenas as cores da mão em estudo, assim, através da busca no histograma pela informação de cor com maior número de ocorrências, é possível utilizar a cor encontrada para estabelecer limiares e gerar uma imagem binária.

A imagem binária muitas vezes precisa de certos ajustes para melhorar os resultados da segmentação. As transformações morfológicas são operações matemáticas que são aplicadas na imagem por meio de um elemento estrutural, uma imagem menor, com objetivo de suavizar as fronteiras entre o preto e branco, preencher pequenos buracos e remover pequenas projeções (UMBAUGH, 2005). Uma destas operações morfológicas que podemos citar é de fechamento, a qual está apresentada na Figura 3.

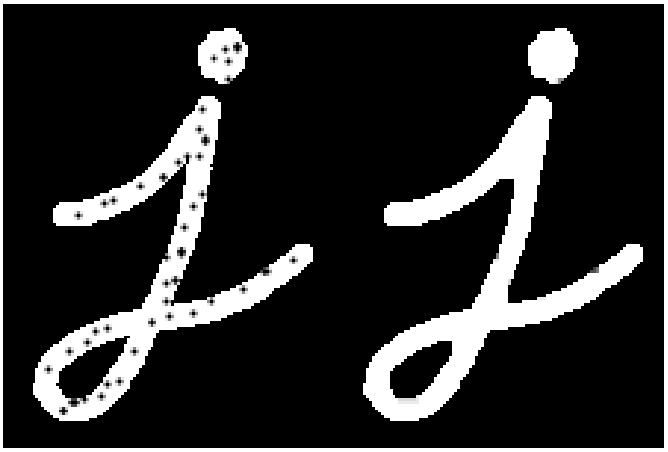

Figura 3: Representação de uma imagem binaria com ruído (esquerda) e após a operação de fechamento (direita) (OPENCV).

De acordo com OPENCV, “os contornos podem ser simplesmente expressos como uma curva que une todos os pontos contínuos ao longo de uma fronteira, mantendo as mesmas cores e intensidades”. Os métodos de segmentação de contornos retornam informações importante para a detecção e reconhecimento de objetos, como também para calcular as características de seus formatos.

3. Método Proposto para o desenvolvimento do sistema embarcado

Durante a fase inicial deste trabalho foi utilizado um computador com sistema operacional Ubuntu 14.04 LTS e uma webcam Logitech C920, para executar as etapas de instalação e configuração das bibliotecas necessárias, familiarizar com a linguagem Python, testar os principais algoritmos disponíveis na documentação do OpenCV e construir os primeiros programas de testes.

O computador foi utilizado posteriormente para auxiliar no desenvolvimento e testes dos códigos que posteriormente foram embarcados no Raspberry Pi, pois a utilização de editores de texto, como o Atom, proporcionam uma maior velocidade na construção dos códigos.

Na fase de construção do sistema embarcado, foram utilizados os seguintes componentes e equipamentos:

- Raspberry Pi 2.
- Cartão micro SD HCI 32GB, classe 10.
- Raspberry Pi Camera Rev. 1.3.
- Adaptador Wi-Fi USB.
- Fonte 5V 2A micro USB.
- Lapdock para Motorola Atrix MB860.
- Fita de LED.

Na Figura 4 é apresentado o diagrama de blocos do sistema embarcado. A fita de LED foi posicionada próxima da região da câmera para promover uma iluminação mais uniforme e constante em relação ao do ambiente em que o sistema poderá ser posicionado.

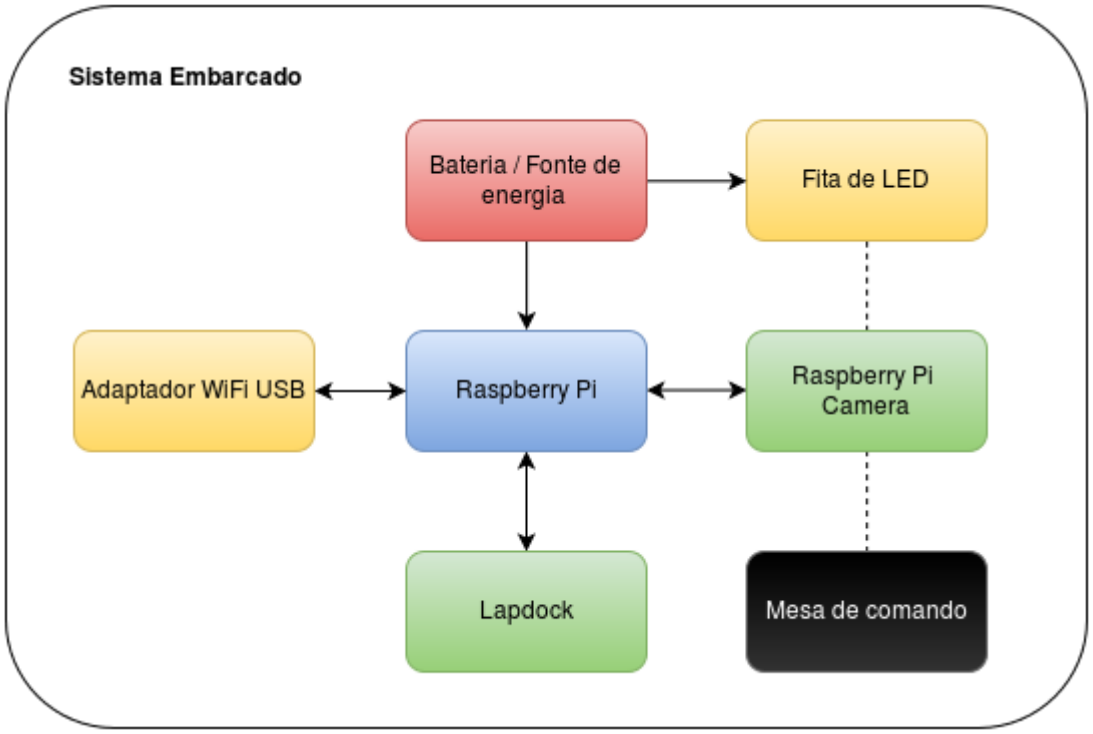

Figura 4: Diagrama de Blocos do sistema embarcado.

As estruturas de suporte e fixação utilizadas foram modeladas no Autodesk Fusion 360 na versão para estudante e fabricadas em uma impressora 3D Sethi3D AiP, com o material sendo PLA (Polylactic acid) branco.

A escolha no uso da Raspberry Pi Camera deve-se ao fato de ser conectada diretamente a GPU, proporcionando menor impacto direto na CPU, quando comparado com outras câmeras USB.

O Lapdock proporcionou uma maior flexibilidade para trabalhar nos testes com o Raspberry Pi, pois possui um display, teclado, touchpad e bateria, tudo integrado em um sistema compacto e portátil.

A mesa de comando tem como objetivo ser um anteparo que contraste com a cor da mão e mantenha a área de captura com uma cor mais uniforme. A câmera é posicionada a aproximadamente 70cm acima da mesa de comando.

Para auxiliar no funcionamento do código, uma pulseira preta ou uma camiseta de manga longa foram utilizados para auxiliar na separação da região do antebraço da região da mão.

O sistema de destino escolhido foi um robô móvel com duas rodas traseiras e uma roda dianteira do tipo castor, com conexão Wi-Fi através da placa ESP8266, a qual transmite via serial os

comandos enviados pelo sistema embarcado para uma placa Arduino Uno que esta interligada com uma placa de controle de motores baseada no circuito integrado L298D. Uma bateria de lítio Polímero de duas células foi utilizada para alimentar o robô. O diagrama de blocos do robô esta apresentado na Figura 5.

