Fundação Educacional do Município de Assis
Instituto Municipal de Ensino Superior de Assis
Campus "José Santilli Sobrinho"

PATRICK FRANCIS GOMES ROCHA

ABORDAGEM TEÓRICA E APLICABILIDADE DE DETECÇÃO FACIAL EM APLICATIVOS MÓVEIS COM OPENCV E GOOGLE ANDROID

Assis

2013

PATRICK FRANCIS GOMES ROCHA

ABORDAGEM TEÓRICA E APLICABILIDADE DE DETECÇÃO FACIAL EM APLICATIVOS MÓVEIS COM OPENCV E GOOGLE ANDROID

Trabalho de Conclusão de Curso apresentado ao
Instituto Municipal de Ensino Superior de Assis,
como requisito do Curso de Graduação.

Orientador: Prof. Esp. Guilherme de Cleva Farto

Área de Concentração: Informática

Assis

2013

FICHA CATALOGRÁFICA

ROCHA, Patrick Francis Gomes

Abordagem teórica e aplicabilidade de Detecção facial em Aplicativos Móveis com *OpenCV* e *Google Android* / Patrick Francis Gomes Rocha. Fundação Educacional do Município de Assis – FEMA – Assis, 2013.

88p.

Orientador: Prof. Esp. Guilherme de Cleva Farto

Trabalho de Conclusão de Curso – Instituto Municipal de Ensino Superior de Assis – IMESA

1. Reconhecimento de padrões. **2**. Biometria **3.** Google Android 4.OpenCV

CDD: 001.6

Biblioteca da FEMA

ABORDAGEM TEÓRICA E APLICABILIDADE DE DETECÇÃO FACIAL EM APLICATIVOS MÓVEIS COM OPENCV E GOOGLE ANDROID

PATRICK FRANCIS GOMES ROCHA

Trabalho de Conclusão de Curso apresentado ao
Instituto Municipal de Ensino Superior de Assis,
como requisito do Curso de Graduação, analisado
pela seguinte comissão examinadora:

Orientador: Prof. Esp. Guilherme de Cleva Farto
Analisador(1): Prof. Dr. Alex Sandro de Souza Romeo Poletto

Assis
2013

DEDICATÓRIA

Dedico este trabalho à Deus, a toda minha família e principalmente aos meus pais, por terem me dado todo carinho e melhor educação possível e serem um grande exemplo de pessoas. Dedico também aos meus amigos e pessoas que estiveram sempre me apoiando para a realização deste meu sonho.

AGRADECIMENTOS

Primeiramente agradeço a **Deus**, autor e princípio da vida, pois sem Ele, não estaríamos aqui vivendo e desfrutando de momentos tão marcantes e preciosos.

Agradeço também a intercessão de **Maria** junto ao Pai, sempre me guiando e orientando em minha vida.

Agradeço à minha mãe **Claudia Beatriz Gomes Rocha** e meu pai **Edivaldo Vieira da Rocha** que estão sempre ao meu lado me apoiando, orientando e me formando para a vida.

A minha namorada, **Patrícia Pereira Francisco**, por me animar sempre, com seu otimismo deixando meus dias mais alegres e belos.

A todos meus **amigos, parentes e familiares**, que me ajudaram de forma direta ou indireta, pois sem a ajuda deles, sem suas orações e apoio não conseguiria sozinho concluir esta etapa de minha vida.

À minha querida Professora e amiga Drª. **Marisa Atsuko Nitto**, por desde o primeiro ano me orientar em todo este processo acadêmico e pelos dois anos me orientando de forma brilhante no Projeto de Iniciação Cientifica desta Instituição.

Ao meu amigo e Professor Esp. **Guilherme de Cleva Farto**, por me orientar e colaborar nesta fase acadêmica tão importante.

A todos os meus **professores** destes quatro anos de curso, que ajudaram em minha formação não só acadêmica, mas também como pessoa.

A todos meus **amigos e amigas de classe**, pelo apoio, amizade e companheirismo.

E por fim agradeço a todos que colaboraram de forma direta ou indiretamente para a execução deste trabalho.

“Por vezes sentimos que aquilo que fazemos não é senão uma gota de água no mar. Mas o mar seria menor se lhe faltasse uma gota.”

Madre Teresa de Calcutá (1910 - 1997)

RESUMO

O reconhecimento facial é uma das mais promissoras das tecnologias biométricas e tem sido um dos temas mais estudados nas áreas da visão artificial, inteligência artificial e alternativas de segurança. A biometria já é bastante utilizada, sendo que o padrão biometrico mais conhecido é a digital. Contudo, o padrão biometrico facial vem sendo muito estudado e pesquisado para a utilização comercial, pois não é necessario contato fisico. A proposta desde trabalho foi a de pesquisar e compreender os conceitos da plataforma *Google Android* e *OpenCV*, fazendo uma integração das duas tecnologias para a construção de um protótipo de sistema de detecção de faces humanas para dispositivos móveis de tecnologia *Android*.

Palavras-chave: Reconhecimento de padrões; Biometria; Google Android; OpenCV.

ABSTRACT

Face recognition is one of the most promising biometric technologies and has been one of the most studied topics in the areas of machine vision, artificial intelligence, and security alternatives. Biometrics is already widely used, and the standard best known biometric is the fingerprint. However, the standard biometric facial has been much studied and researched for commercial use, it is not necessary physical contact. The proposal since work was to research and understand the concepts of the Google Android platform and OpenCV, doing a integration of two technologies for the construction of a prototype system for detecting human faces for mobile Android technology.

Keywords: Pattern Recognition, Biometrics, Google Android, OpenCV.

LISTA DE ILUSTRAÇÕES

Figura 1 – Fases do processo de Visão Computacional 19
Figura 2 – Representação de uma Imagem Digital 20
Figura 3 – Exemplo de operação matemática de Soma com imagem 22
Figura 4 – Exemplo de operações lógicas com imagem 22
Figura 5 – Processamento e Análise Digital de Imagens 24
Figura 6 – Aquisição de Imagens 25
Figura 7 – Pré-processamento da Imagem 26
Figura 8 – Extração de atributos 27
Figura 9 – Reconhecimento e Interpretação 28
Figura 10 – Exemplos de tipos de padrões utilizados. 32
Figura 11 – Padrão biométrico digital 35
Figura 12 – Padrão biométrico facial 36
Figura 13 – Padrão biométrico geometria das mãos 36
Figura 14 – Padrão biométrico íris 37
Figura 15 – Arquitetura de um Sistema Biométrico 39
Figura 16 – Autenticação (1 para 1) 41
Figura 17 – Identificação (1 para N) 42
Figura 18 – Arquitetura Google Android 46
Figura 19 – Emulador Android 49
Figura 20 – Android Studio 50
Figura 21 – Arquitetura geral do protótipo 52
Figura 22 – Fluxo de funcionamento do aplicativo 54
Figura 23 – Integração OpenCV: Descompactação biblioteca 57
Figura 24 – Integração OpenCV: Verificação da versão Android 58
Figura 25 – Integração OpenCV: Configuração Android NDK e OpenCV 59
Figura 26 – Integração OpenCV: Configuração novo projeto 60
Figura 27 – Integração OpenCV: Configuração C/C++ 61
Figura 28 – Fluxo PrincipalActivity e ConfiguracoesActivity 67
Figura 29 – Fluxo Activity x Classe de detecção e marcação 71
Figura 30 – Fluxo HaarCascade 76
Figura 31 – Interface de Menu 79
Figura 32 – Interface Sobre 80
Figura 33 – Interface Configurações 81
Figura 34 – Confirmação das configurações 82
Figura 35 – Interface detecção 83

SUMÁRIO

1 – INTRODUÇÃO 13
1.1 OBJETIVOS 14
1.2 JUSTIFICATIVAS 15
1.3 MOTIVAÇÃO 15
1.4 ESTRUTURA DO TRABALHO 16
2 – VISÃO COMPUTACIONAL 17
2.1 DENIFIÇÕES DE VISÃO COMPUTACIONAL 17
2.2 REPRESENTAÇÃO DE UMA IMAGEM DIGITAL 20
2.3 PROCESSAMENTO E ANÁLISE DIGITAL DE IMAGENS 23
2.3.1 Etapas do Processamento e Análise Digital de Imagens 23
2.3.1.1 Aquisição de Imagens 24
2.3.1.2 Pré-Processamento 25
2.3.1.3 Segmentação 26
2.3.1.4 Extração de atributos 27
2.3.1.5 Reconhecimento e Interpretação 27
2.4 BIBLIOTECA OPENCV 29
3 – RECONHECIMENTO DE PADRÕES E BIOMETRIA 31
3.1 RECONHECIMENTO DE PADRÕES 31
3.2 BIOMETRIA 32
3.2.1 Tipos de Biometria 34
3.2.1.1 Categoria Fisiológica 34
3.2.1.2 Categoria Comportamental 37
3.3 SISTEMAS BIOMÉTRICOS 38
3.3.1 Métodos de Reconhecimento 40
3.3.1.1 Autenticação 40
3.3.1.2 Identificação 41
4 – TECNOLOGIA *GOOGLE ANDROID* 43
4.1 INTRODUÇÃO AO *ANDROID* 43
4.2 *OPEN HANDSET ALLIANCE* E O *ANDROID* 44
4.3 SISTEMA OPERACIONAL LINUX 45

4.4 MAQUINA VIRTUAL DALVIK 45
4.5 PLATAFORMA *ANDROID* 46
4.5.1 Camada *Linux Kernel* 47
4.5.2 Camada *Android Runtime* 47
4.5.3 Camada *Libraries* 47
4.5.4 Camada *Application Framework* 48
4.5.5 Camada *Applications* 48
4.6 *ANDROID SDK* 48
5 – PROPOSTA DO TRABALHO 51
5.1 FUNCIONALIDADE DA APLICAÇÃO 51
5.2 INTEGRAÇÃO COM A BIBLIOTECA OPENCV 53
6 – ESTUDO DE CASO 56
6.1 DEFINIÇÃO DO PROBLEMA 56
6.2 CONFIGURAÇÃO DO AMBIENTE DE DESENVOLVIMENTO 56
6.3 IMPLEMENTAÇÃO DO ESTUDO DE CASO 62
6.3.1 *Activities* do aplicativo 62
6.3.2 Classe de detecção de faces e atributos 68
6.3.3 Utilização de elementos *Haar-Like Features* 72
6.4 INTERFACES DA APLICAÇÃO 78
7 – CONCLUSÃO 84
7.1 CONSIDERAÇÕES FINAIS 84
7.2 TRABALHOS FUTUROS 85
REFERÊNCIAS 86

1 – INTRODUÇÃO

Diante do crescimento acelerado dos sistemas de informação em diversos setores, exige-se um intenso estudo acerca das formas existentes de segurança para tornar os *softwares* mais confiáveis. Vários são os meios que atendem de maneira eficaz e eficiente tais necessidades. Entretanto, o grande ponto em questão é o investimento necessário para utilização desses métodos e protocolos sendo que, na última década, os estudos passaram a ter um desafio ainda maior: a busca por tecnologias de baixo custo, porém, ao mesmo tempo, uma taxa de segurança alta.

Com vista em tais necessidades torna-se necessário o estudo da visão computacional, na qual permite aos sistemas digitais extrair informações de imagens. Diversas informações podem ser extraídas e utilizadas para o reconhecimento de padrões complexos, como texturas, objetos, textos, padrões biométricos entre outros (JAIN; DUIN; MAO, 2000). Essa capacidade de reconhecer padrões proporciona aos sistemas digitais um sistema de visão artificial ainda não tão eficiente quanto o humano, mas que já possui algumas características superiores, tais como visão noturna e *zoom*.

A partir da visão computacional, surge uma grande tendência, a biometria, com enfoque em suprir as necessidades de sistemas que exigem protocolos de segurança de alto nível com custos baixos. O estudo é de extenso uso desde o início do século XX e fora definido, segundo Dantas (2003), como “a aplicação de teorias matemáticas e estatísticas em mensurações de características humanas”.

Mais recentemente, a palavra biometria tem sido utilizada para designar mensurações fisiológicas ou características do comportamento de forma a possibilitar a identificação de uma pessoa a partir de características individuais como veias da palma da mão, impressões digitais, reconhecimento da face, estrutura da retina, estrutura da íris e geometria da mão (ROCHA; NITTO, 2012).

A face humana é o principal atributo por meio da qual as pessoas são reconhecidas. Este fato propicia o desenvolvimento de sistemas automáticos de detecção, rastreamento e reconhecimento facial assim como a promoção de aplicações como mecanismos para a autenticação de usuários, monitoramento de ambientes, controle de acesso e vigilância (NASCIMENTO; GONZAGA, 2008).

O uso de dispositivos móveis tem se expandindo por todo planeta e vem sendo também uma grande oportunidade para a implementação de métodos de segurança, pois os *hardwares* e periféricos destes aparelhos portáteis fornecem um conjunto vasto de possibilidades para a aplicação de métodos, estratégias e maneiras de se implementar tais protocolos.

A partir de 2005, devido ao surgimento do sistema operacional e ambiente de desenvolvimento móvel *Google Android*, projeto *open-source* atualmente mantido pela *Open Handset Alliance* (OHA) sendo liderada pela própria *Google* e diversas outras empresas, a gama de oportunidades de implementação de aplicativos portáteis cresceu consideravelmente de forma a atrair mais programadores e permitir a construção de aplicações de diversas áreas (AQUINO, 2007).

Com o desenvolvimento de um aplicativo de reconhecimento facial para dispositivos móveis baseados em *Android* significa usufruir todos os benefícios da plataforma junto à eficácia dos métodos de reconhecimento biométrico, possibilitando a construção de sistemas razoavelmente seguros e de baixo custo.

1.1 OBJETIVOS

O objetivo deste trabalho é pesquisar e compreender os conceitos acerca dos sistemas biométricos, bem como as ferramentas e arquiteturas para o desenvolvimento de uma aplicação para dispositivos móveis baseados na plataforma *Google Android,* capaz de detectar faces e características humanas, usufruindo de todos os benefícios de um dispositivo baseado na plataforma móvel *Android* juntamente com os métodos e algoritmos de visão computacional proporcionadas na biblioteca *OPENCV*.

A partir dele, será possível realizar um comparativo entre os métodos convencionais de segurança e os métodos biométricos com a finalidade de demonstrar os pontos positivos e negativos.

1.2 JUSTIFICATIVAS

Tendo em vista o grande número de áreas e ambientes restritos tais como *sites* de bancos e caixas eletrônicos, a biometria tem sido utilizada como um excelente meio para simplificar o processo de identificação de indivíduos, pois dispensa o uso de cartões, códigos ou qualquer outro recurso na etapa de reconhecimento.

Além disso, a biometria oferece um alto nível de proteção contra diversos tipos de fraudes ao proporcionar o vínculo entre o usuário e o sistema, uma vez que somente o indivíduo presente poderá ser submetido à identificação. Dessa forma, a preocupação com a falsa identificação realizada por terceiros é eliminada, já que a biometria identifica indivíduos por meio de características únicas pertencentes a cada pessoa (SERRANO, 2010).

De acordo com Marengoni e Stringhini (2013), a biblioteca *OpenCV* possibilita ao desenvolvedor uma vasta opção de métodos de manipulação e processamento de imagens digitais assim como acessos simplificados para a realização de tarefas necessárias com as imagens a serem reconhecidas.

Apesar de a tecnologia *Google Android* oferecer uma interface de programação para reconhecimento facial, o trabalho apresenta grande importância, pois se trata de uma integração entre a biblioteca *OpenCV* e a plataforma *Android*, fornecendo assim, uma nova solução computacional para a área de segurança e identificação além de aumentar as opções de tratamento de imagens e métodos de detecção e reconhecimento de indivíduos a partir das mesmas.

1.3 MOTIVAÇÃO

O desenvolvimento deste trabalho consiste no fato de a biometria ser um tema que está em constante crescimento e que o reconhecimento facial é um dos padrões mais estudados atualmente nesta área.

Adaptar os conceitos estudados e aplicá-los na plataforma *Android* acrescentou, ao protótipo implementado, uma confiabilidade e estabilidade maiores, visto que, o *framework* fornece diversos recursos como bibliotecas e métodos que facilitam e

contribuem para o desenvolvimento de soluções baseadas em computação móvel além de que, por meio da integração com a biblioteca *OpenCV*, novas e grandes oportunidades no aperfeiçoamento do reconhecimento de padrões tornaram-se tangíveis e alcançáveis.

Outra motivação são as consideráveis chances de, em um futuro não tão distante, o mercado de trabalho necessitar de profissionais com conhecimentos e experiências na linha deste tema, uma vez que a área da segurança a informação cresce de forma exponencial.

1.4 ESTRUTURA DO TRABALHO

O trabalho está estruturado nas seguintes partes:

- **Capítulo 1 – Introdução**
- **Capítulo 2 – Visão Computacional**
- **Capítulo 3 – Reconhecimento de Padrões e Biometria**
- **Capítulo 4 – Tecnologia *Google Android***
- **Capítulo 5 – Proposta do Trabalho**
- **Capítulo 6 – Estudo de Caso**
- **Capítulo 7 – Conclusão**
- **Referências**

2 – VISÃO COMPUTACIONAL

O intuito desta sessão é apresentar o “Estado da Arte” da Visão Computacional, bem como definições fundamentais e básicas para o entendimento de manipulações e processamento de imagens.

Neste capitulo, encontram-se também definições e conceitos sobre a biblioteca *OpenCV*, biblioteca construída para atender e implementar as necessidades computacionais e práticas da Visão Computacional.

2.1 DENIFIÇÕES DE VISÃO COMPUTACIONAL

Visão computacional é a ciência responsável em dar visão a uma máquina, pela forma como o computador distingue à sua volta, extraindo dados significativas a partir de imagens capturadas por câmeras de vídeo, sensores, scanners, entre outros dispositivos. Estas informações permitem reconhecer, manipular e detectar sobre os objetos que compõem uma imagem, fazendo com que a máquina interprete seu meio (MILANO; HONORATO, 2011).

Os primeiros estudos com Visão Computacional acontecerão por volta de 1955, e na década de 70 foram aliadas as pesquisas de Inteligência Artificial, onde acreditavam que em pouco tempo conseguiriam simular e representar o sentido da visão de forma completa em uma máquina. Com pesquisas subsequentes, verificaram que a complexidade era muito maior do que o imaginado, pois não tinham informações de como as imagens eram interpretadas dentro do cérebro humano.

