

**UNIVERSIDADE DE SÃO PAULO**  
**ESCOLA DE ENGENHARIA DE SÃO CARLOS**  
**DEPTO. DE ENGENHARIA ELÉTRICA E DE**  
**COMPUTAÇÃO**

**SEL0630 - Aplicação de Microprocessadores II**

**Interpretador de Sinais aplicado à Língua**  
**Brasileira de Sinais - LIBRAS**

**Autor:** Kollins Gabriel Lima, nº. USP 9012931

**Autor:** Laís Paiva Faria Fortes, nº. USP 7960939

**Orientador:** Prof. Dr. Evandro Luis Linhari Rodrigues

São Carlos

2017



# Resumo

A Língua Brasileira de Sinais (LIBRAS) é a segunda língua oficial do Brasil. No entanto, seu uso fora da comunidade surda ainda é pouco difundido, o que acaba causando um isolamento desta comunidade dentro do próprio país. Na tentativa de promover maior integração entre falantes e não falantes da LIBRAS, muitos sistemas têm surgido para auxiliar no processo de tradução. Neste trabalho, foi proposto um sistema (baseado em Android) de reconhecimento gestual utilizando visão computacional que, a princípio, ficou limitado apenas ao reconhecimento dos símbolos numéricos de 0 a 7. O sistema apresentou baixas taxas de erro (cerca de 17%) na classificação de 2 símbolos, mas não se mostrou eficiente com um banco de dados contendo mais símbolos, chegando a taxas de erro de 60%, resultados esses que mostraram uma deficiência do sistema na extração de características das imagens.

Palavras-Chave: LIBRAS, Reconhecimento de Sinais, Visão Computacional.



# **Abstract**

The Brazilian Sign Language (also known as "LIBRAS") is the second official language of Brazil. However, only a few people outside the deaf community have the knowledge of this language, what causes an segregation of this community inside their own country. In order to integrate speakers and non-speakers of sign language, many systems has emerged to help in the translation process. In this project, it was proposed a system (Android based) of sign recognition using computer vision, limited to recognize numerical symbols from 0 to 7. The system operates with low error rates (around 17%) classifying 2 symbols, but it was inaccurate with a bigger database, with up to 60% of error rates, result that shows a deficiency in features extractions of the image.

Keywords: LIBRAS, Sign recognition, Computer Vision.



# Lista de Figuras

2.1	Processo de classificação em cascata para detecção de face. . . . .	20
3.1	Método utilizado. . . . .	24
3.2	Etapas do pré-processamento. . . . .	25
3.3	Resultado do pré-processamento. . . . .	25
3.4	Hash de 64 bits aplicado na imagem binarizada. . . . .	26
3.5	Página web para treinamento do banco de dados. . . . .	27
3.6	Diagrama simplificado do sistema. . . . .	28
4.2	Diagrama simplificado do sistema. . . . .	29
4.1	Símbolos propostos para o sistema de classificação. . . . .	30
4.3	Símbolos alternativos para o sistema de classificação. . . . .	32
4.4	Consumo de recursos pela Raspberry pi. Pode-se observar que a aplicação em python e o banco de dados MongoDB consomem praticamente 10 % dos 434MB de memória principal disponível e quase 95 % do poder de processamento. . . . .	33
4.5	Tamanho do aplicativo android instalado no <i>smartphone</i> . Além dos 167MB consumido pelo aplicativo, é preciso mais 44,43MB de armazenamento para a instalação do <i>OpenCV Manager</i> , necessário para o funcionamento do <i>OpenCV</i> no android. . . . .	34
4.6	Consumo de bateria pelo aplicativo. Percebe-se um consumo elevado de bateria, o que se justifica pelas operações de processamento de imagem que estão sendo realizadas na etapa de pré-processamento. . . . .	35
4.7	Consumo de memória principal e CPU do aplicativo. O consumo de mais de 10 % de CPU e 165MB de memória não afetaram o desempenho do aparelho, mas evidenciam um consumo grande quando comparado à outros aplicativos do sistema. (medição feita com aplicativo <i>Simple System Monitor</i> .) . . . . .	35





# Lista de Tabelas

2.1	Exemplo de características . . . . .	22
2.2	Ranking dos vizinhos mais próximos . . . . .	22



# Siglas

ARM	<i>Advanced RISC Machine</i> - Máquina RISC Avançada
ASL	<i>American Sign Language</i> - Língua de Sinais Americana
k-NN	<i>k-Nearest Neighbors</i> - k-Vizinhos mais próximos
LIBRAS	Língua Brasileira de Sinais
OpenCV	<i>Open Source Computer Vision Library</i> - Biblioteca de Visão Computacional de Código Aberto.
RISC	<i>Reduced Instruction Set Computer</i> - Computador com Conjunto de Instruções Reduzidas.
SQL	<i>Structured Query Language</i> - Linguagem de Consulta Estruturada.



# Sumário

<b>1</b>	<b>Introdução</b>	<b>15</b>
1.1	Objetivo . . . . .	16
1.2	Justificativa . . . . .	16
1.3	Organização do Trabalho . . . . .	16
<b>2</b>	<b>Fundamentação Teórica</b>	<b>19</b>
2.1	Segmentação . . . . .	19
2.1.1	Detecção de Pele . . . . .	19
2.1.2	Detecção de Borda . . . . .	20
2.1.3	Detecção de Face . . . . .	20
2.2	<i>Hash</i> . . . . .	21
2.3	Classificação . . . . .	21
2.3.1	Classificador k-NN . . . . .	21
<b>3</b>	<b>Materiais e Método</b>	<b>23</b>
3.1	Material . . . . .	23
3.2	Método . . . . .	24
3.2.1	Captura da Imagem . . . . .	24
3.2.2	Pré-Processamento . . . . .	24
3.2.3	<i>Hash</i> . . . . .	26
3.2.4	Classificação . . . . .	26
<b>4</b>	<b>Resultados e Discussões</b>	<b>29</b>
4.1	Desempenho . . . . .	33
<b>5</b>	<b>Conclusão</b>	<b>37</b>
	<b>Referências</b>	<b>38</b>



# Capítulo 1

## Introdução

A Língua Brasileira de Sinais (LIBRAS) é reconhecida oficialmente desde 2002 por meio da lei Nº 10.436 (regulamentada pelo decreto Nº 5.626 de 2005). Este marco representou um grande avanço para a comunidade surda, que passaram a possuir uma língua própria e direitos garantidos por lei, como expresso nos artigos 2º, 3º e 4º das obrigações do governo em divulgar a LIBRAS e garantir o atendimento e tratamento à comunidade surda [1].

