

## SISTEMA PARA CORREÇÃO DE GABARITOS USANDO PROCESSAMENTO DIGITAL DE IMAGENS

Otávio Rocha Faria<sup>1</sup>  
Bruno Ramon de Almeida e Silva<sup>2</sup>  
Fábio Roberto Pillat<sup>3</sup>

**Resumo:** O crescente uso da visão computacional está se tornando comum para obtenção e análise de dados, pois é possível obter resultados precisos e com menor tempo se comparado a um ser humano. A visão computacional pode ser aplicada em diferentes áreas, inclusive ao ambiente acadêmico, que possui as necessidades de minimizar o tempo que os docentes levam para realizar a correção de avaliações e a dificuldade em mensurar os resultados obtido pelos discentes. O presente trabalho tem a finalidade de exibir o processo de construção de um aplicativo para realizar a correção de gabaritos utilizando técnicas e algoritmos de processamento digital de imagens. Além do aplicativo conseguir realizar a correção de avaliações, haverá a possibilidade de obter o índice de desempenho por aluno em relação a avaliação, essa função permite o professor utilizar diferentes metodologias de ensino de acordo com a realidade de cada turma.

**Palavras-chave:** Aplicativo, Processamento Digital de Imagens, Visão Computacional, Gabaritos.

**Abstract:** The increasing use of computer vision is becoming common for data collection and analysis, as it is possible to obtain accurate and shorter time results compared to a human being. The computer vision can be applied in different areas, including the academic environment, which has the need to minimize the time teachers take to correct the evaluations and the difficulty in measuring the results obtained by the students. This paper aims to show the process of building an application to perform the correction of jigs using techniques and algorithms of digital image processing. In addition to the application being able to perform the correction of assessments, there will be the possibility of obtaining the performance index per student in relation to assessment, this function allows the teacher to use different teaching methodologies according to the reality of each class.

**Keywords:** Application, Digital Image Processing, Computer Vision, Templates.

### 1 INTRODUÇÃO

A aplicação da visão computacional está revolucionando diversos setores da economia, devido a capacidade de poder abstrair informações relevantes em imagens digitais. As aplicações envolvendo a visão computacional estão se tornando cada vez mais comuns em todas as áreas, devido a facilidade em utilizar ferramentas computacionais e a difusão do conhecimento através da rede mundial de computadores.

<sup>1</sup> Acadêmico do Curso de Otávio Rocha Faria. Linha de Pesquisa. E-mail: otavio-roch@hotmail.com

<sup>2</sup> Professor Orientador do Grupo de Pesquisa xxxxxxxx do Curso xxxxxxxx. E-mail:

<sup>3</sup> Professor Orientador do Grupo de Pesquisa xxxxxxxx do Curso xxxxxxxx. E-mail:

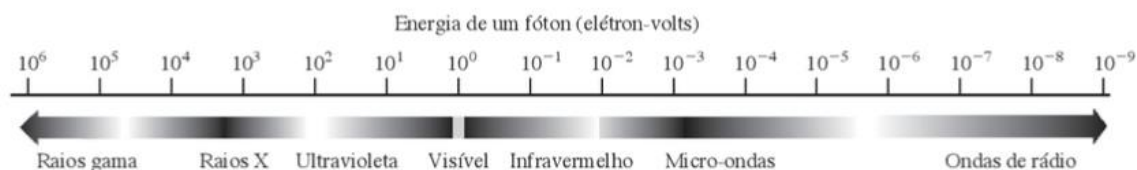
A utilização dessas tecnologias envolvendo processamento digital de imagens está ajudando pessoas a otimizar tempo, adquirir resultados precisos e extrair informações relevantes. No ambiente acadêmico muitos professores destinam o seu tempo fora da carga horária de trabalho corrigindo provas, essa atividade além de ser onerosa e consome um tempo significativo, existem os fatores humanos presentes que podem contribuir com correções errôneas, devido a possibilidade de algumas questões passarem despercebidas pelo professor. Devido a estes problemas esta pesquisa pretende criar um aplicativo utilizando conceitos e aplicações de visão computacional para realizar a correção dessas avaliações. O aplicativo proposto proverá de módulos para obter o índice de desempenho do aluno em relação a avaliação, esta variável permitirá o professor moldar metodologias de acordo com a realidade da turma.

## **2.1. VISÃO COMPUTACIONAL**

A Visão Computacional (VC) permite que uma máquina seja capaz de reconhecer o ambiente através de sensores e abstrair informações do mesmo, isso possibilita a automação de várias tarefas através de um sistema computacional. Segundo Ballard e Brown (1982) conforme citada por Barelli (2018) a Visão Computacional é a ciência que desenvolve e estuda tecnologias que permitem que máquinas extraiam informações do ambiente através de imagens capturadas por sensores, essas informações permitem a manipulação dos objetos que compõe a imagem.

O espectro de cores das imagens que o ser humano consegue enxergar é limitado de acordo com a figura 1, isso impossibilita que o ser humano visualize alguns espectros eletromagnéticos como os raios gama, raios x, ultravioleta, infravermelho, micro-ondas e ondas de rádios. Porém é possível visualizar esses espectros através de sensores e softwares de Processamento digital de imagens (PDI).

**Figura 1 - Representação de espectro eletromagnético de acordo com a energia do fóton**



Fonte: Gonzalez (2010, p. 5)

Devido à existência de sensores capazes de capturar ondas eletromagnéticas e o avanço do PDI foi possível a criação de várias tecnologias. Segundo Barbosa (2017) a Visão Computacional é amplamente utilizada para resolver problemas utilizando PDI e alguns benefícios que surgiram com a visão computacional foram: Resultado de exames obtidos por Raios-X, processamento para reconhecimento biométrico ou facial, Visão Computacional voltada à indústria e aplicações em próteses.

Para poder realizar o reconhecimento em uma imagem com baixos índices de erros e com uma boa otimização é necessário levar em consideração as principais etapas de PDI. De acordo com Barbosa (2017, p. 7) existem processos fundamentais no processo de processamento digitais de imagens que estão relacionados entre si, provocando comportamento que interferem nas fases posteriores.

