EDGAR TAMIO HIRAMA

DESENVOLVIMENTO DE APLICAÇÃO ANDROID PARA RECONHECIMENTO ÓPTICO DE CARACTERES EM BRINCO DE IDENTIFICAÇÃO ANIMAL

Trabalho de Conclusão de Curso apresentado à
Escola de Engenharia de São Carlos, da
Universidade de São Paulo

Curso de Engenharia de Computação com ênfase
em Robótica

ORIENTADOR: Valdir Grassi Jr.

São Carlos

2014

Hirama, Edgar Tamio
Hd Desenvolvimento de aplicação Android para reconhecimento óptico de caracteres em brinco de identificação animal / Edgar Tamio Hirama; orientador Valdir Grassi Junior. São Carlos, 2014.

Monografia (Graduação em Engenharia de Computação) -- Escola de Engenharia de São Carlos da Universidade de São Paulo, 2014.

1. OCR. 2. Tesseract. 3. Visão Computacional. 4. OpenCV. 5. Android. I. Título.

FOLHA DE APROVAÇÃO

Nome: Edgar Tamio Hirama

Título: "Desenvolvimento de aplicação Android para reconhecimento óptico de caracteres em brinco de identificação animal"

Trabalho de Conclusão de Curso defendido em 21 / 11 / 2014.

Comissão Julgadora:	**Resultado:**
Prof. Dr. Valdir Grassi Júnior (Orientador) - SEL/EESC/USP	Aprovado
Prof. Associado Evandro Luís Linhari Rodrigues SEL/EESC/USP	Aprovado
Prof. Associado Alexandre Cláudio Botazzo Delbem SSC/ICMC/USP	APROVADO

Coordenador do Curso Interunidades - Engenharia de Computação:

Prof. Associado Evandro Luís Linhari Rodrigues

Aos meus pais, que sempre souberam lidar com momentos críticos da minha vida e me apoiaram incondicionalmente.

Agradecimentos

A Deus por ter me dado força e saúde para superar as dificuldades com resiliência e paciência.

Ao meu orientador Valdir Grassi Jr., que sempre foi muito prestativo e paciente na realização deste trabalho, e nunca desistiu de me ajudar.

A universidade e ao corpo docente, direção e administração por me proporcionarem um ambiente propício ao estudo e ao desenvolvimento técnico, profissional e pessoal.

Às secretárias Silvana e Jussara, pela excelente prestação de serviços sempre com muita atenção e carinho.

A meus pais, Tamio e Helena, que também tiveram muita paciência e compreensão durante minha graduação. Sem eles nada disso seria possível.

A meus amigos que sempre me apoiaram nos momentos difíceis e fizeram da minha graduação um momento muito agradável e memorável.

A minha segunda família em São Carlos, a Chapahall e meus amigos do Futebol CAASO, com os quais tive a oportunidade de ampliar meus horizontes dentro da graduação.

A EOS e AnimallTag por terem contribuído diretamente com a minha formação profissional e a confecção deste trabalho.

E a todos que direta ou indiretamente fizeram parte da minha formação, o meu muito obrigado.

Sumário

Lista de Figuras

Resumo

A aplicação de algoritmos de reconhecimento óptico de caracteres é cada vez mais ampla, devido à crescente necessidade de digitalizar informações. Neste trabalho, um pouco da história de como a área de OCR cresceu, como funcionam suas etapas, e alguns exemplos de aplicações OCR são brevemente estudados. Verifica-se também a importância da utilização de técnicas da visão computacional para eliminar ruídos e informações irrelevantes, e extrair da imagem as informações de interesse para a respectiva aplicação. O trabalho realizado é compreender o funcionamento do Tesseract, ferramenta própria para a execução de OCR, e aplica-la à imagem de um brinco utilizado para controle e identificação animal, após o devido pré-processamento da imagem para a extração dos 6 algarismos de interesse. Atualmente, a leitura automático do brinco dá-se com a conexão de um bastão leitor de RFID, que reveza a conexão com o *smartphone* com a balança. O objetivo é substituir o uso deste bastão com a leitura através da câmera do *smartphone*. A imagem da câmera é processada em tempo de execução utilizando-se o OpenCV para extrair a informação de interesse, e então esta imagem resultante é passada para o reconhecimento do Tesseract, que retorna um texto que é mostrado para a seleção através de 3 botões na interface, diminuindo o efeito do erro do reconhecimento sobre a experiência do usuário. O resultado foi satisfatório considerando uma amostragem pequena de testes, obtendo uma média de 87.2% de acerto, e obtendo o resultado certo em todas as vezes com o tempo de 2831.39 ms.

Palavras-chave: OCR, Tesseract, Visão Computacional, OpenCV, Android.

Abstract

The application of algorithms related to optical character recognition is developing everyday, due to the growing need of digitalize information. On this paper, the history of how OCR has grown, what are the steps to achieve it and some examples are briefly studied. It turns out that the importance of the use of computer vision is huge on this subject, either to eliminate noise and irrelevant information, or to extract important information from the image itself. One of the steps of the work is to understand how Tesseract – OCR engine – Works, and apply its methods to an ear tag for animal identification and control purposes, after properly preprocessing the image to extract the 6 numbers of interest. Today the automatic Reading of the ear tag is done by an RFID reader, which alternates its smartphone bluetooth connection with the balance. The goal is to replace this RFID reader with the OCR Reading through the phone camera. The camera image is processed in execution time by using OpenCV to extract the infomation of interest, then the resulting image is put to Tesseract recognition, which returns a text shown in 3 buttons in the user interface, lowering the effects in the user experience caused by recognition error. The result was satisfactory considering a small sampling, in which it was able to get a 87.2% average hit, and getting the right result in all times with an average time of 2831.39 ms.

Palavras-chave: OCR, Tesseract, Computer Vision, OpenCV, Android

1. Introdução

As técnicas de visão computacional estão cada vez mais inseridas no mercado de aplicativos para *smartphones*: detecção facial, reconhecimento de movimentos, entre outras aplicações, com o objetivo de melhorar a interação com o usuário. Enquanto isso, essas mesmas técnicas começam a se tornar populares em aplicações comerciais.

Uma das áreas comerciais que pode se beneficiar na aplicação da visão computacional é a pecuária. Para o controle de gado, é comum o uso de brincos contendo uma inscrição numérica para identificação. Dessa forma, técnicas de visão computacional podem ser utilizadas para leitura automática dessa identificação numérica, e assim, ferramentas computacionais podem ser desenvolvidas para os pecuaristas que desejam se modernizar.

Atualmente há um aplicativo, o Sisgado, desenvolvido pela EOS Tecnologia de Informação, em conjunto com a empresa do ramo de identificação animal, a AnimallTag. Este aplicativo permite ao dono de rebanhos manter controle de seus animais através das identificações impressas nos brincos supracitados. Através de comunicação *bluetooth* com a balança é possível registrar o peso do animal, o dispositivo *bluetooth* encontra-se na **Figura 1.1** . Além do peso, é possível registrar outros dados, que ficam guardados no banco de dados do *smartphone*, como o preço por quilo, raça, entre outros, assim como ilustrado na **Figura 1.2.** Outro uso para o *bluetooth* é a comunicação com o bastão leitor de frequência RF, ilustrado na **Figura 1.3**, que lê os dados do brinco – quando equipado com o dispositivo eletrônico – e o transmite para o aplicativo, no caso a identificação numérica. Um problema enfrentado pela aplicação é a troca constante de conexão entre a balança e o bastão, uma vez que a aplicação não suporta conexão simultânea com os dois dispositivos.

Figura 1.1 - Dispositivo *bluetooth* da balança

Figura 1.2 – Exemplo de execução do aplicativo Sisgado.

Figura 1.3 - Bastão RFID para controle de gado bovino

O objetivo deste trabalho é justamente fazer o reconhecimento dessas inscrições numéricas em brincos de identificação de gado utilizando um *smartphone* com sistema operacional Android[1], e tal funcionalidade pode ser inserida no aplicativo que já se encontra em produção, onde hoje é utilizado a leitura RF do brinco eletrônico, permitindo que o aplicativo esteja sempre conectado somente com a balança, e evitando a conexão alternada entre os dispositivos.

A quantidade de informações impressas no brinco pode variar, e o objetivo de estudo será um brinco com quantidade considerada alta de detalhes, 3 (três) linhas menores – duas de texto e número, e uma de código de barras –, uma com a inscrição numérica de interesse, seguida de outra linha numérica. Uma ilustração do brinco a ser estudado encontra-se na **Figura 1.4**.

Figura 1.4 - Brinco de identificação animal.

O processo de identificação e leitura de placas de automóveis é de certa forma semelhante a aplicação que se deseja tratar neste projeto. Na identificação de placas de automóveis, primeiramente é preciso fazer o pré-processamento da imagem do veículo para se encontrar a posição da placa, e então, aplicar o algoritmo de reconhecimento dos caracteres da placa para que seja feita sua leitura. De forma semelhante, pretende-se fazer o pré-processamento da imagem do brinco para identificar a posição dos caracteres, para que estes possam ser reconhecidos.

A biblioteca OpenCV, que será discutida no **Capítulo 2,** tem sido amplamente utilizada para desenvolvimento de aplicações na área de visão computacional, e possui funções básicas de processamento de imagens já implementadas. É possível utilizar a biblioteca para implementações no sistema operacional Android, e por isso será utilizada neste trabalho como base para a etapa de pré-processamento das imagens obtidas para localização do brinco de identificação do gado.

Uma vez localizado o brinco na imagem, será utilizado um algoritmo de reconhecimento de caracteres na imagem, ou seja, um algoritmo de OCR. Existem algumas bibliotecas que utilizam redes neurais e bases treinadas para reconhecer caracteres, uma dessas bibliotecas é o Tesseract, que será discutido no **Capítulo 2.** O Tesseract possui suporte para Android, o que ainda não é comum entre as ferramentas OCR. Além disso, o Tesseract é *open source* e é mantido pela Google. Por essa razão, essa ferramenta foi escolhida como solução de OCR para a aplicação desenvolvida neste trabalho.

Portanto, este trabalho mostra o desenvolvimento de uma aplicação para reconhecimento de caracteres em brincos de identificação de gado, utilizando uma etapa de pré-processamento implementada a partir de funções do OpenCV seguido do uso do Tesseract para reconhecimento de caracteres. Os resultados obtidos mostram que a etapa de pré-processamento para identificação da posição dos caracteres é essencial para melhorar o desempenho do Tesseract no reconhecimento de caracteres.

2. Embasamento teórico

2.1 OCR

Segundo Bhaskar et al.(2010), o algoritmo de reconhecimento óptico de caracteres (*Optical Character Recognition* – OCR) é a tradução, eletrônica ou mecânica, de imagens escaneadas de documentos escritos à mão, datilografados em máquinas de escrever ou impressos, para textos codificados em máquinas. OCR é estudado e desenvolvido a mais de 80 anos, quando a primeira patente para uma máquina OCR foi registrada em nome do alemão Gustav Tauschek, em 1929. Já de acordo com Mori et al.(1992), nos primórdios do estudo de reconhecimento de padrões, grande parte da comunidade científica optou por OCR, pois, segundo expectativas, os caracteres eram provavelmente fáceis de lidar, e, logo, não seria difícil desenvolver um algoritmo para resolver o problema de reconhecimento. Porém, após um início de fácil desenvolvimento nesta área, grandes dificuldades na resolução destes problemas foram encontradas, tais como as causadas por diferenças nas fontes escritas ou então na má qualidade de impressão devido a texturas de diferentes folhas de papel. O desenvolvimento da indústria e a determinações de padrões de fontes tiveram relação direta com o desenvolvimento das técnicas de OCR (CHERIET et al, 2007).

Atualmente, OCR é aplicado em muitas áreas, desde serviços postais, traduções de línguas, e bibliotecas digitais. Há alguns aplicativos móveis que já utilizam a tecnologia OCR disponíveis a usuários. A maioria dos *softwares* disponíveis no mercado não é *open source* (PATEL et al, 2012).

2.1.1 Etapas do OCR

O desenvolvimento do OCR passa pelas etapas de pré-processamento, extração de características e reconhecimento em si. A seguir são descritas algumas delas:

A) Pré-processamento

Segundo Jain et al.(2013), a imagem obtida pela câmera não é apropriada para a tarefa de reconhecimento, e, por isso, deve ser destacada, isto é, o texto deve ter contraste em relação ao fundo da imagem (preto no branco), projetado para um tamanho uniforme, e, sempre que possível, reforçar os caracteres quanto aos seus ângulos agudos. As etapas mais básicas são **redução de ruído**, **rotação** e **correção de perspectiva, binarização**, **extração de características** e **reconhecimento**.

Comparado a *scanners* ópticos, câmeras digitais são inerentemente mais suscetíveis a ruído devido ao seu modo de operação, portanto, na **redução de ruído** devem-se aplicar filtros para que pontos irrelevantes da imagem sejam desconsiderados, e que não atrapalhem nas etapas seguintes do processamento.

Distorção de perspectiva em imagens usando câmeras é comum, o que torna sua correção de suma importância em aplicações OCR. A correção dessas distorções através da **rotação e correção de perspectiva** é importante porque elas têm efeito direto na confiabilidade e eficiência das etapas de segmentação e de extração de características.

Binarização é o processo de converter a imagem em cores ou em escala de cinza para uma imagem de dois níveis. Cada pixel deve ser analisado e classificado como pixel de fundo ou pixel principal. Limiarização é a forma mais popular de realizar essa tarefa.

Na etapa de **extração de características**, cada possível caractere é representado como um vetor de características, que se torna sua identidade. Durante a fase de treinamento, vetores de características de caracteres são utilizados como modelos cuja finalidade é realizar a comparação com os vetores de características encontrados nas imagens adquiridas.

A fase de **reconhecimento** inclui o algoritmo de reconhecimento em si, cada imagem de caractere encontrado deve ser convertida para um código de caractere. As equivalências encontradas entre as características adquiridas da imagem e as características dos modelos são feitas e discriminadas segundo o código do caractere.