A visão humana consegue perceber e interpretar objetos em uma imagem ou ambiente de forma muito rápida e eficaz. Isso acontece no córtex visual do cérebro, uma das partes mais complexas no sistema de processamento do cérebro. Desde ponto, alguns pesquisadores concentram seus estudos na tentativa de decifrar o funcionamento dessa parte do cérebro, para então utilizar tais conceitos na visão computacional. É o que pesquisadores do MIT - *Massachusetts Institute of Technology* (Instituto de Tecnologia de Massachusetts) definem como *"ensinar computadores a enxergarem como humano"* (LOPES, 2009).

Diante de tal complexidade, Marengoni e Stringhini (2013), relata que a visão computacional fornece ao computador inúmeras informações matemáticas precisas a partir de imagens e vídeos, de forma que a máquina consiga executar tarefas pré-programadas e inteligentes, simulando ao máximo da inteligência humana, pois diante de todos os estudos, pesquisas e tecnologias atuais, ainda estão longe de conseguir uma formula ou método que se assemelhe a percepção da visão humana.

Atualmente o estudo de visão computacional mudou sua ênfase, passando de assemelhar para complementar a visão humana, ou seja, muitos aplicativos e sistemas de VC não conseguem atingir a perfeição e a capacidade da visão humana, mas atingem resultados que a visão humana não proporciona, como: visão noturna, *zoom*. Muitos destes sistemas são conhecidos como Sistemas Especialistas, no qual realizam de forma eficiente e eficaz tal procedimento. Dessa forma, não existe um padrão de desenvolvimento para a criação e implementação de um sistema de VC, mas por definição, qualquer *software* que realize algum conceito ou método de visão computacional passa pelas seguintes fases, conhecidas também como Processamento Digital de Imagens:

- **Aquisição da Imagem** – É o processo pelo qual trata da coleta da imagem ou o conjunto de imagens. Esta fase é feita a partir de sensores, câmeras. A imagem ou o conjunto de imagem pode ser digital ou analógica.
- **Pré-processamento:** Fase pela qual a imagem é submetida antes de iniciar a coleta de informações. Quanto maior a refinação com que a imagem é tratada nesta fase, mais fácil será a detecção dos objetos nas próximas fases, como exemplo destaque de contorno, bordas, realce de luzes.
- **Detecção e Segmentação:** Etapa na qual é destacado as regiões relevantes em uma imagem, ou seja, a região na qual a imagem será observada, estudada.
- **Extração de características:** É o processo no qual é retirado da imagem todas as suas características, como exemplo profundidade, bordas, formatos. Lembrando que todas as características são medidas matemáticas.
- **Processamento de alto nível:** Fase em que é feita todo o processo com os dados obtidos, gerando assim um resultado satisfatório. Vale lembrar que esta fase que é a escolha de qual método deve ser usado, de acordo com o

problema em questão. Por exemplo: Caso fosse um sistema de leitor biométrico digital, nesta fase seria feito a comparação entre o dado atual, com o dado salvo no banco de dados, para ser feito a validação do usuário.

A Figura 1 ilustra todas as etapas que uma imagem é submetida durante um processo em um sistema de visão computacional.

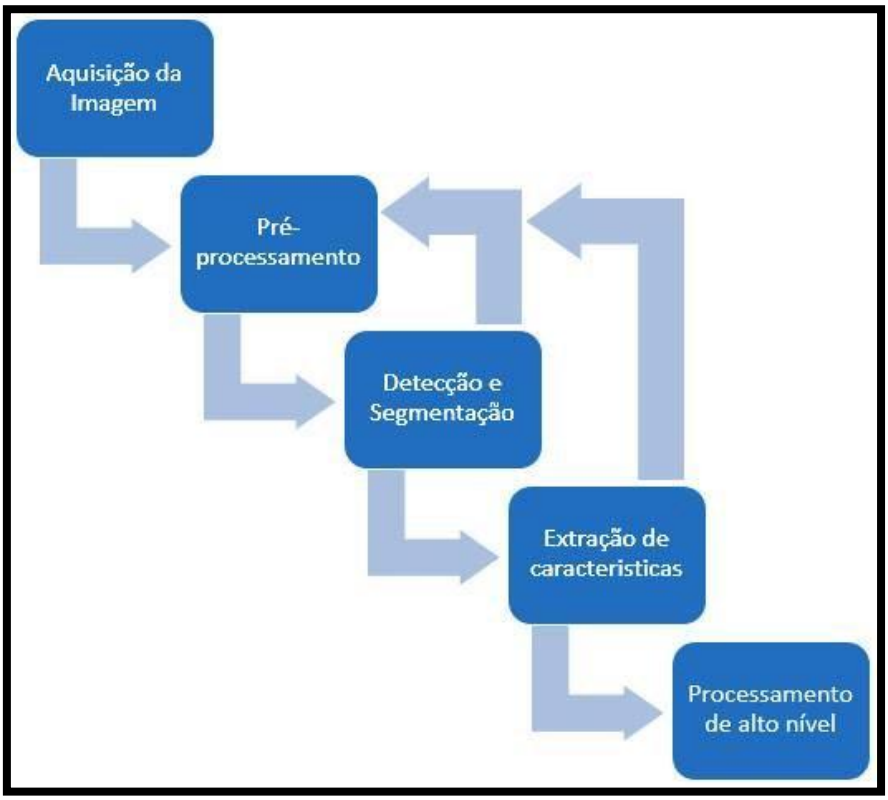

Figura 1 – Fases do processo de Visão Computacional

É importante destacar que se a fase de pré-processamento não for realizada de maneira correta, ocorrerão inúmeras falhas nas fases subsequentes e consequentemente o processo voltará a fase de pré-processamento, causando um retardo e demora para um resultado final.

De acordo com Milano e Honorato (2011), é muito importante um esforço e foco maior na fase de pré-processamento, pois como esta fase não obtém um resultado imediato, ou seja, somente no processamento das próximas fases que o resultado é verificado, a tendência desta etapa é ser feita de forma irregular.

2.2 REPRESENTAÇÃO DE UMA IMAGEM DIGITAL

Diferente da visão humana, o computador visualiza uma imagem em forma de “quadrinhos”, ou seja, de forma matricial, sendo que cada ponto desta matriz, contém informações, e a soma de todas as informações forma-se uma imagem (ALBUQUERQUE; ALBUQUERQUE, 2009).

Uma imagem digital linearizada ou monocromática é uma função bidimensional $f(x, y)$ que representam intensidades luminosas, onde x e y são coordenadas espaciais, que por convenção *x* = [1, 2, ..., N] e *y* = [1, 2, ..., N], sendo que *f* na posição (x, y) é regular ao nível de cinza (ou brilho) da imagem nesta localidade. A Figura 2 demonstra a ideia de um pixel em uma imagem.

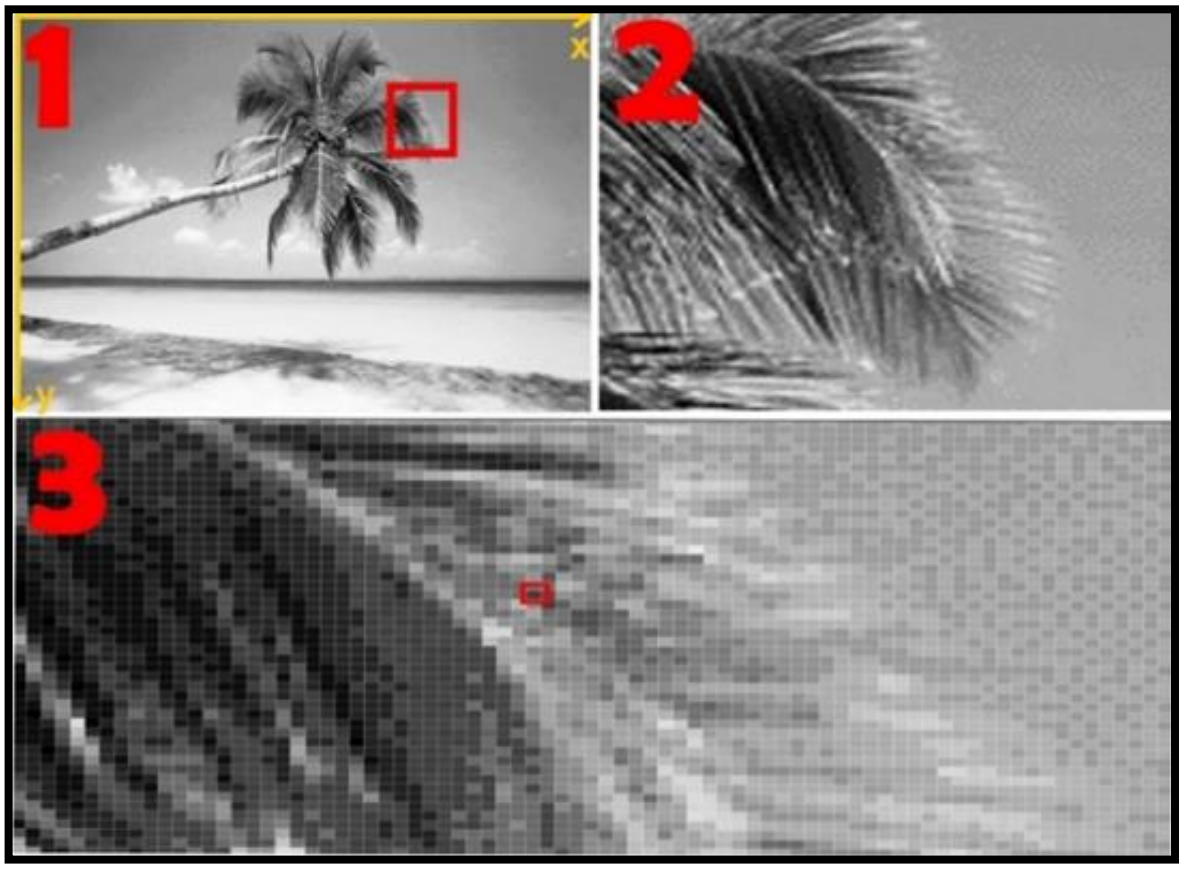

Figura 2 – Representação de uma Imagem Digital

Um pixel é o elemento mais básico de uma imagem. A maneira mais básica para o *pixel* é a forma retangular ou quadrada. Ele também é um componente de dimensões finitas na reprodução de uma imagem digital (SERRANO, 2010).

A Figura 2 fora dividida em três partes: A parte 1, demonstra a figura inteira, na integra, juntamente com o plano cartesiano (x, y), representando a localização dos *pixels* da imagem. Ainda na parte 1, pode-se verificar que a imagem foi selecionada. A parte 2 da figura, é uma imagem ampliada da seleção da parte 1. Já a parte 3, é a ampliação da imagem 2, sendo que nela é possível verificar a representação de um *pixel* da imagem.

Cada *pixel* é composto por um conjunto de três pontos: verde, vermelho e azul (*RGB*). Em imagens com cores, cada um destes três pontos podem ter uma variação de 256 cores (*0 a 255*), o equivalente a 8 bits, e combinando as tonalidades dos pontos é possivel exibir um pouco mais de 16.7 milhões de cores diferentes. Uma imagem de resolução 800 x 600 temos 480.000 pixels, ou seja, mais de 8,1 bilhões de cores. Atualmente, as imagens de alta definição possuem entre 32 a 64 bits de cores em cada *pixel*, aumentando em escala exponelcial a possibilidade de cores das imagens.

A partir de uma imagem digital, pode-se aplicar operações aritméticas e lógicas, pois sua representação é totalmente numérica, uma matriz de *pixels*, ou seja, sua localização, intensidade de luz, tons e cores são números armazenados em cada *pixel e* diante deste conceito, são criados filtros, melhoria na qualidade das imagens, correções, e criação de efeitos artísticos.

De acordo com Müller e Daronco (2000), a habilidade para executar operações é uma das principais vantagens do processamento digital de imagens sobre a fotografia convencional.

As operações aritméticas (ou matemáticas) em imagem são as operações básicas da matemática realizadas nos *pixels* de uma imagem para realçar, retirar ruídos, dar contraste, isto é, melhorar a imagem.

A Figura 3 mostra uma operação matemática:

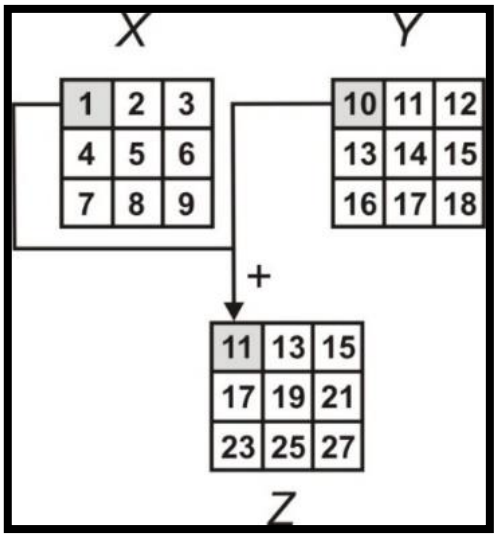

Figura 3 – Exemplo de operação matemática de Soma com imagem

As operações lógicas segue o mesmo conceito de realizar a operação com duas imagens, resultando em uma terceira *imagemR* (Resultante), com a diferença de que neste caso são utilizados os operadores lógicos. A Figura 4 apresenta algumas operações lógicas com imagem:

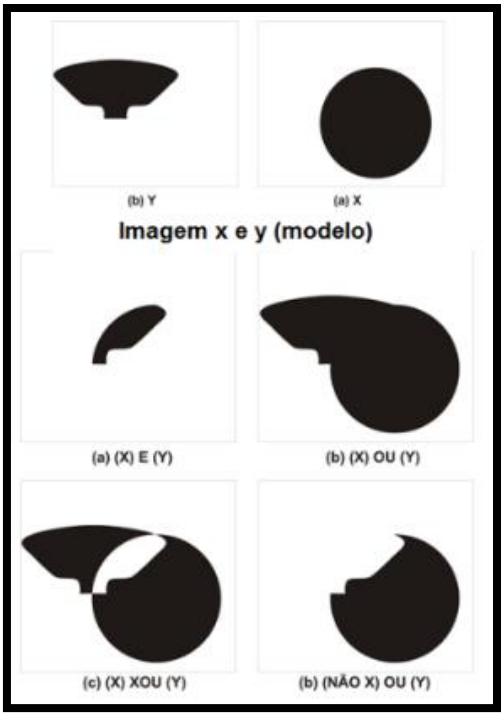

Figura 4 – Exemplo de operações lógicas com imagem
(In: ROCHA; NITTO, 2011)

2.3 PROCESSAMENTO E ANÁLISE DIGITAL DE IMAGENS

O Processamento e a Análise digital de imagens surgem do conceito da Visão Computacional. Muitos pesquisadores relatam que não se sabe determinar qual é a fronteira entre a VC e o Processamento Digital de Imagens, pois em muitos aspectos os dois contextos se assemelham. Conforme Albuquerque e Albuquerque (2009), processar uma imagem consiste em transformá-la sucessivamente com o objetivo de extrair mais facilmente a informação nela presente.

O Processamento Digital de Imagens (PDI) parte da imagem (de uma informação inicial que é geralmente captada por uma câmera) ou de uma sequência de imagens, para a obtenção da informação. Caso a imagem seja analógica, é necessário primeiramente converte-la para digital, em seguida, realizar todo o processamento da mesma.

A visão que um computador tem de uma imagem, como vimos na sessão anterior, é de “quadrinhos”, isto é, *pixel*, o menor elemento num dispositivo de exibição (por exemplo, o monitor), ao qual é possivel atribuir-se cor, realizar operações, correções, sendo que o resultado seja um conjunto de milhares de *pixels* que formam a imagem inteira.

Já a Análise Digital de Imagens (ADI) consiste na extração e tratamento de dados quantitativos a partir das imagens digitais. Apesar do ser humano ser muito mais competente em tarefas de reconhecimento, a ADI pode realizar medições mais rápidas, precisas e apuradas (NETO; FILHO, 1999).

A ADI possibilita ainda a realização de diversas medidas impossíveis de se executar manualmente, com mais precisão, velocidade e eficácia.

2.3.1 Etapas do Processamento e Análise Digital de Imagens

A Figura 5 ilustra as fases do processamento e análise digital de uma imagem. Cada uma dessas etapas é de suma importância no tratamento de uma imagem, dependendo do tipo de aplicação que se pretende realizar. Como se pode observar, as fases coincidem com a ilustração na Figura 1 que representam as etapas de um

sistema de Visão Computacional. Uma das possíveis diferenças seria que o PDI é o campo no qual entrada e saída são imagens. Por sua vez, a VC possui como foco principal simular a capacidade de visão humana em computadores e maquinas, incluindo o aprendizado, de forma a poder realizar deduções e a executar ações baseadas em informações visuais previamente analisadas. Visto que, a área de Análise Digital de Imagens fica envolvida entre o Processamento de Imagens e a Visão Computacional.