**Figura 2 – Fluxo de um sistema baseado na visão computacional**



Fonte: Barelli (2018)

A figura 2 representa as principais etapas do processo de Visão Computacional. De acordo com Barelli (2018) a VC mesmo estando presente em

uma gama de áreas e em diversas tecnologias o processo geralmente apresenta um fluxo em comum. O processo de aquisição de uma imagem destacado pela figura 02 é realizado quando os sensores computacionais conseguem capturar características do ambiente e realizar a conversão em objetos lógicos, esses objetos são conhecidos como imagens digitais.

## 2.2. IMAGENS DIGITAIS

Para a formação de uma imagem digital são necessários elementos finitos dentro de um conjunto, esses elementos são chamados de pixel. Segundo Gonzalez (2010, p. 1) “[...] uma imagem digital é composta de um número finito de elementos, cada um com localização e valor específicos. Esses elementos são chamados de pictóricos, elementos de imagem, pels. Pixel é o termo mais utilizado para representar os elementos de uma imagem digital”.

O processo de aquisição de uma imagem digital consiste na transformação de sinais analógicos em digitais. Esse processo é realizado por sensores que converte o espectro analógico em digital. Para Barbosa (2017, p. 4) as imagens digitais podem ser definidas em 2 tipos, as imagens matriciais que são definidas através de uma matriz bidimensional de pontos  $f(x, y)$  e as imagens vetoriais que são representação de números binários (0, 1).

No ambiente computacional uma imagem digital pode ser representado através de uma matriz bidimensional que possui as coordenadas  $(x, y)$ , que para cada par de coordenadas existe uma variável  $f$  que representa a intensidade do brilho da imagem. De acordo com (GONZALEZ, 2010, p.1)

Uma imagem pode ser definida como uma função bidimensional,  $f(x, y)$ , quem que  $x$  e  $y$  são coordenadas espaciais (plano), e a amplitude de  $f$  em qualquer par de coordenadas  $(x, y)$  é chamada de intensidade ou nível de cinza na imagem nesse ponto. Quando  $x$ ,  $y$  e os valores de intensidade de  $f$  são quantidades finitas e discretas, chamamos de imagem digital.

Também é possível representar uma imagem digital através de um espaço contínuo finito, essa representação de imagens é denominada imagens vetoriais e possui uma estrutura unidimensional formada por conjuntos de bits. Segundo Barbosa (2017, p. 4) as imagens vetoriais possuem uma estrutura linear formada por bits, essa classe de imagem ocupa menos espaço e possui uma eficiência maior em

relação às imagens matriciais.

Devido à presença de cores em algumas imagens digitais é possível representar o nível da cor em uma escala. Para Barbosa (2017, p. 5) as imagens digitais também têm cores que são representadas no sistema padrão RGB. Esse sistema possui três cores primárias que são: vermelho (*Red*), verde (*Green*) e azul (*Blue*). Através da união desses três conjuntos em tonalidades distintas é possível a formação de novas cores.

Segundo Silva (2014, p. 8) há também imagens sem cores, que são denominados de monocromática ou acromáticos. Esse tipo de imagem possui uma característica chamada de intensidade que representa a quantidade de luz. A intensidade da iluminação pode variar de um valor mínimo (preto), um valor médio (cinza) e um valor alto (branco). Isso significa que quanto maior for o valor maior será a intensidade. Para De Campos (2001, p. 22) todos os valores intermediários entre o valor mínimo e o máximo serão tons de cinza e cada ponto pode assumir 256 valores diferentes, variando em uma escala de 0 a 255.

### **2.3. PRÉ-PROCESSAMENTO E SEGMENTAÇÃO DE IMAGENS**


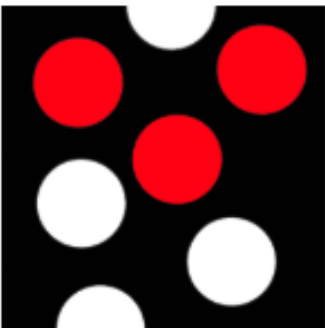
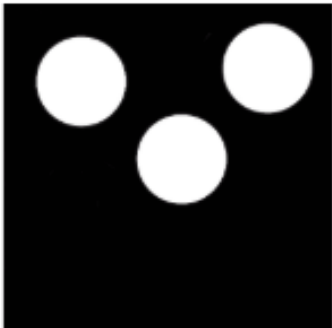
O pré-processamento de imagens é responsável por capturar atributos de uma imagem, realçar e diminuir o nível de ruído, de acordo com Filho e Neto, (1999, apud Barbosa, 2017, p.7), “[...] é a área de processamento de imagens que viabiliza um grande número de aplicações em duas categorias distintas: O aprimoramento de informações visuais para interpretação humana, e a análise automática por computador, de informações extraídas de uma cena”.

Dependendo da parametrização do algoritmo de pré-processamento de imagens o mesmo pode capturar objetos sem sentido. Segundo De Campos (2001, p. 23) além de informações de interesse, são capturados sinais que não dão nexos a imagem e acabam prejudicando a análise. O processo de pré-processamento de imagens é fundamental para extração de características e alterações em imagens, de acordo com Barbosa (2017, p. 7) uma das etapas principais após a obtenção da imagem consiste na manipulação de filtros, suavização, realce, redução de ruídos, e manipulação de histograma. Segundo Silva (2014, p. 10) essa técnica permite

também preparar a imagem para ser utilizada em etapas posteriores.

A técnica de segmentação de imagens é executada posteriormente a técnica de pré-processamento de imagens. Essa técnica tem o objetivo de realçar regiões dentro de áreas capturadas pelo pré-processamento. Existem algumas variáveis que podem ser levadas em consideração para realizar a segmentação, como a cor do objeto, diâmetro ou a forma geométrica. Segundo De Campos (2001, p. 24) “O processo de segmentação pode ser entendido como o particionamento de uma imagem em regiões que apresentam propriedades semelhantes, como textura ou cor”. Essa descrição pode ser entendida através da figura 03, pois o algoritmo de segmentação particionou a imagem e extraiu característica com base em regiões.