2.2 Escolha da solução

Após a pesquisa realizada nos trabalhos supracitados, a opção mais clara era utilizar Tesseract, pois é a ferramenta *open source* de OCR mais precisa atualmente, de acordo com Bhaskar et al.(2012), e tem tido bons resultados. Para o pré-processamento, foi escolhida a ferramenta OpenCV para Android, através da SDK OpenCV4Android.

2.3 OpenCV

OpenCV é uma biblioteca *open source* de visão computacional escrita nas linguagens C e C++, podendo fazer uso de multiprocessadores, possui suporte para os sistemas operacionais Linux, Windows e Mac OS X, e mais recentemente as plataformas móveis Android e iOS, e foi projetada para ser eficiente em termos computacionais e com foco em aplicações de processamento em tempo de execução(BRADSKY, G; KAEHLER,A.,2008).

Um dos objetivos desta biblioteca é disponibilizar uma infraestrutura simples de se utilizar e que possibilite o desenvolvimento de aplicações visuais um tanto quanto sofisticadas. Possui mais de 500 (quinhentas) funções as quais abrangem as mais diversas áreas em visão, incluindo imagens médicas, segurança, interface de usuário, calibração de câmera e robótica.

2.3.1 História

A origem do OpenCV dá-se à uma iniciativa da Intel em avançar nas aplicações com uso intensivo de CPU. A partir de uma pesquisa baseada neste assunto, descobriu-se que alguns alunos de universidades renomadas, como o *MIT Media Lab*, haviam desenvolvido um código aberto bem estruturado que era passado de estudante para estudante, fazendo reuso do código e evitando a reinvenção de código de funções básicas de visão computacional.

O projeto começou com membros do time de bibliotecas do laboratório da Intel e a colaboração do grupo "*Software Performance Libraries"*(Bibliotecas de performance de *software*) e seus experts na implementação e otimização dos algoritmos na Rússia.

O chefe dentre a equipe russa era Vadim Pisarevsky, que gerenciou, implementou e otimizou grande parte do OpenCV. Além dele, Victor Eruhimov ajudou a desenvolver a infraestrutura inicial e Valery Kuriakin gerenciava o laboratório russo e apoiou muito o esforço dos cientistas.

Atualmente algumas companhias como Google, Yahoo, Microsoft, IBM, Sony, Honda, Toyota e a própria Intel fazem uso da biblioteca, e conta com uma comunidade de usuários de mais de 47,000 pessoas, e um número de *downloads* estimado que excede 7 milhões(OPENCV, 2014).

2.3.2 Teoria

A) Mat

Principal representação de uma imagem dentro do OpenCV, a Mat é um objeto que substitui o antigo IplImage do OpenCV implementado em C. Com este objeto, já em C++ não era necessária mais a alocação manual da memória para a imagem, além disso, há o reuso do objeto, caso necessário.

A Mat basicamente é formada pelo cabeçalho, que contém informações como o seu tamanho, endereço, tipo, número de *bits*, entre outras, e um ponteiro para a matriz contendo os valores dos *pixels*. Enquanto que o tamanho do cabeçalho é constante, o tamanho da matriz pode variar de imagem para imagem.

Como o processamento de imagens já é de alto custo computacional, e é comum passar objetos do tipo Mat para as funções, para evitar a cópia de matrizes de grande quantidade de dados, a atribuição deste objeto é feita copiando-se apenas o cabeçalho e o ponteiro para a matriz em questão, logo, se o objetivo é apenas copiar a matriz e manter a original, deve-se haver a preocupação em utilizar métodos como o o Mat.clone() e Mat.copyTo().

B) Espaços de cores

Uma das informações contidas no cabeçalho do objeto Mat é o espaço de cores em que a matriz é definida. O espaço de cores é a definição de como serão separados os componentes que formam uma cor. O mais simples deles é o *grayscale* em que os componentes possuem apenas *pixels* brancos e pretos, formando então imagens em tons de cinza. O mais popular, porém, é o RGB, principalmente por ser esse espaço de cores que compõe a visão do ser humano. Além desses espaços de cores, há muitos outros como o YCrCb, que é utilizado em imagens do formato .JPEG, o CIE L*a*b que é perceptivamente uniforme, que é útil para medir a distância entre duas cores.

Por fim, há o HSV – ou HSL – que é composto por matiz (*hue*), saturação (*saturation*) e valor/iluminação (*value/luminance*), que permite, por exemplo, que se descarte o último componente e torne o algoritmo menos sensível a iluminação.

2.3.3 Funções

Para realizar o processamento de imagens, é necessário utilizar o uso da classe ImgProc, que possui as funções apropriadas para manipulação das matrizes que representam as imagens. A seguir estão descritas algumas dessas funções.

A) PyrDown

Pirâmides de imagens são muito utilizadas em processamento de imagens para realizar a diminuição ou o aumento na resolução da imagem.

No caso da função ***pyrDown***, a intenção é diminuir as dimensões da imagem realizando uma convolução gaussiana para obter uma imagem de dimensões menores e mais simplificada. A convolução é realizada utilizando a seguinte a máscara:

$$\frac{1}{256}\begin{bmatrix}1 & 4 & 6 & 4 & 1\\ 4 & 16 & 24 & 16 & 4\\ 6 & 24 & 36 & 24 & 6\\ 4 & 16 & 24 & 16 & 4\\ 1 & 4 & 6 & 4 & 1\end{bmatrix}$$

Após a convolução, as linhas e colunas pares são excluídas, obtendo então uma imagem de menor resolução.

B) CvtColor

Realiza a conversão do espaço de cores, por exemplo de RGB para HSV. O ultimo parâmetro é a operação de conversão utilizada.

Por motivos de simplificação, seguem apenas as conversões utilizadas no trabalho:

RGB – Gray:

Sendo um pixel formado por 3 canais(R, G e B) no caso de RGB, e um apenas (Y) em uma imagem em Grayscale, podemos considerar a seguinte transformação:

Y = 0.299*R + 0.587*G + 0.114*B.

RGB – HSV:

No caso de imagens 8-bit e 16-bit, os canais R, G e B são transformados para ponto flutuante e reduzidos a uma escala de 0 a 1.

$$V \leftarrow \max(R, G, B)$$

$$S \leftarrow \begin{cases} \dfrac{V - \min(R, G, B)}{V} & se\ V \neq 0 \\ 0 & caso\ contrário. \end{cases}$$

$$H \leftarrow \begin{cases} 60 * \dfrac{(G-B)}{V - \min(R,G,B)} se\ V = R \\ 120 + 60 * \dfrac{(B-R)}{V - \min(R,G,B)} se\ V = G \\ 240 + 60 * \dfrac{R-G}{V - \min(R,G,B)} se\ V = B \end{cases}$$

Se $H < 0$, então $H \leftarrow H + 360$. O que nos daria algo como $0 \leq V \leq 1, 0 \leq S \leq 1, 0 \leq H \leq 360$.

A partir disso, os dados seriam convertidos de acordo com o seguinte cálculo, para imagens de 8-bits:

$$V \leftarrow 255 * V$$

$$S \leftarrow 255 * S$$

$$H \leftarrow H/2 (para\ ficar\ no\ intervalo\ de\ 0\ a\ 255)$$

C) InRangeS

Determina, em uma imagem, os pixels cuja faixa de valores, referente ao seu espaço de cores correspondente, encontra-se dentro do intervalo passado como parâmetro. Caso a imagem tenha mais de um canal eles serão tratados separadamente. Os parâmetros **src** e **dst** devem ser do mesmo tamanho, ter o mesmo número de canais e serem ambas imagens de 8-bits.

D) AdaptativeThreshold

O ***thresholding*** é uma técnica utilizada para obter imagens binarizadas, através de um limiar definido a partir do momento em que quer se descartar os *pixels* abaixo ou acima de um valor. O ***thresholding*** **adaptativo** este valor é variável, de acordo com a região à qual o pixel a ser considerado pertence.

E) FindContours

Busca contornos na imagem segmentada como paramêtro. Essa deve ter 8-bits e possuir somente um canal. Ela retorna o numero total de contornos encontrados na imagem.

F) DrawContours

Desenha contornos na imagem passada como paramêtro. Os contornos podem ser controlados utilizando os parametros do metodo, como por exemplo o tipo da linha e sua espessura.

2.4 Tesseract

Tesseract, segundo Smith(2007) é uma ferramenta utilizada para reconhecimento de caracteres, criada em 1984 como um projeto de PhD no HP Labs, desenvolvida entre 1984 e 1994. Foi apresentada no Teste Anual de Precisão em OCR da UNLV em 1995, no qual foi bem sucedida e obteve resultado similar ou melhor que ferramentas comerciais da época. Em 2005 foi liberada como *Open Source* e atualmente o projeto encontra-se em domínio da empresa Google, cujos interesses com a ferramenta não foram levados a público.

2.4.1 Descrição do algoritmo

A primeira parte do Tesseract consiste em obter possíveis palavras a partir da imagem de entrada, de acordo com os passos do diagrama da **Figura 2.1.**

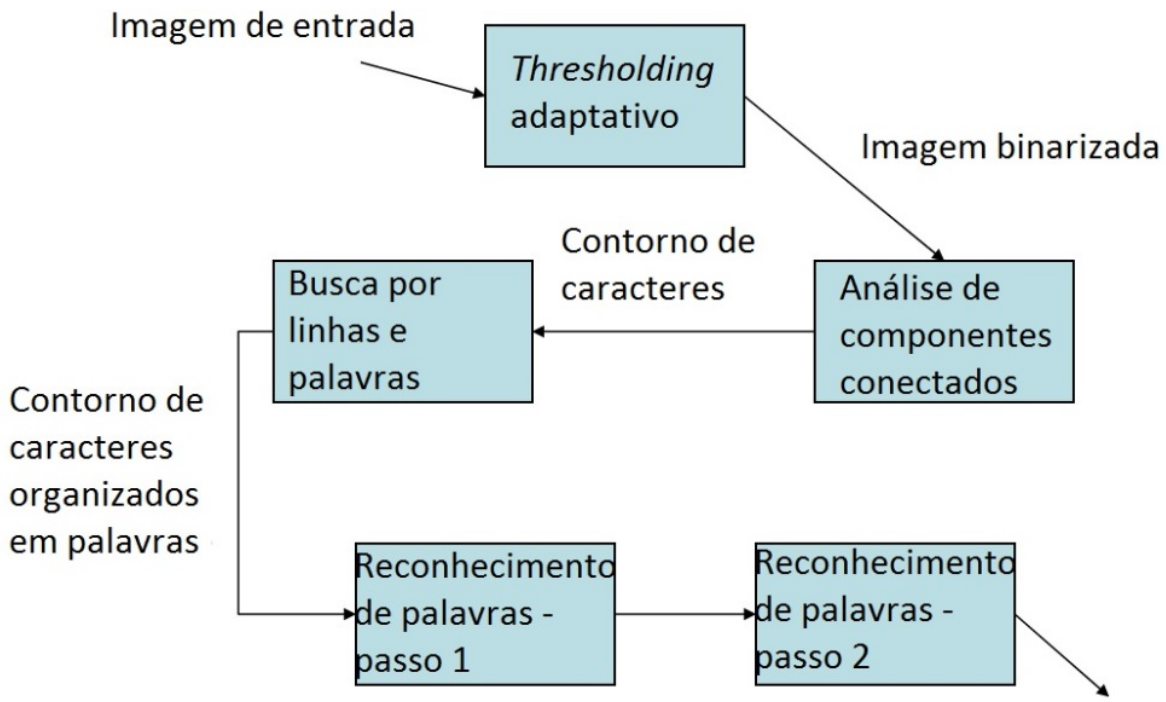

Figura 2.1 - Diagrama de fluxo do algoritmo do Tesseract. Extraída e traduzida de (SMITH,2007)

Primeiro, uma imagem – colorida ou tons de cinza – é dada como entrada para o algoritmo. A imagem é binarizada através de um ***thresholding* adaptativo**, chamado médodo de Otsu (OTSU, 1975), para evitar danos feitos por irregularidade na iluminação, vide diferença entre o resultado do *thresholding* normal, mostrado na **Figura 2.2** e o resultado do *thresholding* adaptativo, mostrado na **Figura 2.3**.