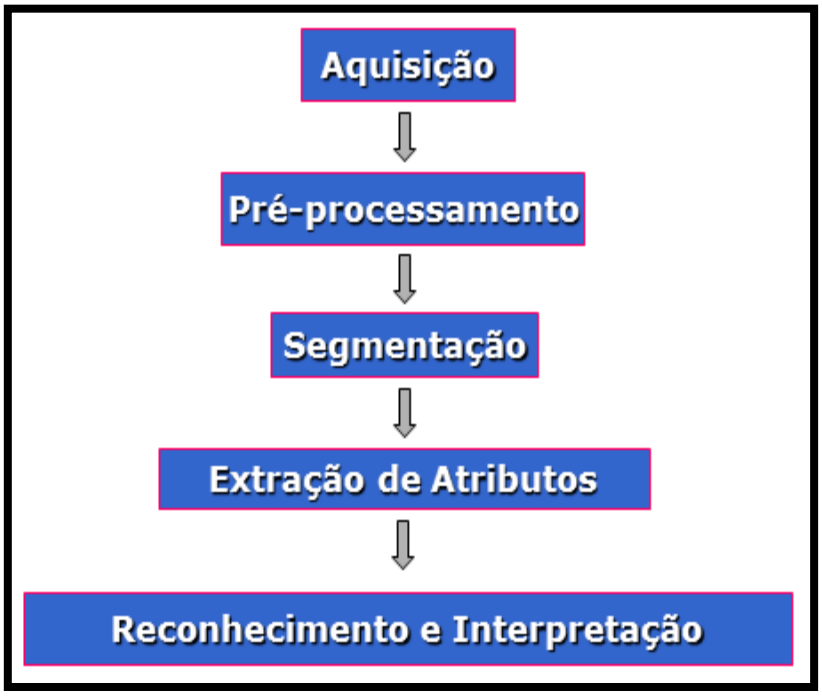

Figura 5 – Processamento e Análise Digital de Imagens
(In: VASCONCELOS, 2013)

2.3.1.1 Aquisição de Imagens

Para obter uma imagem digital é necessário um dispositivo fisico, para capturar a imagem e armazena-la em algum local previamente delimitado. Se a imagem for uma imagem analógica, imagem convencional, ela deve ser convertida para digital. Para este tipo de conversão, usa-se *scaners,* que transformam um sinal totalmente continuo, para binário (0 ou 1).

O conversor analógico/digital converte o sinal elétrico que produz o foto-sensor em pulsos digitais em formato binário, que são entendidos pelo computador. Os *softwares* embarcados nos dispositivos conversores fazem o tratamento e armazenam no computador em um local especifico. A Figura 6 ilustra esta fase:

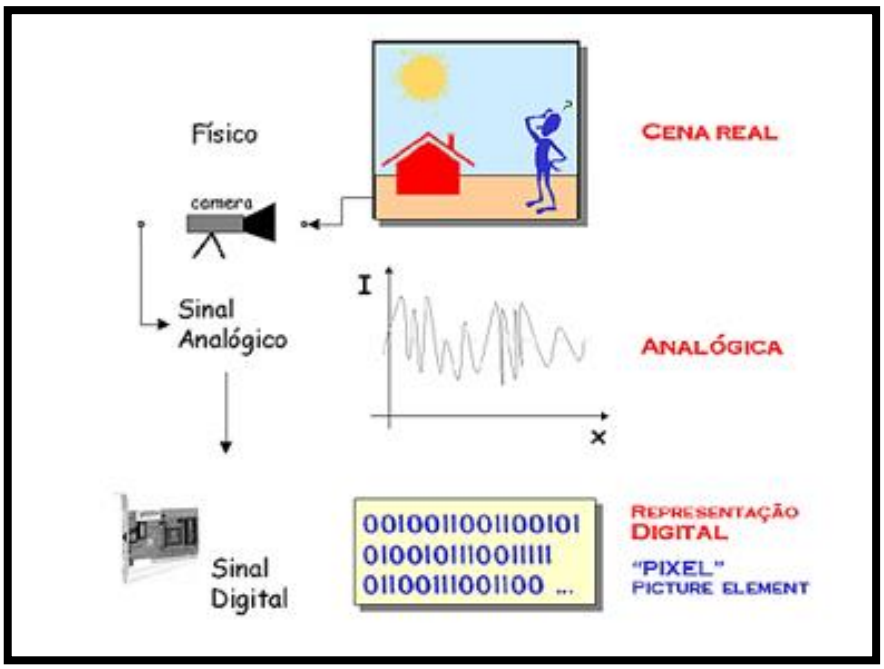

Figura 6 – Aquisição de Imagens
(In: VASCONCELOS, 2013)

2.3.1.2 Pré-Processamento

O Pré-Processamento, consiste em melhorar a imagem de forma a aumentar as chances de sucesso dos processos seguintes (BATISTA et al., 2013). O processo mais simples para esta fase consiste em duas técnicas: para monocromatização da imagem (transforma-la em escala cinza) e a binarização que consiste em transformar a imagem em preto e branco para diminuir o número de possibilidades das cores de cada *pixel* de quase 17 milhões para 2 (0 ou 1). A Figura 7 mostra este conceito:

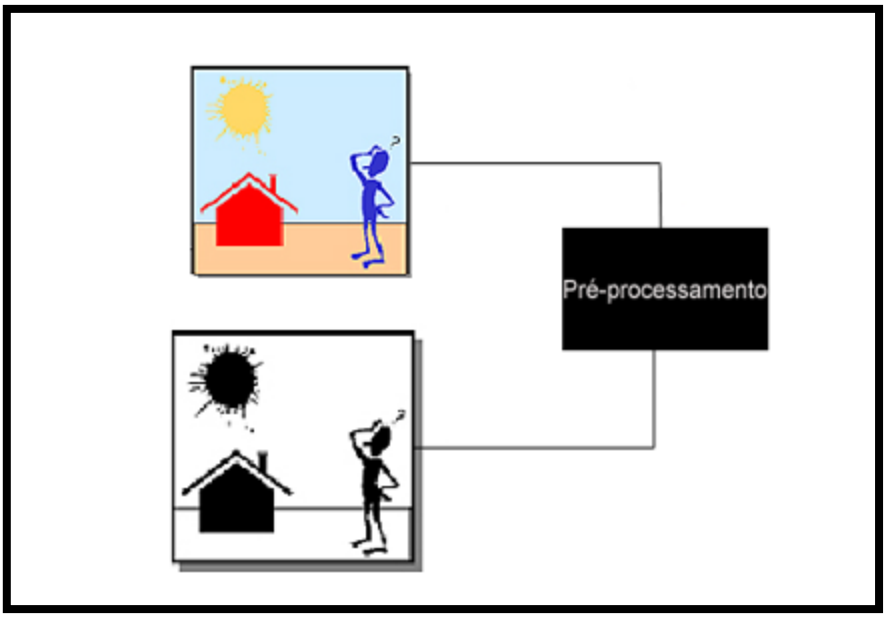

Figura 7 – Pré-processamento da Imagem
(In: VASCONCELOS, 2013)

2.3.1.3 Segmentação

A segmentação consiste no processo de separar na imagem os objetos a serem analisados. É usual denominar objetos da imagem os grupos de *pixels* de interesse, ou que fornecem alguma informação para o processamento digital de imagem (SERRANO, 2010).

Para analisar uma imagem no PDI, é comum desprezar o “fundo” da imagem, ou seja, só trabalhar com os objetos. Seguindo o conceito de que a imagem já foi totalmente preparada no pré-processamento, obtém se uma imagem onde foi corrigido seu brilho, contraste, iluminação e também sem ruídos. Com isso, fica fácil anular seu fundo, já que a imagem pré-processada é totalmente uniforme (com o mesmo valor). Existem dois tipos de segmentação: baseados em pixel, ou baseados em formatos.

- **Baseados em Pixel**
  - Segmentação de Intensidade;
  - Segmentação de Cores.
- **Baseados em Formatos**

- Detecção de bordas;
- Detecção de descontinuidades;
- Detecção de pontos;
- Detecção de linhas.

A fase da segmentação é a etapa do PDI que deve ser mais eficiente, pois a partir dela que é feita a extração de atributos (NETO; FILHO, 1999).

2.3.1.4 Extração de atributos

A etapa da extração de atributos é a fase do que consiste em retirar características (informações) da imagem. Esta etapa retira da imagem informações como: tamanho de uma região, distância entre um *pixel* a outro, altura, área, perímetro, profundidade, entre outros. A Figura 8 ilustra esta etapa:

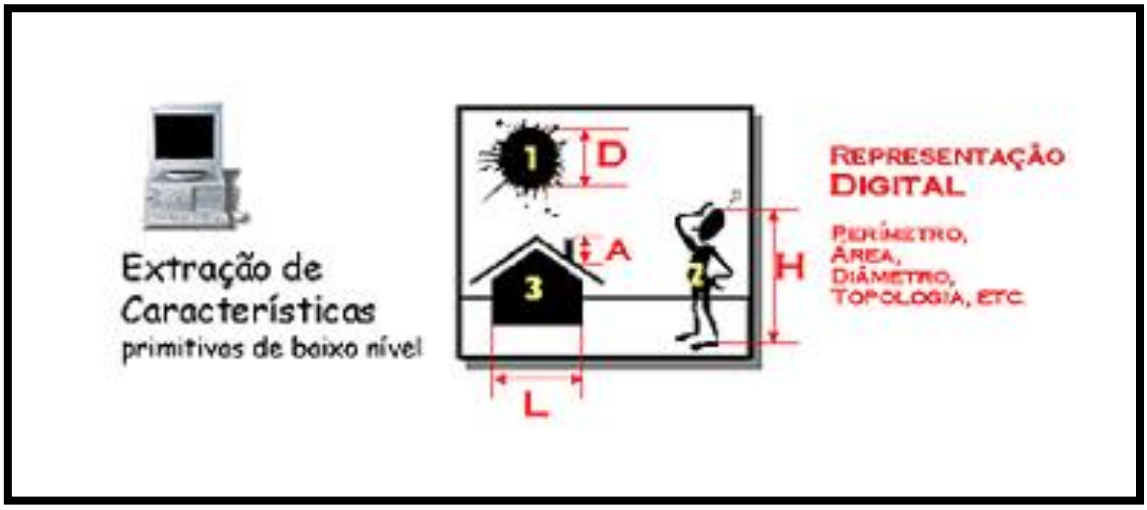

Figura 8 – Extração de atributos
(In: VASCONCELOS, 2013)

2.3.1.5 Reconhecimento e Interpretação

A etapa de reconhecimento e interpretação é a fase que consiste em comparar de forma “automática” todos os atributos retirados na fase de extração de atributos com as informações já pré-cadastradas ou armazenadas no sistema. Nesta fase, a imagem já passou por todo processamento. A Figura 9 demonstra esta etapa e ilustra um simples exemplo de interpretação:

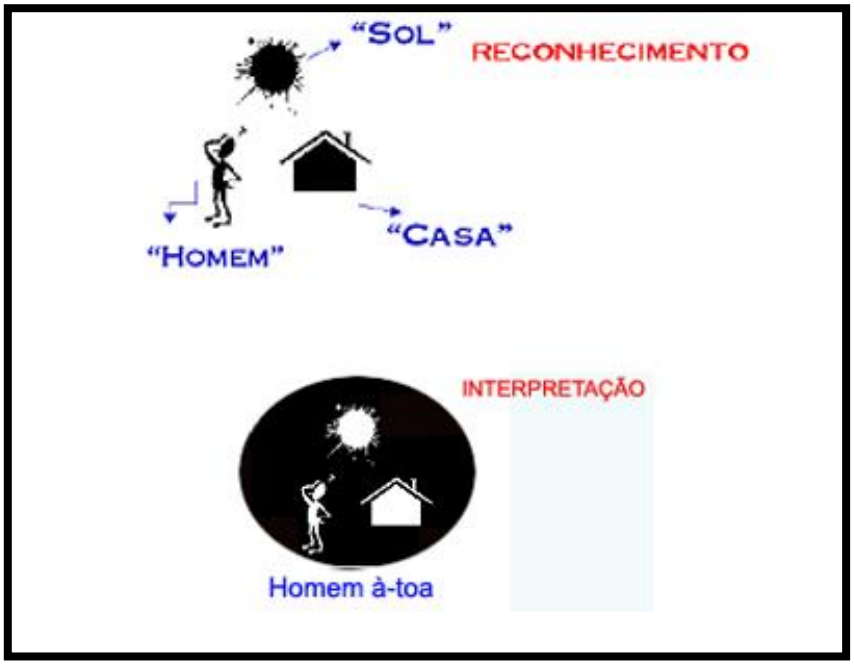

Figura 9 – Reconhecimento e Interpretação
(In: VASCONCELOS, 2013)

A partir da comparação, é feito o reconhecimento, resultando numa resposta satisfatória se for encontrado características similares existentes na imagem processada com alguma armazenada em banco, ou insatisfatória se não encontrar nenhum resultado. Existem dois tipos de reconhecimento: autenticação (um para um) e a busca exaustiva (um para vários).

Em sequência do reconhecimento pode ser feito uma interpretação, dependendo do objetivo do *software*. Existem aplicativos que não só reconhece o usuário, mais também verifica o estado que ele se encontra, feliz, triste, emotivo, doente, dependendo de qual padrão biométrico é utilizado, e aplicativos que reconhecem objetos, ambientes ou cenas.

No PDI, todas as fases são cruciais, mais assim como na Visão Computacional, a fase de pré-processamento, é uma fase muito importante, pois todos os passos a partir desta fase dependem da eficácia e qualidade do pré-processamento.

2.4 BIBLIOTECA OPENCV

Desenvolvida nas linguagens C/C++, no ano 2000 pela Intel, é uma biblioteca totalmente livre ao uso acadêmico e comercial, bastando seguir o modelo de licença da BSD Intel (*Berkeley Software Distribution*), para o desenvolvimento de sistemas com ênfase em Visão Computacional, possuindo interfaces de C, C++, C#, Python e *Java* e suporta Windows, Linux, *Mac OS*, *iOS* e *Android* como plataformas. Adotada em todo o mundo, a biblioteca tem mais de 47 mil pessoas da comunidade de usuários e número estimado de *download* superiores a 7 milhões.

A biblioteca *Open Source Computer Vision Library* (OpenCV), consiste de vários módulos de processamento de imagens e vídeos, estruturas de dados baseados em álgebra linear, interface gráfica do usuário, controle independente de mouse, teclado e periféricos, além de mais de 500 implementações de Visão Computacional como: filtros de imagem, calibração de câmera, reconhecimento de objetos, análise estrutural e outros. Por ser desenvolvida em C/C++, a biblioteca pode tirar proveito do processamento Multi-Core (OPENCV...2013).

Segundo Bradski (2013), a biblioteca *OpenCV* foi arquitetada para a eficiência computacional e com um forte foco em aplicações em tempo real, com o propósito de tornar a Visão Computacional próxima aos usuários e programadores nas áreas da computação e robótica. Um sistema implementado com *OpenCV*, ao ser executado, invoca sempre uma *Dynamic-Link Library* (DLL) que detecta o tipo de processador da máquina invocadora, realizando assim todas as otimizações de códigos. A biblioteca é dividida em cinco principais grupos de funções:

- *“cv.h”* – contém o básico de Processamento de Imagens e algoritmos de visão computacional;
- *“ml.h”* – é biblioteca de Aprendizagem de Máquina e Análise Estrutural, que inclui muitos classificadores estatísticos;

- *“cxcore.h”* – contém as estruturas de dados básicos;
- *“highgui.h”* – contém rotinas I/Oe funções para armazenar e carregar imagens e vídeos;
- *“cvaux.h”* – contém as áreas de reconhecimento facial e algoritmos experimentais.

Como citado nas seções anteriores, a grande preocupação para tornar para o desenvolvimento de um aplicativo ágil e eficaz é um estudo detalhado na fase de pré-processamento, mas o desenvolvimento de um aplicativo para um dispositivo móvel há também a apreensão quanto a taxa de processamento, limitação de memória e quantidade de processos ativos, preocupação esta, que não existe para o desenvolvimento de um aplicativo convencional. A união desta biblioteca com a plataforma *Google Android,* trará ao protótipo um benefício muito grande, pois a OpenCV trata de forma eficiente e rápida as imagens nas quais são submetidas a processamento, utilizando a menor quantidade de memória e taxa de processamento possível.

3 – RECONHECIMENTO DE PADRÕES E BIOMETRIA

Neste capitulo será exposto todos os conceitos necessários para o compreender o funcionamento de um sistema biométrico, bem como, suas características e funcionalidades. Será feita uma descrição dos assuntos de forma a fornecer toda a base de conhecimento exigida pela área estuda.

3.1 RECONHECIMENTO DE PADRÕES

O Reconhecimento de Padrões (RP) é o estudo das classificações de objetos em categorias ou classes. Há tempos existem pesquisas feitas para a implementação e elaboração de algoritmos que resolvam esses estudos. (SERRANO, 2010).

Esta área apresenta inúmeras opções de aplicações em várias áreas, principalmente na área tecnológica. Como exemplo, pode ser citada a classificação de documentos na *internet* (buscadores), sendo classificados por semântica, por categorias; classificação de dados multimídia (por exemplo, o *Windows Media Player* ou *Xbox Music*), que classificam as mídias por gênero, título, autor; e classificações biométricas, que são características físicas ou comportamentais, sendo individuais e pessoais que são capturadas de cada indivíduo e usadas também como forma classificatória.

Os sistemas de reconhecimento, usando padrões biométricos, vêm alavancando e motivando pesquisadores para estudo de algoritmos e formulas, para a melhora na lógica e desempenho destes sistemas. Consequentemente iniciou-se um novo estudo de padrões biométricos, chamado computação afetiva, que dá a um computador a capacidade de reconhecer e expressar emoções e empregar mecanismos que contribuem para a tomada de decisão racional (JAIN; DUIN; MAO, 2000).

Para um ser humano reconhecer algo, é muito simples e automático. O indivíduo captura características do objeto e do meio e compara com características armazenadas no cérebro, sendo assim retorna e identifica que tipo de objeto está tratando, ou observando. Na teoria, computacionalmente é da mesma maneira, captura-se características, do indivíduo em questão, armazena-se em um banco de dados todas as características, e ao receber outras características, realiza

comparações, retornando satisfatório se coincidirem as características ou insatisfatório se não coincidir as características. A Figura 10 mostra alguns tipos de padrões:

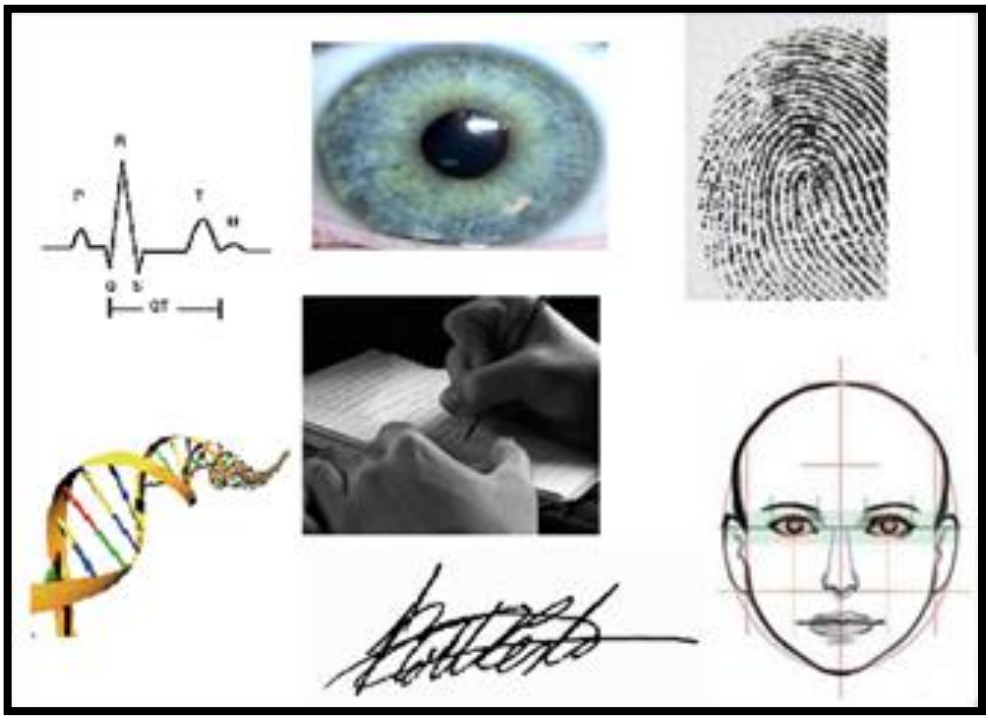

Figura 10 – Exemplos de tipos de padrões utilizados.