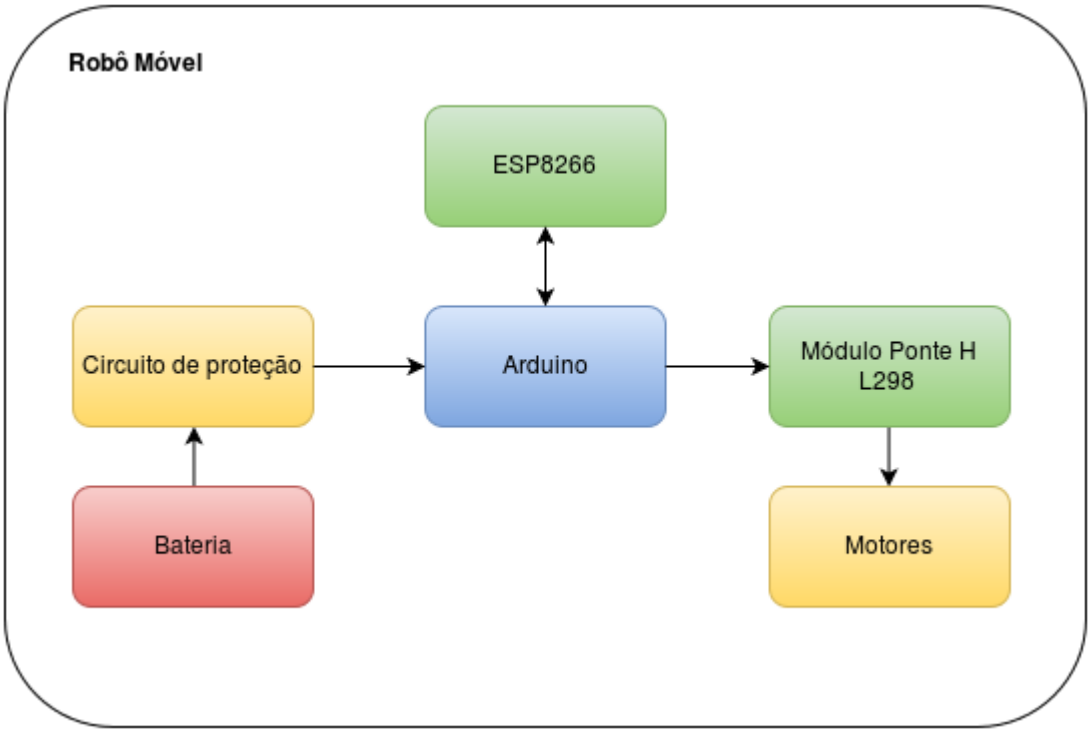

Figura 5: Diagrama de blocos do robô móvel.

Dentre as configurações iniciais para o Raspberry Pi, o sistema operacional escolhido foi o Raspbian, por ser o sistema oficial e com maior documentação e suporte para o desenvolvimento desta aplicação.

Com o sistema operacional instalado no Raspberry Pi, o passo seguinte é a instalação e configuração de bibliotecas e dependências necessárias. As principais bibliotecas utilizadas neste projeto são Python 2.7, OpenCV 3.0.0, Numpy, Matplotlib, Threading, Picamera, Math e Argparse.

Algumas bibliotecas já são nativas do Raspbian e basta atualizar para a última versão, mas as que não são nativas foram instalada via os gerenciadores de pacotes APT e PIP, seguindo as recomendações descritas na sua documentação oficial. As principais configurações utilizadas para instalar o OpenCV, as dependências e bibliotecas necessárias, tanto na Raspberry Pi como no computador, foram feitas de acordo com as recomendações descritas por Rosebrock (2015).

A comunicação entre o computador e o Raspberry Pi foi feita através do protocolo SSH, no qual é possível, no computador, executar o código no Raspberry Pi com interface gráfica. O código foi desenvolvido na maior parte do tempo no computador, através do editor de texto Atom, e sendo enviado para o Raspberry Pi através do protocolo SCP.

3.1 Funcionamento do sistema

Com o posicionamento, iluminação e a inicialização configuradas corretamente, o sistema estabelece a conexão socket com o robô móvel e inicializa o modulo da câmera. A cada iteração da função principal do código, a última imagem capturada pela câmera é chamada para a função principal, onde são feitas a conversão para o espaço de cor HSL, a binarização da imagem com base nos limiares de cores da mão que foi previamente salvo, a aplicação de operadores morfológicos para melhorar o resultado da binarização, a extração dos contornos e classificação do número de dedos abertos da mão.

Com o número de dedos classificados é calculado a moda dos 5 últimos valores detectados e é feita uma verificação para identificar se a mão não foi encontrada na imagem por mais de 2 cálculos da moda e assim, habilitar o modo de calibração.

No modo de calibração uma etapa a mais é habilitada no código, a qual calcula os histogramas de cada canal no espaço de cores HSL e redefine os limiares altos e baixos da cor da mão com base nos valores de maior incidência encontrados em cada canal do histograma. Caso em sequência o programa detecte uma moda referente a 5 dedos ao abrir toda a mão, os valores são salvos e atualizados no código e o modo de calibração é desativado.

Caso o modo de calibração estiver desabilitado e a conexão socket estiver estabelecida, os comandos são enviados para o robô móvel através de uma rede sem fio Wi-Fi por intermédio de um roteador wireless.

O robô móvel recebe os comandos, interpreta-os, retorna para o sistema embarcado uma mensagem que recebeu os dados com sucesso e determina a próxima movimentação do robô através da placa controle dos motores. Caso nenhum comando seja recebido por mais de 1 segundo, o robô desliga todos os motores e aguarda novos comandos.

O funcionamento detalhado será apresentado no tópico 3.2.

3.2 O código.

O código do sistema embarcado foi desenvolvido na linguagem Python, baseado na documentação do OpenCV (2015) e artigos de Rosebrock (2015), além de estar disponibilizado no GitHub (RODRIGUES, Renato, 2016). Algumas funções foram organizadas em bibliotecas e *flags* foram criadas para facilitar no processo de escolha do ambiente de execução e manter a organização do código. O fluxograma resumido do seu funcionamento está apresentado na Figura 6 e será descrito em sequência.

Figura 6: Fluxograma resumido de funcionamento do código.

3.2.1 A execução do código.

A execução do código é feita através de linhas de comando do terminal e possui como entrada 3 argumentos configuráveis: “--display 1”, para habilitar a *flag* de visualização gráfica, “flag_display”, das imagens processadas; “--raspberry 1”, para alternar entre o método de captura da imagem da câmera no Raspberry Pi e no computador através da *flag* “flag_raspberry”; e “--send 1”, para habilitar a transmissão de dados via socket para o sistema de destino através da *flag* “flag_send”. As *flags* são variáveis criadas no código e são consideradas habilitadas caso seu valor for maior que zero.

A configuração dos argumentos é feita logo após a importação das bibliotecas gerais do código, através dos métodos da biblioteca “arg”. Desta forma, é possível trabalhar no mesmo código em ambientes computacionais diferentes, como também facilitar na configuração do sistema através da linha de comando sem precisar alterar o código.

3.2.2 Biblioteca “Configs”

A biblioteca “Configs” é responsável por gerenciar o arquivo de configuração no formato JSON, o qual foi escolhido por ser leve e de fácil interpretação dos dados salvos. Esta é constituída de uma classe chamada “Configs” e os métodos: “__init__” para carregar todas as variáveis do arquivo de configuração, sendo o caminho do arquivo especificado no argumento; “set” para salvar o valor de uma variável; “get” para ler o valor salvo correspondente a variável; “save” para salvar todas as variáveis no arquivo; e “load” para carregar todas as variáveis do arquivo.

As variáveis dos limiares alto e baixo da binarização, da resolução da câmera e das configurações de IP do dispositivo de destino, estão armazenadas no arquivo de configurações.

3.2.3 Biblioteca “PiVideoStream”

A biblioteca PiVideoStream, elaborada por Rosebrock (2015), é responsável por gerenciar a captura das imagens da Raspberry Pi Camera em uma nova *thread*. É constituída pela classe “PiVideoStream”, com os métodos “__init__”, “start”, “update”, “read” e “stop”.

O método “__init__” inicializa e configura o modulo Raspberry Pi Camera e as variáveis utilizadas, o “start” inicializa a thread para capturar as imagens da câmera através do método “update”, o qual mantem atualizado uma variável com a última imagem capturada pela câmera e caso a *flag* “stopped” for verdadeira, desliga a thread e o módulo da câmera.

O método “stop” habilita a *flag* “stopped” com valor verdadeiro e o método “read” retorna a última imagem da câmera que foi atualizada.