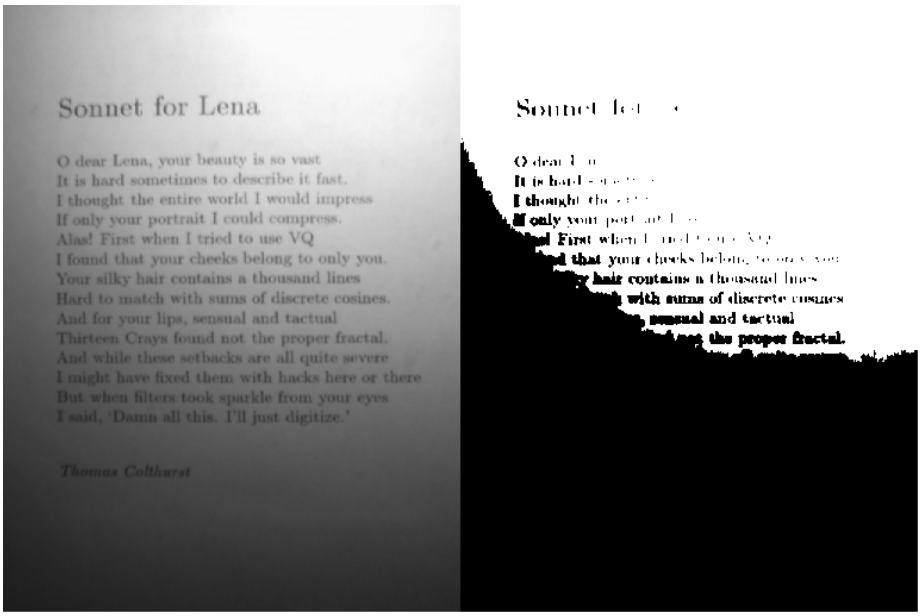

Figura 2.2 – Aplicação de *Thresholding* normal

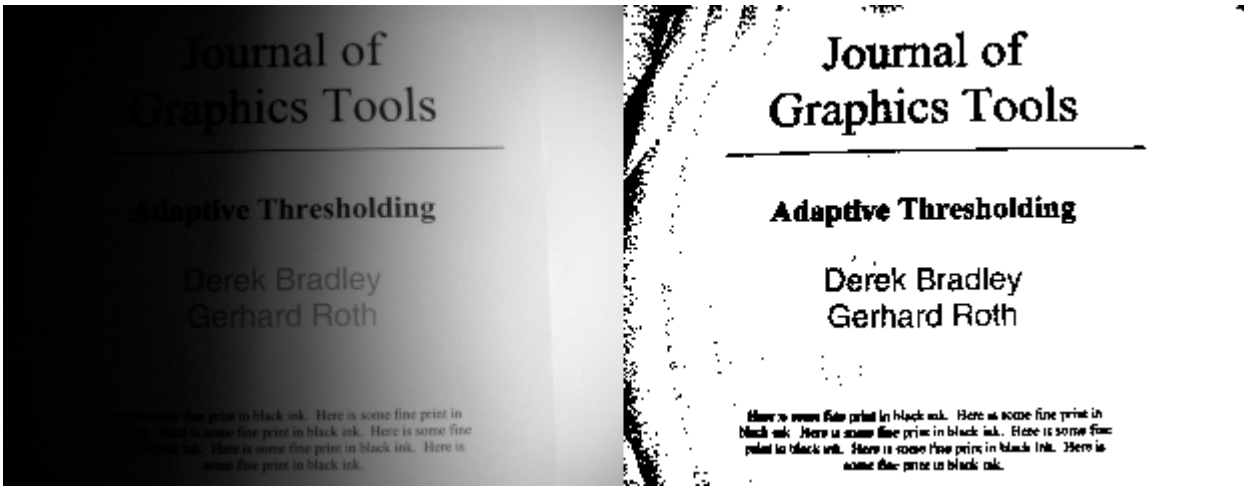

Figura 2.3 – Aplicação de *Thresholding* adaptativo

A imagem binária, então, é utilizada para a realização de uma **análise de componentes conectados**, na qual é feita a comparação de um pixel central com os seus vizinhos, definindo os componentes que estão conectados. Tal análise é importante porque o reconhecimento de textos independe da classe de pixel do fundo e do texto (preto no branco e vice-versa). Os componentes obtidos são organizados em blocos, que por sua vez serão organizados entre as linhas do texto no próximo passo.

A **busca por linhas e palavras** foi feita de tal modo que não fosse necessário rotacionar a página, em casos que a mesma não esteja perfeitamente posicionada, para que não haja perda de qualidade na imagem. Após uma prévia análise da página, as regiões de texto encontradas apresentam uma certa uniformidade em sua altura, sendo possível, então, a eliminação de caracteres de linhas diferentes que se tocam verticalmente, e também manchas menores que a altura média, como pontuação ou ruído. Após essa filtragem, as regiões resultantes, chamadas de *blobs*, devem estar dispostas em linhas inclinadas. Os

blobs são ordenados de acordo com a sua coordenada ‘x’, o que torna possível a associação a uma única linha do texto, enquanto registra a inclinação da página, diminuindo consideravelmente o risco de associar o *blob* a uma linha de texto incorretamente. Assim que os *blobs* são associados às linhas, as retas de orientação das linhas são traçadas com o auxílio do método dos mínimos quadrados. Para consolidar a linha de texto, os *blobs* que se sobrepõem horizontalmente por pelo menos metade de seu comprimento são mesclados, colocando pontuação diacrítica na base correta e associando partes de caracteres partidos. Uma vez que as linhas do texto já foram encontradas, a precisão das retas de orientação das linhas é aperfeiçoada através do método de interpolação de *spline* quadrática. Os parâmetros utilizados para tal método são as diferenças contínuas apresentadas pelos *blobs* em relação à primeira reta de orientação. As curvas traçadas são comparadas e a mais populosa é candidata para ser a nova reta de orientação, e é modelada para tal utilizando novamente o método dos mínimos quadrados. A **Figura 3.4** mostra um exemplo de linha de texto com a reta de orientação e as curvas traçadas pelo método.

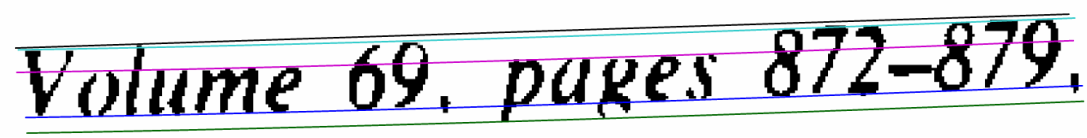

Figura 2.4 - Representação das linhas de *baseline, mean line, ascending line e descender line.* Extraída de (SMITH, 2007)

No **reconhecimento de palavras**, as linhas de texto são separadas em duas categorias: *fixed pitch* e *non-fixed pitch*.

As linhas que se encaixam na categoria *fixed pitch* são as que apresentam o espaçamento entre os caracteres com uma determinada frequência. Nestas linhas, as palavras são quebradas em caracteres de acordo com a frequência encontrada, e estas linhas não passam pelas etapas de corte e associação de caracteres na fase de reconhecimento de palavras.

O espaçamento das linhas que não se encaixam na categoria de *fixed pitch* dificulta a separação dos caracteres e, consequentemente, o reconhecimento das palavras. A **Figura 2.5** ilustra alguns dos problemas típicos desse tipo de espaçamento. O espaço entre dezena e unidade de ‘11,9’ é similar ao espaço entre as palavras do texto, e com certeza maior que o espaço entre o final da palavra ‘Federated’ e o começo da palavra ‘junk’. Já entre as palavras ‘of’ e ‘financial’ não há espaço algum. O Tesseract resolve esse tipo de problema analisando os espaços em uma variação vertical limitada pela *baseline* e a *meanline*. Os espaços próximos do limiar do espaçamento são colocados em estado indeterminado, para que uma decisão seja feita após a fase de reconhecimento de palavras. As linhas que possuem esses problemas de espaçamento são reconhecidas de acordo com o diagrama da **Figura 2.6,** que é executado enquanto o resultado da palavra é insatisfatório**.**

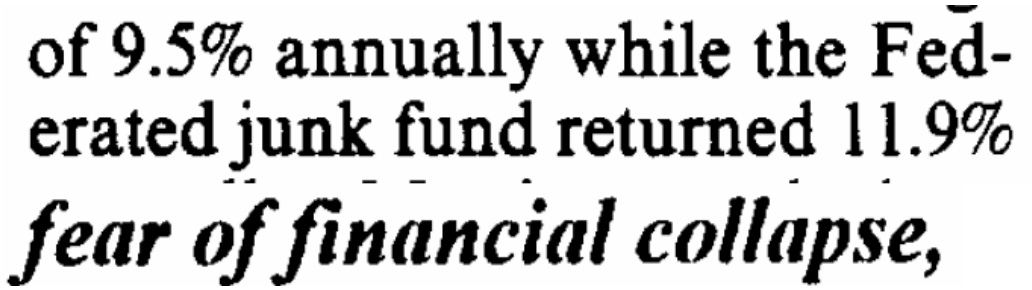

Figura 2.5 – Problemas comumente encontrados de espaçamento. Extraída de (SMITH, 2007)

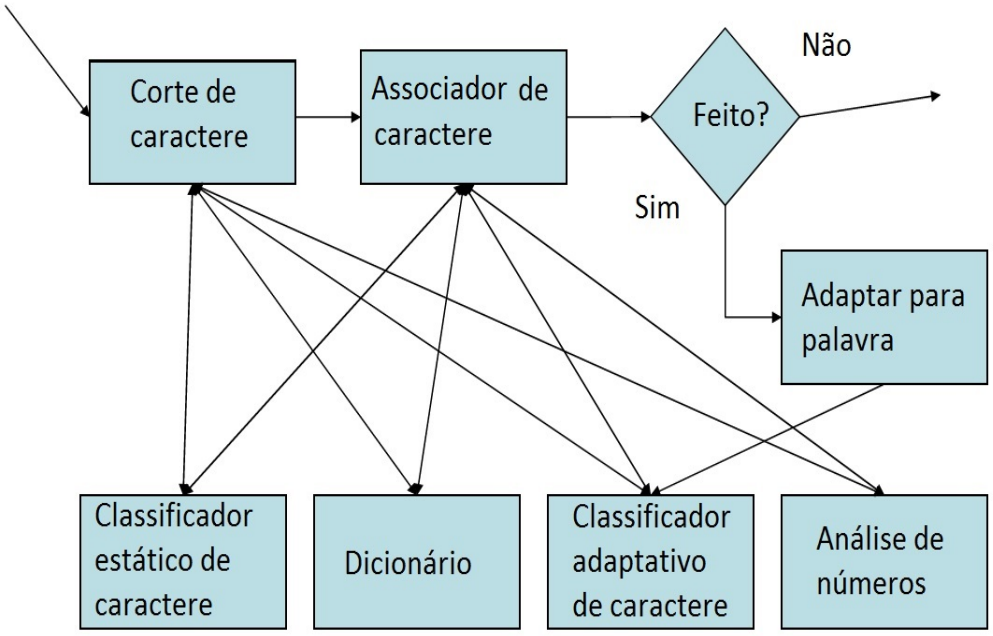

Figura 2.6 – Diagrama de execução do Tesseract para linhas com problemas de espaçamento. Extraída de (SMITH, 2007)

O **corte de caractere** é feito no *blob* de menor confiança do classificador de caractere. Os pontos candidatos ao corte são definidos nos vértices côncavos de uma aproximação poligonal do contorno, e pode ter tanto um vértice côncavo oposto, quanto um segmento de linha. Pode levar até 3 pares de pontos de corte para obter um caractere do conjunto ASCII com sucesso. A **Figura 2.7** mostra, com setas, os pontos candidatos ao corte, e o corte selecionado no encontro das letras 'r' e 'm'. Os cortes são executados de acordo com uma ordem de prioridade, e qualquer corte que não melhore a confiança do resultado é desfeito, mas não descartado, para que possa ser reutilizado na etapa de associação de caractere se necessário.

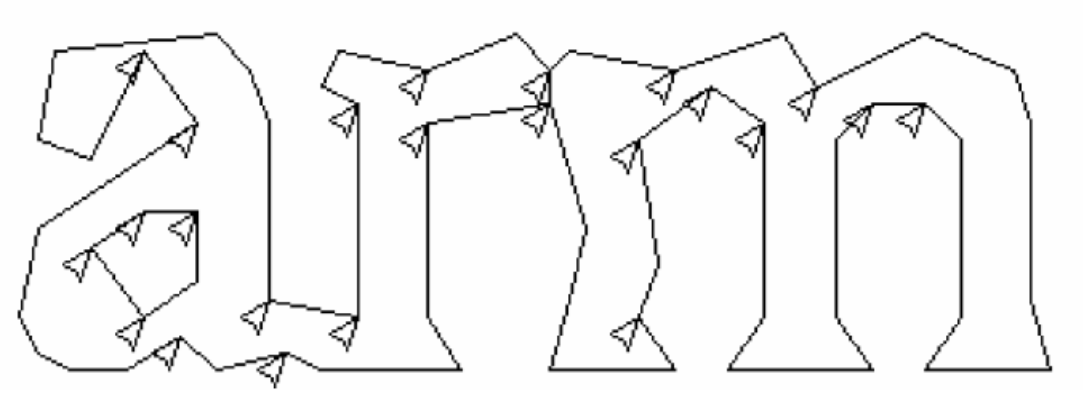

Figura 2.7 - Pontos candidatos ao corte. Extraída de (SMITH,2007)

Após os cortes de caractere serem feitos, se o resultado da palavra ainda não apresenta confiança satisfatória, passa a ser analisada pelo **associador de caractere.** Através de uma busca A* no grafo de segmentação de possíveis combinações dos *blobs* cortados em candidatos a caractere, sem de fato construir o grafo, mas apenas mantendo uma tabela *hash* dos estados visitados. A busca A* avança pegando os novos estados de uma pilha de prioridades e avaliando-os ao classificar novas combinações de fragmentos. O sucesso da técnica está em facilmente reconhecer caracteres "quebrados", como na **Figura 2.8**.

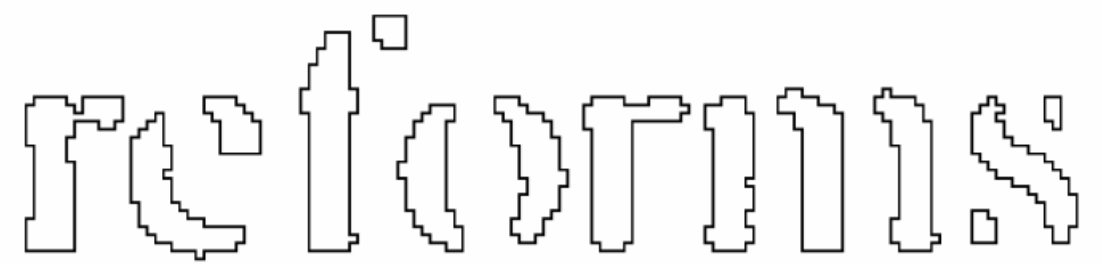

Figura 2.8 - Palavra com caracteres "quebrados". Extraída de (SMITH, 2007)

2.5 Exemplos de aplicações OCR

Após esta breve explicação das etapas que constituem o reconhecimento de caracteres, seguem algumas aplicações e como foram feitas as suas respectivas etapas do processamento:

2.5.1 Reconhecimento de placas veiculares (LPR)

LPR, assim como descreve Anagnostopoulos et al.(2006) – *License Plate Recognition* – é um campo bastante desenvolvido na área de reconhecimento de caracteres, utilizado para controlar tráfego e sistemas de pagamento automático de pedágios ou estacionamentos, por exemplo.

No pré-processamento, algumas aplicações utilizam filtros de Sobel, vide (CHANG et al, 2004), outras aplicam a limiarização mais de uma vez, seguindo parâmetros convenientes para a respectiva aplicação, como feito em Romic et al.(2012).

No primeiro caso, restam algumas possíveis regiões as quais podem ser a placa propriamente dita. Já no segundo, a detecção é feita através da maior concentração de pixels brancos na vertical, sendo possível determinar a localização dos cantos esquerdo e direito da placa. Há ainda uma técnica aplicada (ANAGNOSTOPOULOS et al, 2006) que se refere à detecção e contagem de bordas na linha de pixels. Se o número de bordas passar de um dado limiar de separação, significa a presença da placa. Em caso de insucesso, o limiar é modificado para baixo.