Implementar o conceito em um contexto computacional não é uma tarefa fácil. Pois para obter um resultado satisfatório é exigido não só um algoritmo de comparação e reconhecimento, mas também, conceitos de processamento e armazenamento de imagens, exemplificados e explicados nas sessões anteriores, pois a imagem a ser capturada e a imagem previamente salva no banco de dados, devem estar totalmente processadas e prontas para comparações e análises, submetidas a vários filtros e correções, sendo que a comparação é utilizada para decidir a similaridade entre as entidades do mesmo tipo (ROCHA; NITTO, 2011).

3.2 BIOMETRIA

A Biometria é uma área que está dentro do Reconhecimento de Padrões, é o termo utilizado para designar a ciência que estuda as características físicas ou

comportamentais dos seres vivos, a fim de que estes possam ser manuseados (SERRANO, 2010).

O termo biometria vem do grego: *bios* (vida) e *metron* (medida), isto é, um procedimento automático de reconhecimento individual baseado em medidas biológicas (anatômicas e fisiológicas) e características comportamentais, a fim de identifica-las e abolir o uso de senhas ou cartões magnéticos (CAVALCANTE; BACCI; HOKAMA, 2005).

De acordo com Silva, Carvalho e Santos Neto (2010), atualmente a biometria é utilizada com o homem sob duas circunstâncias: voluntariamente para funções como: emprego, acesso a serviços especiais, informação ou proteção, e involuntariamente para identificação criminal e de controle das fronteiras.

Aeroportos, agências bancárias, urnas eletrônicas e até parques temáticos fazem uso da técnica de reconhecimento das características únicas de cada pessoa, dispensando o uso de qualquer objeto, ou que memorize algo para identificar-se. Partindo do princípio de que cada indivíduo tem sua característica biométrica única, a identificação do usuário é feita por meio dessas características em questão (SILVA et al., 2012).

Teoricamente, quaisquer características humanas, podem ser utilizadas para a identificação de pessoas, desde que satisfaçam os cinco principais requisitos:

- **Universalidade**: significa que todas as pessoas devem possuir a característica;
- **Unicidade**: indica a característica não pode ser igual em pessoas diferentes;
- **Permanência**: significa que a característica não deve variar com o tempo;
- **Coleta**: indica que a característica pode ser medida quantitativamente;
- **Aceitação**: significa que o padrão biométrico a ser usado pelo sistema, deve ser de fácil captura e aceitação pelos usuários.

Atualmente, que devido à evolução tecnológica, surgi outra característica fundamental na qual vem sendo vista como mais um requisito: *a Proteção*, referindo-se que o

padrão biométrico e o sistema a serem utilizados, dificultem a forma de fraudar. Na pratica nenhum padrão, atende com perfeição todos os requisitos, mas consegue-se resultados satisfatórios e muito mais eficazes em comparação a tecnologias convencionais, como senha e cartões magnéticos.

3.2.1 Tipos de Biometria

O ser humano possui diversas características singulares que podem ser utilizadas para sua identificação. Os tipos biométricos são normalmente classificados em duas categorias: fisiológicas e comportamentais.

3.2.1.1 Categoria Fisiológica

Esta categoria define as características físicas contidas em cada indivíduo. Como citado anteriormente, qualquer característica fisiológica pode ser considerada uma característica biométrica e utilizada para implementação e identificação de uma pessoa, basta atender os cinco principais requisitos. Seguem algumas das principais características biométricas utilizadas para reconhecimento e identificação de pessoas:

- **Impressão digital:** é a forma mais simples e comum de identificação. Além de ser o processo mais rápido é também o que exige menos recursos, porém, a confiabilidade neste método é bem baixa. As impressões digitais são únicas para cada dedo de uma pessoa, incluindo os gêmeos idênticos. Este padrão identifica as cristas e vales de fricção da digital de um indivíduo. A partir da detecção, o algoritmo realiza cálculos de distancias, áreas e medidas dos pontos encontrados e faz uma análise entre a digital armazenada e a fornecida atual identificando ou não o indivíduo. A Figura 11 demonstra o padrão biométrico com seu dispositivo de captura:

Figura 11 – Padrão biométrico digital

- **Reconhecimento facial:** um dos métodos de identificação menos vantajosos para identificação simples, ou seja, controle de acesso, pois sua baixa confiabilidade em união com o alto custo computacional para leitura, reconhecimento e pesquisa faz deste recurso um dos menos utilizados para este tipo de acesso, entretanto, quando necessita-se de detecções e reconhecimento em sistemas em tempo real, como estádios, aeroportos, festas, este método é o mais seguro e eficaz. O grande problema para este tipo de padrão biométrico é a alta taxa da necessidade de filtros de processamento de correções de imagens, pois nem sempre as imagens capturadas estão com luminosidade adequada para realizar os métodos e processos que este padrão exige. O método de identificação mais utilizado para este padrão biométrico é o processo que calcula as distancias entre olhos, distância entre as mandíbulas, entre outras nuanças. A Figura 12 ilustra a face de uma pessoa, com os pontos mais importantes em serem detectados e calculados:

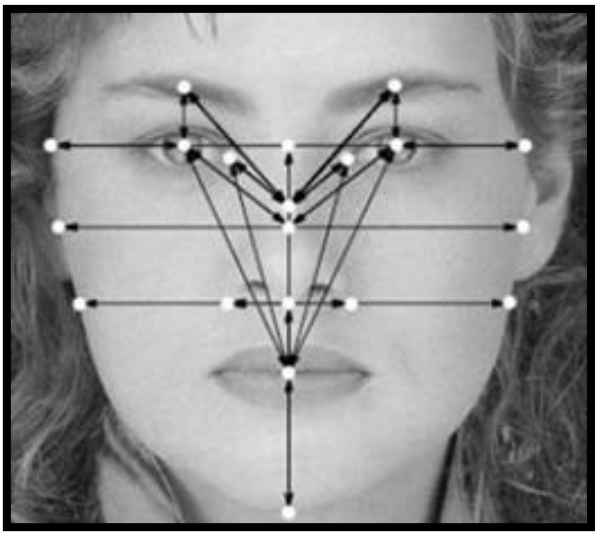

Figura 12 – Padrão biométrico facial
(In: VASCONCELOS, 2013)

- **Geometria da mão:** menos confiável do que a impressão digital, o reconhecimento por meio da geometria palmar enfrenta problema quanto ao correto posicionamento da mão na superfície do scanner, sem falar nos anéis e acessórios que muitas pessoas utilizam. Este processo, semelhante ao de impressão digital, identifica as minúcias das mãos de uma pessoa. Este padrão é muito utilizado em caixas-eletrônicos de bancos.
  A Figura 13 ilustra o dispositivo de captura deste padrão biométrico:

Figura 13 – Padrão biométrico geometria das mãos

- **Retina:** a reconhecimento de uma pessoa por meio da retina já é muito utilizada em sistemas de alta segurança. É totalmente confiável e a retina também é imutável, isto é, não muda com o passar o tempo. Os problemas ficam por conta da dificuldade na obtenção de uma imagem da retina e no alto custo financeiro exigido para a implantação deste sistema biométrico.
- **Íris:** embora exija equipamentos de alto custo para identificação, a íris é uma característica muito confiável para reconhecer as pessoas. É imutável com o passar dos anos e quase impossível de ser clonada, além de substituir o padrão biométrico da retina, pois seus níveis de segurança são equivalentes. A Figura 14 demonstra o padrão biométrico, bem como seu dispositivo de captura:

Figura 14 – Padrão biométrico íris

3.2.1.2 Categoria Comportamental

Cada usuário reage de modo diferente em determinadas situações. Alguns choram, outros ficam agressivos, tudo depende das características comportamentais de cada um. Existem estudos complexos sobre as atitudes tomadas pelas pessoas (SILVA et al., 2012).

Os cientistas apostam que, em um futuro muito próximo, seja possível destacar uma pessoa no meio de uma multidão, apenas pelo seu jeito de andar, mexer as mãos ou reconhecendo alguma “mania” dela. Este tipo de análise recebe o nome de biometria comportamental e, mesmo que ainda não haja sistemas profissionais em operação para este tipo de reconhecimento, as pesquisas não param.

A característica comportamental mais empregada atualmente é a assinatura, mesmo não sendo implementada com o foco em pesquisa de um comportamento de um indivíduo, este padrão biométrico é muito empregado em bancos e cartórios para a identificação e legitimidade entre assinaturas. Outro padrão biométrico comportamental que vem sendo muito pesquisado é a voz. Muitos cientistas estão empregando este modelo para identificar o humor de um indivíduo: triste, alegre, calmo. Esta pesquisa vem contribuindo para os estudos da computação afetiva (JAIN; DUIN; MAO, 2000).

3.3 SISTEMAS BIOMÉTRICOS

Um sistema biométrico, de acordo com Canedo (2013) é, essencialmente, um sistema de reconhecimento de padrões que busca extrair o padrão mais característico de uma pessoa, armazená-lo para depois comparar como novas amostras e determinar a identidade de cada usuário dentro de uma população. Assim sendo, um sistema biométrico típico tem pelo menos cinco componentes principais:

- **Componente de apresentação e captura de dados biométricos**: o traço biométrico precisa ser apresentado a um sensor, que irá transformar a informação em uma representação digital (foto, vídeo, áudio);
- **Componente de processamento do dado biométrico e extração do *template***: esse componente transforma a representação digital do traço biométrico em um *template*. Normalmente isso é feito por meio da segmentação da representação digital, avaliação (e melhoria) da qualidade e extração de características únicas. O processo usado para extração é muitas vezes patenteado e sempre mantido em segredo pelas empresas, pois este processo é um dos mais fundamentais do sistema.

- **Componente de armazenamento do *template***: o armazenamento do *template* é um componente que apresenta crescente desafio e muitas opções de implementação. Os *templates* podem ser armazenados de três formas em sistemas biométricos: no sensor ou computador local, repositório central (bando de dados) ou cartão.
- **Componente de comparação de *templates* e decisão**: na comparação de *templates* os valores matemáticos das características extraídas dos traços biométricos são comparados para determinar o seu grau de correlação ou similaridade, num processo chamado de *matching* (comparação);
- **Canal de transmissão**: o canal de transmissão é um componente comum a todos os outros sistemas de informação. E assim como em outros sistemas, nos sistemas biométricos ele é muito importante para definir a arquitetura do sistema, a política de segurança e privacidade.

A Figura 15 ilustra todo conceito definido anteriormente, desde a captura da imagem até seu armazenamento e, em seguida, a inserção de um novo modelo (*template*) para realizar a comparação e tomada de decisão:

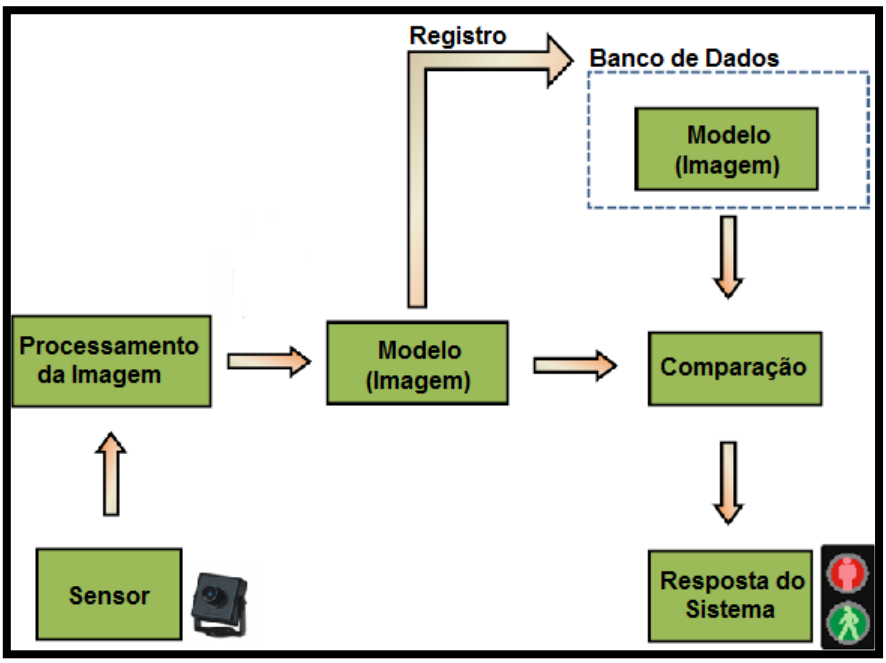

Figura 15 – Arquitetura de um Sistema Biométrico
(In: ROCHA; NITTO, 2012)

O modelo ilustrado na Figura 15 fora parcialmente trabalhado e implementado no Projeto de Iniciação Cientifica (PIC), dividido em dois anos, desenvolvido nesta instituição e apresentados nos Fóruns de Ciência e Tecnologia dos respectivos anos, sendo que a primeira fase, (ROCHA; NITTO, 2011), tinha o objetivo de conhecer todo o processo de tratamento e armazenamento da imagem, já o segundo no ano do projeto, (ROCHA; NITTO, 2012) com o foco nos estudos de processos de reconhecimento facial e implementação de detecção de faces humanas.

Este trabalho, como descrito em seus objetivos, prima utilizar os conceitos estudados durante os anos de desenvolvimento do PIC, fazendo uma abordagem e utilização de uma biblioteca especifica para tratamento e processamento de imagens, unindo com os conceitos estudados nos anos anteriores, com a robusta plataforma móvel *Google Android*, desenvolvendo assim, um protótipo mais alentado e voltado com a tecnologia atual.

3.3.1 Métodos de Reconhecimento

Os sistemas biométricos realizam o reconhecimento de forma automática e pessoal de duas maneiras: a Autenticação e a Identificação (conhecida como Busca Exaustiva).

3.3.1.1 Autenticação

É um tipo de comparação onde é feito a partir da imagem processada, gerada atual, com somente uma imagem de dentro do banco. Também conhecido como *“processo de 1 pra 1”*, o método é muito utilizado para pontos eletrônicos, pelo qual o usuário antes de realizar a comparação digita seu código ou ID (Registro de Identificação). Ao digitar seu código, já previamente cadastrado no sistema, o mesmo retornará todos seus dados, inclusive a imagem a ser comparada. Em seguida, o usuário fornece seu padrão biométrico (digital, face, íris, entre outros). Capturado por algum dispositivo, o *template* fornecido pelo usuário, passa por todo processamento é submete-se a

comparação com o *template* já cadastrado. Caso os padrões coincidem, o sistema retorna uma mensagem satisfatório.

A Figura 16 ilustra este tipo de reconhecimento:

Figura 16 – Autenticação (1 para 1)
(In: ROCHA; NITTO, 2011)

De acordo com Chiachia, Penteado e Marana (2011), este processo, elimina o uso de senhas, ou cartões, assim fornecendo ao usuário uma confiabilidade exponencialmente maior, em comparação aos métodos convencionais. Um dos pontos positivos deste método é a eliminação de fraudes, pois mesmo submeter a uma autenticação uma fotografia de um usuário cadastrado, o método não reconhecerá a foto do usuário cadastrado, pois os algoritmos e métodos de identificação e reconhecimento, além de reconhecerem distancias, medem também a profundidade da imagem submetida.

3.3.1.2 Identificação

É o tipo de comparação conhecido como *"1 pra N"*, em que o usuário não fornece nenhum código antes do reconhecimento, fornecendo somente o padrão biométrico.

A partir deste ponto, realiza-se uma busca entre o padrão biométrico fornecido, com todos os existentes e armazenados dentro do banco. A busca tem por consequência duas saídas: insatisfatório (se comparado todos os *templates* e não encontrar nenhuma característica do padrão fornecido, com os salvos em banco) ou satisfatório (se encontrar alguma característica semelhante, com os padrões já cadastrados). A Figura 17 demonstra o método, ilustrando uma face a ser verificada e identificada com as demais imagens já cadastradas em banco de dados:

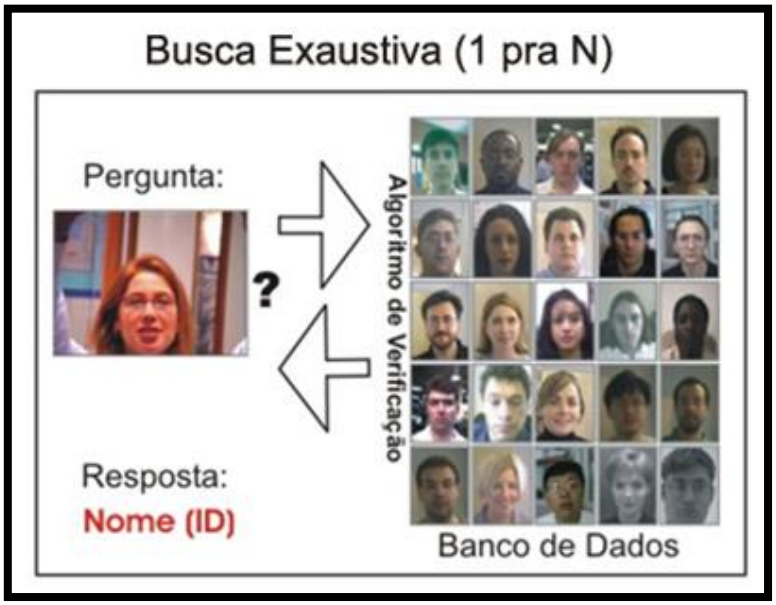

Figura 17 – Identificação (1 para N)
(In: ROCHA; NITTO, 2011)

Devido seu alto padrão de eficácia e segurança, este processo será utilizado nos estádios da Copa do Mundo de 2014 e nos ginásios das Olimpíadas de 2016, inibindo assim, violência entre torcedores e atentados terroristas. Pois uma vez cadastrado, o usuário será constantemente monitorado por câmeras de alta qualidade e resolução, e caso aconteça algo inesperado no evento, será fácil identificar o autor da ação.