**Tabela 1 - Representação da etapa de segmentação de imagens**

		
Figura A	Figura B	Figura C

Fonte: Barelli (2018)

A figura A da tabela 2 demonstra o processo de aquisição de uma imagem, no qual os sensores computacionais capturaram características do ambiente e converteram em uma imagem digital representada por um conjunto de elementos em uma matriz bidimensional.

A figura B da tabela 2 demonstra o processo de pré-processamento de imagens, nessa etapa contém a extração de características de acordo com a forma geométrica do objeto, minimizando o nível de ruído em uma imagem digital.

A figura C da tabela 2 demonstra o processo de segmentação de uma imagem, nela é possível destacar apenas os objetos de interesse a ser analisado. O algoritmo utilizado para extração de características da imagem utilizou como



parâmetro a cor do objeto a ser estudado.

É possível analisar a partir da tabela 02 que houve uma redução de ruídos significantes para aumentar a eficiência do algoritmo de PDI e aumentar a probabilidade de sucesso ao extrair características de uma imagem digital. Nas etapas de pré-processamento e segmentação de imagens houve um tratamento da imagem para que o algoritmo de processamento de imagens dê sentido ao objeto a ser analisado.

## **2.4. PROCESSAMENTO DE IMAGENS**

O processamento digital de imagens envolve várias áreas do conhecimento pois permite a criação de aplicação para vários setores. Segundo Silva (2014, p. 8) O PDI possui ferramentas para criação de aplicações para o mundo real como indústria, comércio e saúde.

De acordo com Gonzalez (2010, p.1) não existe um acordo para definir os limites em relação ao processamento de imagens e outras áreas relacionadas, como a análise de imagens e a visão computacional, devido a entrada e saída de dados do processo serem imagens. É difícil dimensionar os limites de cada área de PDI, pois nas etapas de pré-processamento, segmentação e processamento de imagens há a necessidade de processar dados.

Para Barbosa (2017, p. 6) há parâmetros criados para conceituar o que se refere a PDI. Ele define esses parâmetros como processo de nível baixo, médio e alto. Nível baixo é o processo onde há um pré-processamento, geralmente aplicado em redução de ruídos, realce de contraste, aguçamentos de imagens, no qual a entrada e saída do processo é uma imagem. Nível médio é a etapa onde há uma separação de imagem em pedaços ou objetos, e a entrada é uma imagem, e a saída é um objeto. Nível alto é onde no processo há uma análise do objeto, fazendo com que uma imagem tenha sentido.

## **2.5 OPENCV**

A biblioteca *OpenCV* permite a extração de características de uma imagem através de algoritmos de visão computacional. Segundo Barbosa (2017, p.9) a

tecnologia *OpenCV* possui funções de processamento de imagens, vídeos, estruturas de dados, álgebra linear e interfaces gráficas. Ele menciona também que a biblioteca possui mais de 350 algoritmos de reconhecimento instalado nativamente.

Para Barelli (2018) a biblioteca *OpenCV* foi projetada para criação de aplicativos na área de visão computacional, processamento de imagens, estrutura de dados e álgebra linear. De acordo com ele essa tecnologia foi desenvolvida no ano 2000 e é totalmente livre para uso acadêmico e comercial.

Segundo Silva (2014, p.47) nas primeiras versões da plataforma *OpenCV* a tecnologia estava limitada a aplicações para desktop, posteriormente essa tecnologia foi portada para outros sistemas operacionais e atualmente é possível compilar aplicações para funcionar nas plataformas *Linux*, *Windows*, *Mac* e *Android*. Esse fato possibilita a exportação de aplicações para diferentes tipos de dispositivos com softwares distintos.

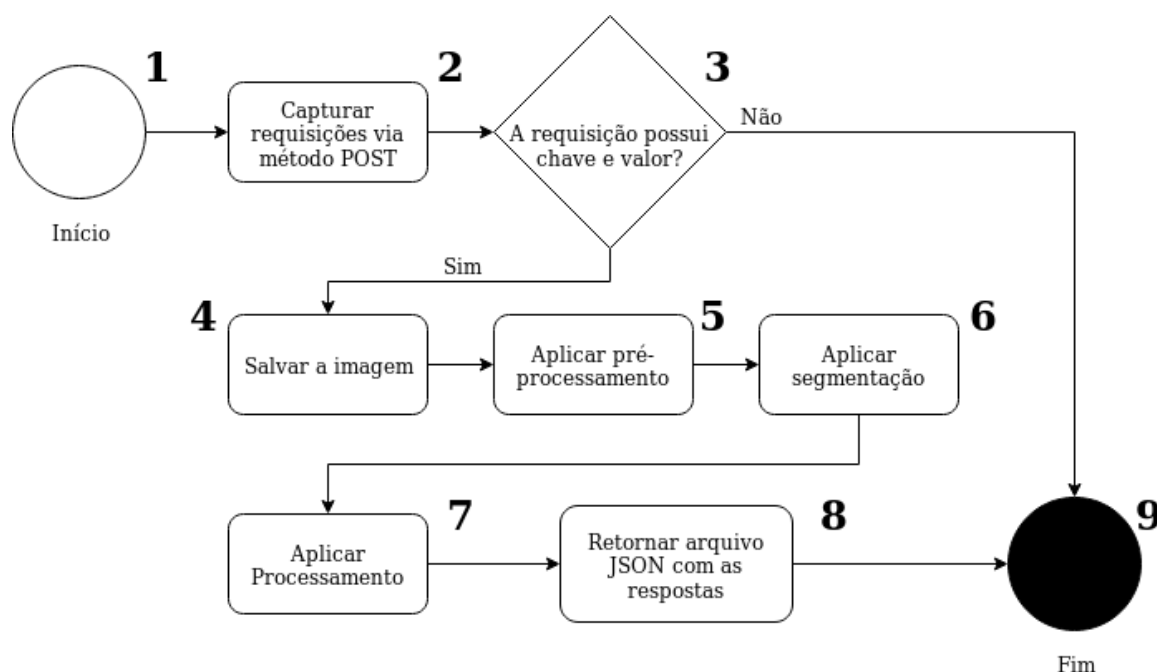


### 3 DESENVOLVIMENTO

Para realizar esta pesquisa foram utilizadas as tecnologias *OpenCV* para o processamento digital de imagens, a biblioteca *Flask* para a comunicação entre a linguagem Python e *Flutter*, a linguagem de programação Python para comunicação com o *OpenCV* e *Flask* e a linguagem *Dart* junto com o framework *Flutter* para a criação do aplicativo mobile.