Segundo o Portal Brasil, existem mais de 9,7 milhões de surdos no país (dados do Censo de 2010), no entanto, a acessibilidade para essas pessoas ainda é um desafio devido a falta de intérpretes [2]. O avanço da tecnologia tem contribuído para diminuir essa barreira de comunicação, por meio de dispositivos e aplicativos capazes de fazer traduções, mas os resultados atuais ainda não são capazes de oferecer independência linguística aos surdos.

É vasta a quantidade de estudos envolvendo o reconhecimento de gestos aplicadas à línguas de sinais. Na literatura, encontramos dois métodos principais para a leitura dos sinais: abordagem visual, baseada em visão computacional, ou por meio de dispositivos eletromecânicos, utilizando luvas, sensores, etc. Embora a segunda abordagem facilite o processo de aquisição do sinal, ela requer o uso de um hardware adicional que pode ser desconfortável e custoso. A primeira abordagem, entretanto, apesar de ser mais simples para uso no dia-a-dia, requer um maior pré-processamento para identificação e reconhecimento das mãos [3].

Também é vasta a coleção de aplicativos disponíveis para traduzir o português para LIBRAS, dois grandes exemplos a serem citados são o *Hand Talk* e o *ProDeaf*, duas soluções nacionais premiadas internacionalmente.

No entanto, sistemas capazes de fazer o processo inverso de tradução e que estão disponíveis à sociedade ainda são escassos. Em 2016, o projeto *SignAloud*, desenvolvido por Navid Azodi e Thomas Pryor, ambos estudantes da Universidade de Washington, ganharam o prêmio Lemelson,

fornecido pelo Instituto de Tecnologia de Massachusetts, pelo projeto de reconhecimento da língua ASL (*American Sign Language* ou Língua de Sinais Americana). O projeto é baseado em uma luva equipada com placas de arduino e acelerômetros para o reconhecimento de sinais e ainda não possui uma versão comercial.

Um projeto semelhante, apresentado em julho de 2017 por engenheiros da Universidade da Califórnia em San Diego, é o *The Language of Glove*, com a mesma proposta de reconhecimento de sinais ASL, mas utilizando um hardware de baixo custo (menos de 100 dólares, segundo os autores).

Aqui no Brasil, foi lançado em junho de 2017 o projeto “Giulia” para Android. Idealizado pelo Professor Manuel Cardoso da UFAM (Universidade Federal do Amazonas), o projeto utiliza os próprios sensores do *smartphone*, que ficam presos ao braço do usuário, para fazer o reconhecimento dos sinais (após o treinamento dos movimentos), eliminando a necessidade de hardwares adicionais.

A grande motivação deste trabalho é atuar de forma a fornecer ferramentas para que um usuário de LIBRAS possa interagir com pessoas que não tem conhecimento da língua de sinais. A proposta inicial é utilizar o método de visão computacional para a identificação de gestos estáticos, focando no reconhecimento dos números de 0 a 7.

## 1.1 Objetivo

Este trabalho tem como objetivo principal o desenvolvimento de um aplicativo para interpretação de sinais em LIBRAS (números de 0 a 7), fornecendo uma resposta textual ao sinal identificado.

## 1.2 Justificativa

Este projeto se justifica por atuar de forma a buscar alternativas à falta de intérpretes disponíveis aos usuários de LIBRAS, não com o intuito de substituí-los, mas na tentativa de aumentar a autonomia dos surdos em duas tarefas diárias, e agregar valor tecnológico e inclusivo ao Brasil.

## 1.3 Organização do Trabalho

Este trabalho está distribuído em 5 capítulos, incluindo esta introdução, dispostos conforme a descrição que segue:

Capítulo 2: Contém uma base teórica dos principais conceitos envolvidos no desenvolvimento do projeto.

Capítulo 3: Discorre sobre os materiais e método envolvidos.

Capítulo 4: Apresenta os resultados obtidos, junto à discussões à respeito deles.



Capítulo 5: Finaliza o trabalho concluindo à respeito dos resultados obtidos e do sistema como um todo.



## Capítulo 2

# Fundamentação Teórica

### 2.1 Segmentação

Ao se capturar uma imagem para processamento, é muitas vezes conveniente separar regiões de interesse, eliminando elementos alheios. Para este projeto, estamos interessados em obter uma imagem das mãos com o maior nível de detalhe possível.

Na tentativa de isolar as mãos em uma imagem foram adotadas 3 técnicas:

- Detecção de pele
- Detecção de bordas
- Detecção de face

#### 2.1.1 Detecção de Pele

Uma técnica utilizada para detecção de pele é a baseada em cores, ou seja, é considerada pele todos os pixels que estão dentro de uma faixa de cores. Matematicamente, seja a imagem  $F(x,y)$  e dois limitantes de cor  $\tau_1$  e  $\tau_2$ , a detecção de pele é feita conforme a equação 2.1.

$$F'(x,y) = \begin{cases} F(x,y), & \text{se } \tau_1 > F(x,y) > \tau_2 \\ 0, & \text{caso contrário} \end{cases} \quad (2.1)$$

Nos trabalhos consultados, não houve unanimidade quanto à melhor opção de espaço de cores para detecção de pele. Em [4], é utilizado o espaço de cores YCbCr, o mesmo utilizado por [5]. Em [6] é utilizado o espaço de cores YUV enquanto [7] faz o reconhecimento de pele no espaço RGB.

Experimentalmente, o melhor resultado foi obtido com o uso do espaço de cor HSV, como feito por [8], que também aponta as limitações do uso da técnica baseada em cores devido a grande variação

de tonalidade de peles.

### 2.1.2 Detecção de Borda

A detecção de borda realça os limites das regiões, baseando-se na variação de luminosidade [9]. Este processo se mostra necessário pois realiza a binarização da imagem, evidenciando o elemento de interesse (mãos) e diminuindo a quantidade de características presentes na imagem, o que facilita a classificação posteriormente [9].

Existe diversos algoritmos que fazem a detecção de borda, tais como operador Roberts, Sobel, Prewitt, etc. Segundo [10], as etapas básicas para fazer a detecção de borda são:

1. Suavização dos ruídos: nesta etapa, busca-se eliminar a maior quantidade de ruído possível, tomando o cuidado para não destruir as bordas.
2. Ressaltar Bordas: aplica-se um filtro que realça posição das bordas.
3. Localizar Bordas: decidir, entre as regiões destacadas, onde existe uma borda efetivamente (eliminação de falsos-positivos).

### 2.1.3 Detecção de Face

A detecção de face é uma técnica que visa identificar a existência ou não de uma face humana na imagem. Difere do reconhecimento de face, já que não se deseja identificar a pessoa, mas apenas identificar que existe uma pessoa (face).