A grande dificuldade deste processo é a exigência de equipamentos sofisticados e de custo alto, pois para obter um padrão elevado de identificações precisas, é necessário muitos métodos de processamento de imagens, obrigando a compra de computadores com níveis de processamento alto, e também os *templates* armazenados e comparados em tempo real, devem ser de alta qualidade, forçando a obtenção de câmeras de alta qualidade (CHIACHIA; PENTEADO; MARANA, 2011).

4 – TECNOLOGIA *GOOGLE ANDROID*

Neste capitulo serão apresentadas todas as características e funcionalidades principais da tecnologia *Google Android*, bem como seus aspectos positivos em comparação a outras plataformas de dispositivos móveis.

4.1 INTRODUÇÃO AO *ANDROID*

O mercado de dispositivos móveis vem crescendo admiravelmente. De acordo com Ávila (2013), a penetração de celulares no mundo é de 87%. De acordo com a pesquisa, o serviço de telefonia móvel vem barateando gradativamente, existem atualmente cerca de 8 bilhões de linhas móveis ativas no mundo, ou seja, praticamente existe um aparelho móvel para cada pessoa no mundo, incluindo crianças.

O que antes eram diferenciais, só em aparelhos de custo elevado, hoje em dia, usuários comuns estão buscando cada vez mais celulares com recursos como câmeras de alta definição, musicas, *bluetooth*, capacidade para jogos, GPS, acesso à internet e principalmente a redes sociais e TV digital (LECHETA, 2013, p. 20).

Na área corporativa a procura por dispositivos móveis, como tabletes ou smartphones, vem crescendo também. Diversas empresas procuram incorporar suas aplicações convencionais para dispositivos móveis, tornando seus negócios mais ágeis e flexíveis e alimentando cada vez mais o conceito “mobilidade” (FARTO, 2010).

Para acompanhar o nicho de mercado e a evolução tecnológica, satisfazendo usuários, corporações e desenvolvedores, o *Android* é uma resposta da Google para ocupar este espaço. Atraindo a atenção de milhares de empresas e usuários, o anuncio da tecnologia causou muito impacto por ser uma plataforma Open Source e liderada pela Google, empresa revolucionaria e líder na internet. Além da Google, outras empresas de telefonia móvel investem e apoiam esta tecnologia, o grupo foi chamado de OHA (Open Handset Alliance).

4.2 *OPEN HANDSET ALLIANCE* E O *ANDROID*

Criado com a intenção de padronizar uma plataforma de código aberto e livre para dispositivos móveis, precisamente para atender as expectativas e tendências do mercado atual, a *Open Handset Alliance* (OHA), grupo formado por empresas referencias no mercado de celulares como entre eles: HTC, LG, Motorola, Samsung, Sony Ericsson, Toshiba, Strint Nextel, China Mobile, T-Mobile, ASUS, Intel, Garmun e liderado pela poderosa Google tem o objetivo de mudar a experiência de dispositivos móveis no mercado, construindo um moderna plataforma: única, aberta, moderna e flexível para desenvolvimento de aplicações a usuários e corporações, resultando na tecnologia *Google Android* (FARIA, 2010).

A tecnologia *Android*, é uma plataforma de desenvolvimento inovadora para aplicativos móveis, que contém um sistema operacional baseado em *Linux*, com uma interface visual rica, e capacidade de integração a diversas funcionalidades, como GPS, 3G e 4G, *Wi-Fi*, *Bluetooth*, câmeras de alta definição, entre outros. Outra característica forte da plataforma, é a capacidade de implementação em *Java* e todas suas funcionalidades e recursos.

De acordo com a OHA a plataforma *Android* dará as empresas de telefonia móvel (operadoras móveis, fabricantes de aparelhos e desenvolvedores) dará total alicerce para construção de dispositivos inovadores, software e serviços (OPEN ...2013).

De acordo com Haseman (2008, p. 12), o fato de existir uma plataforma única, padronizada e livre é uma vantagem essencial para a criação de novos dispositivos móveis. Outra vantagem enorme é que a licença *Android* é flexível, deixando os fabricantes bem à vontade para realizar alterações no código-fonte para customizar seus produtos e deixar o dispositivo mais seguro e rápido, sendo que o melhor de tudo é a não-obrigatoriedade do compartilhamento desta customização.

Segundo Lecheta (2013, p. 22), um ponto muito importante desta aliança e do tipo de licença nela solidificada, onde todo fabricante ou desenvolvedor de qualquer parte do mundo, pode contribuir com o projeto, assim evitando *“bugs”* e erros futuros, pois uma vez encontrado um problema, toda comunidade *Android*, se volta corrigindo todas as falhas.

4.3 SISTEMA OPERACIONAL LINUX

A sistema operacional *Android* foi fundamentada no kernel 2.6 do Linux, sendo responsável por todo gerenciamento de memória, processos, threads, segurança dos arquivos e pastas, monitoramento e instalação dos drivers e redes.

Toda a lógica de programação para o desenvolvimento de um aplicativo *Android*, se assemelha com a programação convencional, sendo que a única diferença crucial é a preocupação com a memória e processamento, pois uma vez que a plataforma executa em vários dispositivos de marcas e arquitetura diferentes, isto é, cada dispositivo tem suas peculiaridades (MEDNIEKS et al., 2012, p. 24).

Toda a vida útil de um aplicativo da plataforma *Android,* também se assemelha com a execução de um programa em um sistema operacional convencional. Cada aplicativo dispara ao sistema operacional um processo, alguns deles exibindo ao usuário diferentes interfaces, outros ficando em segundo plano.

Outra vantagem muito grande desta fundamentação é a possibilidade de diversos aplicativos sendo executados simultaneamente, sendo que o *kernel* do sistema operacional é o responsável por todo este gerenciamento e controle. Caso necessário o próprio *Android*, tem a possibilidade de tomar decisões em encerrar um processo, liberando memória e recursos a outros aplicativos, ou simplesmente pausa-lo, para posteriormente voltar a fase atual quando a situação estiver controlada.

4.4 MAQUINA VIRTUAL DALVIK

Para desenvolver um aplicativo *Android*, o desenvolvedor terá que utilizar *Java* como linguagem de programação, sendo a arquitetura não utiliza a JVM (*Java Virtual Machine*) como sua máquina virtual, mas uma máquina virtual otimizada para execuções em dispositivos móveis, chamada *Dalvik*.

Ao desenvolver aplicativos *Android*, o desenvolvedor contará com todos os recursos *Java*, mais quando o *bytecode* (*.class*) é submetido à compilação, ele é convertido para um formato (*.dex*), isto é, *Dalvik Executable*, que representa uma aplicação *Android* compilada.

Em seguida, todos os arquivos com extensão (*.dex*), outros recursos como imagens, sons e vídeos são compactados em um único arquivo com a extensão (*.apk*) – (*Android Package File*), que representa a aplicação final, pronta para ser instalada e distribuída a qualquer dispositivo que execute o sistema operacional *Android* (LECHETA, 2013, p. 20).

4.5 PLATAFORMA *ANDROID*

A Figura 18 demonstra toda arquitetura da plataforma *Android*. Será feita uma descrição detalhada de cada nível/camada da arquitetura, pois é de fundamental importância um entendimento detalhado para o desenvolvimento de um aplicativo mais eficiente.

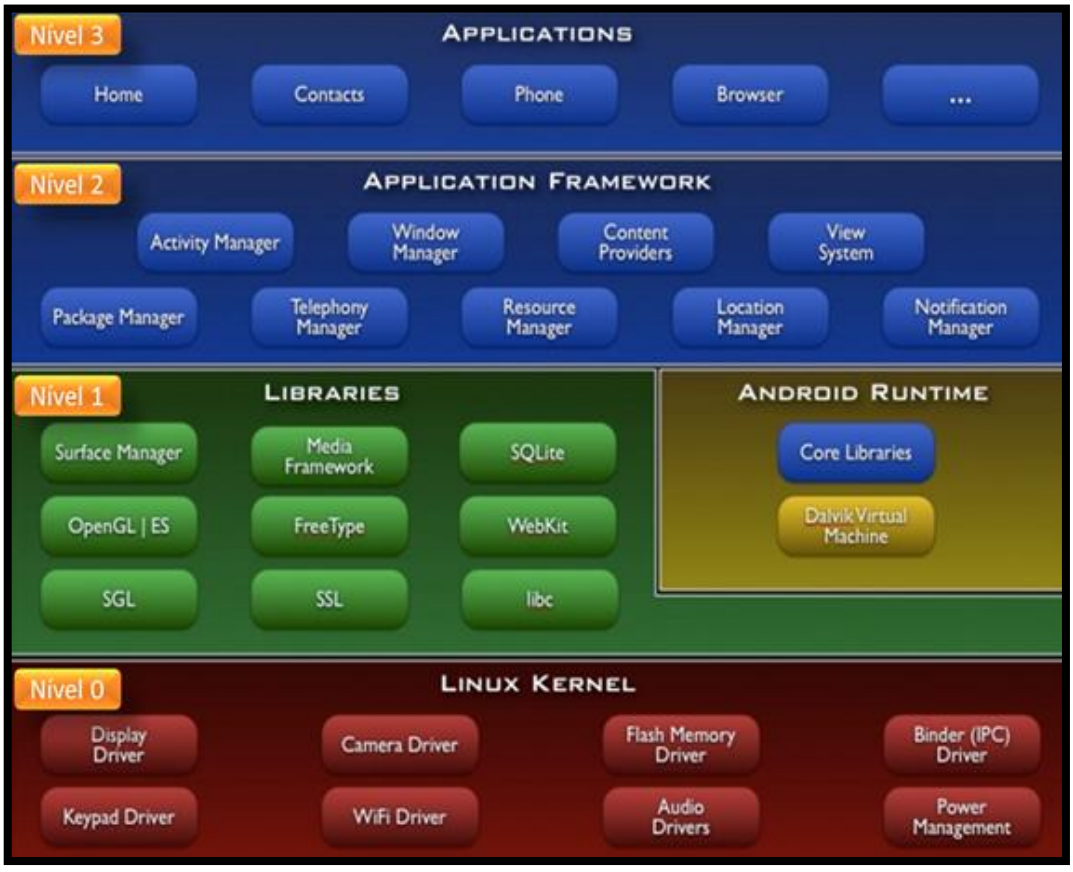

Figura 18 – Arquitetura *Google Android*
(In: PEREIRA; SILVA, 2009, p. 5).

A ilustração da arquitetura *Google Android*, foi dividida na ilustração da Figura 18 em níveis, sendo que o nível mais baixo presta serviço e da base ao nível acima, sendo que nível será discutido nas seções seguintes.

4.5.1 Camada *Linux Kernel*

Utilizando a versão 2.6 no *kernel* do robusto sistema operacional *Linux*, este nível é responsável pelos serviços centrais do sistema *Android*, como gestão de memória, processos, segurança e tarefas. Este nível também opera como uma camada de *middleware* entre as camadas das plataformas *Android,* abstraindo toda a complexidade do *hardware* (PEREIRA; SILVA, 2009, p. 9).

4.5.2 Camada *Android Runtime*

Cada aplicativo do *Android* executa seu próprio processo, instanciado previamente pela máquina virtual *Dalvik*. Os processos são executados no formato *Dalvik Executable* (.dex) e otimizados para ocupar uma pequena quantidade de memória. Esta camada tem o objetivo de controlar as bibliotecas centrais no núcleo (*core libraries*), e consequentemente controlar as instancias da máquina virtual *Dalvik* e seus processos otimizados (PEREIRA; SILVA, 2009, p. 8).

4.5.3 Camada *Libraries*

A arquitetura *Android* possui um conjunto de bibliotecas C/C++ usadas por diversos componentes. Tais bibliotecas permitem trabalhar com arquivos de mídia comuns como MPEG4, H.264, MP3, AAC, AMR, JPG e PNG. Componentes para elaboração de projetos em 2D como em 3D, inclusive, uma biblioteca cuja implementação foi baseada em *Open Graphics Library* (OpenGL), fornecendo acesso a praticamente todos os recursos do hardware de vídeo. Juntamente com esta camada, foi disponibilizado o *SQLite*, um poderoso e leve banco de dados relacional (PEREIRA; SILVA, 2009, p. 7).

4.5.4 Camada *Application Framework*

A arquitetura deste framework foi desenvolvida com o objetivo de simplificar a programação e a reutilização dos componentes, sendo possível qualquer desenvolvedor construir um aplicativo e permitindo que este seja utilizado por outros programas. Sendo assim, está camada é responsável dos disponibilizar todas as APIs e recursos necessários para pacotes e aplicativos (PEREIRA; SILVA, 2009, p. 6).

4.5.5 Camada *Applications*

É a camada onde se encontram todos os aplicativos *Google Android,* como clientes de e-mail, contatos, navegador web, aplicativos GPS. Todos os projetos implementados encontram-se neste nível (PEREIRA; SILVA, 2009, p. 6).

4.6 *ANDROID SDK*

O *Android* SDK é um *software* que fornece as bibliotecas da API e instrumentos de desenvolvimento necessários para construir, testar e depurar aplicativos para a plataforma móvel *Google Android* (LECHETA, 2013, p. 27).

A plataforma possui um emulador de simulação de um dispositivo móvel, fornecendo praticamente todas as funções que um celular ou *tablet*, dando ao desenvolvedor total condição de realizações de testes.

Apesar de o Android SDK possuir o emulador que é executado normalmente no computador, como um aplicativo convencional, a plataforma fornece um *plug-in* para o Ambiente de Desenvolvimento *Eclipse*, cujo o objetivo é integrar a programação em *Java* com o Emulador (LECHETA, 2013, p. 27).

A Figura 19 ilustra o emulador *Android*:

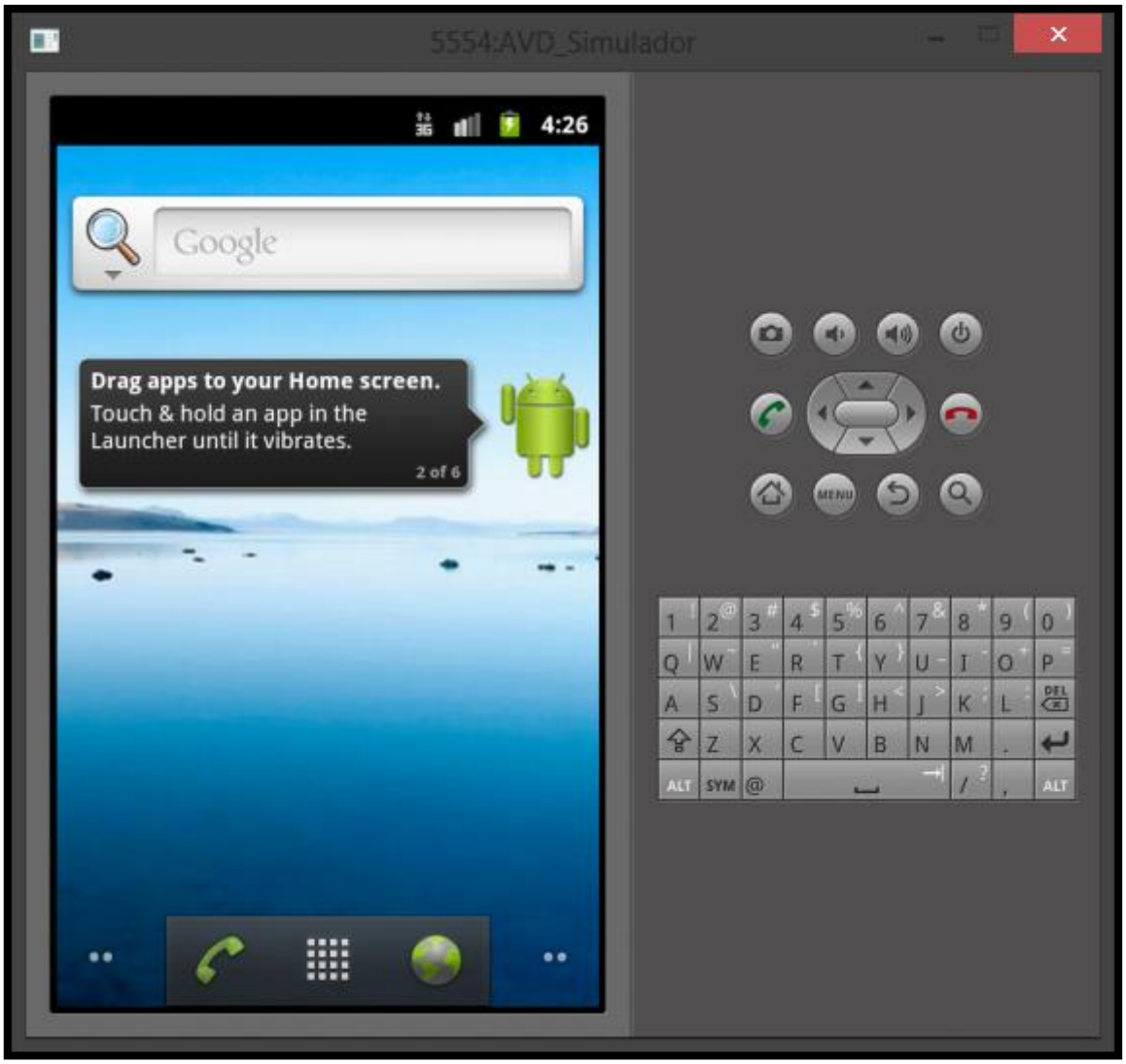

Figura 19 – Emulador *Android*

A partir do *plug-in,* é possível iniciar o emulador junto ao *Eclipse,* dando ao mesmo a capacidade de realizar depurações de no código de forma automática, como qualquer outra aplicação.