Nesta etapa do desenvolvimento será abordada o fluxo necessário para realizar o processamento digital de imagens, a *API* responsável pela comunicação entre a linguagem *Python* e *Dart* e o processo para criação do aplicativo.

**Figura 03 – Fluxo geral do reconhecimento de gabaritos**



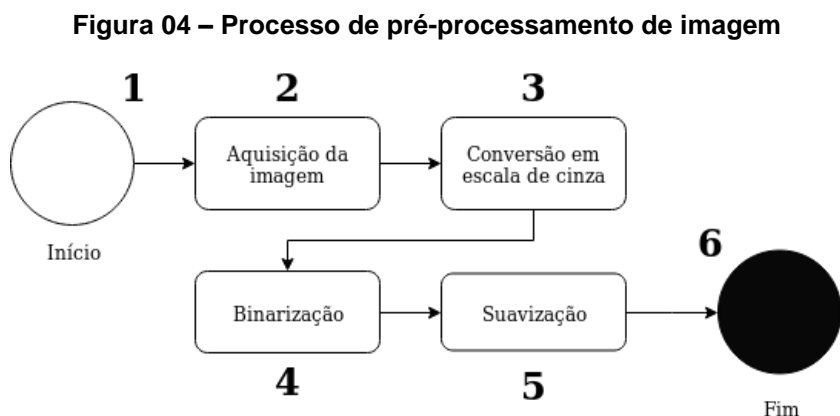
Fonte: Autor (2019)

O fluxo abordado na Figura 03 demonstra o processo geral de entrada, processamento e saída de dados do algoritmo proposto. A etapa 02 desse fluxograma demonstra a entrada de dados via requisições *POST*, caso o algoritmo não encontre dados relacionados a chave ou valor o fluxo se encerra. A próxima etapa é salvar a imagem caso haja sucesso na tarefa anterior, e posteriormente aplicar as técnicas de pré-processamento, segmentação e processamento de imagens. Após o resultado final da etapa de processamento digital de imagens o

algoritmo retornará um arquivo no formato JSON com as respostas respectiva as marcações dos alunos. Este fluxograma representado através da Figura 03 representa o processo geral do sistema de correção de gabaritos, há outras variáveis que será abordada nos passos posteriores.

### 3.1 PRÉ-PROCESSAMENTO DE IMAGENS

O primeiro passo é realizar a aquisição da imagem. Foi feita a aquisição através de uma foto de celular e realizado o redimensionamento para que a resolução não ultrapasse 1000 pixels de altura e largura. A modificação segue a sequência proposta na figura 3. Esse pré-processamento demonstrado na figura 3 serviu para eliminar ruídos e retornar as informações relevantes para a próxima etapa.



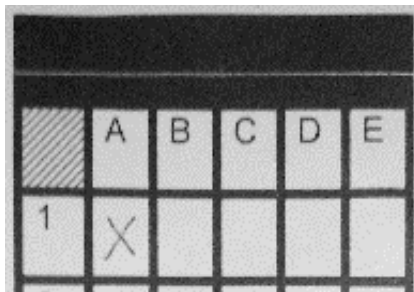
Fonte: Autor (2019)

A figura 04 representa o fluxo do processo de pré-processamento de imagens elaborado no projeto em questão, nele é possível visualizar as tarefas que serão abordadas posteriormente.

Primeiramente foram reduzidos os espectros de cores da imagem através do método `cv2.COLOR_BGR2GRAY`, essa função transforma toda a imagem cromática em monocromática. Isto significa que há a redução dos espectros de cores presentes nas imagens para tons de preto e branco. Há uma escala linear entre esses tons, que são definidos entre os valores 0 e 255. Esses valores são

responsáveis por definir a tonalidade entre essas duas cores, dependendo desses valores é possível criar diferentes tons de cinza.

**Figura 05 - Transformação da imagem em monocromática**



Fonte: Autor (2019)

A imagem 05 demonstra o resultado obtido através da função `cv2.COLOR_BGR2GRAY`. Este resultado é importante pois foram reduzidos os espectros de cores desnecessários e isso facilitará a busca por informações relevantes presentes na imagem.

Após transformar a imagem em monocromática é necessário converter todas as tonalidades de cinza em tons de preto ou branco, este processo chama-se binarização. A função `cv2.THRESH_BINARY` é responsável por definir apenas 2 valores para todos os pixels da imagem, que é os valores 0 representado pela cor preta e o valor 255 representado pela cor branca. Essa função também permite a escolha de uma faixa de tonalidade para cada cor, inicialmente para o algoritmo elaborado todos os valores entre 1 e 175 passavam a ter o valor 0, e os valores entre 176 e 254 recebiam o valor 255. Porém devido as diferentes condições de iluminações e sombras nas imagens analisadas percebe-se que um parâmetro padrão para aplicação do método `cv2.THRESH_BINARY` se torna ineficiente, devido a possibilidade de a figura adquirir segmentos com tonalidades desnecessárias da cor preta ou branca. A biblioteca *OpenCV* possui diversos algoritmos para a binarização de imagens, e percebeu-se que a função `cv2.adaptiveThreshold` consegue realizar a binarização levando em consideração as características do ambiente e com taxa de erros reduzidas.

Percebe-se que a utilização do pré-processamento de imagens é essencial para minimizar os índices de erros e maximizar o número de acertos no processo de processamento digital de imagens. A figura 06 ilustra o resultado final obtido nessa etapa sendo perceptível observar a diminuição do número de ruídos na imagem.