3.2.4 Configurações Iniciais

Após o carregamento das bibliotecas, as variáveis globais são inicializadas e pré-configuradas. Em seguida, são definidas as funções utilizadas pela função principal.

A configuração da transmissão de dados é feita anteriormente apenas se a flag “flag_send” estiver habilitada. O endereço de IP e a porta do servidor do sistema de destino são carregados do arquivo de configurações e o método de conexão socket é iniciada. Caso ocorra um erro e não seja possível conectar ao sistema de destino, o código é finalizado.

A função “send_comand” envia a mensagem passada no parâmetro de entrada através da conexão socket previamente configurada, caso a *flag* “flag_send” estiver habilitada e a mensagem não for vazia. Após cada envio, deve ser recebido de volta uma mensagem de confirmação no formato “0!” para validação da transmissão dos dados.

As funções “calc_distance”, “calc_angles” e ”calc_meanpoint” tem como objetivo executar operações matemáticas nos pontos, pares ordenados, definidos como parâmetros de entrada da função. Estas retornam informações de distancias, ângulos e valores médios entre pontos.

As funções “start_timer” e “end_timer” são utilizadas para monitorar o tempo de execução entre duas partes do código. Primeiro, a função “start_timer” é executada para salvar o tempo inicial e ao executar a função “end_timer”, o intervalo entre a execução das duas funções é calculado e adicionado à lista da variável “timer_list”.

Para facilitar no processo de configuração e calibração do sistema, algumas trackbars foram inseridas na janela principal de visualização da imagem, caso a *flag* “flag_display” estiver habilitada. As trackbars são barras horizontais graduadas, configuradas através da função “cv2.createTrackbar()” e seu valor atual é lido através da função “cv2.getTrackbarPos()”. A

cada mudança na posição de seu cursor, a função “ update_trackbars()” é executada para fazer a leitura e atualizar os novos valores das variáveis “span_h”, “span_l” e “span_s”, e salvar no arquivo de configuração.

A última configuração inicial é a inicialização e configuração do modulo da câmera. No caso da *flag* “flag_raspberry” estar habilitada, a classe “PiVideoStream()” é inicializada, caso contrário, a classe cv2.VideoCapture() é inicializada com parâmetro 0, o qual corresponde ao primeiro dispositivo de câmera detectado pelo dispositivo. Um atraso de 2 segundos é executado para aquecer os sensores da câmera [x3].

3.2.5 Função principal

A função principal está contida dentro da instrução “try”, com objetivo de poder finalizar o programa ao ocorrer alguma exceção, como possíveis erros ou interrupção do teclado através do comando “Ctrl + c”. Esta função é executada até que uma exceção ou as teclas “q” ou “Esc” forem pressionadas, para que na sequencia o modulo da câmera seja desligado e, caso a *flag* “flag_display” estiver habilitada, fechar todas as janelas de visualização.

Em cada passo foi utilizado as funções “start_timer()” e “end_timer()” para calcular o tempo em milissegundos que este intervalo foi executado e, assim, poder fazer a avaliação de desempenho do método em que este fora aplicada, com exceção do cálculo do FPS, o qual é feito no início da função principal. Os valores calculados são atualizados na lista “timer_list” e, caso a tecla “t” do teclado for pressionado, esta lista é apresentada no terminal de comando.

O primeiro passo - após a inicialização das variáveis locais - é a leitura da imagem da câmera, a qual é feita através do método selecionado pela *flag* “flag_raspberry”. Uma região de interesse da imagem é definida na variável “frame”, mantendo apenas uma região central da imagem, com metade do comprimento e da largura original.

O segundo passo é a conversão do espaço de cores de uma base para outra. A imagem na variável “frame” está em RGB e, o espaço de cores escolhido para a conversão foi o HLS, armazenado na variável “frame_hsl”.

Caso a *flag* de calibração estiver habilitada, uma segunda região de interesse é definida, na variável “frame_roi”, sendo um quadrado no centro, com dimensões de 1/10 do comprimento da variável “frame_hsl”, no qual é calculado o histograma para um de seus canais e organizados

de forma decrescente em relação a quantidade de pixels para cada nível de cada canal. Os limiares baixo e alto de cor para a binarização, nas variáveis “hand_color_l” e “hand_color_h” respectivamente, são definidos através dos níveis de maior ocorrência do histograma, distanciando seus valores através das variáveis “span_h”, “span_l” e “span_s”, somando-as no limar alto e subtraindo-as no limiar baixo, e restringindo os valores entre 0 e 255.

O terceiro passo é a binarização do “frame_hls” dentro dos limiares de cor “hand_color_l” e “hand_color_h”, resultando em uma imagem binária armazenada na variável “mask0”. Nesta imagem binária ainda são necessárias efetuar algumas operações morfológicas para suavizar os contornos externos dos objetos e preencher pequenos buracos. Neste caso, com o fundo da imagem possui uma cor uniforme, foi apenas necessário executar uma operação de fechamento, através da aplicação do método “cv2.dilate()” e em seguida o “cv2.erode()”.

Com uma imagem binária bem definida, o método “cv2.findContours()” é aplicado com o parâmetro “cv2.RETR_EXTERNAL” para encontrar e armazenar os contornos mais externos na lista “cnts”. Nesta aplicação será utilizado apenas o contorno externo que possuir maior área.

O contorno mais externo é avaliado por diversos filtros baseados em características da mão humana, os quais utilizam-se de parâmetros como coordenadas, distancia, ângulos e área.

O filtro inicial é feito pela área do contorno e por meio de testes, foram identificados um valor mínimo e máximo de área para a primeira validação. Caso a área não estiver dentro dos limites, a variável “finger_num” é definida com o valor -1 e termina a análise dos contornos, retornando na variável “status_detector” uma mensagem com o valor da área encontrada.

Os momentos da imagem são calculados através do método “cv2.moments()” e utilizados para encontrar o centro de massa do contorno. As distancias entre cada ponto e o centro de massa são calculadas e armazenadas na variável “dist_center_c”. A distância média entre os pontos do contorno e o centro de massa é calculada e armazenada na variável “mean_dist_center”.

A menor envoltória convexa do contorno é calculada através do método “cv2.convexHull()” e então, o segundo filtro é aplicado. Caso a área do contorno for maior que 90% da área da envoltória é considerado que nenhum dedo foi encontrado, assim, a variável “finger_num” é definida com o valor 0, representando que a mão está fechada, e a avaliação dos contornos é finalizada.

A partir desta etapa da avaliação dos contornos, sabe-se que a área do contorno é válida e a mão não está fechada, assim, um ou mais dedos da mão devem estar abertos algumas cavidades são formadas entre a envoltória e o contorno. O método “cv2.convexityDefects()” foi utilizado para encontrar em cada cavidade os pontos de início e fim da cavidade, o comprimento e o ponto mais distante perpendicularmente a reta entre os pontos de início e fim da cavidade, os quais respectivamente foram armazenados nas variáveis “start”, “end”, “d” e “far”. Para complementar esta análise das cavidades, estes pontos encontrados e o ponto do centro de massa são passados como parâmetros para a função “calc_angle()”, na qual são calculadas as seguintes variáveis:

- “angle”: O angulo formado entre o ponto mais distante cavidade e os pontos de início e fim;
- “dist_start” e “dist_end”: As distâncias entre o ponto mais distante da cavidade e os pontos de início e de fim, respectivamente;
- “dist_start_center”, ”dist_end_center” e “dist_far_center”: As distancias entre o ponto do centro de massa e os pontos de início, fim e o mais distante da cavidade, respectivamente.

Todos estes pontos, ângulos e distâncias calculadas foram adicionados à lista “defects_list” para serem utilizados nos próximos filtros.

O terceiro filtro separa os pontos que são candidatos a representar um ou mais dedos da mão aberta. A primeira condição deste filtro é a variável “angle” estar entre 5º e 100º, na qual 100º corresponde ao máximo de abertura encontrada durante os testes entre o dedo indicador e o polegar. A segunda e a terceira condição diz respeito aos pontos “start” e “end”, os quais são testados individualmente se o seu comprimento em relação ao centro de massa é maior que a “mean_dist_center”, e se a relação entre seu comprimento até o ponto “far” e até o centro de massa estar contida entre 0.5 e 1.3, assim, caso de as três condições forem verdadeiras, o ponto em questão é adicionado à lista “finger_list”.