Atualmente, mas ainda como em fase de testes, a própria Google disponibilizou uma plataforma de desenvolvimento e depuração para aplicativos *Android* chamada *Android Studio,* cujo o objetivo é fornecer ferramentas integradas para o desenvolvimento de aplicativos na plataforma, e dar ao desenvolvedor uma vasta opção de edições de *layouts* e ferramentas de soluções rápidas, tudo já incluso dentro da suíte. O ambiente foi apresentado na última conferência anual da empresa, *Google I/O* (BARROS, 2013).

A Figura 20 ilustra o novo ambiente de desenvolvimento:

Figura 20 – *Android Studio*

De acordo com Barros (2013), a plataforma tem grande possibilidade de ascensão, pois é totalmente *Open Source e* está sendo desenvolvida por uma poderosa e potente empresa que dará ao produto mais benefícios e facilidades ao programador. Ainda que em fase testes, muitos desenvolvedores já estão migrando para este ambiente de desenvolvimento, pois poderão fazer sugestões de mudanças e indicações para a versão final, assim atraindo um público alvo de programadores ainda maior.

Uma das funções mais interessantes deste novo ambiente de desenvolvimento é a capacidade de exibir uma pré-visualização em tempo real do aplicativo em desenvolvimento, nos múltiplos formatos. A medida que se desenvolve, é possível avaliar como o layout da aplicação fica em smartphones e tabletes de vários tamanhos e resoluções, permitindo realizar ajustes e aproveitar espaços livres (no caso de telas maiores).

5 – PROPOSTA DO TRABALHO

Neste capítulo será apresentada a proposta do trabalho, delineando o problema e modelo proposto, bem como o fluxo de funcionamento e processamento de elementos faciais para comprovar a integração da plataforma móvel *Android* com a biblioteca *OpenCV*.

5.1 FUNCIONALIDADE DA APLICAÇÃO

A fase de detecção é uma das fases internas da etapa de extração das características, na qual exige uma taxa alta de processamento em comparação a outras fases do sistema de forma que haja a captura de todos os atributos, sendo necessária uma arquitetura bem definida para realizar a detecção dos padrões biométricos a serem analisados pelo aplicativo.

De maneira similar, todo sistema biométrico implementa uma arquitetura geral que é utilizada em qualquer sistema de padrões, onde a fase de detecção dos atributos é o ponto onde se caracteriza o tipo de padrão em que o sistema é implementado.

Esta fase consome muito processamento e memória da máquina e, por tratar-se de um dispositivo móvel, a preocupação eleva-se, pois mesmo com a gama de oportunidades e configurações altas atualmente encontradas em aparelhos móveis, ainda existe a necessidade de métodos eficazes para um consumo de pouca memória e um baixo nível de processamento.

Com vista em tais aspectos, a implementação do trabalho tem por objetivo desenvolver parte de um sistema biométrico capaz de realizar detecções de faces humanas em tempo real. A implementação baseia-se na etapa de extração das características físicas, conforme anteriormente ilustrada na Figura 15.

Para a implementação das funcionalidades propostas, fora elaborada a Figura 21 que ilustra a arquitetura básica do aplicativo móvel:

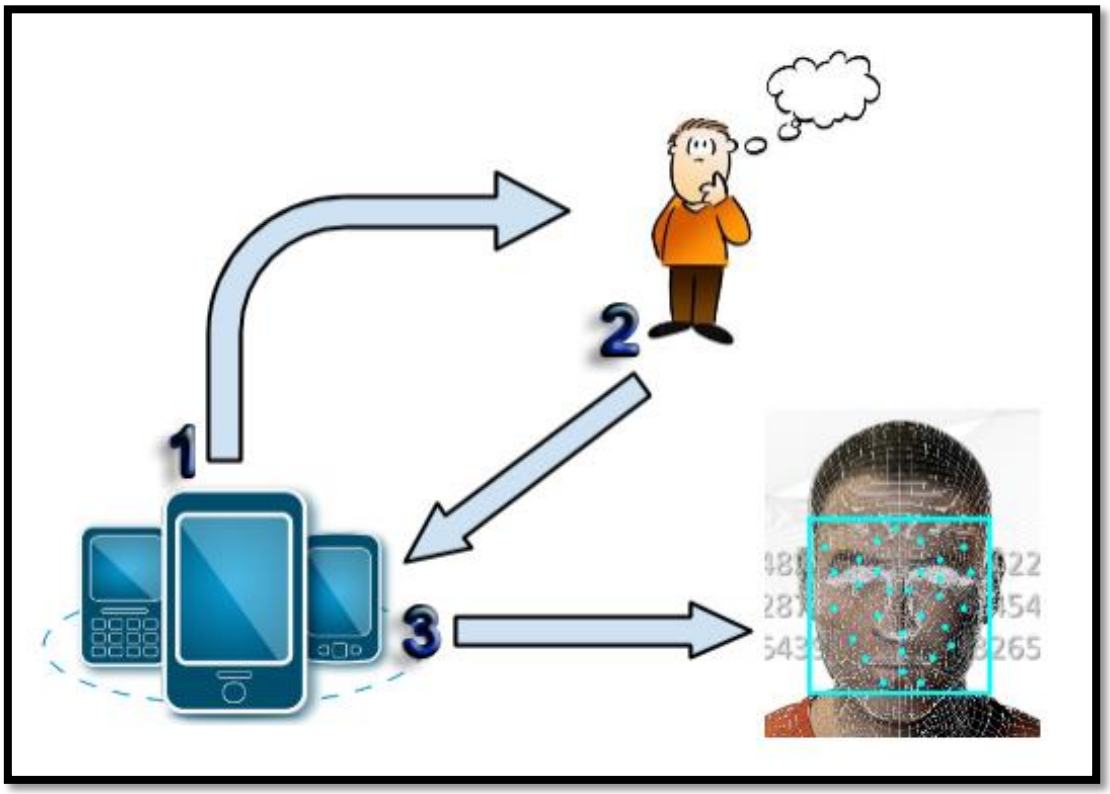

Figura 21 – Arquitetura geral do protótipo

A arquitetura proposta contribui para a implementação de um aplicativo simples que ajude na realização de trabalhos futuros e sirva como base para a construção de um sistema capaz de reconhecer faces humanas. Para realizar a detecção, ao iniciar o sistema (passo 1), o usuário deve fornecer quais padrões que o aplicativo deverá detectar tais como faces, olhos, nariz e boca e em seguida confirmar os atributos fornecidos (passo 2). Logo após a confirmação, o sistema será capaz de realizar detecções dos atributos selecionados pelo usuário (passo 3).

Certamente que, para realizar a implementação de um sistema completo capaz de realizar o reconhecimento facial, não será necessário o (passo 2), pois o sistema realizará de forma automática a detecção das faces nas imagens e de seus respectivos atributos. Por esse motivo, a implementação deste projeto contribuirá com pesquisas e desenvolvimentos futuros.

5.2 INTEGRAÇÃO COM A BIBLIOTECA OPENCV

A biblioteca OpenCV fornece o alicerce necessário para atender questões de processamento e cálculo das imagens de forma automática, visto que seu foco é o de implementar algoritmos de técnicas de visão computacional.

Outra grande contribuição para este projeto é a capacidade de integração da plataforma *Android* com a biblioteca OpenCV, aproveitando e usufruindo das características dos dois segmentos. Objetivando facilitar essa integração, far-se-á uso de métodos e técnicas *haarcascades*, recursos implementados e disponibilizados pela própria biblioteca OpenCV.

Um classificador *haarcascade* é um conjunto de classificadores, isto é, um arquivo formado por algumas centenas de *templates* com amostras de características faciais.

A aplicação dos *haarcascades*, implementados na biblioteca OpenCV segue o conceito de somente possuir dois tipos de retorno: verdadeiro, no caso de a região submetida aos testes for semelhante à amostra *haarcascade*, e falso, no caso de a região não ser semelhante. O classificador é gerado com a premissa de que ele pode ser prontamente redimensionado, com o objetivo de localizar as imagens e detectá-las em diferentes tamanhos, proporcionando uma eficácia maior do que o redimensionamento da imagem propriamente dita. Isso se deve ao fato de que, ao encontrar uma imagem de resolução e tamanhos diferentes, o classificador se auto ajusta para se assemelhar com o tamanho e resolução da imagem e em seguida realiza a detecção.

Para que exista a integração entre a plataforma móvel e a biblioteca é necessário que o dispositivo possua instalado o aplicativo *OpenCV Manager*, disponível no *Google Play*, mercado *online* de aplicações *Android*. O *OpenCV Manager* é a biblioteca OpenCV para o dispositivos móveis com *Android*, contendo todos os algoritmos da biblioteca, bem como um verificador de *hardware* que, ao ser instalado, é executado de forma automática, detectando o desempenho máximo que o dispositivo pode

oferecer para que, no futuro, ao necessitar de um alto processamento, a biblioteca saiba qual é o limite do aparelho.

A Figura 22 ilustra o fluxo de funcionamento, navegando pelos principais níveis que o projeto focaliza na implementação: *Android*, OpenCV e *haarcascade*:

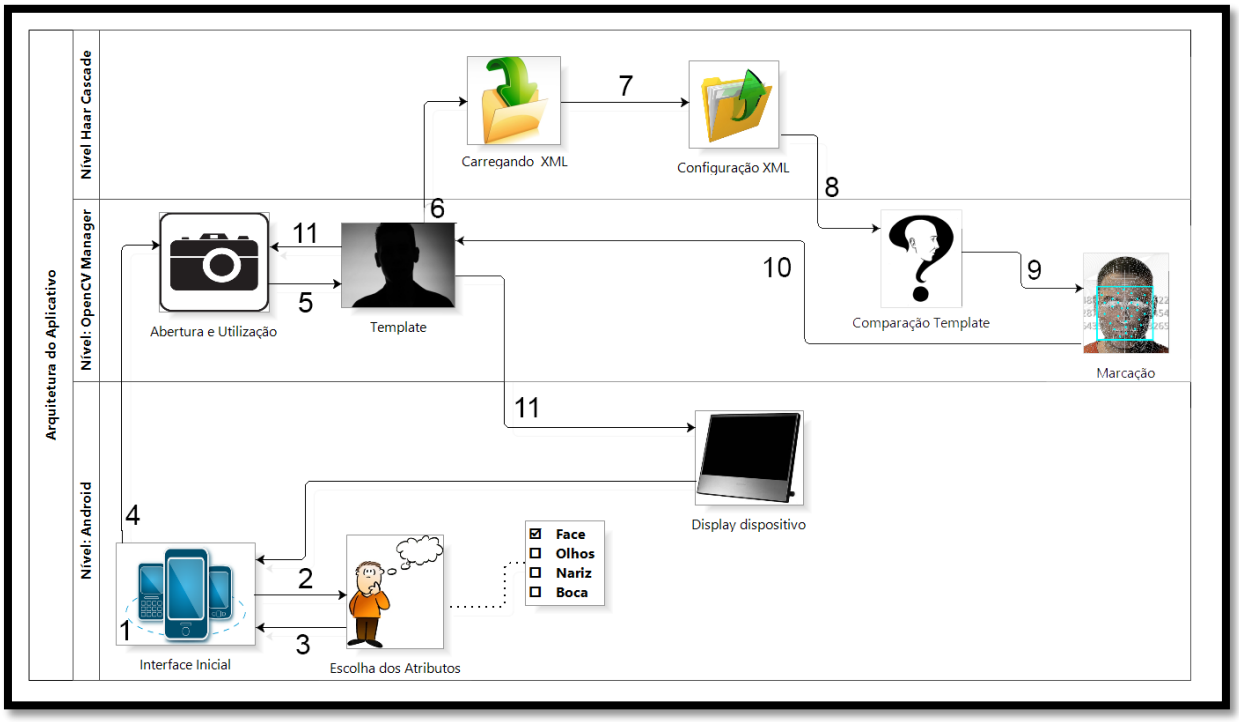

Figura 22 – Fluxo de funcionamento do aplicativo

O implementação deste projeto apresenta os seguintes passos:

- ***Passo 1:*** Interface Inicial, onde o usuário terá as opções de configuração e verificação de informações do protótipo.
- ***Passo 2:*** Escolha dos atributos, onde o usuário terá a opção de escolher quais atributos o aplicativo irá detectar.
- ***Passo 3:*** Ao voltar da interface de configurações, o usuário o aplicativo já estará apto a realizar as detecções.
- ***Passo 4:*** O aplicativo abrirá a câmera do dispositivo por meio da biblioteca.
- ***Passo 5:*** O sistema irá capturar o *template* (imagem).
- ***Passo 6:*** O aplicativo instanciará carregará o arquivo XML *haarcascade,* de acordo com a escolha no passo 2.

- ***Passo 7:*** O arquivo *haarcascade* se adaptará de acordo com o *template* e em seguida criará outro arquivo.
- ***Passo 8:*** o aplicativo realizará a comparação do *template* com o arquivo XML salvo no passo anterior.
- ***Passo 9:*** Caso verdadeiro, o aplicativo realizará a marcação no *template*.
- ***Passo 10:*** É realizado o retorno do *template* para classe invocadora.
- ***Passo 11:*** A imagem marcada (se verdadeiro) é transmitida no *display* do dispositivo. Ao mesmo tempo o fluxo volta ao fluxo no passo 5.

É de extrema importância dar ênfase nos passos 7 ao 9 onde o uso da biblioteca OpenCV fica evidenciada. O passo 7, etapa muito importante do fluxo do aplicativo, tem por objetivo configurar o arquivo XML, de acordo com o *template* (imagem) carregada pela câmera, colocando a imagem e o arquivo em resoluções e dimensões equivalentes, em seguida é criado um novo arquivo para que no próximo passo ser realizada a comparação do *template* com o arquivo salvo no passo 7.

Outro etapa de ampla relevância é a marcação dos atributos, fase localizada no passo 9, onde de acordo com a detecção (passo anterior), o aplicativo destaca os atributos detectados.

6 – ESTUDO DE CASO

Neste capítulo serão apresentados a especificação e a implementação do modelo proposto por este trabalho. O principal objetivo consistiu em definir e desenvolver um protótipo de um aplicativo para dispositivos móveis na plataforma *Google Android* capaz de realizar detecção de faces humanas, integrando a robusta plataforma de dispositivos móveis com a pujante biblioteca OpenCV, fornecendo maior estabilidade, segurança e performance ao aplicativo.

6.1 DEFINIÇÃO DO PROBLEMA

A definição do problema iniciou-se com um estudo de caso para a fixação do escopo. Para o desenvolvimento deste estudo foram utilizados conceitos de padrões de projeto para a criação do protótipo na plataforma *Google Android,* bem como um estudo aprofundado para a integração da biblioteca OpenCV. Com o ambiente devidamente configurado e preparado para a implementação, o enfoco do problema fora o de desenvolver um modelo de aplicativo móvel capaz de realizar detecções de faces humanas assim como outras partes da face: olhos, nariz, boca e outros, utilizando conceitos de reconhecimento de padrões e biometria, visão computacional e processamento e análise digital de imagens estudados nos capítulos anteriores.

6.2 CONFIGURAÇÃO DO AMBIENTE DE DESENVOLVIMENTO

Uma das principais dificuldades encontradas para realizar a integração *Google Android* e *OpenCV* é a configuração do ambiente de desenvolvimento. Para contribuir com trabalhos futuros, esta seção tem objetivo demonstrar os passos para a configuração do ambiente de aplicação.