No exemplo ilustrado na **Figura 2.9** são utilizadas duas TDNN – *time delay neural network* – redes neurais. Uma vez localizada a placa, é feita a segmentação dos caracteres nela contidos, e o reconhecimento de cada um deles. O fluxograma deste procedimento está ilustrado na **Figura 2.10** e as etapas são mostradas na **Figura 2.11**.

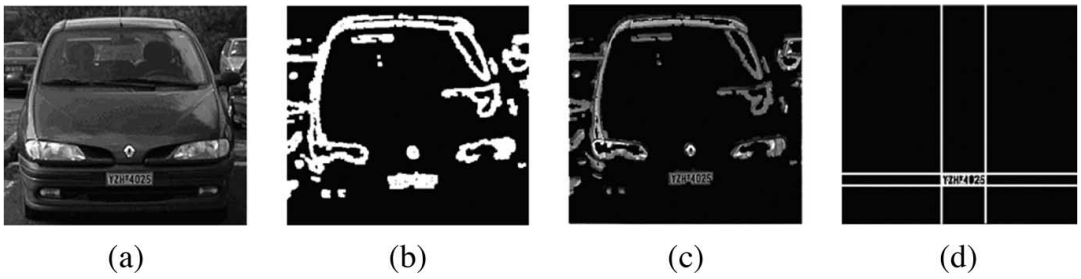

Figura 2.9 - (a) Imagem em tons de cinza obtida. (b) Aplicação de thresholding após segmentação. (c) Aplicação de máscara na figura (a). (d) Detecção da placa. Extraída de (ANAGNOSTOPOULOS et al, 2006)

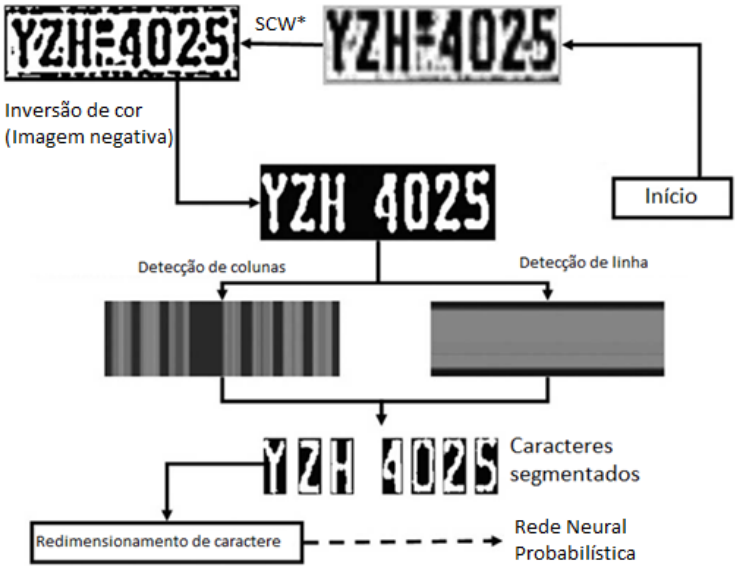

Figura 2.10 - Diagrama de fluxo da segmentação e reconhecimento de caractere. Extraída e traduzida de (ANAGNOSTOPOULOS et al, 2006) *SCW – *Sliding concentric Windows* – técnica de segmentação

Figura 2.11 - Segmentação bem-sucedida dos caracteres da placa veicular. Extraída de (ANAGNOSTOPOULOS et al, 2006)

2.5.2 Reconhecimento de *Business Card*

Um projeto na Universidade de Stanford de (BHASKAR; LAVASSAR; GREEN, 2010) teve como propósito a identificação e leitura de cartões de contatos profissionais, também conhecido como *business card*, mostrado na **Figura 2.12**.

O projeto foi implementado em Android, mas teve parte do processo realizada em MATLAB, através de comunicação com um servidor *online*.

O processo consistia em realizar um *thresholding* adaptado, e então eram aplicados os filtros de operação morfológica. Da imagem binarizada obtida, mostrada na **Figura 2.13,** são detectadas as bordas e cantos do cartão através da diferença de entre as imagens dilatadas e erodidas, e posterior aplicação da transformada de Hough e intersecção das quatro linhas encontradas. O último passo do pré-processamento era a transformação da perspectiva, utilizando a proporção de um *business card* padrão dos EUA e fazer a operação usando ainda o MATLAB, o resultado encontra-se na **Figura 2.14**. O *thresholding* adaptado era novamente aplicado apenas nos limites do retângulo encontrado nas etapas anteriores. O reconhecimento foi feito utilizando o Tesseract.

Figura 2.12 - Imagem original de um cartão de negócios padrão.

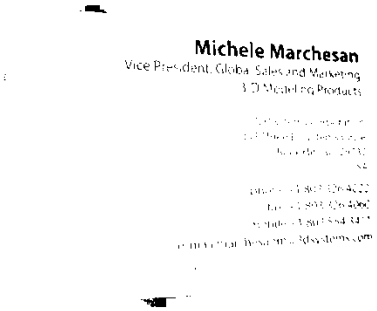

Figura 2.13 - Imagem após pré-processamento da Figura 2.12

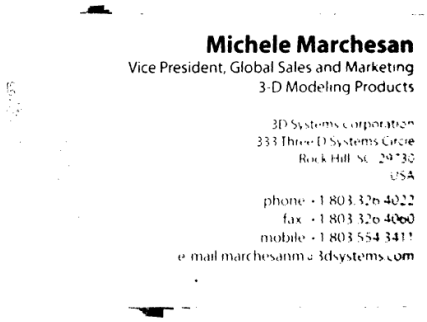

Figura 2.14 – Imagem após correção de perspectiva da Figura 2.13

2.5.3 Reconhecimento de bilhetes de loteria

Outro projeto realizado na Universidade de Stanford, no qual um *smartphone* foi utilizado para detectar informações importantes do bilhete de loteria, ilustrado na **Figura 2.15**, para que a conferência dos resultados torne-se mais fácil, para quem joga com muitos bilhetes por sorteio (GUPTA, A.; ZOU, T., 2012). O diagrama de fluxo do reconhecimento das informações de interesse encontra-se na **Figura 2.16.**

Neste caso em particular, viu-se que a componente vermelha da imagem continha as informações importantes e "limpava" a imagem de fundo, com as marcas d'água, inclusive. A partir da imagem contendo apenas o componente 'R' da imagem RGB, ilustrada na **Figura 2.17**, foi feito o *threshold* utilizando um filtro de média. O resultado desta etapa encontra-se na **Figura 2.18**.

A imagem binária foi então invertida, e foi aplicada a rotulagem de regiões (ou *region labeling*), e filtrada por área para remoção de pequenas regiões. Através da dilatação dessa imagem, com o objetivo de eliminar o espaço entre a estrutura do código de barras, foi possível detectá-lo, como é possível observar na **Figura 2.19**.

Com o código de barras em mãos, foi possível determinar as coordenadas dos quatro cantos do mesmo através da transformada de Hough, método de detecção de características (BALLARD,1981), com processo semelhante à aplicação anterior, e tais coordenadas foram utilizadas para realizar a correção da distorção de perspectiva, rotação e escalonamento, . A mesma transformação resultante deste processo foi aplicada na imagem binária e na imagem que contém apenas a componente 'R', o resultado encontra-se na **Figura 2.20**.

Então, a localização das colunas de números e da data é feita, como mostrado na **Figura 2.21**, e os dados são extraídos e passados para o Tesseract.

Figura 2.15- Imagem original do bilhete de loteria

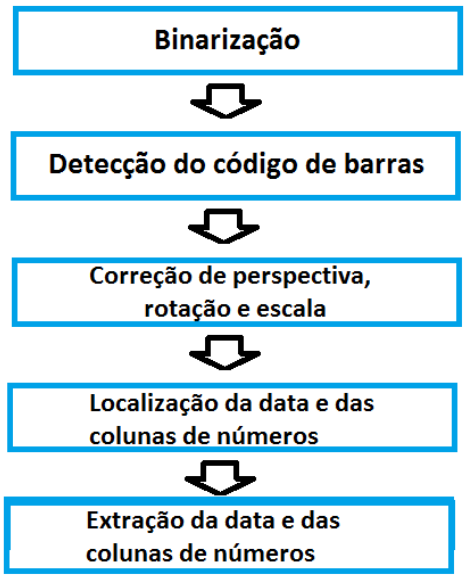

Figura 2.16 - Diagrama de fluxo do reconhecimento do bilhete de loteria

Figura 2.17 - Imagem em *grayscale* do bilhete de loteria

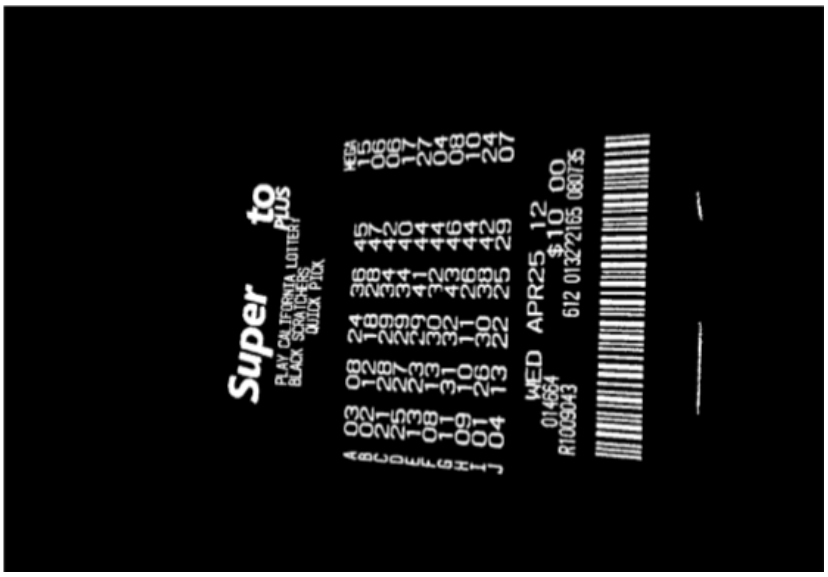

Figura 2.4 - Imagem binarizada do bilhete de loteria

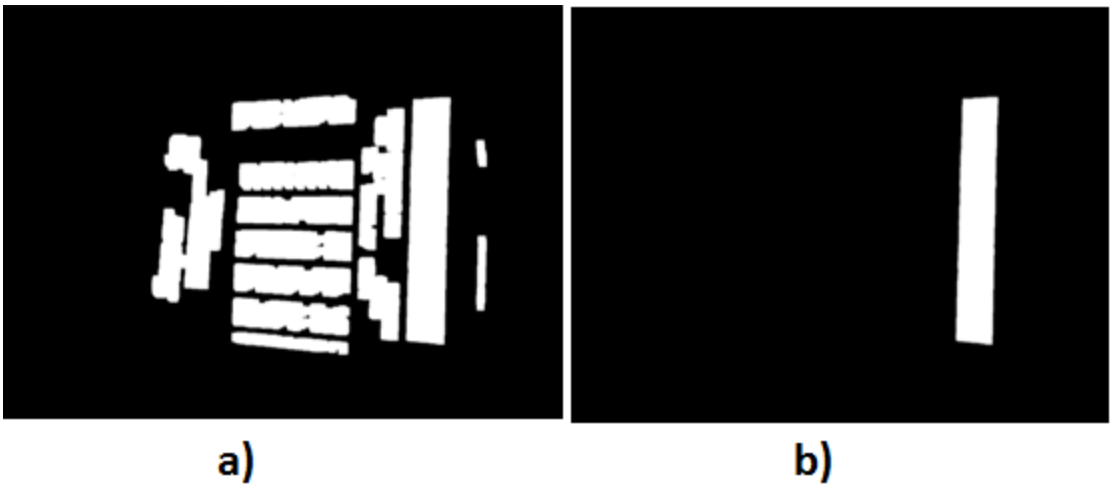

Figura 2.5 - a) Imagem dilatada para facilitar a localização do código de barras. b) Extração do código de barras.

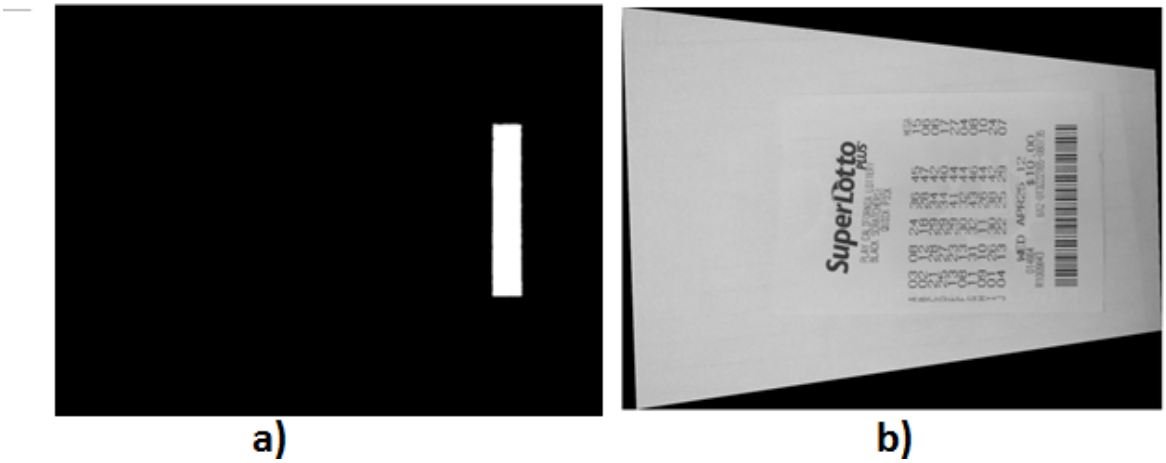

Figura 2.20 - Aplicação das correções de perspectiva, rotação e escala no código de barras em a) e a aplicação da correção na imagem em *grayscale* em b).