No caso de apenas um dedo estar aberto na mão, as condições do terceiro filtro não são atingidas na maioria dos casos testados. O quarto filtro verifica se a variável “finger_list” está vazia para definir a condição de um dedo aberto da mão, no qual é considerado o ponto do contorno mais distante do centro e a variável “finger_num” é definida com o valor 1.

Com dois ou mais dedos abertos da mão, a lista “finger_list” pode conter pontos muito próximos e até pontos coincidentes. O quinto e último filtro calcula as distancias entre todos os pontos e armazena na lista “finger_list_filtered” o ponto médio entre todos os pontos com distâncias menores que a variável “nearst_dist”, apenas se este ponto médio já não está na lista. A variável “finger_num” é definida com o valor do comprimento da lista “finger_list_filtered”.

Para todos os casos apresentados anteriormente, a variável “finger_num” é definida com um valor. O quarto passo é a verificação para habilitar a *flag* de calibração. A variável “finger_num” é adicionada à lista “command_buffer” e uma variável é utilizada como contador para a cada 20 iterações da função principal calcular a moda dos valores armazenados na lista “command_buffer” e armazena-la na variável “min_mode_command”. O uso da moda neste caso tem como objetivo filtrar possíveis variações do número de dedos encontrados, sempre arredondando para o menor número.

De acordo com o valor encontrado para a variável “min_mode_command”, algumas condições são estabelecidas:

- Caso a variável “min_mode_command” for menor que zero, ou seja, ocorreu na maioria dos casos um filtro por área, e a *flag* auxiliar “flag_calibration_timeout” estiver desabilitada, se esta condição for detectada três vezes consecutivas a *flag* “flag_calibration_timeout” é habilitada e o contador é zerado.
- Caso a *flag* “flag_calibration_timeout” estiver habilitada é habilitado a *flag* “calibration_hand_flag” e é armazenado os últimos limiares calculados da cor da mão em duas variáveis temporárias.
- Caso a variável “min_mode_command” for igual a 5 e *flag* “flag_calibration_timeout” estiver habilitada, a *flag* de calibração é desabilitada e os limiares da cor da mão são atualizados com os valores salvos nas variáveis temporárias.

Após a verificação da calibração, o quinto passo é a classificação dos comandos, definidos de acordo o número de dedos encontrados e suas respectivas posições:

- Caso “finger_num” for menor que 0, a variável “command” é definida como “0”
- Caso “finger_num” for igual a 0, a variável “command” é definida como “S”

- Caso “finger_num” for igual a 1, é calculado o menor angulo entre a horizontal e a reta do centro de massa até ponta do dedo. A variável “finger_angle_range” define um intervalo para as condições a seguir:
    - Se o angulo for maior que 90º somado com “finger_angle_range”, a variável “command” é definida como “FL”.
    - Se o angulo for menor que 90º subtraído com “finger_angle_range”, a variável “command” é definida como “FR”.
    - Caso contrário, a variável “command” é definida como “F0”.
- Caso “finger_num” for igual a 2, é calculado a média entre os dois pontos de dedos encontrados e é calculado o menor angulo entre a horizontal e a reta do centro de massa até a média dos pontos encontrada. As condições para definir a variável “command” são análogas ao caso de “finger_num” igual a um, apenas substituindo a inicial “F” por “B”.
- Caso “finger_num” for igual a 3 ou 4, a variável “command” é definida como “N”
- Caso “finger_num” for igual a 5 ou mais, a variável “command” é definida como “X”

Cada comando definido representa um comportamento para o outro sistema embarcado de destino. A função “send_command()” é responsável por enviar estes comandos para o robô móvel caso a *flag* “flag_send” estiver habilitada. A Tabela 1 apresenta os seguintes comportamentos atribuídos no robô móvel para cada comando.

Tabela 1: Lista de comandos e comportamento definidos no robô móvel.

Comando	Comportamento
“0”	Nada a ser executado - comando inválido
“S”	Reduz gradativamente a velocidade das rodas
“F0”	Andar para frente
“FL”	Andar para frente e para esquerda
“FR”	Andar para frente e para direita
“B0”	Andar para frente

“BL”	Andar para frente e para esquerda
“BR”	Andar para frente e para direita
“N”	Manter as velocidades
“X”	Parar imediatamente

Caso a *flag* “flag_display” estiver habilitada, o sexto passo é a apresentação da variável “frame”, com os pontos e valores relevantes encontrados nos passos anteriores, e da variável mask2.

Por fim, é feito a verificação das teclas pressionadas durante a função principal, responsáveis por habilitar e desabilitar diretamente algumas *flags* e finalizar a função principal, enviando o comando “X” para o robô móvel para que ele para imediatamente.

3.2.6 Finalização do código

Após a finalização da função principal, o módulo da câmera é desligado de acordo com a *flag* “flag_raspberry” para selecionar o método correto de desligamento e caso a *flag* “flag_display” estiver ativada, as janelas de visualização das imagens apresentadas são finalizadas.

4. Resultados e Discussões.

A mesa de comando tem como anteparo uma cartolina preta para contrastar com a mão que será testada. Os testes foram realizados em um ambiente interno, com iluminação de duas lâmpadas LED de 35W no teto e com a fita de LED posicionada na região da câmera voltada para a mesa. A Figura 7 apresenta o ambiente de teste utilizado e ilustra o posicionamento das duas câmeras, do computador, do Raspberry Pi 2, da fita de LED e da mesa de operações.

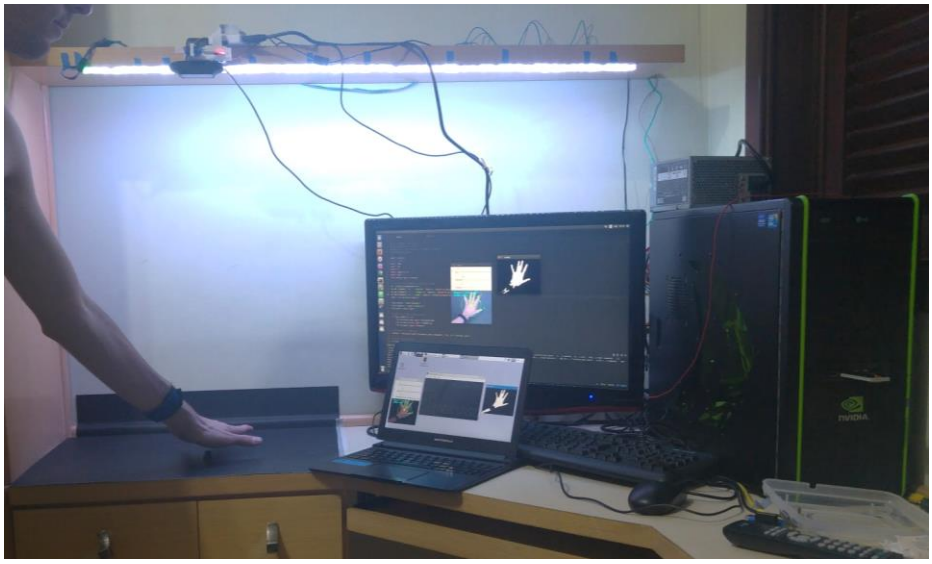

Figura 7: Ambiente de teste do sistema pelo computador e pelo Raspberry Pi.

O robô móvel utilizado como destino para os comandos do sistema embarcado está apresentado na Figura 8.