Para que a integração seja realizada, torna-se necessário instalar e configurar o ambiente de desenvolvimento. Para a implementação deste projeto fora utilizado a

Integrated Development Environment (IDE) *Eclipse* na versão Indigo, devidamente configurada para a programação *Java* e *Android,* bem como seus *plug-ins* para desenvolvimento móvel *Google Android.*

O primeiro passo, para a integração, é realizar o *download* do projeto com as bibliotecas *OpenCV* que serão adicionadas dentro do ambiente de desenvolvimento. O *download* é de fácil acesso e está disponível no *site* oficial da biblioteca *OpenCV.* O nome do pacote de *download* é *OpenCV-2.4.3.2-android-sdk.*

Em seguida, é necessário descompactar o projeto baixado adicionando-o na *workspace* do ambiente de desenvolvimento, diretório onde ficam salvos todos os projetos desenvolvidos. A Figura 23 ilustra o primeiro passo para a integração:

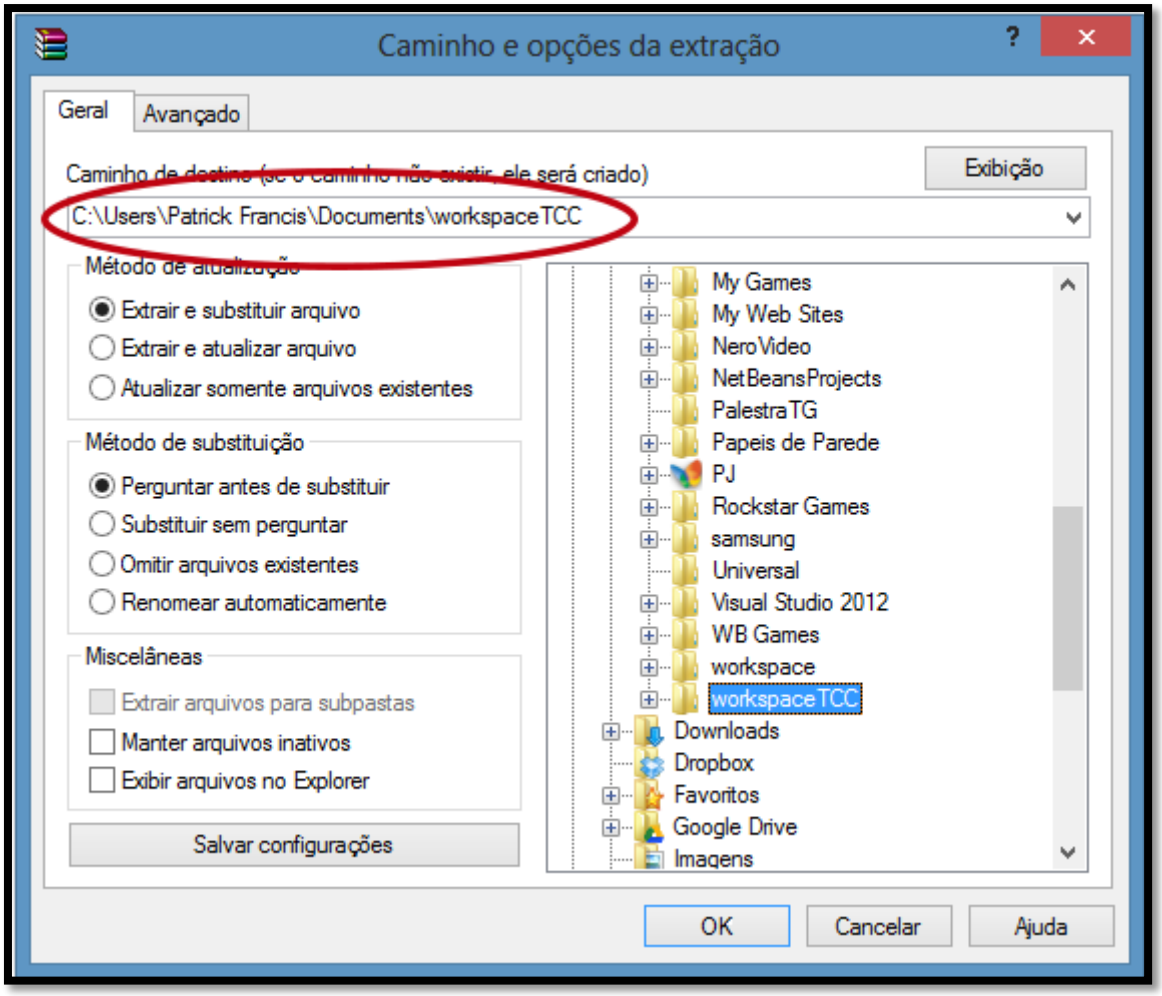

Figura 23 – Integração *OpenCV*: Descompactação biblioteca

É de extrema importância adicionar o projeto dentro do *workspace* pois o arquivo descompactado será adicionado como um projeto dentro da IDE. Caso seja descompactado em outro local, será necessário manter o endereço para sua utilização nos próximos passos.

Logo após é realizado a importação do projeto dentro do *Eclipse*. Em seguida é necessário verificar se a versão *Android* em que o projeto está configurado esteja em 3.0 ou acima para que não ocorra problemas futuros na integração. A Figura 24 ilustra o procedimento:

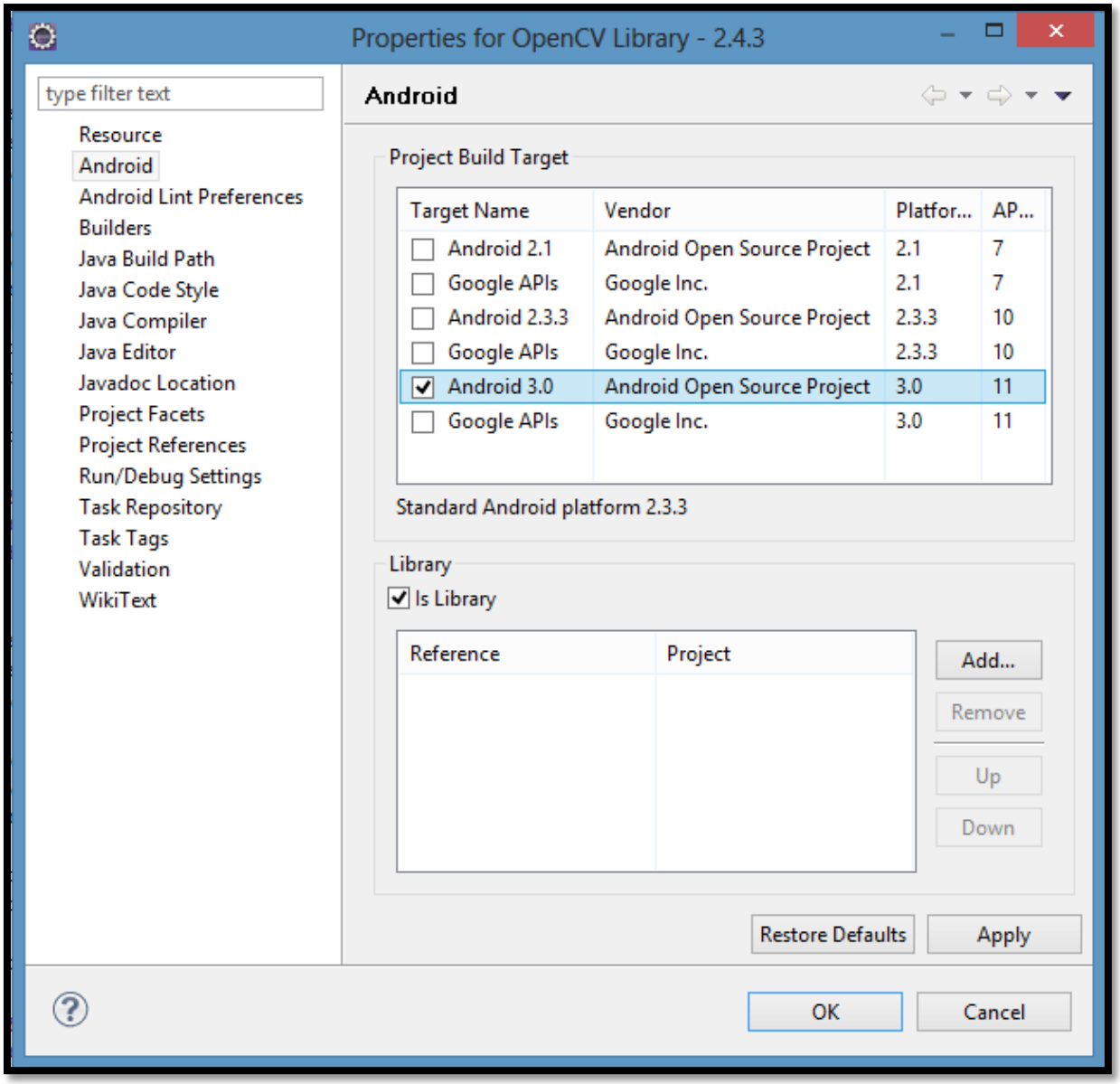

Figura 24 – Integração *OpenCV*: Verificação da versão *Android*

Para abrir a interface de propriedades do projeto basta selecionar o projeto e apertar os botões (*Alt + Enter*) ou clicar com o botão direito do *mouse* sobre o projeto e em seguida clicar em *Properties.*

Após realizado os procedimentos, conclui-se o primeiro passo da integração. A próxima etapa é criar um novo projeto com o OpenCV já integrado, para isto é necessário saber o endereço do arquivo NDK *Android*, e em seguida realizar uma cópia do projeto descompactado no *workspace* para a pasta raiz onde se localiza o arquivo NDK *Android.* A Figura 25 ilustra o requisito:

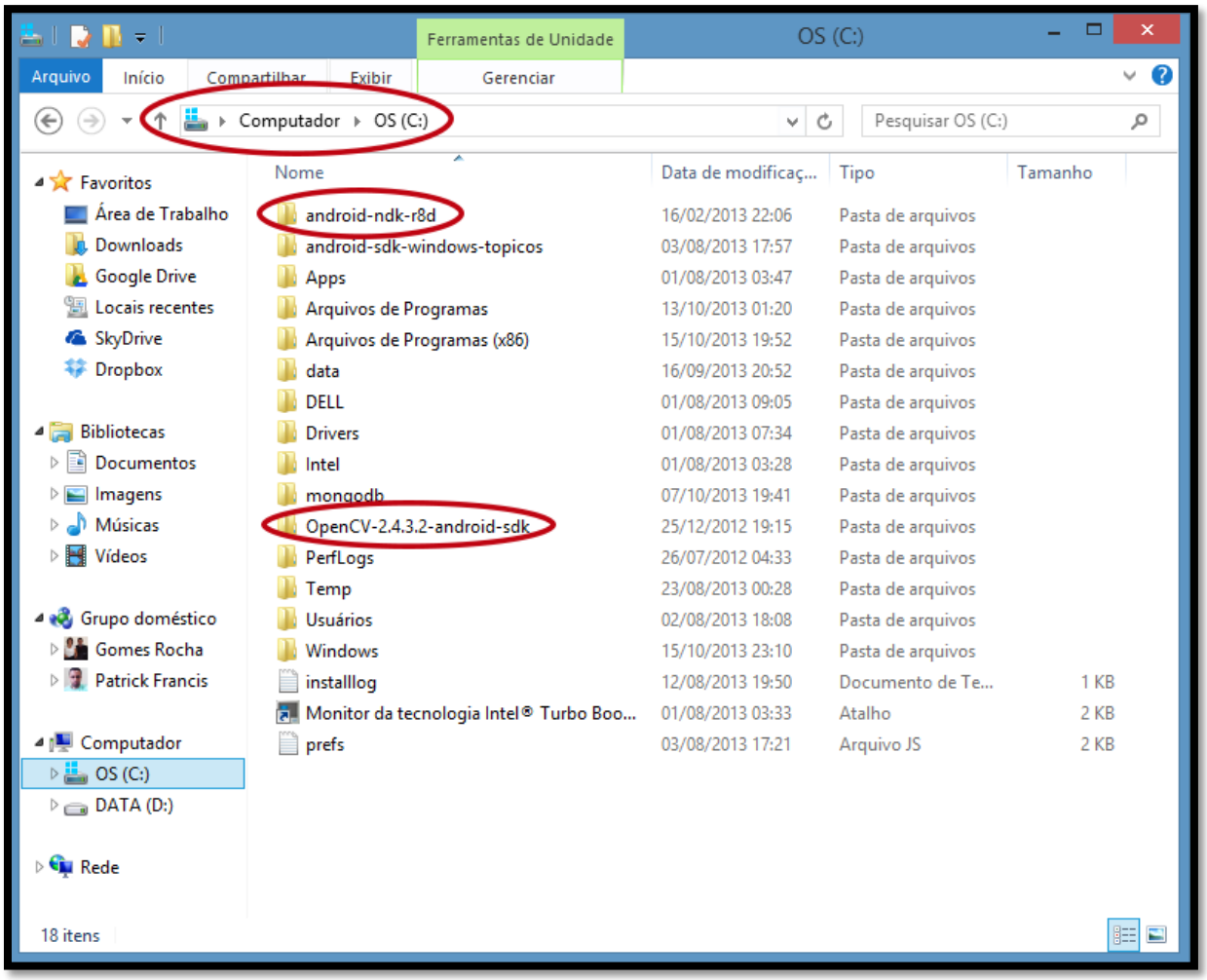

Figura 25 – Integração *OpenCV*: Configuração *Android* NDK e OpenCV

Outro requisito de grande importância, é que o caminho onde se encontra a pasta do *Android NDK* e a biblioteca *OpenCV*, deve ser curto. A solução para este trabalho foi alocar as duas pastas dentro da raiz da partição *C:* como ilustrado na figura.

A próxima etapa é criar novo projeto no *Eclipse* seguindo os seguintes passos: *File / New / Android Aplication Project.*

Após a criação e configuração do novo projeto é necessário realizar a importação da biblioteca *OpenCV* dentro das configurações *Android* no desenvolvimento. A Figura 26 exemplifica o procedimento:

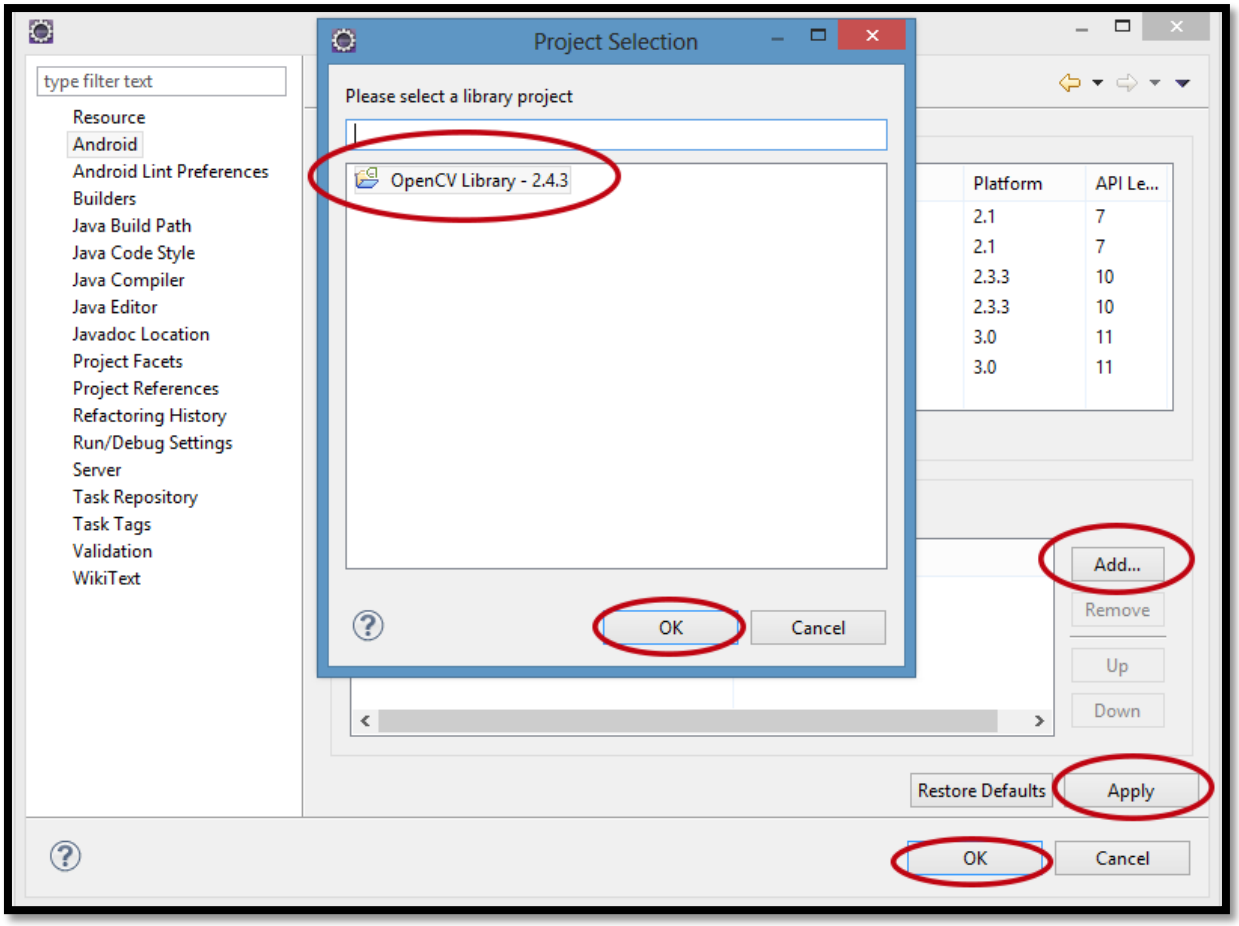

Figura 26 – Integração *OpenCV*: Configuração novo projeto

Para abrir essa interface é necessário entrar nas propriedades do projeto, em seguida clicar em *Android,* logo após acionar o botão *Add* e adicionar a biblioteca *OpenCV Library*. Para concluir a configuração deve-se clicar em *Apply* seguido do botão *OK*.

O último passo para a integração *OpenCV e Android* é realizar a conversão nativa C++ para o projeto. Esta conversão deve ser realizada pois a biblioteca *OpenCV* é totalmente nativa e implementada da linguagem de programação C++.

Para realizar o processo de conversão deve-se seguir os seguintes passos:

1. Clicar com o botão direito sobre o Projeto
2. *New*
3. *Other*
4. *Convert to a C/C++ Project (Adds C/C++ Nature)*
5. *Next*
6. *Finish*

Ao concluir a conversão basta entrar nas propriedades do projeto e realizar a configuração, como ilustrado na Figura 27:

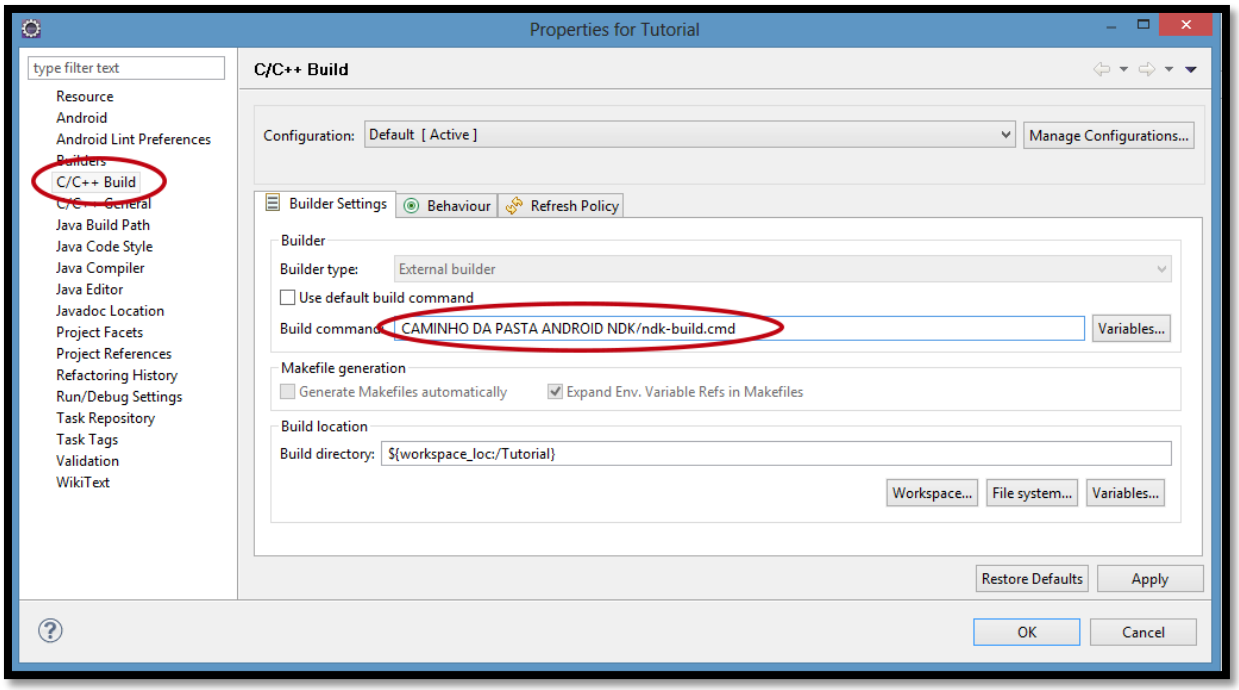

Figura 27 – Integração *OpenCV*: Configuração C/C++

Para realizar a configuração desta última etapa, basta desmarcar a opção: *Use default build command* e em seguida no campo *Build command,* colocar o caminho da pasta *Android* NDK + “\ndk.build.cmd”.