Figura 2.6 - Imagem da segmentação realizada no bilhete de loteria

3. Desenvolvimento do Trabalho

Com o conhecimento das ferramentas e técnicas necessárias para o trabalho, o objetivo torna-se a aplicação destas no desenvolvimento do trabalho em si. A seguir está descrito o problema, as tentativas de resolução e os respectivos resultados.

3.1 Descrição do Problema

O problema consiste na identificação dos 6 números mais representativos de um brinco eletrônico ilustrado na **Figura 3.1**, que é preso a orelha de animais bovinos para controle de gado. O reconhecimento de caracteres é feito através de um algoritmo OCR, o Tesseract, que como explicado anteriormente, tenta reconhecer qualquer elemento da imagem como um caracter propriamente dito.

Figura 3.1 - Brinco utilizado para controle de gado

3.2 Ferramentas e Tecnologias Utilizadas

Diversas ferramentas foram utilizadas no decorrer do projeto, assim como algumas linguagens de programação. Abaixo estão descritas as ferramentas utilizadas, bem como os motivos que levaram à escolha das mesmas.

- IDE Eclipse(ECLIPSE, 2014):

  É um IDE desenvolvido em Java e *open source*, sendo o IDE Java mais utilizado no mundo. Com o uso de plug-ins pode ser usado para desenvolver em outras linguagens além do Java, como C/C++, PHP, ColdFusion, Python e Android.

  Com o plug-in do Android SDK (*Software Development Kit*) o Eclipse pode ser usado para fazer desenvolver aplicativos Android, rodando diretamente em um aparelho. E o *layout* das telas do aplicativo pode ser feito de forma gráfica, podendo visualizar o resultado parcial sem ser preciso rodar o aplicativo em um aparelho.

  Essa ferramenta foi escolhida por ser gratuita e até então a mais utilizada e estável para o desenvolvimento de aplicativos Android.

- Java(JAVA, 2014) :

  Linguagem de programação orientada a objetivo desenvolvida na década de 90, que é executada por uma máquina virtual. A linguagem Java é a linguagem mais utilizada no desenvolvimento de aplicativos Android.

  Essa ferramenta foi utilizada pois é a que possui mais documentação e a mais utilizada por desenvolvedores Android.

- Android SDK(ANDROID SDK, 2014) :

  Conjunto de ferramentas que ajudam no desenvolvimento de aplicativos Android. Estão incluídos: *debugger*, bibliotecas, emulador Android, documentação e exemplos de projetos. O IDE oficialmente suportado é o Eclipse, com o plug-in ADT (Android Development Tools). O Android SDK foi utilizado no projeto pois é importante para se compilar e rodar os aplicativos, já que contém todas as bibliotecas do Android.

- XML(*eXtensible Markup Language*)(XML, 2014) :

  Linguagem de marcação, para criação de documentos com dados organizados de forma hierárquica, mas sem conter informações de como exibí-los. Em sua estrutura são utilizadas *tags* definidas pelo próprio usuário. Seu propósito principal é a facilidade de compartilhamento de informações através da internet.

  No caso projeto, foi utilizado para definir os *layouts* das telas do aplicativos, para ajudar na tradução do aplicativo para outros idiomas, e também para ajudar na atualização do banco de dados.

- Android NDK(ANDROID NDK, 2014):

  Conjunto de ferramentas que possibilita a implementação de código de linguagens nativas, como C e C++. Para alguns aplicativos, pode ser útil para reutilizar bibliotecas prontas escritas nessas linguagens, mas a maioria dos aplicativos não utiliza esta ferramenta.

- OpenCV4Android SDK (OPENCV4ANDROID SDK, 2014):

  Conjunto de ferramenteas que permite a utilização da biblioteca OpenCV para aplicações Android.

3.3 Resolução do problema e implementação

O desenvolvimento do trabalho foi feito através da integração e modificação dos códigos dos aplicativos "OpenCV Sample – color-blob-detection", projeto utilizado de exemplo de aprendizado no desenvolvimento com a biblioteca para Android do OpenCV, e "OCR Test", de Robert Theis (THEIS, R., 2014).

3.3.1 OpenCV Sample – color-blob-detection

Projeto simples de detecção de regiões de certa faixa de cor, que é definida de acordo com o toque do usuário na imagem da câmera em tempo real. A interface deste aplicativo é ilustrada na **Figura 3.2.**

Figura 3.2 - Interface do aplicativo Blob Detector.

O aplicativo conta apenas com duas classes:

ColorBlobDetectionActivity.java: Classe que implementa a interface OnTouchListener, o que permite sobrescrever o método OnTouch, que trata o toque do usuário na tela enquanto a imagem da câmera é exibida, selecionando a cor da região para criar os *blobs*.

ColorBlobDetector.java: Como o próprio nome diz, detecta os *blobs* na imagem da cor selecionada pelo toque do usuário.

3.3.2 OCR Test

Projeto que realiza o reconhecimento de caracteres através da API do Tesseract para Android, através de uma área, definida pelo usuário, da imagem da câmera. A **Figura 6.3** mostra a interface deste aplicativo.

Figura 3.3 - Interface do aplicativo OCR Test.

Este aplicativo é mais complexo e conta com 27 classes distribuídas em 3 pacotes:

BeepManager.java : Controla a função de som e vibração da CaptureActivity.

CaptureActivity.java: Abre a câmera e delega a função de scanner para uma thread que roda em background. Possui o viewFinder para ajudar a selecionar a área da imagem corretamente.

CaptureActivityHandler.java: Handler da classe anterior, e trata as mensagens relacionadas à máquina de estado para captura de imagens.

DecodeHandler: Envia o bitmap para o OCR.

DecodeThread: Thread na qual são feitas as decodificações das imagens.

FinishListener.java: Listener que é usado para fechar o app em determinadas ocasiões.

HelpActivity.java: Disponibiliza informações gerais para o usuário.

LuminanceSource.java: Classe abstrata utilizada para abstrair diferentes implementações do bitmap entre as plataformas.

OcrCharacterHelper.java: Habilita blacklists/whitelists em diferentes idiomas.

OcrInitAsyncTask.java: Instala os dados necessários para o OCR e inicializa a engine com uma thread que roda em background.

OcrRecognizeAsyncTask.java: Utilizada em modo não-contínuo do OCR, envia requisições para a engine em uma thread, solta uma mensagem de sucesso/falha e dispensa o progress dialog

OcrResult.java: Encapsulamento da resposta do OCR.

OcrResultFailure.java: Carrega a metadata para resultados que falharam no OCR.

OcrResultText.java: Encapsulamento do texto resultante do OCR.

PlanarYUVLuminanceSource.java: Classe que estende a LuminanceSource em um array de dados YUV do driver da câmera, com a opção de cortar um retângulo dentro de toda a imagem. Pode ser utilizado para excluir pixels supérfluos e acelerar a decodificação.

PreferencesActivity.java: Salva as preferências do usuário, que estão disponíveis no botão Menu.

ViewfinderView.java: Sobreposta sobre o preview da câmera, utilizado para mostrar a interface do viewFinder e o resultado do OCR.

AutoFocusManager.java: Gerencia a opção de auto foco da câmera.

CameraConfigurationManager.java: Classe que lida com os parâmetros do hardware da câmera.

CameraManager.java: Classe que lida diretamente com a câmera e encapsula as imagens prévias utilizadas tanto para prévia quanto para a decodificação.

PreviewCallBack.java: Chamada quando a próxima preview é recebida.

ShutterButton.java: Botão para ser utilizado dentro da imagem para disparar a captura da imagem.

LanguageCodeHelper.java: Lida com a conversão de idiomas e de códigos de idiomas.

TranslateAsyncTask.java: Classe que realiza traduções em background.

Translator.java: Delega a tradução para o serviço apropriado.

TranslatorBing.java: Traduz utilizando a API da Microsoft para tradução.

TranslatorGoogle.java: Traduz utilizando a API da Google para tradução.

3.3.3 Integração e implementação do trabalho

A implementação do projeto passa pelo fluxo exibido na **Figura 3.4**.

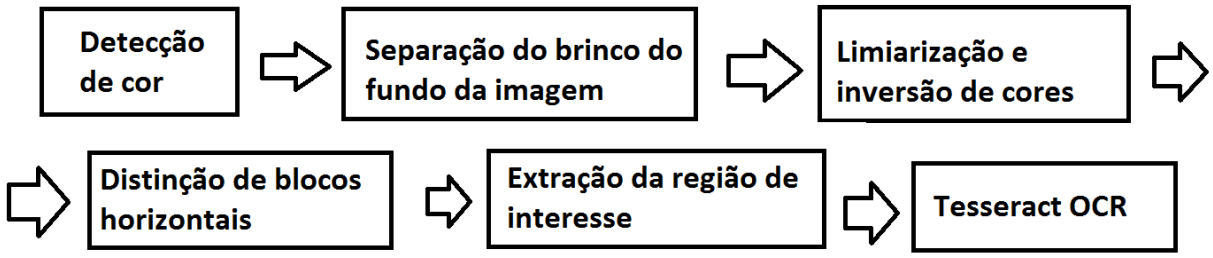

Figura 3.4 - Fluxo da aplicação

Através do projeto citado em 3.3.1, pode-se utilizar a função de detectar blobs para detectar o brinco de identificação, através da troca da lógica de obter a faixa de cor pelo toque do usuário, e fixar na cor do brinco, a amarela. Isso é possível através da função ***InRangeS*** já citada no **capítulo 2**. Através do método ***findContours*** podemos obter o contorno que cerca o brinco, devido a sua cor, e então podemos desenhá-lo na imagem através da função ***drawContours***. O resultado dessa etapa encontra-se na **Figura 3.5.**

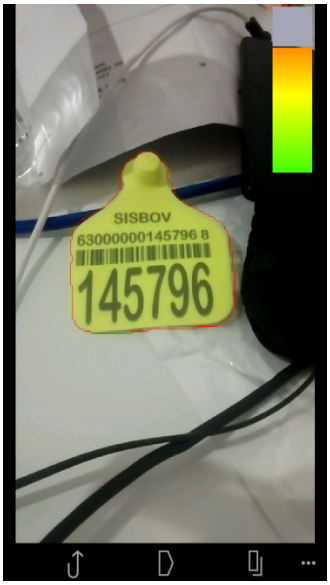

Figura 3.5 - Contornos desenhado em torno do amarelo do brinco.

Podemos fazer uso do contorno e copiar apenas a parte que cerca o brinco para uma imagem de fundo preto, através da função ***crop***. O resultado encontra-se na **Figura 3.6**.

Figura 3.6 - Brinco isolado do resto da imagem.

Como a cor amarela é clara, em contraste com o fundo e com os inscritos no brinco, podemos realizar o ***threshold* adaptativo**, e assim obter a **Figura 3.7(a)**. E assim invertemos as cores de dentro do brinco para melhor identificar os contornos mais tarde, o resultado é a **Figura 3.7 (b)**.

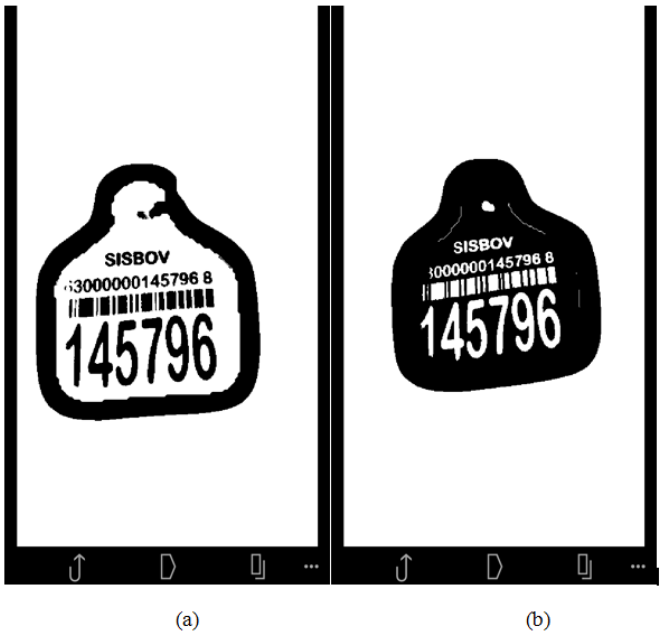

Figura 3.7 - (a) Aplicação do Threshold adaptativo. (b) Inversão das cores de dentro do brinco

O próximo passo é fazer com que o destaque seja realmente apenas nos caracteres, logo, eliminaremos o fundo branco, deixando uma imagem preta com os caracteres em branco, assim como ilustrado na **Figura 3.8**.

Figura 3.8 - Caracteres e contorno do brinco em branco com fundo preto.

Então, para facilitar o reconhecimento da região de interesse, juntamos as regiões em blocos horizontais, utilizando a função ***morphologyEx*** – em alternativa ao método de dilatação - com uma máscara de altura de um *pixel* e 15(quinze) de largura e o resultado encontra-se na **Figura 3.9**.

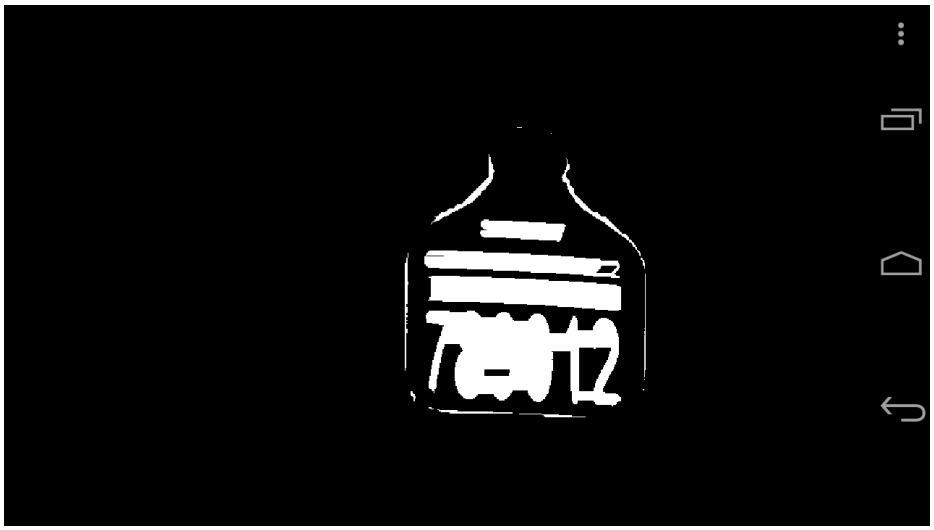

Figura 3.9 - Imagem aglomerada em blocos horizontais.