**Figura 06 - Transformando uma imagem em binária**



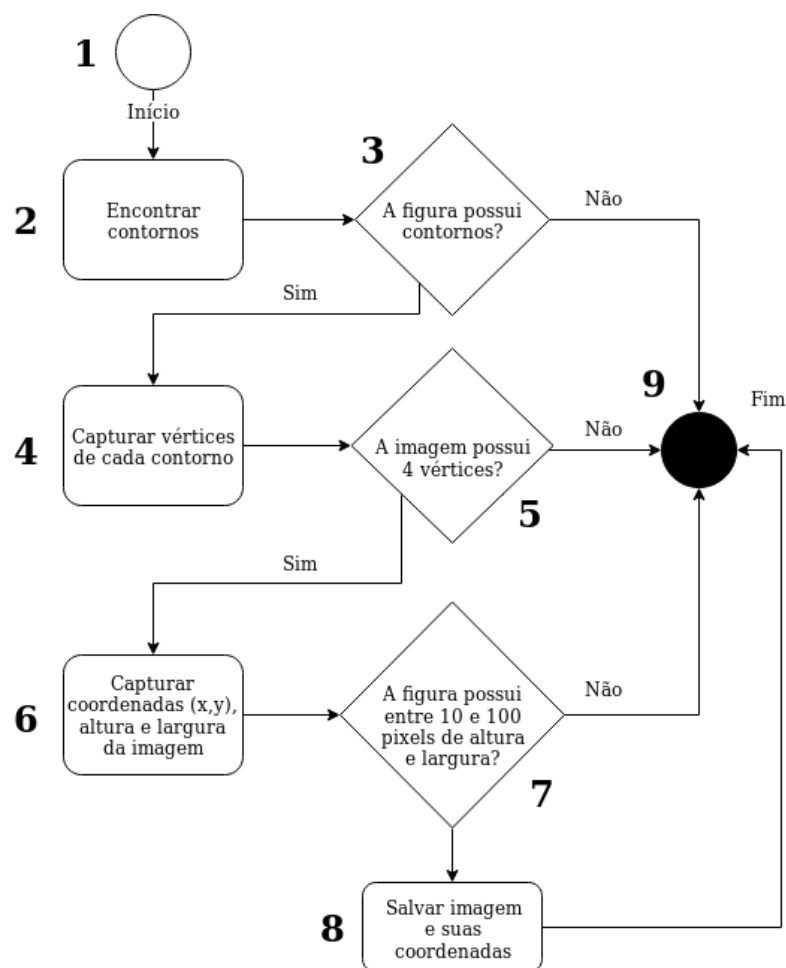
Fonte: Autor (2019)

A imagem 06 demonstra o resultado obtido após o uso da função *cv2.adaptiveThreshold*, nela é perceptível que apenas as cores branca e preta se encontram presente dentro do espectro de cores. E a utilização desta técnica é importante para localização dos contornos na imagem.

### **3.2 SEGMENTAÇÃO DE IMAGENS**

Após reduzir os ruídos na imagem é necessário utilizar ferramentas para encontrar padrões e capturar regiões de interesse, para isso faz necessário utilizar a técnica de segmentação de imagens. No processo de visão computacional existe a possibilidade de utilizar classificadores para reconhecimento de objetos e figuras geométricas. Esses classificadores são arquivos em formato *XML* que possui coordenadas que ajudam a reconhecer a figura de interesse, eles também são conhecidos como *haarcascade*. Porém neste momento não é necessário à sua utilização, visto que o formato quadrado do gabarito são figuras geométricas de baixa complexidade. Devido a forma geométrica da tabela presente no gabarito ser um quadrilátero, as propriedades internas da biblioteca *OpenCV* consegue realizar o processo de segmentação de imagens.

Figura 07 – Processo de segmentação de imagens



Fonte: Autor (2019)

A figura 07, demonstra o processo de segmentação de imagens, onde é necessário extrair as informações relevantes para o processo de processamento de imagens. Este processo representado visualmente através do fluxograma da figura X é o resultado dos algoritmos e práticas implementados neste trabalho.

Primeiramente para encontrar os polígonos de quatro lados é necessário a utilização da função *cv2.findContours* que permite encontra os contornos e bordas em uma imagem. Após capturar os contornos da imagem é importante a utilização do método *cv2.approxPolyDP*, que é responsável por retornar à quantidade de vértices presente em um polígono. Através do retorno dessa propriedade é possível testar se uma determinada figura geométrica é um quadrilátero ou não. Veja o trecho de código a seguir.

Figura 08 - Algoritmo para detecção de quadriláteros

```
15 contours,h = cv2.findContours(thresh,cv2.RETR_LIST, cv2.CHAIN_APPROX_SIMPLE)
16
17 contador = 0
18 for cnt in contours:
19     # captura o perímetro da imagem
20     perimetro = cv2.arcLength(cnt, True)
21     # aproxima os contornos da imagem, neste caso estou declarando a variável
22     approx = 0
23
24     if perimetro > 80 and perimetro < 300:
25         # aproxima os contornos da forma correspondente
26         approx = cv2.approxPolyDP(cnt, 0.04 * perimetro, True)
27         # verifica a quantidade de vértices, caso seja igual a 4 pode ser um...
28         # quadrado ou triângulo
29         if len(approx) == 4:
```

Fonte: Autor (2019)

O trecho de código descrito na Figura. 08, demonstra os métodos descritos anteriormente, com a finalidade de mostrar o processo de obtenção de um quadrilátero. Na linha 18 da figura 08 demonstra a captura dos contornos em uma imagem matricial, e para percorrer as coordenadas referente a cada contorno faz necessário o uso de um laço de repetição, sua utilização é demonstrada na linha 18. Outro detalhe importante é a condição presente na linha 29, que testa se a figura geométrica é um quadrilátero ou não. Se `len(approx)` fosse comparado com o número três o sistema retornaria os triângulos da imagem, ou se houvesse a comparação com o número cinco o sistema encontraria os pentágonos da figura. Como a comparação é realizada com o número quatro há o retorno de todos os quadrados presentes.

A próxima etapa foi realizar alguns testes para identificar as figuras geométricas com tamanho proporcional. Primeiramente é necessário obter as coordenadas x, y, altura e largura, para isso faz necessário o emprego da função `cv2.boundingRect`.