Uma vez que se deseja isolar as mãos, é preciso fazer a identificação e a remoção da face, já que a detecção de pele destaca estas duas regiões [5].

Para fazer a detecção de faces foi utilizada a classificação em cascata fornecida pela biblioteca *OpenCV*. Este método recebe um arquivo de características pré-treinado e faz a classificação da imagem com base em diversas características. A face só é detectada se for aprovada ao longo da cascata de classificação, conforme mostra a figura 2.1.

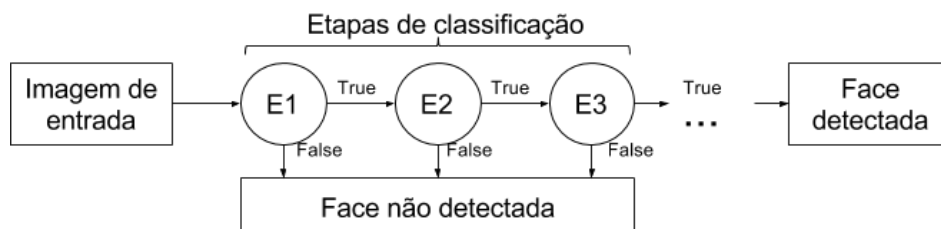


Figura 2.1: Processo de classificação em cascata para detecção de face.

Estas 3 etapas mostradas constituem uma fase de pré-processamento da imagem, com o intuito de identificar e isolar as mãos do ambiente.

## 2.2 Hash

Uma função *hash* é uma função unidirecional que mapeia dados de entrada em uma faixa limitada de valores. No presente trabalho é utilizada a função *hash* para mapear uma imagem em um valor numérico de 256-bits.

Para fazer a transformação, é utilizado um algoritmo de *Average Hash* seu funcionamento é descrito a seguir [11], para a criação de um *hash* de 64-bits:

1. A imagem de entrada é reduzida à 64 pixels (8x8) de modo a eliminar detalhes e componentes de altas frequências.
2. Conversão para escala de cinza.
3. Cálculo do valor médio (nível de cinza médio).
4. A imagem é binarizada seguindo o critério: se o pixel estiver acima da média ele é setado (recebe 255), do contrário recebe o valor 0.
5. A leitura da imagem (da esquerda para direita, de cima para baixo) gera um código binário, que representa o *hash* desejado.

## 2.3 Classificação

Para fazer o reconhecimento das imagens é preciso fazer uso de um sistema de classificação. Este sistema faz a identificação da imagem com base em um conjunto de treinamento previamente fornecido.

Vários são os métodos para fazer a classificação. Em [4] é utilizado máquina de vetores de suporte ou SVM (*Support Vector Machine*); classificador de Bayes é utilizado em [6] e em [7] utiliza-se Modelo oculto de Markov. Para simplificar o desenvolvimento, no entanto, foi adotado um classificador simples para este projeto, o k-NN.

### 2.3.1 Classificador k-NN

O classificador k-NN (*k-Nearest Neighbors*) verifica a distância do elemento de entrada para cada elemento no conjunto de treinamento e faz a classificação baseado nos k elementos mais próximos. A medida de distância deve ser definida com base nos elementos a serem classificados.

O k-NN suporta naturalmente uma aprendizagem incremental e utiliza uma abordagem *lazy*, ou seja, o algoritmo de aprendizagem retém informações de treinamento e apenas constroi um modelo ao receber uma entrada a ser classificada [12].

Para entender o processo de classificação, considere a tabela 2.1 que relaciona uma entrada numérica à uma cor. Suponha que ela represente um conjunto de treinamento.

Tabela 2.1: Exemplo de características

Número	Cor
10	Azul
21	Vermelho
12	Azul
20	Vermelho

Se uma entrada de valor 17 for aplicada neste classificador, pode-se utilizar como medida de distância o módulo da diferença entre os dados do conjunto de treinamento e a entrada.

$$d1 = \sqrt{(17 - 10)^2} = 7$$

$$d2 = \sqrt{(17 - 21)^2} = 4$$

$$d3 = \sqrt{(17 - 12)^2} = 5$$

$$d4 = \sqrt{(17 - 20)^2} = 3$$

Para  $k=3$  (considerando 3 vizinhos mais próximos), devemos considerar apenas as 3 menores distâncias ( $d4, d2$  e  $d3$ ). Atribui-se uma pontuação à cada um dos vizinhos, que deve ser maior quanto menor for a distância, conforme mostra a tabela 2.2.

Tabela 2.2: Ranking dos vizinhos mais próximos

Pontuação (1/d)	Cor
1/3	Vermelho
1/4	Vermelho
1/5	Azul

Por fim, verifica-se qual elemento (cor) possui a maior pontuação. Neste caso, a cor vermelha é a cor mais bem pontuada considerando os 3 vizinhos e portanto o classificador escolhe esta cor como resultado.

## Capítulo 3

# Materiais e Método

### 3.1 Material

O projeto principal consiste em um aplicativo Android conectado a um sistema embarcado em uma Raspberry Pi (com Raspian) em rede e com comunicação *web* para um *website* para treinamento dos símbolos. Ou seja, generalizando a divisão do projeto, tem-se os seguintes materiais envolvidos:

- Android Studio (3.0.1): para a criação do aplicativo, o qual processa as imagens da câmera do *smartphone* para transferi-las à Raspberry pi.
- Raspberry Pi (modelo B): microcomputador embarcado, com uma distribuição Linux e programado para as necessidades do projeto.
- Raspian (4.9.35): distribuição variante do Debian do Linux baseada na arquitetura ARM do hardware da Raspberry Pi.
- Python 3: linguagem de programação interpretada de alto nível utilizada durante toda execução do projeto, tanto para rotinas internas de classificação, quanto para criação do *website*.
- Flask (0.12.2): *framework* leve e poderoso para Python, utilizada na criação do sistema *web*.
- OpenCV (3.3): significando *Open Source Computer Vision Library*, é uma biblioteca multiplataforma utilizada no aplicativo Android afim de processar as imagens obtidas na câmera.
- Materilize (v0.100.2): *framework* para customização da interface *web*.
- MongoDB (2.4.10): modelo de banco de dados *NoSQL*, o que facilita sua criação, servindo como armazenamento das imagens dos símbolos para posteriores comparações e classificações.

## 3.2 Método

O método utilizado para realizar este trabalho baseia-se no que foi utilizado em [4], [5] e [7]. Basicamente o processo se divide conforme mostra a figura 3.1.