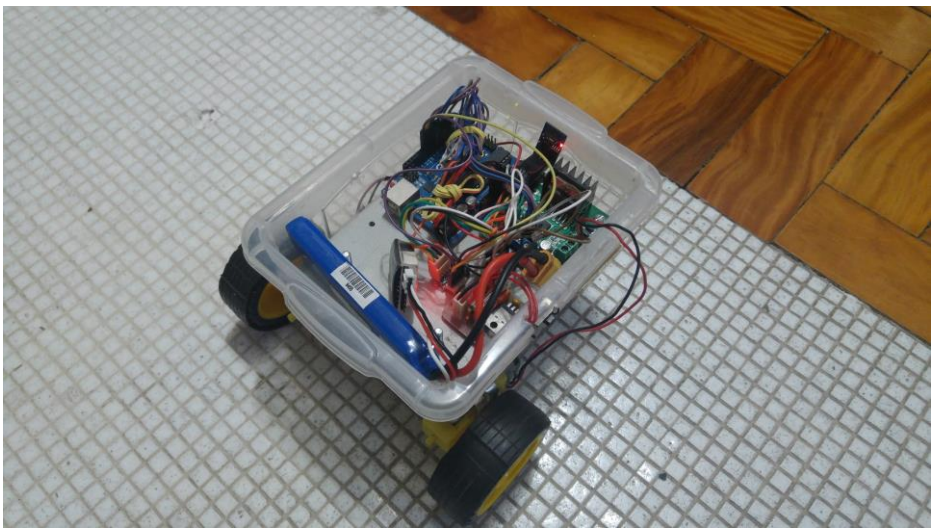

Figura 8: Robô móvel que recebe os comandos via Wi-Fi do sistema embarcado.

Os testes que serão apresentados a seguir foram executados separadamente, com as *flags* "flag_send" e "flag_display" habilitadas. Os tempos de execução de cada função do código no computador e no Raspberry Pi estão apresentados na Tabela 2. Os testes foram executados no mesmo ambiente e nas condições descritas anteriormente neste capítulo, seguindo uma mesma sequência de comandos e tempos de execução similares, em torno de 1 minuto. A taxa média de processamento no computador foi de aproximadamente 30,18 FPS e no Raspberry Pi foi de 4,27 FPS.

Tabela 2: Tempos de execução do código no computador e no Raspberry Pi.

Passos do código	Computador (us)	Raspberry Pi (us)	$\frac{Raspberry\ Pi}{Computador} - 1$ (%)
Captura da imagem	25322,42	9901,58	-60,90%
Conversão RGB para HSL	1513,44	35866,73	2269,88%
Binarização por limiares	171,62	9332,47	5337,87%
Operações morfológicas	389,53	69256,01	17679,38%
Classificação dos contornos	3799,22	26304,66	592,37%
Verificação da calibração	2,70	97,88	3525,19%
Definição do comando	7,15	143,09	1901,26%
Janelas de visualização	628,00	26443,62	4110,77%
Envio de comandos	15,68	2243,78	14209,82%
Tempo total da iteração	31,84	179,58	463,87%

Os tempos de execução no Raspberry Pi 2 foram superiores do que no computador, como esperado. As operações morfológicas foram responsáveis pelo passo com maior tempo de execução. O fato do tempo de captura da imagem no computador ser maior que no Raspberry Pi deve-se ao método read() da classe cv2.VideoCapture() manter o processo em espera até que uma nova imagem seja capturada e recebida, logo o tempo de execução do código é menor que o tempo entre cada captura da imagem.

Os comandos foram classificados pelo algoritmo com sucesso dentro das suas limitações de processamento e seus métodos. Foi observado que nesta versão do código com a taxa em torno de 5 FPS é possível controlar o robô com um leve atraso no tempo de resposta, atendendo as expectativas da aplicação do sistema.

As imagens obtidas na janela de visualização para cada comando serão apresentadas a seguir. O círculo amarelo representa o centro do contorno, a linha azul consiste na envoltória mais externa dos contornos, o círculo verde corresponde ao raio de "mean_dist_center" e quando são encontradas cavidades, o ponto "far" é representado na imagem com um número em vermelho correspondente a sequência das cavidades que foram encontradas pelo método. No caso em que o dedo encontrado for originado de uma cavidade, uma linha branca representa. A Figura 9 mostra o caso em que nenhum dedo da mão encontra-se aberto e a área do contorno está em 94% da área da envoltória, resultando assim no comando "S".

Figura 9: Imagem para o comando "S".

Os comandos de movimentação do robô móvel para frente estão apresentados nas Figuras 10 a 12. Nos comandos "F0" e "FL" nota-se que foi detectado uma cavidade válida e apenas a componente relacionada ao indicador foi classificada como um dedo válido, diferente do caso

do comando “FR”, em que nenhuma cavidade válida foi detectada e o dedo foi classificado pelo ponto do contorno mais distante do centro de massa. O dedo utilizado para estes comandos foi o indicador.

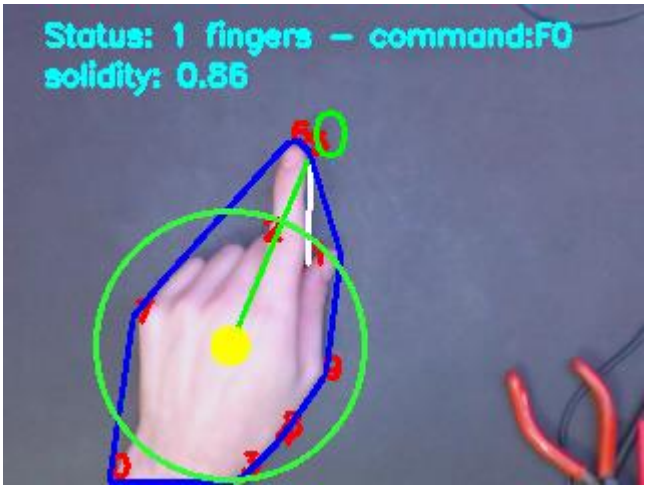

Figura 10: Imagem para o comando “F0”.

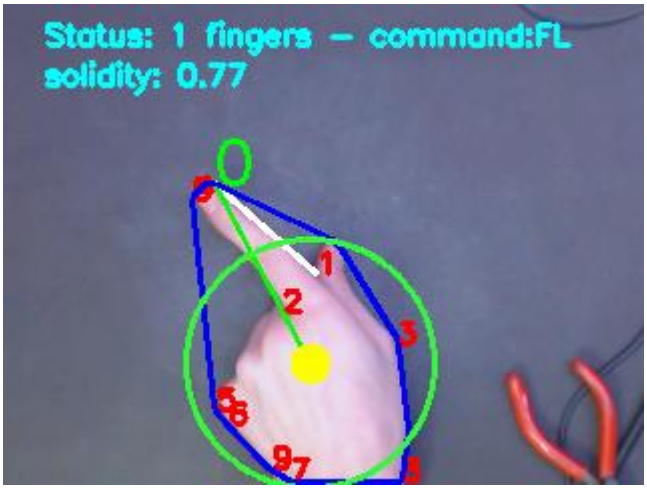

Figura 11: Imagem para o comando “FL”.

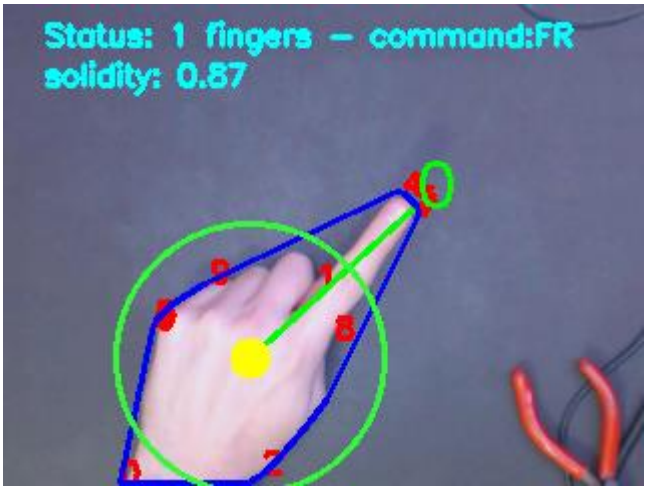

Figura 12: Imagem para o comando “FR”.

Os comandos de movimentação do robô móvel para trás estão apresentados nas Figuras 13 a 15. Para estes casos, ao menos uma cavidade válida foi encontrada e dois dedos foram classificados. Os dedos utilizados para estes comandos foram o indicador e médio.