Com esta imagem, é possível buscar os contornos que a contém através novamente do ***findContours***, e desenhar retângulos ao redor deles, usando a função ***boundingRect*** para todo contorno encontrado. O que resulta na **Figura 3.10.**

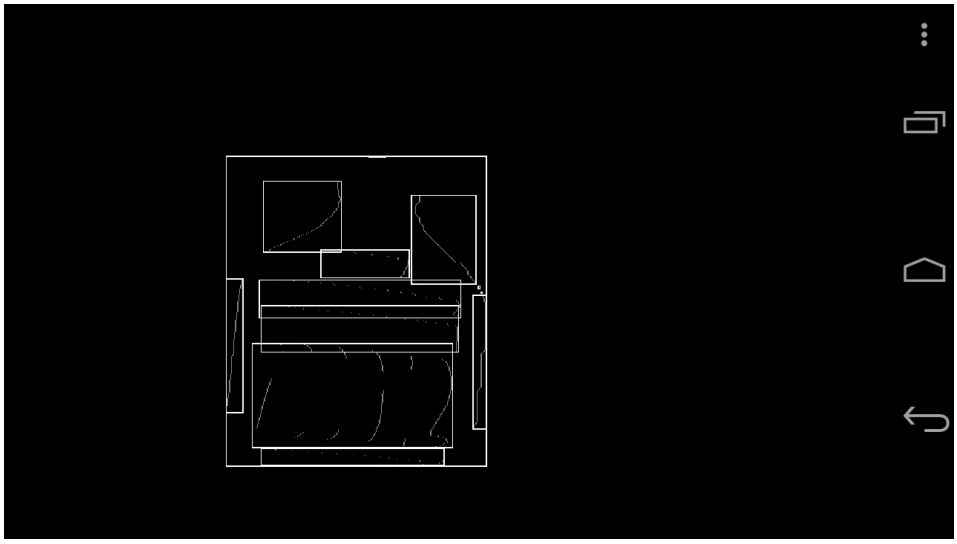

Figura 3.10 - Retângulos desenhados em volta dos contornos

Tendo estes retângulos, basta filtrar por altura e largura, tendo como referência o contorno maior, do brinco em questão, e conseguimos passar o retângulo que contém o número de 6 dígitos que era o objetivo do trabalho. O retângulo resultante encontra-se na **Figura 3.11.**

Figura 3.11 - Região dos 6 dígitos destacada na imagem original.

Pensando em uma possível integração com o aplicativo Sisgado citado anteriormente, foi implementada uma interface com uma tela simples, com um botão e um texto, para simular essa integração. A tela encontra-se na **Figura 3.12.**

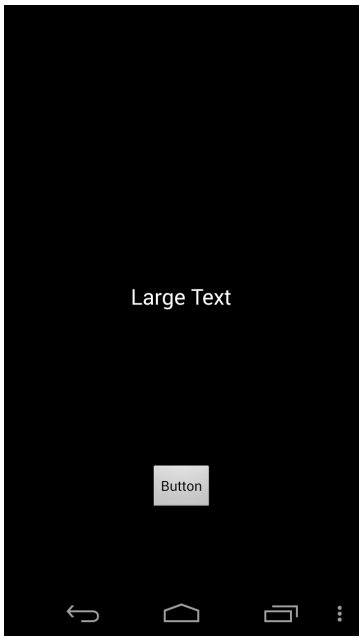

Figura 3.12 - Interface da tela inicial do aplicativo

A tela na qual se encontra o processamento da imagem e o reconhecimento de caracteres abre ao toque do botão, e além da imagem da câmera, possui três botões nos quais serão mostrados os textos reconhecidos, possibilitando ao usuário escolher o resultado certo, em caso de alguma falha. A execução dessa interface encontra-se na **Figura 3.13.**

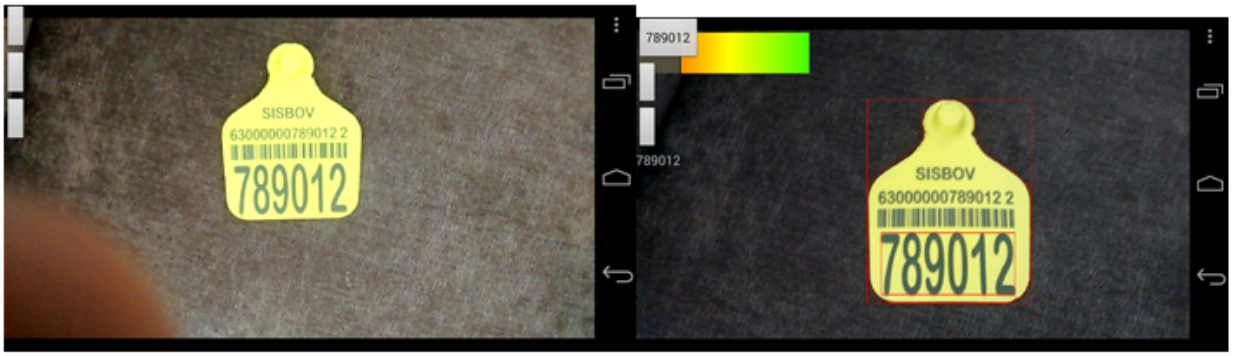

Figura 3.13 -Execução da tela de processamento

3.4 Testes

Os testes realizados foram de dois tipos: um teste de acurácia, na qual o *smartphone* já estaria a 0,25m do brinco em posição favorável ao reconhecimento, com iluminação de uma lâmpada fluorescente a aproximadamente 1,5m de distância do brinco. Assim que o toque na tela aconteça, só é considerado acerto se a primeira tentativa de reconhecimento possui os 6 dígitos corretos. A **Tabela 1** mostra o resultado destes testes.

Tabela 1 - Resultados do teste de acurácia

Brinco	Acertos/Tentativas	% de acerto	Desvio
789012	24/25	96	8.8
567891	25/25	100	12.8
123456	17/25	68	19.2
901234	24/25	96	8.8
345678	22/25	88	0.8
358702	21/25	84	3.2
482538	21/25	84	3.2
135790	21/25	84	3.2
145796	19/25	76	11.2
246807	24/25	96	8.8
Total	218/250	87.2	-

Desvio padrão: $\sigma = 9.19$

O segundo teste é de eficiência, ou seja, nas mesmas condições, mede-se o tempo que leva para o número correto do brinco estar disponível para o usuário selecionar em um dos 3 botões. Deve-se enfatizar que nenhum dos outros processos rodando no *smartphone* foi parado para a realização dos testes. Os resultados encontram-se na **Tabela 2.**

Tabela 2 - Resultados do teste de eficiência

Brinco	Tempo em média (ms)
789012	1611.36
567891	1185.08
123456	4497.2
901234	1364.92
345678	3629.84
358702	1448.32
482538	2057.88
135790	5803.08
145796	4419.68
246807	2296.56
Média	2831.39

4. Conclusão

A disponibilização de código aberto em forma de bibliotecas facilita muito a vida do desenvolvedor Android, no caso deste trabalho, por exemplo, não foi necessária a implementação de nenhuma função complexa de visão computacional, tampouco de reconhecimento de caracteres, graças ao OpenCV e ao Tesseract.

Neste trabalho pude aplicar os conhecimentos de programação e visão computacional adquiridos ao longo do curso, além de ampliar meus conhecimentos sobre a importação de código nativo de outras linguagens que não a oficial do Android, a linguagem Java.

Enquanto que o Tesseract realiza bem a sua tarefa de reconhecer caracteres, desde que em condições favoráveis para tal, o processamento da imagem da câmera do dispositivo através do OpenCV possibilitou extrair uma informação de uma imagem mais ampla, com mais detalhes a ser ignorados.

Apesar de não ter uma acurácia totalmente confiável, devido a trepidações na imagem e distorções causadas pelo movimento, a interface criada com as opções de escolha ajudam a contornar esse problema, visando uma aplicação comercial, em que o usuário está interessado no resultado e dinamismo da aplicação.

4.1 Trabalhos Futuros

Para os próximos passos, devemos ter a integração dos dois aplicativos sendo concretizada, além da possível integração com o aplicativo Sisgado.

Além disso, há a possibilidade de treinar o Tesseract com as fontes utilizadas nos brincos, aumentando a confiança dos resultados, ou até mesmo criar uma rede neural baseada nesses caracteres, sem a necessidade de utilizar a API do Tesseract.

Outro ponto que pode ser melhorado é o processamento de imagens em si, melhorando o reconhecimento da região de interesse, fazer com que o aplicativo fique menos sensível a mudanças na iluminação, rotação, e imagens tremidas.

Com a melhora do algoritmo, pode-se pensar também em utilizá-lo em outras aplicações não relacionadas ao Sisgado, que possam fazer uso do reconhecimento de caracteres. Caso chegue a um ponto de maturidade razoável, é possível a publicação do código para contribuir com a comunidade open source.

Referências Bibliográficas

ANAGNOSTOPOULOS, C. N. E. et al. A License-Plate-Recognition Algorithm for Intelligent Transportation System Applications. **IEEE Transactions on Intelligent Transportation Systems**, v. 7, n. 3, p. 377-392, set. 2006.

BALLARD, D.H. **Generalizing the Hough Transform to Detect Arbitrary Shapes**, Pattern Recognition, Vol.13, No.2, p.111-122, 1981

BHASKAR, S.; LAVASSAR, N.; GREEN, S. **Implementing Optical Character Recogniton on the Android Operating System for Business Cards.** Final Project – Digital Image Processing – Universidade de Stanford, Stanford. 2010.

BRADSKY,G.;KAEHLER, A. **Learning OpenCV.** O'Reilly, 2008.

CHANG, S. et al. Automatic License Plate Recognition**. IEEE Transactions on Intelligent Transportation Systems**, v. 5, n. 1, p. 42-53, mar. 2004.

CHERIET, M. et al. **Character Recognition Systems: A Guide for Students and Practioners**. New Jersey: Wiley, 2007.

GUPTA, A.; ZOU, T. **Mobile Lottery Ticket Recognition Using Android Phone**. Final Project – Digital Image Processing – Universidade de Stanford, Stanford. 2012.

JAIN, A. et al. Fundamental Challenges to Mobile Based OCR. **International Journal of Innovative Research & Studies**, v. 2, Issue 5, p. 87-101, maio 2013.

MORI, S.; SUEN, C. Y.; YAMAMOTO, K. Historical Review of OCR Research and Development. **Proceedings of the IEEE**, v. 80, n. 7, p. 1029-1058, jul. 1992.

PATEL, C; PATEL, A.; PATEL, D. Optical Character Recognition by Open Source OCR Tool Tesseract: A Case Study. **International Journal of Computer Applications,** v. 55, n. 10, p. 50-56, out. 2012.

OTSU, N. (1975). **A threshold selection method from gray-level histograms**. Automatica, 11 (285-296). IEEE

ROMIC, K; GALIC, I; BAUMGARTNER, A. Character Recognition Based On Pixel Concentration For License Plate Identification. **Jornal Técnico, Croácia**, v. 19, n.2, p. 321-325, jun. 2012.

SMITH, R. **An Overview of the Tesseract OCR Engine**. In proceedings of Document Analysis and Recognition. ICDAR 2007. IEEE Ninth International Conference.

THEIS, R.. Android-OCR. Disponível em: <https://github.com/rmtheis/android-ocr>. Acesso em:4 de dezembro de 2014.

Eclipse. Disponível em: <http://www.eclipse.org>. Acesso em:4 de dezembro de 2014.

JAVA. Disponível em: <http://www.java.com/>. Acesso em:4 de dezembro de 2014.ANDROID SDK. Disponível em: <http://developer.android.com/sdk/index.html>. Acesso em:4 de dezembro de 2014.

SQLITE. Disponível em: <http://www.sqlite.org>. Acesso em:4 de dezembro de 2014.

XML. Disponível em: <http://www.w3schools.com/XML/xml_whatis.asp>. Acesso em:4 de dezembro de 2014.

ANDROID NDK. Disponível em: <http://developer.android.com/tools/sdk/ndk/index.html>. Acesso em:4 de dezembro de 2014.OPENCV4ANDROID SDK.Disponível em: <http://docs.opencv.org/doc/tutorials/introduction/android_binary_package/O4A_SDK.html>

TESS-TWO, Fork of Tesseract Tools for Android. Disponível em: < https://github.com/rmtheis/tess-two>. Acesso em:4 de dezembro de 2014.

OPENCV. Disponível em: <http://opencv.org/l>. Acesso em:4 de dezembro de 2014.

TESSERACT-OCR. Disponível em: <https://code.google.com/p/tesseract-ocr/>. Acesso em:4 de dezembro de 2014.

5. Apêndice A – Imagens dos brincos em situação de teste

Imagens capturadas com o smartphone à ~25cm do brinco.