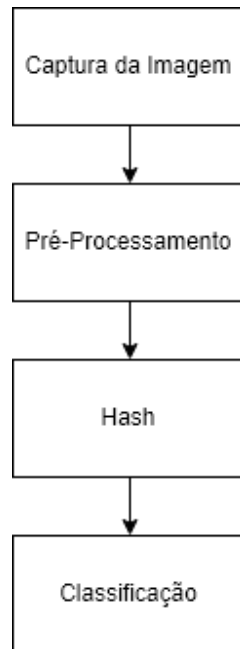


Figura 3.1: Método utilizado.

### 3.2.1 Captura da Imagem

A captura da imagem é feita pela câmera do aparelho Android. Não foi imposta nenhuma restrição quanto às características da câmera para este projeto.

### 3.2.2 Pré-Processamento

O pré-processamento é realizado com auxílio da biblioteca *OpenCV* e consiste em tratar a imagem capturada pela câmera de modo isolar as características desejadas (neste caso, busca-se isolar a região das mãos).

O processo de pré-processamento é mostrado na figura 3.2. São realizadas 3 etapas de maneira independente:

- Detecção de bordas: foi utilizado o detector de bordas de *Canny*, por meio da função *Imgproc.Canny*.
- Detecção de face: utiliza o *CascadeClassifier* da biblioteca *OpenCV* para fazer a detecção da região quadrada que contém uma face. Esta região é então preenchida (preto) de modo a eliminar



a face da imagem.

- Detecção de pele: detecta regiões dentro de uma faixa de cores (utilizando o espaço HSV). O ajuste para detecção de pele foi feito manualmente.

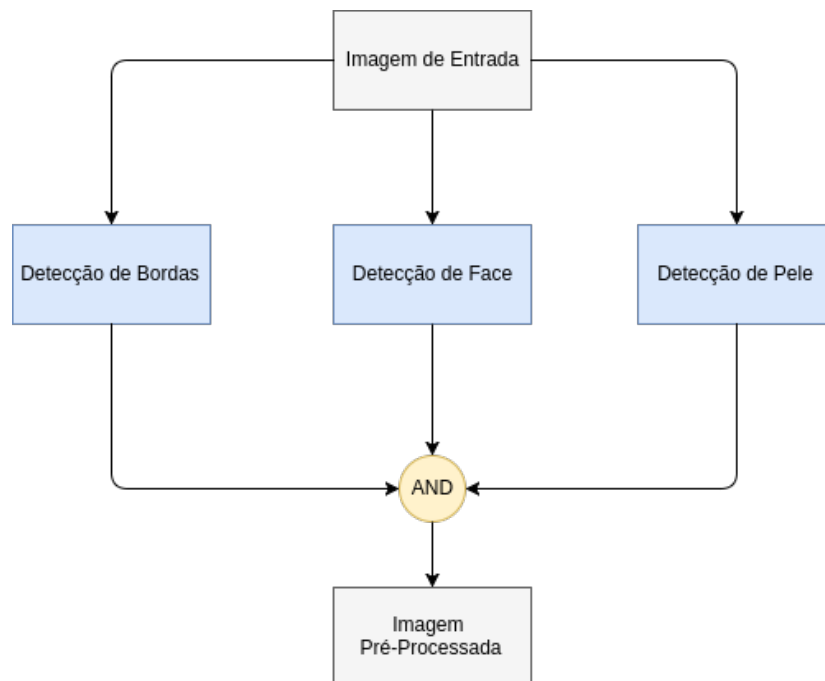


Figura 3.2: Etapas do pré-processamento.

Ao final, o resultado dos 3 processos são combinados em uma única imagem. O resultado é uma imagem como mostrado na figura 3.3, onde apenas o contorno das mãos fica visível na tela. Esta imagem é convertida para "jpg" e enviada pela rede à Raspberry onde ocorrerá a segunda parte do processo.

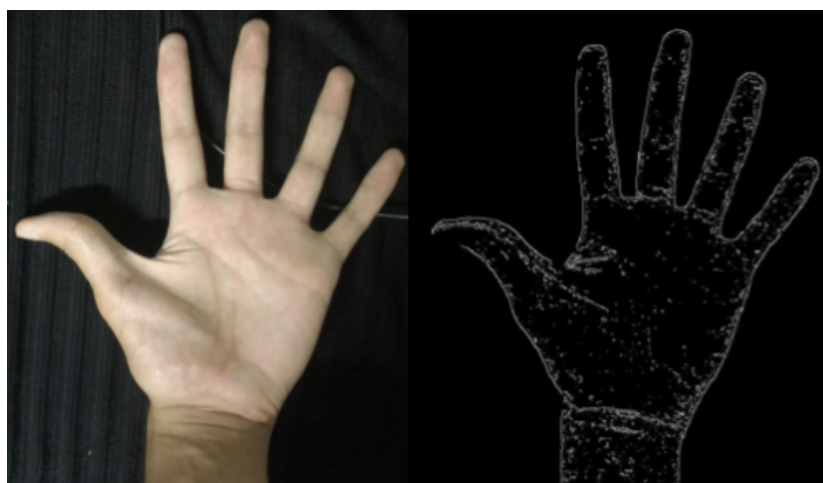


Figura 3.3: Resultado do pré-processamento.

### 3.2.3 Hash

A imagem recebida passa pelo processo de *Average Hash*, disponibilizado pela biblioteca *ImageHash* do Python. Este processo é feito para facilitar a comparação das imagens pois, ao final do processo, as imagens serão representadas por números, que podem ser facilmente armazenadas na base de dados e comparada com operações aritméticas simples.

Experimentalmente, chegou-se ao valor de 256-bits para o tamanho do *hash*, abaixo deste valor não era possível diferenciar as imagens com o classificador utilizado. A figura 3.4 mostra um exemplo de como o *hash* atua na imagem binarizada mostrada na figura 3.3.



Figura 3.4: *Hash* de 64 bits aplicado na imagem binarizada.

### 3.2.4 Classificação

Por meio da interface *web* desenvolvida, mostrada na figura 3.5, é possível visualizar cada imagem recebida pela Raspberry e fazer o treinamento do banco de dados relacionando a imagem visualizada ao seu número correspondente.

Ao mesmo tempo, em segundo plano, o classificador *k-NN* executa continuamente a classificação de cada imagem recebida. Foram utilizados os 3 resultados melhores classificados (3 vizinhos). Este valor também foi decidido experimentalmente de forma a não consumir muito tempo no processo de classificação.

Antes de enviar o resultado novamente ao Android, foi aplicado um *threshold* buscando diminuir a taxa de erro. Desta forma, cada novo resultado obtido do classificador só é enviado pela rede como resultado final se for, pelo menos, 10% diferente do resultado enviado anteriormente. O valor de 10% foi obtido verificando como o classificador se comporta quando não há mudança no símbolo de entrada (qual o erro nesta condição).