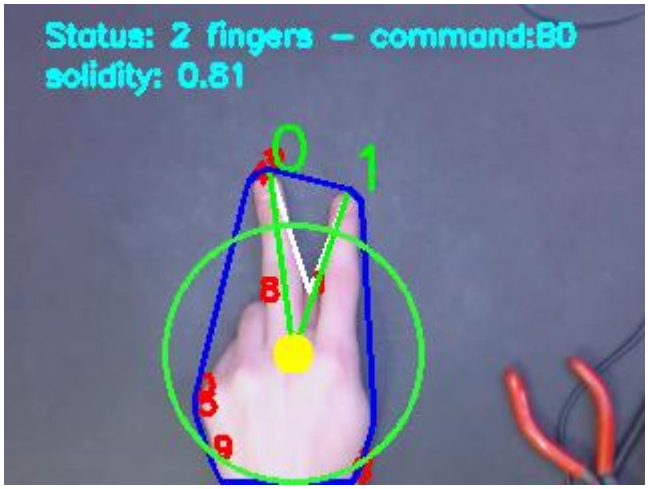

Figura 13: Imagem para o comando “B0”.

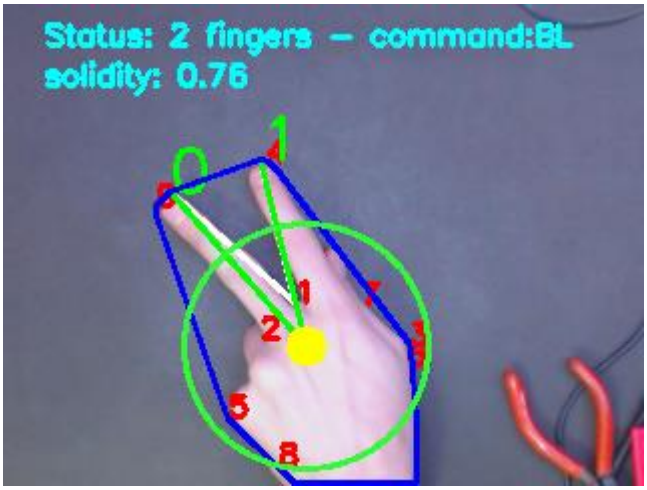

Figura 14: Imagem para o comando “BL”.

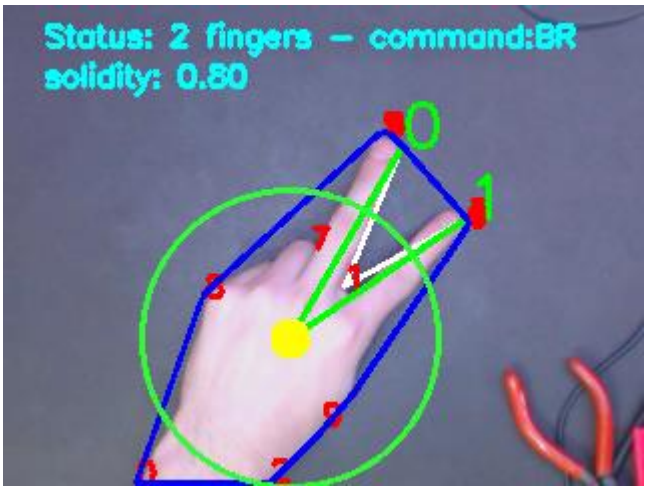

Figura 15: Imagem para o comando “BR”.

O comando “N”, apresentados nas Figuras 16 e Figura 17, é classificado no caso em que três dedos válidos foram detectados e tem como função manter o robô executando o último comando recebido diferente de “N”. Os dedos utilizados para este comando foram o indicador, médio, anelar e mínimo.

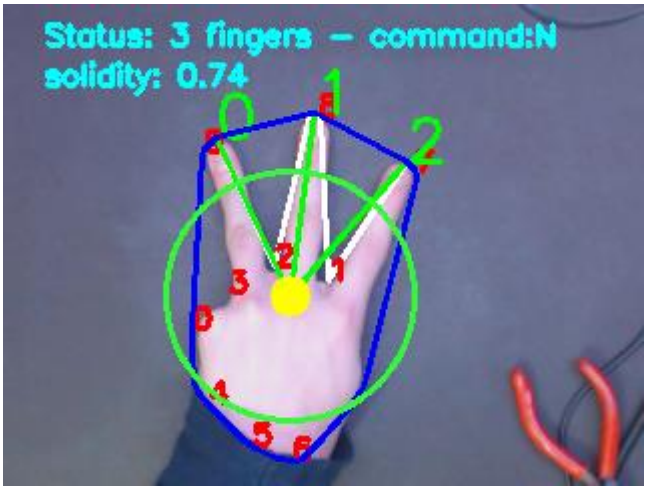

Figura 16: Imagem para o comando “N” no caso de 3 dedos.

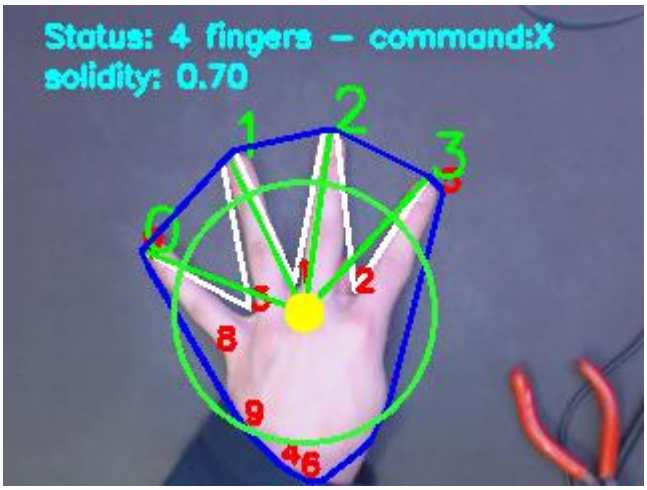

Figura 17: Imagem para o comando “N” no caso de 4 dedos.

Para o caso em que os cinco dedos forem detectados, apresentado na Figuras 18, o comando é classificado como “X” e o robô móvel deve parar imediatamente.

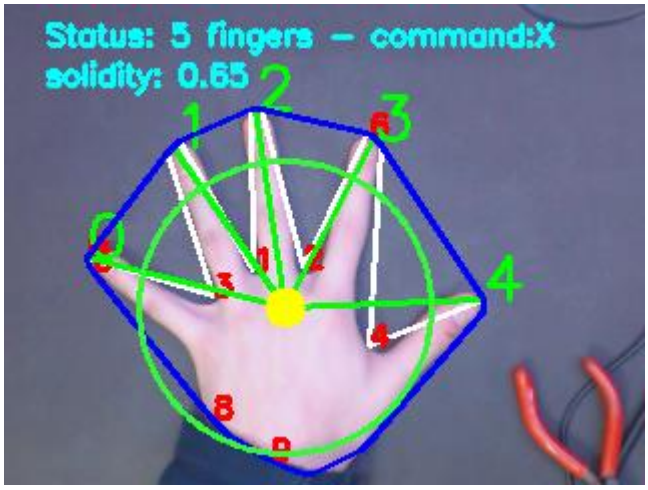

Figura 18: Imagem para o comando “X”.

No caso de outros gestos, o sistema pode ter interpretações diferentes decorrente do algoritmo que foi desenvolvido. As Figuras 19 a 23 mostram o resultado para outros tipos de gestos.

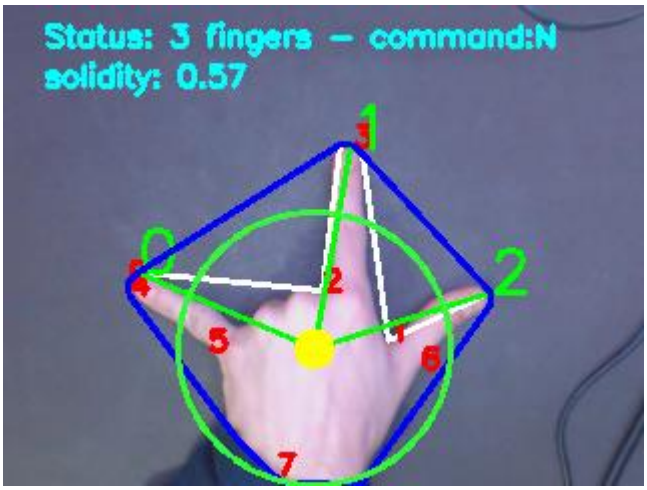

Figura 19: Imagem para um gesto classificado como “N”.