Figura 09 - Algoritmo para extração de polígonos

```
30
31
32
33
34
35
36
37
38
39
40
41
42

# boundingRect captura as coordenadas x,y e a altura e largura
(x, y, a, l) = cv2.boundingRect(cnt)
# A condição seguinte testa as seguintes propriedades:
# A altura e largura é maior que 15px e altura e largura é menor que 300px;
# O resultado da altura - largura é menor que 15, essa condicional serve...
# para encontrar figuras geométricas próximas de um quadrado.
if (a > 15 and l > 15 and a < 300 and l < 300 and a-l < 15):
    # cv2.rectangle é responsável por desenhar um retângulo na imagem
    cv2.rectangle(imagem, (x, y), (x + a, y + l), (0, 255, 0), 2)
    # variável roi captura a imagem "dentro" do retângulo
    roi = imagem[y:y + l, x:x + a]
    # cv2.imwrite grava a imagem em um arquivo de imagem no formato jpg
    cv2.imwrite("imagens/roix"+str(contador)+".jpg", roi)
```

Fonte: Autor (2019)

O trecho de código descrito na Figura. 09, demonstra o emprego da função `cv2.boundingRect` e sua finalidade foi descrita anteriormente. Também há uma condicional presente que testa se a altura e largura é maior que 15 pixels e menor que 300 pixels. Outra propriedade para a condição ser aceita é a capacidade de identificar se uma determinada figura geométrica se assemelha com um quadrado ou não. Caso a subtração da altura e largura for inferior a 15 a condição é aceita e caso o contrário não, este método ajuda a eliminar alguns retângulos com a largura ou altura desproporcional ao padrão estabelecido.

Para visualizar os resultados obtidos através do algoritmo é necessário destacar as regiões que a máquina conseguiu capturar. Para tal finalidade, é necessário o emprego do método `cv2.rectangle`, que desenha um retângulo de acordo com as coordenadas inseridas em seu parâmetro. Isto significa que é necessário informar as coordenadas (x,y) do plano cartesiano, as variáveis correspondente a altura e largura, código de cor correspondente a borda do retângulo e sua espessura. Veja a figura a seguir.



**Figura 10 - Resultado visual da segmentação de imagens**

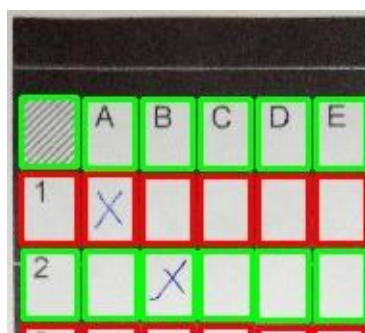


Fonte: Autor (2019)

A disposição dos contornos presentes na Figura. 10, foi desenhado através da função *cv2.rectangle* e demonstra o resultado dos processos anteriores para encontrar o formato correto do gabarito.

Para identificação das camadas do gabarito é necessário o mapeamento da variável que representa o eixo y do plano cartesiano, para isso foi necessário comparar as camadas de cima e baixo do gabarito e salva-la em um arquivo de texto.

**Figura 11 - Segmentação de imagens com as camadas reconhecidas**



Fonte: Autor (2019)




A figura 11 representa as camadas reconhecidas através da comparação entre as camadas do gabarito, essa tarefa é importante para salvar cada alternativa em sua pasta, para após isso ser analisada pelo processo de processamento digital de imagens.

O último passo para finalizar o processo de segmentação de imagens é salvar as áreas destacadas de acordo com a Figura. 11, para realizar esse processo

é necessário capturar uma parte na imagem que está inserida em cada quadrado. As variáveis  $x$ ,  $y$ ,  $a$ ,  $l$  representadas pela Figura. 09, são as coordenadas dos pontos de cada polígono. A linha 40 da Figura. 09, exemplifica o processo de captura da área selecionada. Já na linha 42 da Figura. 09, demonstra o método de armazenamento da imagem, que é realizado através da função `cv2.imwrite`. O resultado final do processo de segmentação de imagem é a captura da imagem interna presente dentro de cada quadrilátero.

### 3.3 REALCE DE IMAGENS E RECONHECIMENTO DE MARCAÇÕES.

Tabela 2 - Métodos para realce em imagens

Propriedades	Resultados
<code>cv2.COLOR_BGR2GRAY</code>	
<code>cv2.threshold</code>	
<code>cv2.GaussianBlur</code>	



















Fonte: Autor (2019)

Percebe-se através do primeiro resultado da Tabela. 02, que a imagem em questão apresenta uma resolução baixa. Isto pode implicar em resultados errôneos quando o algoritmo for analisar o objeto em evidência. Com o objetivo de solucionar este problema é necessário aplicar técnicas de realce nas imagens. As propriedades `cv2.COLOR_BGR2GRAY` e `cv2.threshold` foram explanadas anteriormente, porém é possível visualizar o processo através da Tabela. 02. Já a propriedade `cv2.GaussianBlur` é a responsável por atribuir um pequeno desfoque na imagem. Estes processos são importantes para aumentar a resolução e a nitidez do objeto conforme mostrado visualmente na Tabela. 02.

Inicialmente foi proposto a utilização da biblioteca *tesseract OCR* para realizar o reconhecimento óptico de caracteres. Porém através de testes utilizando essa tecnologia percebeu-se algumas variáveis que implicariam negativamente no processo em questão. Observa-se que a biblioteca é ineficiente para detectar apenas um caractere e devido aos diferentes tipos de caligrafias presentes no gabarito não é possível realizar o reconhecimento com esta tecnologia.

Para conseguir realizar o reconhecimento óptico de caracteres surgiu a necessidade do emprego de um arquivo *haarcascade*, porém não foi encontrado nenhum arquivo que atendesse os requisitos do algoritmo em questão. Devido a esse fator, foi utilizado a técnica de *machine learning* para criação de um arquivo *harrcascade* que atendesse os requisitos da aplicação. A técnica de *machine learning* possui o objetivo de aprender a reconhecer padrões conforme os parâmetros inseridos. Para a criação do arquivo *haarcascade* foi necessário utilizar dois tipos de imagens, que são as imagens positivas e negativas. As imagens positivas são aquelas que apresentam características em comum ao objeto a ser analisado e as imagens negativas são aquelas que não apresentam evidências com o contexto da imagem. Observe a tabela a seguir.