A imagem 3.6 apresenta um diagrama simplificado do funcionamento do sistema como um todo.

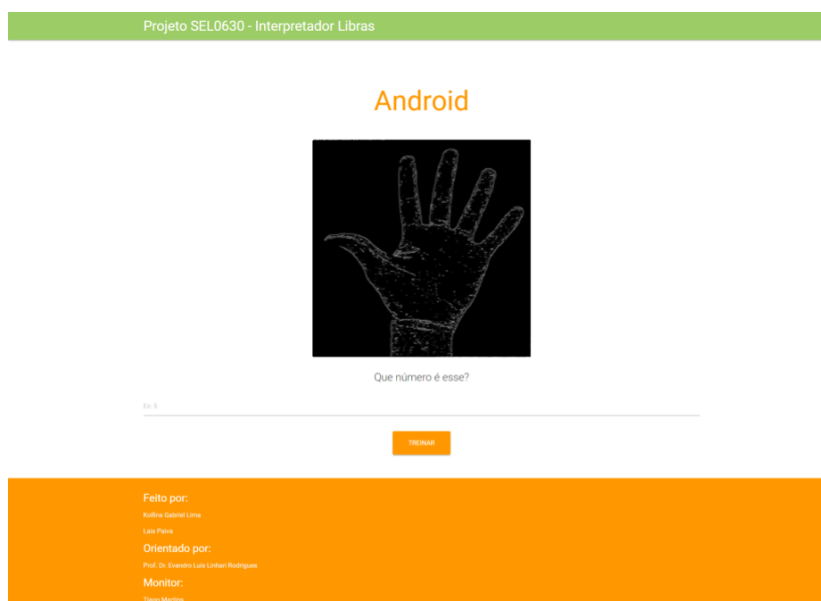


Figura 3.5: Página *web* para treinamento do banco de dados.

Os códigos desenvolvidos para esta aplicação podem ser encontrados no Github.

- Android: [https://github.com/kollinslima/ProjetoFinal\\_micros2\\_android](https://github.com/kollinslima/ProjetoFinal_micros2_android)
- Raspberry: [https://github.com/kollinslima/ProjetoFinal\\_micros2\\_rasp.git](https://github.com/kollinslima/ProjetoFinal_micros2_rasp.git)

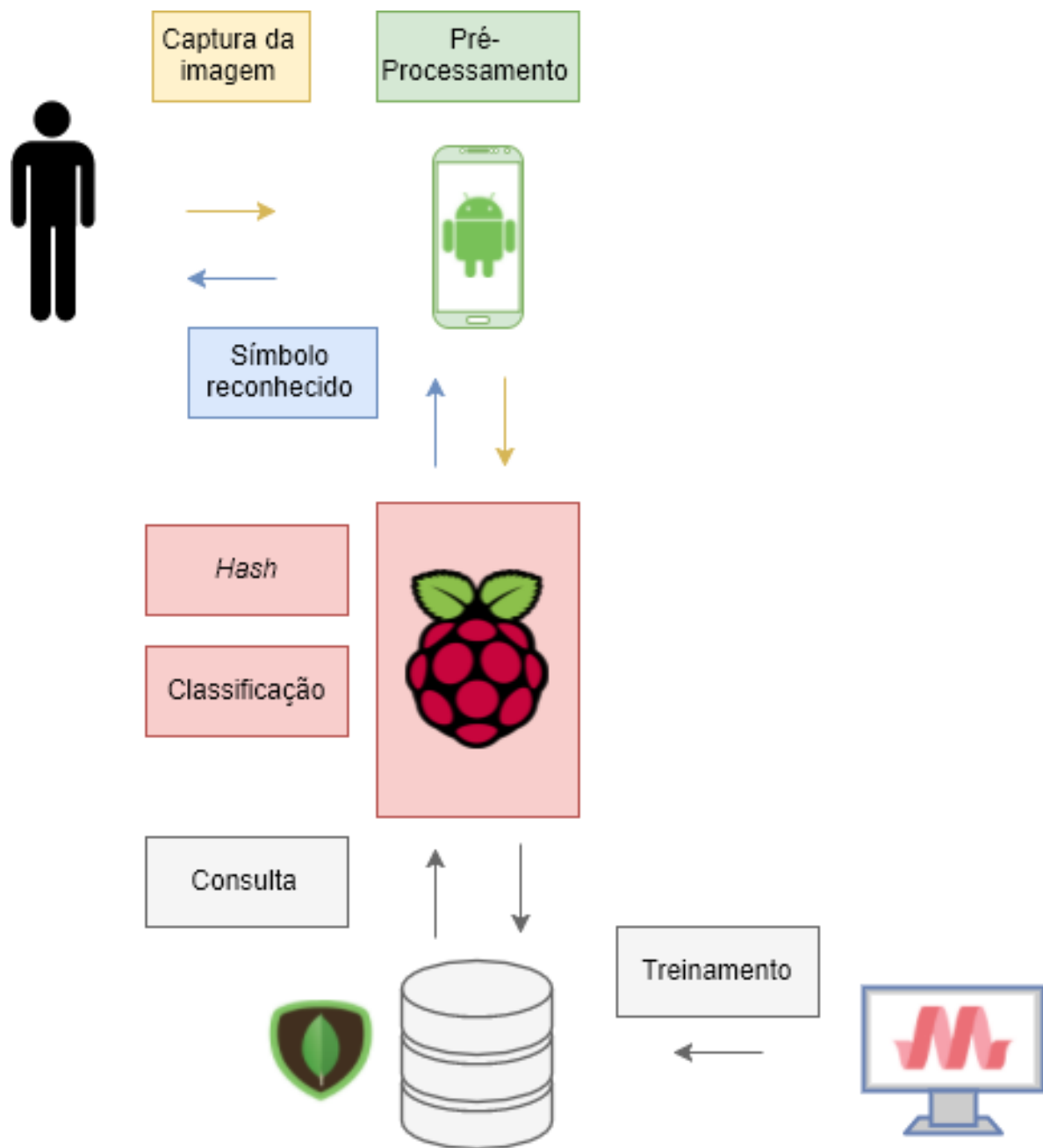


Figura 3.6: Diagrama simplificado do sistema.

## Capítulo 4

# Resultados e Discussões

Os símbolos numéricos a serem classificados são mostrados na figura 4.1.

O sistema funciona com baixas taxas de erro quando trabalha com poucos símbolos, no entanto o processo de classificação se torna impreciso conforme a quantidade de símbolos diferentes aumenta. A figura 4.2 mostra a taxa de erro obtida em função da quantidade de símbolos diferentes que foram adicionados ao banco de dados.