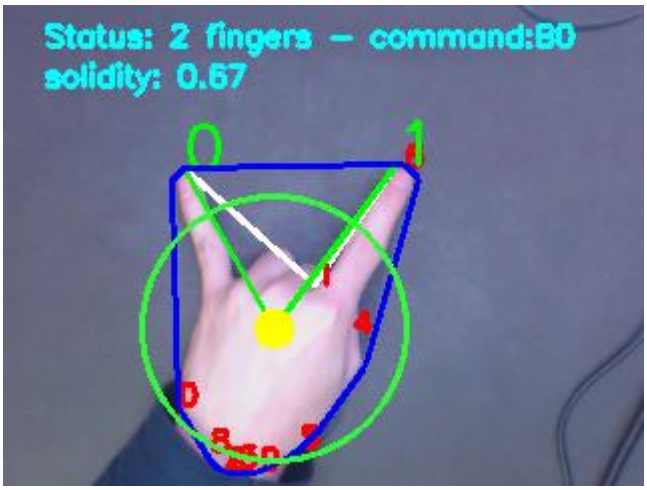

Figura 20: Imagem para um gesto classificado como “B0”.

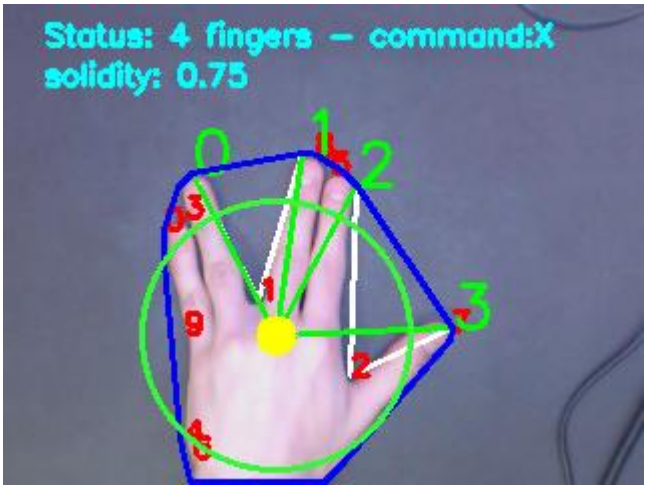

Figura 21: Imagem para um gesto classificado como “X”.

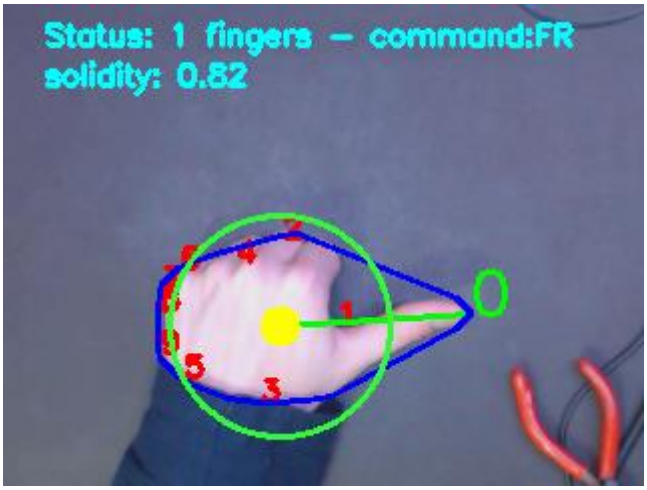

Figura 22: Imagem para um gesto classificado como “FR”.

A obtenção de uma binarização mais eficiente pode ser observada nas Figuras 23 e 24, onde a aplicação da fita de LED na região da mesa de comando proporciona uma iluminação mais uniforme que resulta em uma máscara mais uniforme e robusta a variações de posicionamento da mão e da iluminação local, como nos casos apresentados nas Figuras 25 e 26.

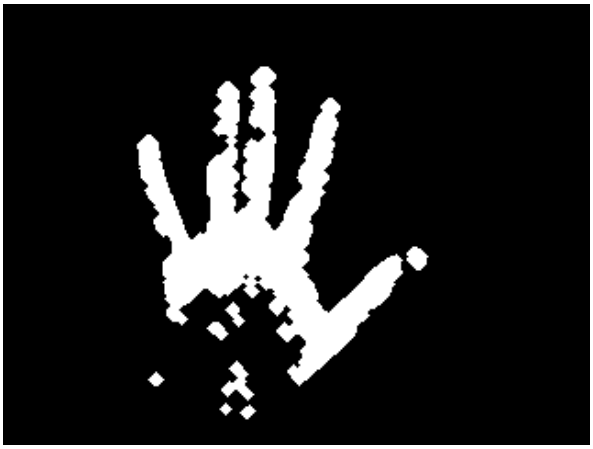

Figura 23:Resultado da binarização com a fita de LED desligada.

Figura 24:Resultado da binarização com a fita de LED ligada.

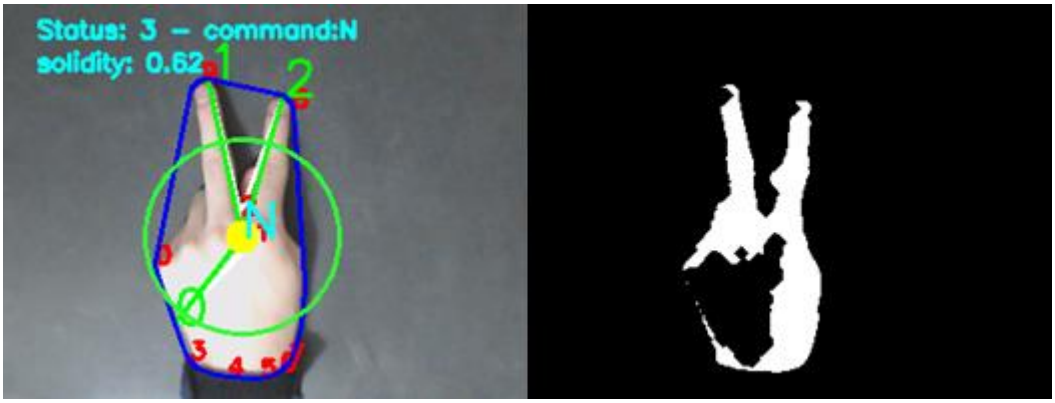

Figura 25: Resultado divergente para o comando “B0” com a fita de LED desligada.

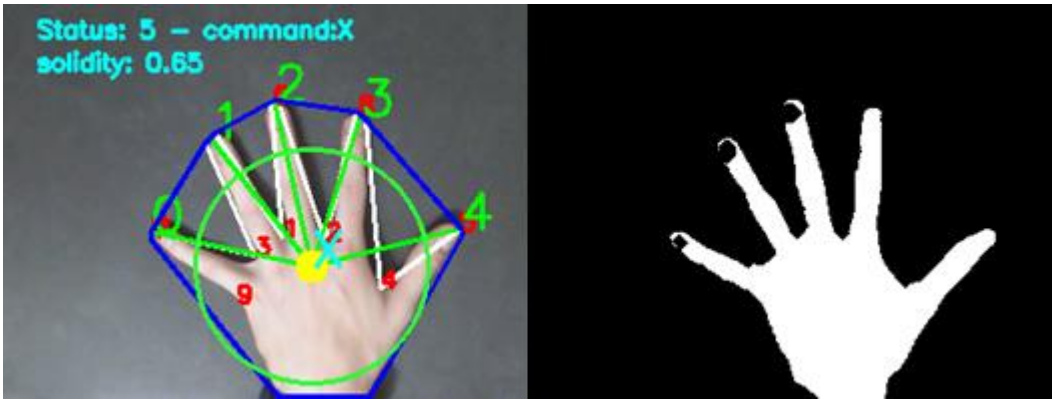

Figura 26: Resultado para o comando “X” com a fita de LED desligada.