**Tabela 03 - Ilustração das imagens positivas e negativas**

Imagens positivas			Imagens negativas		
					
0-ocrf.jpg	0-ocry.jpg	1-ocrf.jpg	0-ocr.jpg	0-ocr5.jpg	0-ocrf.jpg
					
4-ocrr.jpg	4-ocry.jpg	5-ocr4.jpg	1-ocrx.jpg	2-ocr.jpg	2-ocr5.jpg
					
6-ocrr.jpg	7-ocr4.jpg	7-ocrm.jpg	3-ocrq.jpg	3-ocrs.jpg	3-ocrx.jpg

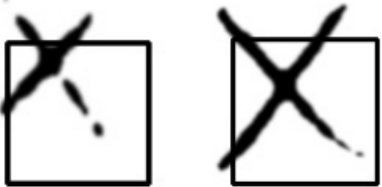

Fonte: Autor (2019)

A disposição das imagens positivas e negativas é representado através da Tabela. 03, e para a criação do arquivo *haarcascade* que foi utilizado no algoritmo

presente, foram necessárias 823 imagens negativas e 359 imagens positivas. Após realizar a aquisição da imagem é necessário convertê-las em imagens vetoriais para mapear os padrões presentes. O resultado do processo de *machine learning* é um arquivo no formato *XML* com as características, coordenadas e padrões do objeto especificado. O interessante desse arquivo é a possibilidade da lógica de reconhecimento do objeto funcionar em qualquer linguagem de programação que possui a compatibilidade com o *OpenCV*.

Para testar a funcionalidade do algoritmo, foi necessário mapear a área em que o classificador *haarcascade* capturou as coordenadas da imagem e desenhar um polígono em seu local de atuação. Foi utilizada a função *cv2.rectangle* para exibir um retângulo na área em que o algoritmo *haarcascade* detectou o objeto.

**Tabela 04 - Ilustração das imagens positivas e negativas**

Teste do classificador em imagens positivas	Teste do classificador em imagens negativas
	

Fonte: Autor (2019)

Observa-se através da Tabela. 04, que o classificador funcionou perfeitamente para encontrar a figura geométrica que foi especificada. Em outros testes realizado foram inseridos 3 tipos de gabaritos com todas as opções marcadas. Nesses testes o classificador conseguiu identificar todas as imagens corretamente, totalizando 100% de acertos.

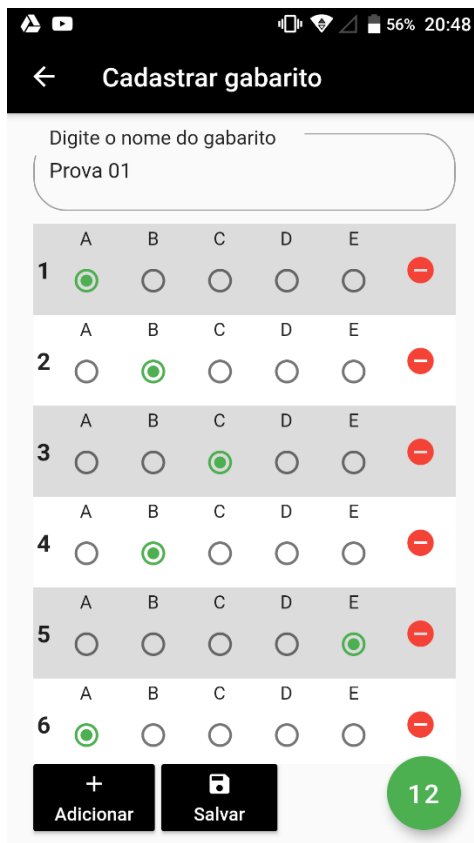
Após o reconhecimento do gabarito é necessário salvar a alternativa em um arquivo no formato *JSON* para o retorno das alternativas marcada pelo o usuário através do servidor *Flask*. Devido o algoritmo de processamento digital de imagens (PDI) executar na linguagem *Python* foi necessário criar um meio de comunicação

entre a parte de PDI com a parte mobile que é desenvolvida com a tecnologia *Flutter*. Para conseguir comunicar essas 2 tecnologias foi necessário criar uma *API* responsável por coletar a imagem a ser analisada e retornar um arquivo *JSON* para a aplicação mobile. A tecnologia *Flask* foi utilizado para realizar essa comunicação, devido ao fato da biblioteca gerar um servidor *WEB* que suporta os métodos *GET* e *POST*.

### 3.4 APLICATIVO

Para a criação do aplicativo foi necessário a utilização do framework *Flutter* e a linguagem de programação *Dart*. Primeiramente foi criado um módulo para cadastro das alternativas corretas que posteriormente são salvas em um arquivo *JSON* na memória do *Smart Phone*.

Figura 12 – Cadastro das alternativas



Digite o nome do gabarito

Prova 01

	A	B	C	D	E	
1	<input checked="" type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	-
2	<input type="radio"/>	<input checked="" type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	-
3	<input type="radio"/>	<input type="radio"/>	<input checked="" type="radio"/>	<input type="radio"/>	<input type="radio"/>	-
4	<input type="radio"/>	<input checked="" type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	-
5	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input checked="" type="radio"/>	-
6	<input checked="" type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	-

+ Adicionar

Salvar

12

Fonte: Autor (2019)

A figura 12 representa visualmente a tela de cadastro de questões do gabarito, essa janela possui as opções de inserção, leitura, exclusão e atualização de dados além das informações serem salvos permanentemente na memória do dispositivo.

Após o cadastro das alternativas é necessário capturar uma imagem de um gabarito para ser corrigida, este processo pode ser realizado dentro do aplicativo ou através da captura de uma imagem presente na galeria. Em relação a resolução da imagem não é necessário se preocupar com o tamanho da imagem, pois o algoritmo de processamento digital de imagem redimensionará a figura para que ela se adeque aos padrões estabelecidos pela lógica do algoritmo.

**Figura 13 – Resultado obtido**

Item	Resultado do sistema	Resultado aluno	Status
5	Resultado aluno: E		✗
6	Resultado do sistema: A Resultado aluno: D		✗
7	Resultado do sistema: B Resultado aluno: C		✗
8	Resultado do sistema: C Resultado aluno: B		✗
9	Resultado do sistema: D Resultado aluno: A		✗
10	Resultado do sistema: E Resultado aluno: C		✗
11	Resultado do sistema: D Resultado aluno: B		✗
12	Resultado do sistema: C Resultado aluno: C		✓

O aluno conseguiu obter 8.33% da nota.