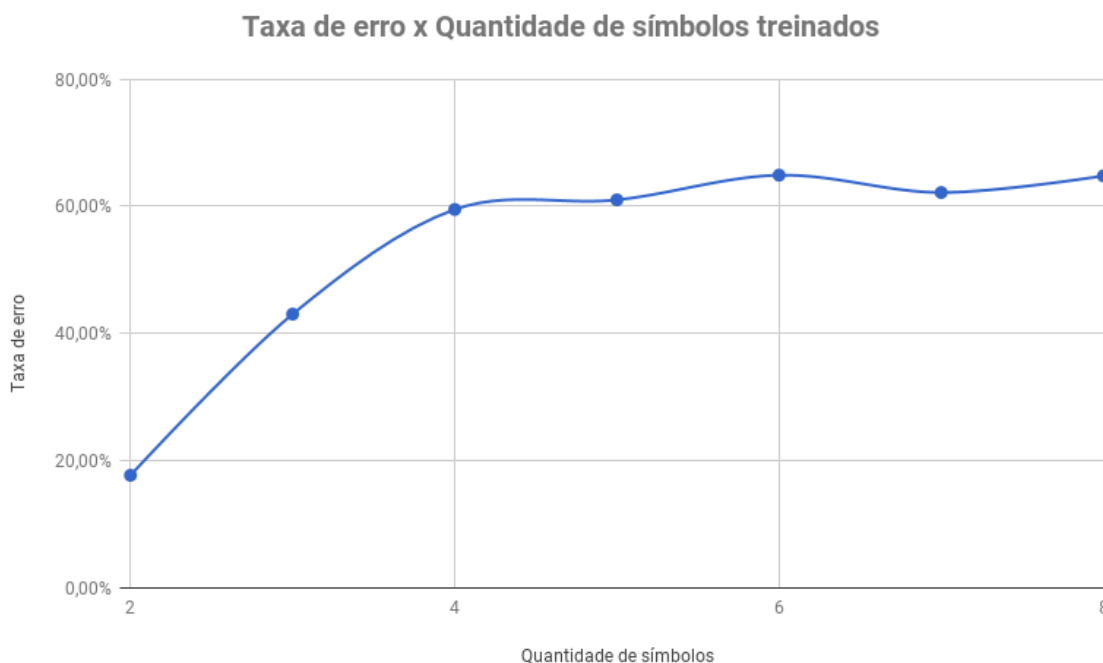


Figura 4.2: Diagrama simplificado do sistema.

Para realizar os testes, o banco de dados foi treinado com a mesma quantidade de exemplos para cada símbolo (3 exemplos) e foi dada como entrada a mesma quantidade de instâncias de cada símbolo


	0
	1
	2
	3
	4
	5
	6
	7

Figura 4.1: Símbolos propostos para o sistema de classificação.

(3 instâncias para cada símbolo treinado). O gráfico apresenta a taxa de desequilíbrio entre o esperado e o obtido.

Como mostrado na figura 4.2, acima de 2 símbolos o sistema é caótico, com baixas taxas de acerto. Esse resultado indica duas possibilidades para que o sistema não funcione como o esperado:

1. O processo de *hash* elimina muitas características da imagem de forma que diferenciar muitas imagens se torna uma tarefa complicada.
2. O classificador utilizado (*k-NN*) não é bom o suficiente para classificar muitos símbolos.

Muito embora o classificador *k-NN* seja bastante simples se comparado às técnicas utilizadas nos trabalhos citados anteriormente, a transformação *hash* é a principal responsável pelos resultados obtidos, uma vez que características importantes da imagem são descartadas neste processo.

Isso ficou evidente em um segundo teste realizado com um novo conjunto de símbolos, mostrados na figura 4.3. Como se pode notar, estes novos símbolos possuem um nível de detalhamento menor e exploram diferentes regiões da imagem, o que faz com que o *hash* de cada imagem tenha uma maior diferença.

Com estes novos símbolos, obteve-se uma melhora na classificação de aproximadamente 10% (em relação ao caso de teste com 8 símbolos). Embora essa taxa de erro ainda seja elevada (acima de 50%), na prática pôde-se notar um sistema muito mais estável e preciso quando comparado ao caso de teste anterior.









	0
	1
	2
	3
	4
	5
	6
	7

Figura 4.3: Símbolos alternativos para o sistema de classificação.



## 4.1 Desempenho

Quanto ao desempenho dos sistemas, podemos observar na figura 4.4 que a Raspberry está utilizando todo o processamento disponível para manter a interface *web* e as operações de comunicação com a rede e o banco de dados.