Figura 5.1 - Imagem do brinco 789012

Figura 5.2 - Imagem do brinco 567891

Figura 5.3 - Imagem do brinco 123456

Figura 5.4 - Imagem do brinco 901234

Figura 5.5 - Imagem do brinco 345678

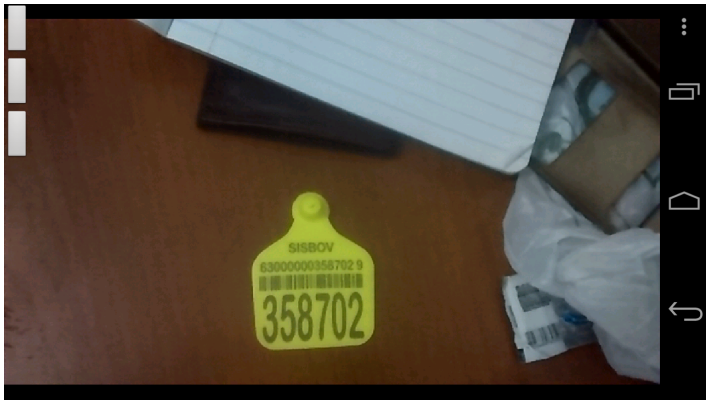

Figura 5.6 - Imagem do brinco 358702

Figura 5.7 - Imagem do brinco 482538

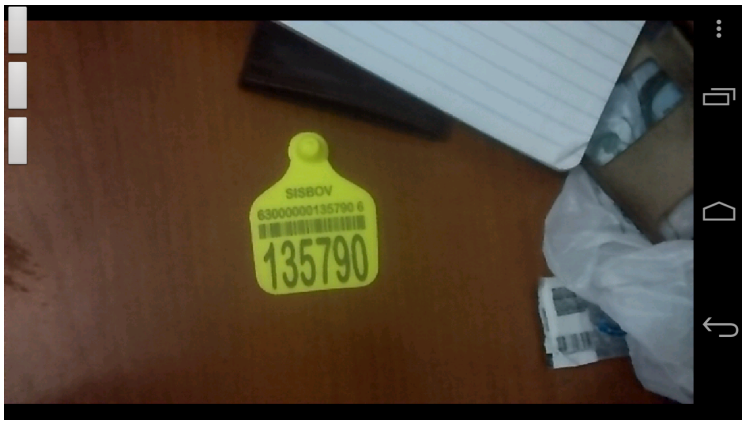

Figura 5.8 - Imagem do brinco 135790

Figura 5.9 - Imagem do brinco 145796

Figura 5.10 - Imagem do brinco 246807

6. Apêndice B – Código-fonte do projeto

MainActivity.java:

```
package org.opencv.samples.colorblobdetect;

import android.app.Activity;
import android.content.Intent;
import android.os.Bundle;
import android.view.View;
import android.view.View.OnClickListener;
import android.widget.Button;
import android.widget.TextView;

public class MainActivity extends Activity {
    protected static final int START_IMAGE_PROCESSING = 0;
    private Button mButton;
    private TextView mTextView;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
            super.onCreate(savedInstanceState);
            setContentView(R.layout.activity_main);

            mButton = (Button) findViewById(R.id.button1);
            mTextView = (TextView) findViewById(R.id.textView1);

            mButton.setOnClickListener(new OnClickListener() {

                    @Override
                    public void onClick(View v) {
                            Intent intent = new Intent(getApplicationContext(),
ColorBlobDetectionActivity.class);
                            startActivityForResult(intent, START_IMAGE_PROCESSING);
                    }
            });
    }

    @Override
    protected void onActivityResult(int requestCode, int resultCode, Intent data) {

            if (requestCode == START_IMAGE_PROCESSING && resultCode == RESULT_OK){
                    mTextView.setText(data.getExtras().getString("result"));
            }
            super.onActivityResult(requestCode, resultCode, data);
    }
}
```

Activity_main.xml:

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent" >

    <TextView
        android:id="@+id/textView1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_centerHorizontal="true"
        android:layout_centerVertical="true"
        android:text="Large Text"
        android:textAppearance="?android:attr/textAppearanceLarge" />

    <Button
        android:id="@+id/button1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_alignParentBottom="true"
        android:layout_centerHorizontal="true"
        android:layout_marginBottom="78dp"
        android:text="Button" />

</RelativeLayout>
```

ColorBlobDetectionActivity.java:

```
package org.opencv.samples.colorblobdetect;

import java.io.File;
import java.util.ArrayList;
import java.util.List;

import org.opencv.android.BaseLoaderCallback;
import org.opencv.android.CameraBridgeViewBase;
import org.opencv.android.CameraBridgeViewBase.CvCameraViewFrame;
import org.opencv.android.CameraBridgeViewBase.CvCameraViewListener2;
import org.opencv.android.LoaderCallbackInterface;
import org.opencv.android.OpenCVLoader;
import org.opencv.android.Utils;
import org.opencv.core.Core;
import org.opencv.core.CvType;
import org.opencv.core.Mat;
import org.opencv.core.MatOfPoint;
import org.opencv.core.Rect;
import org.opencv.core.Scalar;
import org.opencv.core.Size;
import org.opencv.imgproc.Imgproc;

import android.app.Activity;
```

```
import android.app.AlertDialog;
import android.content.DialogInterface;
import android.content.Intent;
import android.graphics.Bitmap;
import android.os.AsyncTask;
import android.os.Bundle;
import android.os.Environment;
import android.util.Log;
import android.view.MotionEvent;
import android.view.View;
import android.view.View.OnClickListener;
import android.view.View.OnTouchListener;
import android.view.Window;
import android.view.WindowManager;
import android.widget.Button;
import android.widget.TextView;

import com.googlecode.tesseract.android.TessBaseAPI;

public class ColorBlobDetectionActivity extends Activity implements
        OnTouchListener, CvCameraViewListener2 {
    private static final String TAG = "OCVSample::Activity";

    private boolean mIsColorSelected = false;
    private Mat mRgba;
    private Mat mBin;
    private Scalar mBlobColorRgba;
    private Scalar mBlobColorHsv;
    private ColorBlobDetector mDetector;
    private Mat mSpectrum;
    private Size SPECTRUM_SIZE;
    private Scalar CONTOUR_COLOR;

    private CameraBridgeViewBase mOpenCvCameraView;
    private TextView mOcrText;

    private Button button1;
    private Button button2;
    private Button button3;

    private long startTime;
    private long endTime;

    private TessBaseAPI tessAPI;

    private BaseLoaderCallback mLoaderCallback = new BaseLoaderCallback(this) {
        @Override
        public void onManagerConnected(int status) {
            switch (status) {
```

```
                case LoaderCallbackInterface.SUCCESS: {
                        Log.i(TAG, "OpenCV loaded successfully");
                        mOpenCvCameraView.enableView();
                        mOpenCvCameraView
                                        .setOnTouchListener(ColorBlobDetectionActivity.this);
                }
                        break;
                default: {
                        super.onManagerConnected(status);
                }
                        break;
                }
        }
};

private Rect minRect;

public ArrayList<String> numbersList;

public ColorBlobDetectionActivity() {
        Log.i(TAG, "Instantiated new " + this.getClass());
}

/** Called when the activity is first created. */
@Override
public void onCreate(Bundle savedInstanceState) {
        Log.i(TAG, "called onCreate");
        super.onCreate(savedInstanceState);
        requestWindowFeature(Window.FEATURE_NO_TITLE);
        getWindow().addFlags(WindowManager.LayoutParams.FLAG_KEEP_SCREEN_ON);

        setContentView(R.layout.color_blob_detection_surface_view);

        mOpenCvCameraView = (CameraBridgeViewBase)
findViewById(R.id.color_blob_detection_activity_surface_view);
        mOpenCvCameraView.setCvCameraViewListener(this);
        mOcrText = (TextView) findViewById(R.id.textOCR);
        button1 = (Button) findViewById(R.id.button1);
        button2 = (Button) findViewById(R.id.button2);
        button3 = (Button) findViewById(R.id.button3);
        ButtonClickListener buttonClickListener = new ButtonClickListener();
        button1.setOnClickListener(buttonClickListener);
        button2.setOnClickListener(buttonClickListener);
        button3.setOnClickListener(buttonClickListener);
        // button1.setVisibility(View.INVISIBLE);
        // button2.setVisibility(View.INVISIBLE);
        // button3.setVisibility(View.INVISIBLE);
        // mOcrText.setVisibility(View.INVISIBLE);
}
```

```
@Override
public void onPause() {
	super.onPause();
	if (mOpenCvCameraView != null)
		mOpenCvCameraView.disableView();
}

@Override
public void onResume() {
	super.onResume();
	OpenCVLoader.initAsync(OpenCVLoader.OPENCV_VERSION_2_4_3, this,
			mLoaderCallback);

	numbersList = new ArrayList<String>();
}

public void onDestroy() {
	super.onDestroy();
	if (mOpenCvCameraView != null)
		mOpenCvCameraView.disableView();
}

public void onCameraViewStarted(int width, int height) {
	mRgba = new Mat(height, width, CvType.CV_8UC4);
	mBin = new Mat(height, width, CvType.CV_8UC4);
	mDetector = new ColorBlobDetector();
	mSpectrum = new Mat();
	mBlobColorRgba = new Scalar(255);
	mBlobColorHsv = new Scalar(255);
	SPECTRUM_SIZE = new Size(200, 64);
	CONTOUR_COLOR = new Scalar(255, 0, 0, 255);
}

public void onCameraViewStopped() {
	mRgba.release();
}

public boolean onTouch(View v, MotionEvent event) {
	int cols = mRgba.cols();
	int rows = mRgba.rows();

	int xOffset = (mOpenCvCameraView.getWidth() - cols) / 2;
	int yOffset = (mOpenCvCameraView.getHeight() - rows) / 2;

	int x = (int) event.getX() - xOffset;
	int y = (int) event.getY() - yOffset;

	Log.i(TAG, "Touch image coordinates: (" + x + ", " + y + ")");
```

```
        if ((x < 0) || (y < 0) || (x > cols) || (y > rows))
                return false;

        Rect touchedRect = new Rect();

        touchedRect.x = (x > 4) ? x - 4 : 0;
        touchedRect.y = (y > 4) ? y - 4 : 0;

        touchedRect.width = (x + 4 < cols) ? x + 4 - touchedRect.x : cols
                        - touchedRect.x;
        touchedRect.height = (y + 4 < rows) ? y + 4 - touchedRect.y : rows
                        - touchedRect.y;

        Mat touchedRegionRgba = mRgba.submat(touchedRect);

        Mat touchedRegionHsv = new Mat();
        Imgproc.cvtColor(touchedRegionRgba, touchedRegionHsv,
                        Imgproc.COLOR_RGB2HSV_FULL);

        // Calculate average color of touched region
        mBlobColorHsv = Core.sumElems(touchedRegionHsv);

        Log.i(TAG, "Touched hsv color: (" + mBlobColorHsv.val[0] + ", "
                        + mBlobColorHsv.val[1] + ", " + mBlobColorHsv.val[2] + ", "
                        + mBlobColorHsv.val[3] + ")");
        int pointCount = touchedRect.width * touchedRect.height;
        for (int i = 0; i < mBlobColorHsv.val.length; i++)
                mBlobColorHsv.val[i] /= pointCount;

        mBlobColorRgba = converScalarHsv2Rgba(mBlobColorHsv);

        Log.i(TAG, "Touched rgba color: (" + mBlobColorRgba.val[0] + ", "
                        + mBlobColorRgba.val[1] + ", " + mBlobColorRgba.val[2] + ", "
                        + mBlobColorRgba.val[3] + ")");

        mDetector.setHsvColor(mBlobColorHsv);

        Imgproc.resize(mDetector.getSpectrum(), mSpectrum, SPECTRUM_SIZE);

        mIsColorSelected = true;

        touchedRegionRgba.release();
        touchedRegionHsv.release();
        startTime = System.nanoTime();
        return false; // don't need subsequent touch events
}

public Mat onCameraFrame(CvCameraViewFrame inputFrame) {
```

```
mRgba = inputFrame.rgba();
Mat crop = null, crop_1 = null, mask = null;

Mat black = null, binary = null, negative = null;

Mat crop_black = new Mat(mRgba.rows(), mRgba.cols(), CvType.CV_8UC1);
Mat colorMask = null;
if (mIsColorSelected) {
        mDetector.process(mRgba);
        List<MatOfPoint> contours = mDetector.getContours();
        Log.e(TAG, "Contours count: " + contours.size());

        Imgproc.Canny(mRgba, mBin, 80, 90);
        Imgproc.drawContours(mRgba, contours, -1, CONTOUR_COLOR);

        // Mat
        // let's create a new image now
        crop = new Mat(mRgba.rows(), mRgba.cols(), CvType.CV_8UC3);
        crop.setTo(new Scalar(255, 255, 255));
        // mask = Mat.ones(mRgba.rows(), mRgba.cols(), CvType.CV_8UC1);
        mask = new Mat(mRgba.rows(), mRgba.cols(), CvType.CV_8UC1);
        colorMask = new Mat(mRgba.rows(), mRgba.cols(), CvType.CV_8UC1);
        black = new Mat(mRgba.rows(), mRgba.cols(), CvType.CV_8UC1);
        binary = new Mat(mRgba.rows(), mRgba.cols(), CvType.CV_8UC1);
        negative = new Mat(mRgba.rows(), mRgba.cols(), CvType.CV_8UC1);
        colorMask = new Mat(mRgba.rows(), mRgba.cols(), CvType.CV_8UC1);
        crop_black = new Mat(mRgba.rows(), mRgba.cols(), CvType.CV_8UC1);

        Imgproc.drawContours(mask, contours, -1, new Scalar(255), -1);

        mRgba.copyTo(crop, mask);
        crop_1 = new Mat(mRgba.rows(), mRgba.cols(), CvType.CV_8UC1);
        Imgproc.cvtColor(crop, crop_1, Imgproc.COLOR_RGB2GRAY);
        Imgproc.adaptiveThreshold(crop_1, binary, 255,
                        Imgproc.ADAPTIVE_THRESH_MEAN_C,
Imgproc.THRESH_BINARY, 75,
                        10);

        Core.bitwise_not(binary, negative, mask);
        crop_black.setTo(new Scalar(0));
        negative.copyTo(crop_black, mask);
        crop_black.copyTo(black);

        // borrar a imagem
        Mat element = new Mat();
        element = Imgproc.getStructuringElement(Imgproc.MORPH_RECT,
                        new Size(15, 1));
        Imgproc.morphologyEx(black, black, Imgproc.MORPH_CLOSE, element);
        black.copyTo(colorMask);
```

```
            Mat mHierarchy = new Mat();
            Imgproc.findContours(black, contours, mHierarchy,
                        Imgproc.RETR_EXTERNAL, Imgproc.CHAIN_APPROX_NONE);
            ArrayList<Rect> rects = new ArrayList<Rect>();
            for (int i = 0; i < contours.size(); i++) {

                  Rect rect = Imgproc.boundingRect(contours.get(i));
                  rects.add(rect);
            }
            minRect = new Rect(0, 999, 0, 0);
            for (Rect rect : rects) {
                  if (rect.y < minRect.y) {
                        minRect = rect;
                  }