Fonte: Autor (2019)

A Figura 13 representa o resultado final do processo de correção de gabarito, o arquivo *JSON* retornado pela *API* e o resultado foi demonstrado visualmente pelo software de correção de gabaritos. Na parte inferior da figura é possível visualizar a

nota do aluno e também é possível observar a nota cadastrada no sistema interno do aplicativo e a nota que a etapa de processamento digital de imagens retornou.



#### 4. CONCLUSÃO:

Com o aperfeiçoamento do projeto proposto é completamente viável a utilização deste software para correção de gabaritos. Devido ao framework *Flutter* executar de forma nativa nos aparelhos *Android* e *IOS* é possível executá-lo nesses sistemas operacionais distintos, além de permitir ao docente manusear esta ferramenta com facilidade e otimizar suas tarefas diárias.

Com a construção desse aplicativo foi necessário a utilização de diversas tecnologias e esse fato ajudou a moldar e aperfeiçoar o conhecimento adquirido ao longo desses anos, além de contribuir socialmente para melhorar o trabalho dos professores.

Foi percebido que há fatores que podem contribuir com correções imprecisas neste software, um exemplo é se a largura da borda do gabarito for de aproximadamente 1px, pois o sistema detecta com imprecisão alguns quadrantes. Para solucionar esse problema é necessário aumentar a borda para aproximadamente 4px.

O arquivo *haarcascade* criado teve um desempenho excelente, visto que os processos de pré-processamento e segmentação de imagens ajudam a eliminar ruídos para que o algoritmo de *Machine Learning* funcione perfeitamente.

É possível otimizar o sistema de correção de gabaritos com a utilização de processos que ajudem nas atividades dos docentes, não foi possível implementar o módulo de geração de relatórios em decorrência do tempo, porém toda a parte de reconhecimento e geração de resultados está funcionando completamente, sendo possível adicionar módulos futuramente.

O formato de gabarito suportado pelo sistema é o formato padrão oficializado pela instituição de ensino Unibalsas, é possível adicionar outros formatos alterando a parametrização e funções na linguagem de programação Python, isso permite a inserção de diferentes modelos futuramente e a utilização do software para correção de outros modelos de gabaritos institucional.

## 5. REFERÊNCIAS

BARBOSA, Alexandre. **Processamento digital de imagens para o Reconhecimento de placas de veículos.** Disponível em: <<https://www.unibalsas.edu.br/wp-content/uploads/2017/01/Alexandre.pdf>> Acesso em: 20 de maio de 2019.

BARELLI, Felipe. **Introdução à Visão Computacional Uma abordagem prática com Python e OpenCV.** Casa do Código. 2018.

BARTH, Vitor Bruno de Oliveira *et al.* **Desenvolvimento de um sistema web, Baseado no raspberry pi e node.js, para monitoramento e controle de periféricos.** Disponível em: <<http://brjd.com.br/index.php/BRJD/article/view/381/325>>. Acesso em: 21 de junho de 2019.

DE CAMPOS, Tatiane Jesus. **Reconhecimento de caracteres alfanuméricos de placas em imagens de veículos.** 2001. 120 p. Disponível em: <<https://lume.ufrgs.br/handle/10183/2329>> Acesso em: 20 de maio de 2019.

GONZALES, RAFAEL C. **Processamento digital de imagens.** 3 ed. São Paulo. Pearson Prentice Hall, 2010.

LINS, Luiz. **Reconhecimento ótico de caracteres (OCR) e análise de sistemas OCR baseados em código aberto.** Disponível em: <<http://www.fatecsp.br/dti/tcc/tcc00068.pdf>>. Acesso em: 21 de maio de 2019.

RIBEIRO, V. M.; RIBEIRO, V.M.; GUSMÃO, J.B. **Indicadores de qualidade para a mobilização da escola.** Disponível em: <<http://www.scielo.br/pdf/%0D/cp/v35n124/a1135124.pdf>>. Acesso em: 21 de junho de 2019.

SILVA, Bruno Ramon de Almeida. **Sistema de contagem automática de objetos utilizando processamento digital de imagens em dispositivos móveis.** Disponível em: <<https://ppgcc.ufersa.edu.br/wp-content/uploads/sites/42/2014/09/bruno-ramon-de-almeida-silva.pdf>>. Acesso em: 21 de maio de 2019.

ZENORINI, R. P. C.; SANTOS, A. A. A.; MONTEIRO, R. M. **Motivação para aprender: relação com o desempenho de estudantes.** Disponível em: <<https://www.revistas.usp.br/paideia/article/view/7278/8761>> Acesso em: 21 de maio de 2019.

## **ANEXO A – LISTA DE TABELAS**

### **LISTA DE TABELAS**

Tabela 1 - Representação da etapa de segmentação de imagens	7
Tabela 2 - Métodos para realce em imagens	18
Tabela 3 - Ilustração das imagens positivas e negativas	19
Tabela 4 - Resultado obtido após a aplicação do classificador	20

## **ANEXO B – LISTA DE FIGURAS**

### **LISTA DE FIGURAS**

Figura 1 - Representação de espectro eletromagnético de acordo com a energia do fóton	04
Figura 2 – Fluxo de um sistema baseado na visão computacional	04
Figura 3 – Fluxo geral do reconhecimento de gabaritos	10
Figura 4 – Processo de pré-processamento de imagem	11
Figura 5 – Transformação da imagem em monocromática	12
Figura 6 – Transformando uma imagem em binária	13
Figura 7 – Processo de segmentação de imagens	14
Figura 8 – Algoritmo para detecção de quadriláteros	15
Figura 9 – Algoritmo para extração de polígonos	16
Figura 10 – Resultado visual da segmentação de imagens	17
Figura 11 – Segmentação de imagens com as camadas reconhecidas	17
Figura 12 – Cadastro das alternativas	21
Figura 13 – Resultado obtido	22