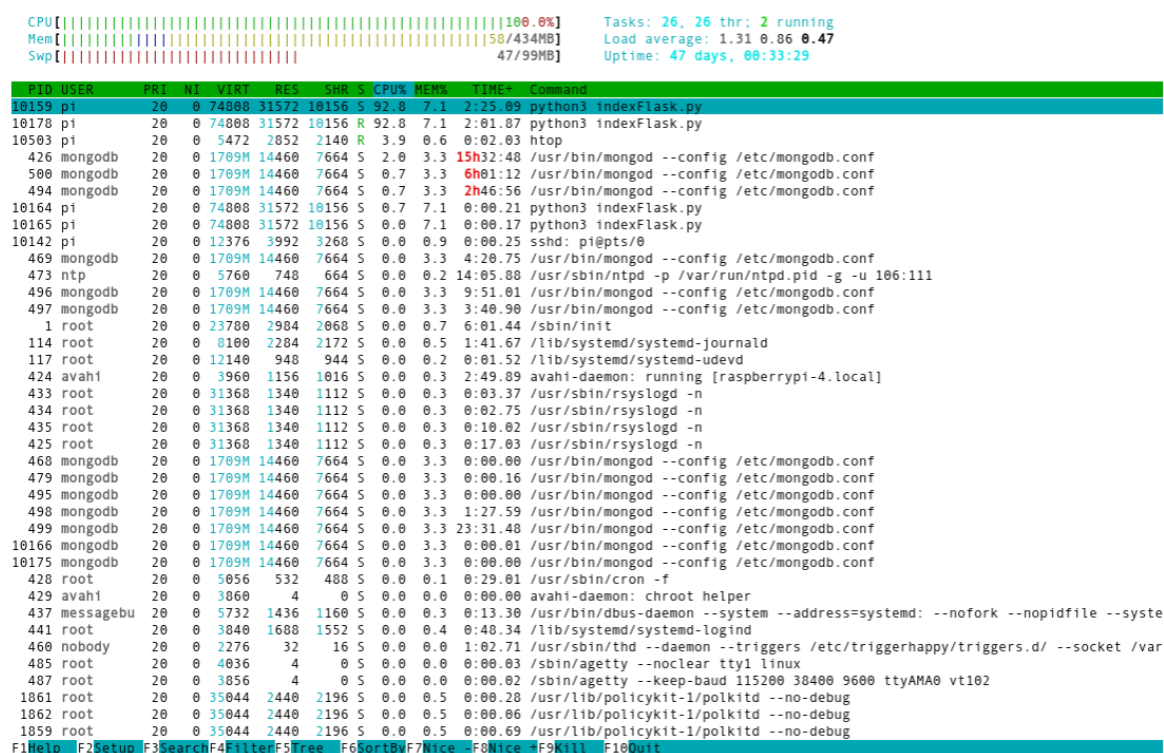


Figura 4.4: Consumo de recursos pela Raspberry pi. Pode-se observar que a aplicação em python e o banco de dados MongoDB consomem praticamente 10 % dos 434MB de memória principal disponível e quase 95 % do poder de processamento.

Embora o consumo de processamento esteja elevado, não foi percebido problema no desempenho da aplicação (apesar do sistema ser lento na inicialização). Para o desenvolvimento na Raspberry pi, o consumo de recurso foi pensado desde o início do projeto, sendo esse o principal motivo do uso da biblioteca *ImageHash* em alternativa ao *OpenCV*.

Quanto ao aplicativo Android, as figuras 4.5, 4.6 e 4.7 mostram as características de desempenho referentes ao tamanho do aplicativo, consumo de bateria e consumo de recursos de CPU e memória. Para testar o sistema, foi utilizado um *smartphone* ASUS ZenFone 2, com 4 núcleos de processamento (2,33GHz), 4GB de memória principal, 32GB de armazenamento, executando o Android 5.0.

O que se pode observar do aplicativo é que o processamento de imagem o tornou muito “pesado”, fazendo-o consumir muita bateria, CPU, memória e armazenamento, tornando essa aplicação inviável para uso no dia-a-dia.

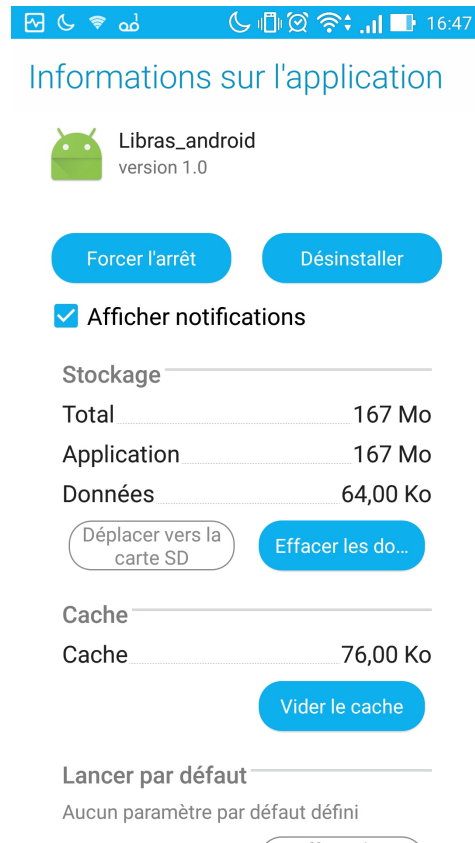


Figura 4.5: Tamanho do aplicativo android instalado no *smartphone*. Além dos 167MB consumido pelo aplicativo, é preciso mais 44,43MB de armazenamento para a instalação do *OpenCV Manager*, necessário para o funcionamento do *OpenCV* no android.

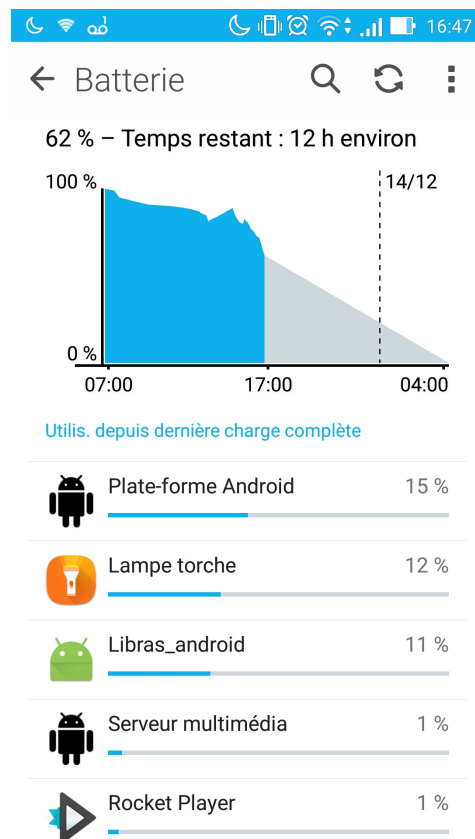


Figura 4.6: Consumo de bateria pelo aplicativo. Percebe-se um consumo elevado de bateria, o que se justifica pelas operações de processamento de imagem que estão sendo realizadas na etapa de pré-processamento.

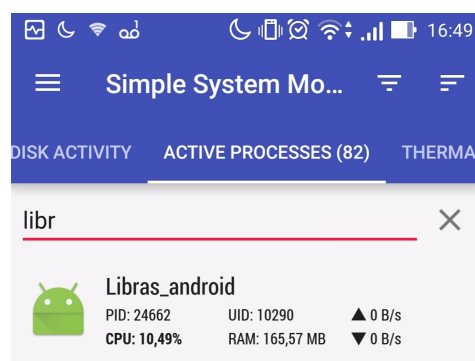


Figura 4.7: Consumo de memória principal e CPU do aplicativo. O consumo de mais de 10 % de CPU e 165MB de memória não afetaram o desempenho do aparelho, mas evidenciam um consumo grande quando comparado à outros aplicativos do sistema. (medição feita com aplicativo *Simple System Monitor*.)

Apesar disso, provavelmente a transferência dessas tarefas para a Raspberry pi acarretaria em perda de desempenho, já que, como foi mostrado na figura 4.4, o sistema já trabalha praticamente à 100% de carga.

Como solução à esse problema, provavelmente o melhor a se fazer é trocar a Raspberry pi por uma plataforma mais poderosa que possa executar com tranquilidade tanto o pré-processamento quanto o processamento em sí, ou ainda utilizar soluções em nuvem para transferir parte do trabalho a ser executado à servidores mais poderosos, podendo assim continuar com a Raspberry pi. De qualquer modo, é certo que utilizar o Android para processamento de imagem não parece ser uma solução muito efetiva quando o objetivo é criar um sistema portátil para ser usado no dia-a-dia (como é a proposta do projeto) devido, principalmente, ao consumo de bateria.