                  // Core.rectangle(black, rect.br(), rect.tl(), new Scalar(255));
            }
            Rect maxRect = minRect;
            Core.rectangle(mRgba, maxRect.br(), maxRect.tl(), new Scalar(255));
            rects.remove(minRect);
            minRect = new Rect(0, crop_1.height(), 0, 0);
            for (Rect rect : rects) {
                  if (rect.height > minRect.height
                              && rect.width > maxRect.width * 0.7) {
                        minRect = rect;
                  }
            }

            Core.rectangle(mRgba, minRect.br(), minRect.tl(), new Scalar(255));

            Mat colorLabel = mRgba.submat(4, 68, 4, 68);
            colorLabel.setTo(mBlobColorRgba);

            Mat spectrumLabel = mRgba.submat(4, 4 + mSpectrum.rows(), 70,
                        70 + mSpectrum.cols());
            mSpectrum.copyTo(spectrumLabel);
      }
      if (black != null && crop_1 != null && minRect != null
                  && minRect.area() > 0) {
            Mat rectMat = new Mat(crop_1, minRect);
            Bitmap bitmap = Bitmap.createBitmap(rectMat.width(),
                        rectMat.height(), Bitmap.Config.ARGB_8888);
            Utils.matToBitmap(rectMat, bitmap);
            String destinationDirBase = getStorageDirectory().toString();
            File tessdataDir = new File(destinationDirBase + File.separator
                        + "tessdata");
            if (!tessdataDir.exists() && !tessdataDir.mkdirs()) {
                  Log.e(TAG, "Couldn't make directory " + tessdataDir);
```

```
                } else {
                        if (tessAPI == null) {
                                tessAPI = new TessBaseAPI();
                                tessAPI.init(destinationDirBase + File.separator, "eng");
                        }
                        tessAPI.setImage(bitmap);

                        Log.d("result", "mask result: " + tessAPI.getUTF8Text());
                        // mOcrText.setText(tessAPI.getUTF8Text());
                }

                OCRAsyncTask ocrAsyncTask = new OCRAsyncTask();
                ocrAsyncTask.execute(tessAPI);

                return mRgba;

        } else {
                Log.d("IMAGE", "mRgba");
                return mRgba;
        }
}

private Scalar converScalarHsv2Rgba(Scalar hsvColor) {
        Mat pointMatRgba = new Mat();
        Mat pointMatHsv = new Mat(1, 1, CvType.CV_8UC3, hsvColor);
        Imgproc.cvtColor(pointMatHsv, pointMatRgba, Imgproc.COLOR_HSV2RGB_FULL,
                        4);

        return new Scalar(pointMatRgba.get(0, 0));
}

/** Finds the proper location on the SD card where we can save files. */
private File getStorageDirectory() {
        // Log.d(TAG, "getStorageDirectory(): API level is " +
        // Integer.valueOf(android.os.Build.VERSION.SDK_INT));

        String state = null;
        try {
                state = Environment.getExternalStorageState();
        } catch (RuntimeException e) {
                Log.e(TAG, "Is the SD card visible?", e);
                showErrorMessage("Error",
                                "Required external storage (such as an SD card) is unavailable.");
        }

        if (Environment.MEDIA_MOUNTED.equals(Environment
                        .getExternalStorageState())) {

                // We can read and write the media
```

```
                        // if (Integer.valueOf(android.os.Build.VERSION.SDK_INT) > 7) {
                        // For Android 2.2 and above

                        try {
                                return getExternalFilesDir(Environment.MEDIA_MOUNTED);
                        } catch (NullPointerException e) {
                                // We get an error here if the SD card is visible, but full
                                Log.e(TAG, "External storage is unavailable");
                                showErrorMessage("Error",
                                                "Required external storage (such as an SD card) is full or
unavailable.");
                        }

                } else if (Environment.MEDIA_MOUNTED_READ_ONLY.equals(state)) {
                        // We can only read the media
                        Log.e(TAG, "External storage is read-only");
                        showErrorMessage(
                                        "Error",
                                        "Required external storage (such as an SD card) is unavailable for data
storage.");
                } else {
                        // Something else is wrong. It may be one of many other states, but
                        // all we need
                        // to know is we can neither read nor write
                        Log.e(TAG, "External storage is unavailable");
                        showErrorMessage("Error",
                                        "Required external storage (such as an SD card) is unavailable or
corrupted.");
                }
                return null;
        }

        /**
         * Displays an error message dialog box to the user on the UI thread.
         *
         * @param title
         *          The title for the dialog box
         * @param message
         *          The error message to be displayed
         */
        void showErrorMessage(String title, String message) {
                new AlertDialog.Builder(this).setTitle(title).setMessage(message)
                                .setOnCancelListener(new FinishListener(this))
                                .setPositiveButton("Done", new FinishListener(this)).show();
        }

        /**
         * Simple listener used to exit the app in a few cases.
         *
```

```
 * The code for this class was adapted from the ZXing project:
 * http://code.google.com/p/zxing
 */
final class FinishListener implements DialogInterface.OnClickListener,
                DialogInterface.OnCancelListener, Runnable {

        private final Activity activityToFinish;

        FinishListener(Activity activityToFinish) {
                this.activityToFinish = activityToFinish;
        }

        public void onCancel(DialogInterface dialogInterface) {
                run();
        }

        public void onClick(DialogInterface dialogInterface, int i) {
                run();
        }

        public void run() {
                activityToFinish.finish();
        }

}

private class OCRAsyncTask extends AsyncTask<TessBaseAPI, Void, String> {
        @Override
        protected String doInBackground(TessBaseAPI... params) {
                TessBaseAPI tess = params[0];
                return tess.getUTF8Text();
        }

        @Override
        protected void onPostExecute(String result) {
                if (mOcrText != null) {
                        mOcrText.setText(result);
                }
                Log.d("result", "result size: " + result.length());
                if (result.length() == 6 && !numbersList.contains(result)) {
                        if (result.equals("145796")) {
                                //utilizado para o teste de resposta do algoritmo.
                                endTime = System.nanoTime();
                                long duration = endTime - startTime;
                                float timeInMili = duration / 1000000;
                                Log.d("Time", "time elapsed: " + timeInMili);
                        }
                        if (button1 != null && button1.getText().length() == 0) {
                                button1.setText(result);
```

```
                        } else if (button2 != null
                                        && button2.getText().toString().isEmpty()) {
                                button2.setText(result);
                        } else if (button3 != null
                                        && button3.getText().toString().isEmpty()) {
                                button3.setText(result);
                        }
                        numbersList.add(result);
                }
                super.onPostExecute(result);
        }

    }

    private class ButtonClickListener implements OnClickListener {

        @Override
        public void onClick(View v) {
            //retorna a string da label do botão para a activity anterior
            Button button = (Button) v;
            Intent resultIntent = new Intent();
            resultIntent.putExtra("result", button.getText().toString());
            setResult(RESULT_OK, resultIntent);
            ColorBlobDetectionActivity.this.finish();
        }

    }
}
```

Color_blob_detection_surface_view.xml:

```
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent" >

    <org.opencv.android.JavaCameraView
        android:id="@+id/color_blob_detection_activity_surface_view"
        android:layout_width="fill_parent"
        android:layout_height="fill_parent" />

    <Button
        android:id="@+id/button1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_alignParentLeft="true"
        android:layout_alignParentTop="true" />

    <Button
        android:id="@+id/button2"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
```

```
        android:layout_alignParentLeft="true"
        android:layout_below="@+id/button1" />

    <Button
        android:id="@+id/button3"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_alignParentLeft="true"
        android:layout_below="@+id/button2" />

    <TextView
        android:id="@+id/textOCR"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_alignParentLeft="true"
        android:layout_below="@+id/button3" />

</RelativeLayout>
```

ColorBlobDetector.java:

```
package org.opencv.samples.colorblobdetect;

import java.util.ArrayList;
import java.util.Iterator;
import java.util.List;

import org.opencv.core.Core;
import org.opencv.core.CvType;
import org.opencv.core.Mat;
import org.opencv.core.MatOfPoint;
import org.opencv.core.Scalar;
import org.opencv.imgproc.Imgproc;

import android.util.Log;

public class ColorBlobDetector {
    // Lower and Upper bounds for range checking in HSV color space
    private Scalar mLowerBound = new Scalar(0);
    private Scalar mUpperBound = new Scalar(0);
    // Minimum contour area in percent for contours filtering
    private static double mMinContourArea = 0.1;
    // Color radius for range checking in HSV color space
    private Scalar mColorRadius = new Scalar(25,50,50,0);
    private Mat mSpectrum = new Mat();
    private List<MatOfPoint> mContours = new ArrayList<MatOfPoint>();

    // Cache
    Mat mPyrDownMat = new Mat();
    Mat mHsvMat = new Mat();
    private Mat mMask = new Mat();
```

```
    Mat mDilatedMask = new Mat();
    Mat mHierarchy = new Mat();

    public void setColorRadius(Scalar radius) {
        mColorRadius = radius;
    }

    public void setHsvColor(Scalar hsvColor) {
        //valor de HSV para o amarelo do brinco.
        hsvColor = new Scalar(49.0, 154.3, 188.6, 0.0);
        double minH = (hsvColor.val[0] >= mColorRadius.val[0]) ? hsvColor.val[0]-mColorRadius.val[0] : 0;
        double maxH = (hsvColor.val[0]+mColorRadius.val[0] <= 255) ? hsvColor.val[0]+mColorRadius.val[0] :
255;

        mLowerBound.val[0] = minH;

        mUpperBound.val[0] = maxH;

        mLowerBound.val[1] = hsvColor.val[1] - mColorRadius.val[1];
        mUpperBound.val[1] = hsvColor.val[1] + mColorRadius.val[1];

        mLowerBound.val[2] = hsvColor.val[2] - mColorRadius.val[2];
        mUpperBound.val[2] = hsvColor.val[2] + mColorRadius.val[2];

        mLowerBound.val[3] = 0;
        mUpperBound.val[3] = 255;

        Mat spectrumHsv = new Mat(1, (int)(maxH-minH), CvType.CV_8UC3); //[80.1875, -40.40625, 126.0,
0.0] [130.1875, 59.59375, 226.0, 255.0]

        for (int j = 0; j < maxH-minH; j++) {
            byte[] tmp = {(byte)(minH+j), (byte)255, (byte)255};
            spectrumHsv.put(0, j, tmp);
        }

        Imgproc.cvtColor(spectrumHsv, mSpectrum, Imgproc.COLOR_HSV2RGB_FULL, 4);
    }

    public Mat getSpectrum() {
        return mSpectrum;
    }

    public void setMinContourArea(double area) {
        mMinContourArea = area;
    }

    public void process(Mat rgbaImage) {

        Log.d("image", "size: " + rgbaImage.size());
```

```
        Imgproc.pyrDown(rgbaImage, mPyrDownMat);
        Imgproc.pyrDown(mPyrDownMat, mPyrDownMat);

        Imgproc.cvtColor(mPyrDownMat, mHsvMat, Imgproc.COLOR_RGB2HSV_FULL);

        Core.inRange(mHsvMat, mLowerBound, mUpperBound, getmMask());
        Imgproc.dilate(getmMask(), mDilatedMask, new Mat());

        List<MatOfPoint> contours = new ArrayList<MatOfPoint>();

        Imgproc.findContours(mDilatedMask, contours, mHierarchy, Imgproc.RETR_EXTERNAL,
Imgproc.CHAIN_APPROX_SIMPLE);

        // Find max contour area
        double maxArea = 0;
        Iterator<MatOfPoint> each = contours.iterator();
        while (each.hasNext()) {
            MatOfPoint wrapper = each.next();
            double area = Imgproc.contourArea(wrapper);
            if (area > maxArea)
                maxArea = area;
        }

        // Filter contours by area and resize to fit the original image size
        mContours.clear();
        each = contours.iterator();
        while (each.hasNext()) {
            MatOfPoint contour = each.next();
            if (Imgproc.contourArea(contour) > mMinContourArea*maxArea) {
                Core.multiply(contour, new Scalar(4,4), contour);
                mContours.add(contour);
            }
        }
    }

    public List<MatOfPoint> getContours() {
        return mContours;
    }

        public Scalar getmUpperBound() {
                return mUpperBound;
        }

        public void setmUpperBound(Scalar mUpperBound) {
                this.mUpperBound = mUpperBound;
        }

        public Scalar getmLowerBound() {
                return mLowerBound;
```

```
        }

        public void setmLowerBound(Scalar mLowerBound) {
                this.mLowerBound = mLowerBound;
        }

        public Mat getmMask() {
                return mMask;
        }

        public void setmMask(Mat mMask) {
                this.mMask = mMask;
        }
    }
```

AndroidManifest.xml:

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="org.opencv.samples.colorblobdetect"
    android:versionCode="21"
    android:versionName="2.1" >

    <application
        android:icon="@drawable/icon"
        android:label="@string/app_name"
        android:theme="@android:style/Theme.NoTitleBar.Fullscreen" >
        <activity
            android:name="ColorBlobDetectionActivity"
            android:configChanges="keyboardHidden|orientation"
            android:label="@string/app_name"
            android:screenOrientation="landscape" >
        </activity>
        <activity
            android:name=".PhotoCaptureExample"
            android:label="Photo" >
        </activity>
        <activity
            android:name="MainActivity"
            android:label="Main"
            android:screenOrientation="landscape" >
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />

                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
    </application>

    <supports-screens
        android:anyDensity="true"
        android:largeScreens="true"
        android:normalScreens="true"
        android:resizeable="true"
```

```
    android:smallScreens="true" />

 <uses-sdk
    android:minSdkVersion="8"
    android:targetSdkVersion="11" />

 <uses-permission android:name="android.permission.CAMERA" />

 <uses-feature
    android:name="android.hardware.camera"
    android:required="false" />
 <uses-feature
    android:name="android.hardware.camera.autofocus"
    android:required="false" />
 <uses-feature
    android:name="android.hardware.camera.front"
    android:required="false" />
 <uses-feature
    android:name="android.hardware.camera.front.autofocus"
    android:required="false" />

</manifest>
```