Nos casos em que uma parte significativa do antebraço for capturada pela câmera e o usuário estiver sem a pulseira, o código pode classificar alguns comandos indesejados como o observado na Figura 27, onde as primeiras condições para encontrar a mão estavam corretas e nenhuma cavidade foi encontrada, assim este classificou o comando no caso de um dedo aberto da mão e o ponto mais distante do contorno em relação ao centro está na região do antebraço.

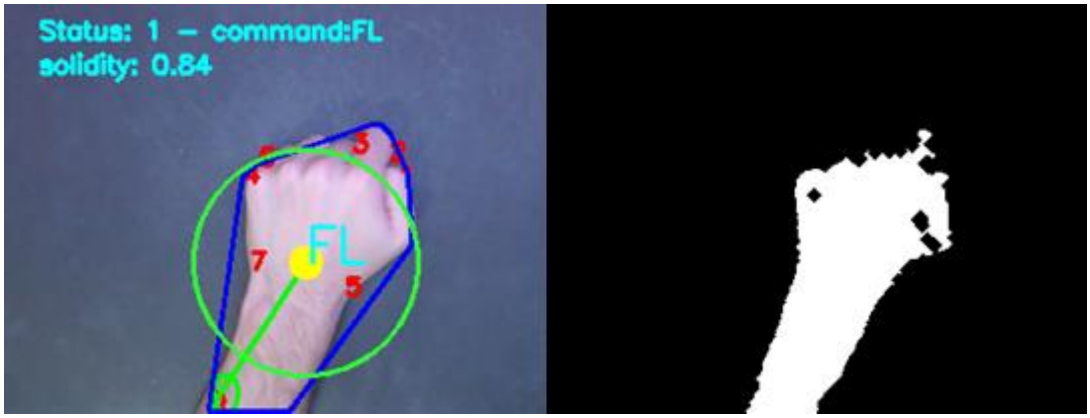

Figura 27: Resultado divergente para o comando "S" sem a utilização da pulseira

As limitações estabelecidas neste trabalho foram na diminuição do tamanho da imagem processada, através do estabelecimento de regiões de interesse, a divisão do antebraço da mão através de uma pulseira, o método de calibragem de cor e a classificação de comandos pelo número de dedos encontradas e informações sobre seu posicionamento na imagem. Estas condições estabelecidas foram necessárias para que fosse viável o desenvolvimento dentro das limitações deste sistema embarcado.

O método de calibração automática desenvolvido foi implementado com sucesso e otimizou o processo de calibragem, com mais flexibilidade na troca do usuário que utilizará o sistema e mantem o sistema sempre calibrado nas condições do instante que será utilizado.

As maiores dificuldades encontradas foram em relação a iluminação para a execução do processo de binarização, pois os resultados demonstraram que estas técnicas utilizadas são sensíveis as variações na iluminação e a não uniformidade das cores do objeto que será analisado. Quando necessário, o usuário pode ajustar os *trackbars* disponíveis na janela apresentada na tela de visualização para aumentar ou diminuir os intervalos de limiarização de cada canal da imagem em HSV.

5. Conclusões e Perspectivas

Este trabalho demonstrou a viabilidade do desenvolvimento de um sistema embarcado baseado na placa Raspberry Pi 2 que por meio de algoritmos de visão computacional fosse possível detectar e classificar alguns gestos feitos com a mão e controlar um robô móvel conectado à rede sem fio.

Os estudo e testes feitos ao longo deste trabalho evidenciaram os grandes desafios da visão computacional para obter os resultados esperados. A definição das etapas do processo de determinação dos limiares alto e baixo utilizados na binarização da imagem foi feita por meio de diversos testes em diferentes ambientes e condições de iluminação, o qual mostrou-se melhores resultados com o uso da fita de LED e um anteparo que contraste com a cor da mão testada.

A escolha dos comandos através da classificação pelo número de dedos atendeu aos objetivos atribuídas ao robô móvel. A transmissão dos comandos por meio da rede sem fio permite que o sistema embarcado interaja com outros sistemas na mesma rede ou até da internet, como por exemplo, gerar relatórios e notificações de uso do robô móvel.

As dificuldades encontradas em relação ao desempenho do sistema no Raspberry Pi 2 podem ser aprimoradas utilizando outras placas de desenvolvimento de mesmo propósito que possuam capacidade de processamento superiores, como a versão da placa Raspberry Pi 3 lançada este ano.

O sistema embarcado desenvolvido pode futuramente ser aprimorado através de um encapsulamento mais robusto para toda a solução e de fácil instalação no ambiente. A câmera poderia também ter um mecanismo de movimentação e executar outras funções para extrair mais informações do ambiente. Alguns LEDs de sinalização podem ser adicionados as GPIOs do Raspberry Pi para trazer de forma simplificada algumas informações da execução do código, como os comandos e *flags*.

Referências Bibliográficas

BARROS, Edna; CAVALCANTE, Sérgio. **Introdução aos Sistemas Embarcados**. [200-]. Disponível em: <http://www.cin.ufpe.br/~vba/periodos/8th/s.e/aulas/STP%20-%20Intro%20Sist%20Embarcados.pdf>. Acesso em: 16/10/2015.

BLACKICE. **HSI Color Space - Color Space Conversion**. Disponível em: <http://www.blackice.com/colorspaceHSI.htm>. Acesso em: 14/07/2016.

COELHO, Alessandra; DELAI, Ricardo. **VISÃO COMPUTACIONAL COM A OPENCV – MATERIAL APOSTILADO E VEÍCULO SEGUIDOR AUTÔNOMO**. 2010. Disponível em: <http://maua.br/files/082014/visao-computacional-opencv-material-apostilado-veiculo-seguidor-autonomo.pdf>. Acesso em: 14/06/2016.

ELINUX. **RPi Low-level peripherals**. 2015. Disponível em: <http://elinux.org/RPi_Low-level_peripherals>. Acesso em: 10/07/2016

KÄMMERER, Bernhard; MAGGIONI, Christoph. GestureCompuetr – History, Design and Applications. CIPOLLA, Roberto; PENTLAND, Alex. **Computer Vision for Human-Machine Interaction**. Reino Unido: Biddles Short Run Books, 1998. p.23-25.

OPENCV. **About**. 2016. Disponível em: <http://opencv.org/about.html>. Acesso em 10/07/2016.

OPENCV. **Image Processing in OpenCV**. 2016. Disponível em: <http://docs.opencv.org/master/d6/d00/tutorial_py_root.html>. Acesso em: 10/07/2016

RASPBERRY PI FOUNDATION. ***RASPBERRY PI 2 MODEL B***. [2015]. Disponível em: <https://www.raspberrypi.org/products/raspberry-pi-2-model-b/>. Acesso em :10/07/2016

RASPBERRY PI FOUNDATION. ***WHAT IS A RASPBERRY PI?***. [201-]. Disponível em: <https://www.raspberrypi.org/help/what-is-a-raspberry-pi/>. Acesso em :10/07/2016

RODRIGUES, Renato. **Raspi-hand-detector**. Disponível em: <https://github.com/rodriguesrenato>. Acesso em: 14/07/2016.

ROSEBROCK, Adrian. **How to Install Opencv 3 on Raspbian Jessie**. 2015. Disponível em: < http://www.pyimagesearch.com/2015/10/26/how-to-install-opencv-3-on-raspbian-jessie/>. Acesso em: 19/01/2016

ROSEBROCK, Adrian. **Increasing webcam FPS with Python and OpenCV**. 2015. Disponível em: <http://www.pyimagesearch.com/2015/12/21/increasing-webcam-fps-with-python-and-opencv>. Acesso em: 14/02/2016

SANTOS, Danilo Moura. **Projeto de sistemas embarcados: um estudo de caso baseado em microcontrolador e seguindo AOSD**. 2006. Disponível em: <http://www.lisha.ufsc.br/pub/Santos_BSC_2005.pdf>. Acesso em: 15/06/2016.

SHAPIRO, Linda; STOCKMAN, George. **Computer Vision**. New Jersey: Prentice-Hall. 2001.

UMBAUGH, Scott. **Computer Imaging**: Digital Image Analysis and Processing. 1. Ed. Florida; CRC Press. 2005.