## Capítulo 5

# Conclusão

Neste trabalho foi apresentado uma proposta para realizar o reconhecimento da linguagem de sinais, aplicada à Língua Brasileira de Sinais - LIBRAS, através de um sistema Android .

Pode-se dizer que o trabalho atinge parcialmente seus objetivos. Por um lado, foi implementado um sistema capaz de ler, processar, classificar e responder textualmente os símbolos apresentados por meio da câmera de um *smartphone* Android. Por outro, com uma taxa de erro superior à 60%, o sistema não é capaz de fazer a classificação correta para todos os símbolos numéricos propostos.

Ademais, a grande quantidade de erros não permite que o sistema possa ser confiável e usado livremente como aplicativo comerciável e disponibilizado para download na comunidade Android, pois seu uso ainda não estaria finalizado com sucesso. Também foi identificado que o uso da biblioteca *OpenCV* no aplicativo o tornou demasiado grande, e seu download implicaria na necessidade de complementar com o da biblioteca *OpenCV*, o que traria desconfiança ao usuário, além de que testes demonstraram que o aplicativo demandou bastante da bateria do *smartphone*.

Analisando o que foi desenvolvido, notou-se que esta proposta não é suficiente para prover independência ao indivíduo surdo, que é o objetivo principal, pois além de se limitar aos números somente, o sistema não é fácil uso no dia-a-dia, mesmo se estivesse em perfeito funcionamento, uma vez que depende de uma câmera apontando para o usuário, fazendo com que o aplicativo dependesse de duas pessoas para seu uso (uma segurando a câmera e outra fazendo os gestos), do contrário restringiria os movimentos por exigir que uma das mãos segure a câmera. Seu uso seria mais eficiente em um contexto de palestra ou algum outro evento onde já existe uma câmera externa monitorando o ambiente, também com exemplo de pesquisa e desenvolvimento na área acessibilidade com processamento de imagens.

Apesar dos resultados obtidos, o desenvolvimento deste projeto exigiu conhecimento de diversas áreas antes pouco exploradas, o que foi de grande valia para o aprendizado. Entre os conhecimentos

obtidos, pode-se destacar as diversas técnicas de processamento de imagem que foram necessárias na etapa de pré-processamento (com auxílio da biblioteca *OpenCV*); o uso da Raspberry-pi como elemento central no projeto e seu Linux Embarcado; o uso do *Flask* para desenvolver a interface *web*; o uso do banco de dados MongoDB além da prática em desenvolvimento na linguagem Python e para Android.

Por fim, foi observado que para a conclusão efetiva dos objetivos propostos e para benfeitoria da comunidade surda o sistema ainda precisaria ser mais desenvolvido e trabalhado para chegar a um resultado mais utilizável com bom desempenho, com esta análise foi possível concluir que em trabalhos futuros, é preciso pensar em outras maneiras de fazer o reconhecimento de gestos, seja por um processamento de imagem diferente, como utilizar a biblioteca *OpenCV* na Raspiberry-pi diretamente, reduzindo o consumo de bateria no *smartphone* e eliminando a técnica de *hash*, que se mostrou ineficiente para a aplicação e o uso de um sistema mais sofisticado de classificação, que traga uma resposta mais rápida e assertiva ao sistema. Também é preciso que seja de fácil uso por uma única pessoa e que não envolva uma quantidade excessiva de hardware externo, tanto para prover um sistema mais acessível, quanto devido à estética.

## Referências

- [1] Claudio Alves Benassi. Da lei n. 10.436 de 24 de abril de 2002, 2014.
- [2] Portal Brasil (Org.). Apesar de avanços, surdos ainda enfrentam barreiras de acessibilidade. Disponível em: <<http://www.brasil.gov.br/cidadania-e-justica/2016/09/apesar-de-avancos-surdos-ainda-enfrentam-barreiras-de-acessibilidade>>. Acesso em: 08 out. 2017, 2016.
- [3] Luciano A. Digiampietri Beatriz Teodoro. Desenvolvimento de um sistema de reconhecimento automático de língua brasileira de sinais. Disponível em: <http://www.each.usp.br/digiampietri/bibtex/TeodoroEDigiampietri2014a.pdf>. Acesso em: 22 out. 2017, 2014.
- [4] João D. S. Almeida Ruberth A. A. Barros, Aitan V. Pontes. Reconhecimento de linguagem de sinais: aplicação em libras, 2014.
- [5] Hyotaek Lim Hui-Shyong Yeo, Byung-Gook Lee. Hand tracking and gesture recognition system for human-computer interaction using low-cost hardware, 2013.
- [6] Yanyu Niu Xiaotang Wen. A method for hand gesture recognition based on morphology and fingertipangle, 2010.
- [7] Chung-Lin Huang Feng-Sheng Chen, Chih-Ming Fu. Hand gesture recognition using a real-time tracking method and hidden markov models, 2003.
- [8] Adrian Rosebrock. Skin detection: A step-by-step example using python and opencv. Disponível em: <<https://www.pyimagesearch.com/2014/08/18/skin-detection-step-step-example-using-python-opencv/>>. Acesso em: 15 out. 2017, 2014.
- [9] Fábio Alexandre Caravieri Modesto Adilson Roberto Pavan. Reconhecimento de gestos com segmentação de imagens dinâmicas aplicadas a libras, 2010.

- [10] Valdir Grassi Jr. Processamento de imagens arquivo. Disponível em: [https://edisciplinas.usp.br/pluginfile.php/4145717/mod\\_resource/content/1/Processamento%20de%20Imagens.pdf](https://edisciplinas.usp.br/pluginfile.php/4145717/mod_resource/content/1/Processamento%20de%20Imagens.pdf). Acesso em: 18 nov. 2017, 2017.
- [11] Neal Krawetz. Looks like it. Disponível em: <http://www.hackerfactor.com/blog/index.php/?archives/432-Looks-Like-It.html>. Acesso em: 12 nov. 2017, 2011.
- [12] Solange Oliveira Rezende Rafael Geraldeli Rossi. Aprendendo com os vizinhos. Disponível em: [http://java.icmc.usp.br/moodle/pluginfile.php/5331/mod\\_resource/content/0/S14-Aprendendo%20com%20Vizinhos.pdf](http://java.icmc.usp.br/moodle/pluginfile.php/5331/mod_resource/content/0/S14-Aprendendo%20com%20Vizinhos.pdf). Acesso em: 18 nov. 2017, 2017.