Neste ponto o projeto será capaz de executar qualquer implementação que utilize a biblioteca *OpenCV.*

É importante relatar que, dependendo da versão da IDE, a configuração do *Build command* fica sujeita a alterações, sendo que a mudança se concentra na eliminação da extensão "*.cmd" do* caminho da pasta resultando apenas em: Endereço *Android* NDK + "\ndk.build".

A partir dessas configurações, a biblioteca *OpenCV* funcionará como um conector para o OpenCV, isto é, será realizada a integração utilizando *AndroidCV*, sendo que a biblioteca, em tempo real, realizará a tradução e chamada dos verdadeiros métodos C++ nativos da biblioteca.

6.3 IMPLEMENTAÇÃO DO ESTUDO DE CASO

A implementação deste estudo de caso objetiva colocar em prática toda a pesquisa deste trabalho, desde os estudos e práticas levantadas sobre visão computacional até as regras de sistemas biométricos.

Outro ponto importante é a integração da biblioteca OpenCV com a plataforma móvel *Google Android*.

As seções seguintes descreverão todos os componentes utilizados para o desenvolvimento deste projeto, bem trechos do código empregados na implementação.

6.3.1 *Activities* do aplicativo

Uma *Activity* é um componente de aplicação que provê uma tela na qual os usuários podem interagir, a fim de realizar alguma tarefa. Cada *Activity* é equivalente a uma janela. Esta normalmente preenche a tela do dispositivo mas pode ser menor e flutuar em cima de outras.

Uma aplicação geralmente consiste de múltiplas *Activities* que são fracamente ligadas umas às outras. Normalmente, uma *Activity* em um aplicativo é especificada como *principal*, que é apresentada ao usuário ao iniciar o aplicativo pela primeira vez.

Cada *Activity* pode iniciar outra *Activity*, a fim de executar ações diferentes. Cada vez que uma nova é iniciada, a *Activity* anterior é interrompida, mas o sistema preserva a atividade em uma pilha, conhecida como *pilha back*. Quando uma nova *Activity* é iniciada, a chamadora é empurrada para a pilha, em seguida tirada o foco do usuário.

A lógica de programação para esta pilha é o conhecido *last in, first out*, que significa o último a entrar é o primeiro a sair, por isso, quando o usuário e pressiona o botão de voltar do dispositivo, o sistema retorna a *Activity* anterior, onde automaticamente será removida e destruída da pilha, sem que o usuário perceba.

Para cada Activity deverá ser criada uma classe *Java* para implementar seus métodos e funcionalidades, onde cada um destes *Activities* podem ter um arquivo *Extensible Markup Language* (*XML*) correspondente para a definição e criação de botões, textos e *layouts*.

Para a implementação deste trabalho foram criadas quatro *Activities*, sendo que a primeira é definida como principal para o usuário, exercendo o papel de menu que dispõe todas as funcionalidades do protótipo.

A segunda *Activity tem* o objetivo de fornecer ao usuário o resumo e informações deste trabalho. Já na terceira *Activity* implementada o objetivo é prover ao usuário uma escolha, onde serão dispostos todos os atributos no qual o aplicativo poderá detectar e a quarta *Activity* terá a função de apresentar ao usuário o resultado das detecções feitas pelo OpenCV.

O trecho do código *XML* a seguir demonstra a programação feita para a criação do *layout* principal do aplicativo:

```
<ScrollView <!-- CRIAÇÃO BARRA DE ROLAGEM -->
  android:layout_width="fill_parent"
  android:layout_height="wrap_content"
  android:gravity="center"
  android:orientation="vertical" >

  <LinearLayout
```

```
            android:layout_width="fill_parent"
            android:layout_height="wrap_content"
            android:gravity="center"
            android:orientation="vertical" >

            <TextView <!-- TEXTO DO TOPO NO MENU -->
                android:id="@+id/txtnome"
                android:layout_width="wrap_content"
                android:layout_height="wrap_content"
                android:text="@string/txtPrincipal"
                android:textAppearance="?android:attr/textAppearanceLarge"
                android:textColor="#FFFFFF"
                android:textStyle="bold" />

            <ImageView <!-- CRIAÇÃO IMAGEM -->
                android:id="@+id/imageView1"
                android:layout_width="match_parent"
                android:layout_height="match_parent"
                android:src="@drawable/logo" />

            <LinearLayout
                android:layout_width="fill_parent"
                android:layout_height="wrap_content"
                android:gravity="center"
                android:orientation="horizontal" >

                <Button <!-- CRIAÇÃO BOTÃO DETECTAR -->
                    android:id="@+id/btnDeteccao"
                    style="@style/ButtonText"
                    android:layout_width="match_parent"
                    android:layout_height="wrap_content"
                    android:layout_margin="10dp"
                    android:layout_weight="0.05"
                    android:background="@color/bluebutton"
                    android:text="@string/btnDeteccao"
                    android:textStyle="bold" />
            </LinearLayout>
        </LinearLayout>
    </ScrollView>

</LinearLayout>
```

O trecho descrito anteriormente determina a criação de uma barra de rolagem, uma imagem e texto que ficarão dispostos no menu do aplicativo, a criação de um botão para início da detecção dos atributos.

Todos os arquivos *XML* referentes à criação dos *Activities* são armazenados na pasta *layouts* que está localizada dentro da pasta *"res"* do projeto.

Como explicado anteriormente, cada *Activity* tem uma classe *Java* correspondente. A código a seguir demonstrará um trecho da classe principal.

```
package org.opencv.samples.fd;


import android.app.Activity;
import android.app.AlertDialog;
import android.content.DialogInterface;
import android.content.Intent;
import android.os.Bundle;
import android.view.View;
import android.widget.Button;


public class Principal extends Activity {


	// ATRIBUTOS REFERENTES AOS COMPONENTES DO ACTIVITY
	private Button btnDeteccao = null;
	private Button btnConfig = null;
	private Button btnSobre = null;
	public static final int CONFIG_CODE = 123456;


	// ATRIBUTOS PARA AS CARACTERISTICAS FACIAIS
	private Boolean face = null;
	private Boolean olhos = null;
	private Boolean zoom = null;
	private Boolean nariz = null;
	private Boolean boca = null;


	// CRIAÇÃO DA ACTIVITY
	@Override
	protected void onCreate(Bundle savedInstanceState) {

		super.onCreate(savedInstanceState);
		setContentView(R.layout.principal);

		iniciarComponentes();

		// CLICK DO BOTÃO COFIGURAÇÕES
		btnConfig.setOnClickListener(new View.OnClickListener() {

			@Override
			public void onClick(View v) {
				Intent Configurar = new Intent(Principal.this,
						Configuracao.class);


				startActivityForResult(Configurar, CONFIG_CODE);
			}
		});

		// CLICK DO BOTÃO DETECÇÃO
		btnDeteccao.setOnClickListener(new View.OnClickListener() {

			@Override
			public void onClick(View arg0) {
				chamarDetecao(); // CHAMANDO METODO DE DETECÇÃO
PASSANDO OS PARAMETROS
			}
```

```
            });
      }

      // RETORNO ACTIVITY CONFIGURAÇÕES
      @Override
      protected void onActivityResult(int requestCode, int resultCode,
Intent data) {
            super.onActivityResult(requestCode, resultCode, data);

                  if (requestCode == CONFIG_CODE) {

                    if (resultCode == Activity.RESULT_OK) {

                        Bundle parametros = data.getExtras();

                         face = (Boolean) parametros.getBoolean("face");
                         olhos = (Boolean) parametros.getBoolean("olhos");
                         zoom = (Boolean) parametros.getBoolean("zoom");
                         boca = (Boolean) parametros.getBoolean("boca");
                         nariz = (Boolean) parametros.getBoolean("nariz");

                         btnDeteccao.setEnabled(true);

                     } else {

                         btnDeteccao.setEnabled(false);
                     }
            }
      }

      // INICIANDO COMPONENTES
      private void iniciarComponentes() {

            btnDeteccao = (Button) findViewById(R.id.btnDeteccao);
            btnConfig = (Button) findViewById(R.id.btnConfig);
            btnSobre = (Button) findViewById(R.id.btnSobre);

            btnDeteccao.setEnabled(false);
      }

      // ...
}
```

No trecho do código onde acontece a chamada da *Activity* configurações, o processo de invocação é diferente, pois diferente dos outros métodos, o processo aguarda um retorno. O método utilizado para este procedimento é o *startActivityForResult,* e o procedimento utilizado para capturar a resposta da *Activity* configurações é o método *onActivityResult,* onde testa-se a chave para verificar se realmente é a *Activity* referida e outro teste para verificar se houve um retorno positivo ou negativo, isto é, se foi completo o ciclo da *Activity* configurações ou o usuário somente acionou o botão voltar.

A Figura 28 ilustra o fluxo entre as duas *Activities*:

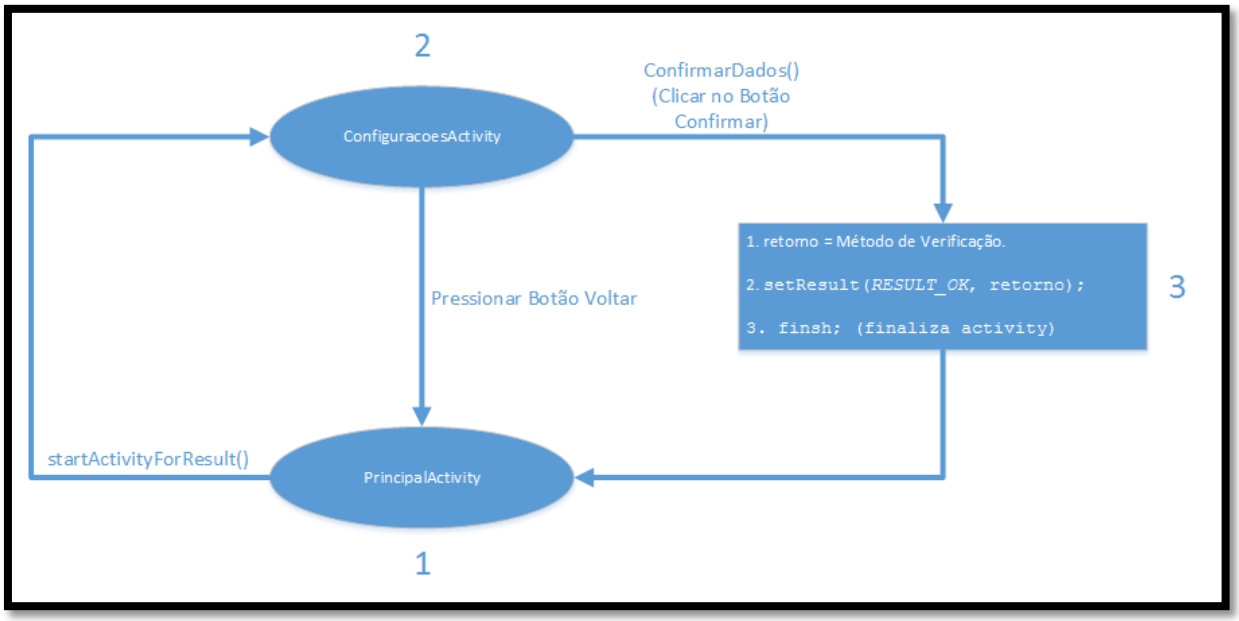

Figura 28 – Fluxo PrincipalActivity e ConfiguracoesActivity

O fluxo ilustra a chamada da *Activity* configurações, onde no (passo 2) terá dois tipos de retorno: se o usuário pressionar o botão voltar, não confirmando os dados das configurações o retorno será negativo, pois a *Activity* Principal não capturará os atributos a serem detectados mas, se houver confirmação de dados o retorno será positivo pois, na classe *ConfiguracoesActivity* haverá um método de verificação e captura de todos os dados configurados e em seguida o retorno para a *Activity* Principal.

Outro detalhe bastante significante, ainda dentro da Activity principal é a passagem dos atributos a serem detectados, sendo que estes atributos são os mesmos que foram pré-configurados na *Activity* Configurações. O trecho do código que fornece a passagem destes parâmetros é o que está descrito a seguir:

```
// ...
      private void chamarDetecao() {
            Bundle a = new Bundle();
            a.putBoolean("face", face);
            Bundle b = new Bundle();

            b.putBoolean("olhos", olhos);
            Bundle c = new Bundle();
            c.putBoolean("zoom", zoom);
```

```
            Bundle d = new Bundle();
            d.putBoolean("nariz", nariz);
            Bundle e = new Bundle();
            e.putBoolean("boca", boca);
            Intent Detectar = new Intent(Principal.this, FdActivity.class);
            Detectar.putExtras(a);
            Detectar.putExtras(b);
            Detectar.putExtras(c);
            Detectar.putExtras(d);
            Detectar.putExtras(e);
            startActivity(Detectar);
            btnDeteccao.setEnabled(false);
      }

// ...
```

Uma característica deste protótipo é que ao iniciar o aplicativo, o botão que aciona a detecção estará desabilitado, forçando o usuário a configurar os atributos desejados. Caso o usuário entre nas configurações e não escolha nenhum atributo e em seguida confirme as configurações, o aplicativo não detectará nada em nenhum *template.*

Vale lembrar que um sistema de reconhecimento de faces humanas, este processo não ocorrerá, visto que o aplicativo terá que detectar todos os atributos e faces humanas em todos os *templates*.

6.3.2 Classe de detecção de faces e atributos

Neste projeto, foi designado uma classe para realizar as detecções das faces e atributos, nela são implementados métodos e procedimentos utilizando a biblioteca *OpenCV* mas, antes da invocação desta classe o Principal invoca o *Activity* responsável abrir a câmera do dispositivo e transmitir ao usuário os *templates* detectados por meio dos atributos configurados. A seguir o trecho do código da *Activity* responsável por estes processos.

```
package org.opencv.samples.fd;
```

```
import org.opencv.android.BaseLoaderCallback;
import org.opencv.android.LoaderCallbackInterface;
import org.opencv.android.OpenCVLoader;

import android.app.Activity;
import android.app.AlertDialog;
import android.content.DialogInterface;
import android.os.Bundle;
import android.util.Log;
import android.view.Window;
import android.widget.RelativeLayout;

public class FdActivity extends Activity {
      private static final String TAG = "Sample::Activity";

      private FdView mView;
      public static int method = 5;

      private Boolean face = null;
      private Boolean olhos = null;
      private Boolean zoom = null;
      private Boolean nariz = null;
      private Boolean boca = null;

      private BaseLoaderCallback mOpenCVCallBack = new
BaseLoaderCallback(this) {
            @Override
            public void onManagerConnected(int status) {
                  switch (status) {
                  case LoaderCallbackInterface.SUCCESS: {
                        Log.i(TAG, "OpenCV carregada com Sucesso!");

                        //CARREGANDO CLASSE DE MARCAÇÃO DOS ATRIBUTOS
                        mView = new FdView(mAppContext, face, olhos, zoom,
nariz, boca);
                        mView.setDetectorType(mDetectorType);
                        mView.setMinFaceSize(0.2f);

                        // ...

                        // CHECANDO E ABRINDO CAMERA DO DISPOSITIVO MÓVEL
                        if (!mView.openCamera()) {
                              AlertDialog ad = new
AlertDialog.Builder(mAppContext)
                                          .create();
                              ad.setCancelable(false); // CANCELANDO BOTAO
VOLTAR
                              ad.setMessage("Erro Fatal: Camera não
encontrada!");
                              ad.setButton("OK", new
DialogInterface.OnClickListener() {
                                    public void onClick(DialogInterface
dialog, int which) {
                                          dialog.dismiss();
                                          finish();
                                    }
                              });
                              ad.show();
                        }
                  }
                        break;
```

```
                    default: {
                          super.onManagerConnected(status);
                    }
                          break;
                    }
              }
      };

      // ...
      // CRIAÇÃO DA ACTIVITY
      @Override
      public void onCreate(Bundle savedInstanceState) {
            Log.i(TAG, "onCreate");
            super.onCreate(savedInstanceState);

            Bundle parametros = getIntent().getExtras();

            face = parametros.getBoolean("face");
            olhos = parametros.getBoolean("olhos");
            zoom = parametros.getBoolean("zoom");
            nariz = parametros.getBoolean("nariz");
            boca = parametros.getBoolean("boca");

            requestWindowFeature(Window.FEATURE_NO_TITLE);

            Log.i(TAG, "Trying to load OpenCV library");
            if (!OpenCVLoader.initAsync(OpenCVLoader.OPENCV_VERSION_2_4_2,
this,
                        mOpenCVCallBack)) {
                  Log.e(TAG, "Cannot connect to OpenCV Manager");
            }
      }

// ...
```

Uma característica importante desta *Activity* é a ausência de um *XML* para sua criação, pois tudo que nela conter será configurada e ajustada por meio da classe de detecção e marcação dos atributos.

A *Activity* funciona em *loop*, isto é, enquanto o usuário não pressionar o botão voltar, a *Activity* invocará a classe responsável por realizar a detecção e marcação dos atributos. Ao realizar a invocação, é transmitido para a classe o *template* que ao concluir sua marcação, a classe retornará ao *Activity* para que o usuário tenha acesso na tela do dispositivo móvel.

A Figura 29 demonstra o fluxo entre a *Activity* que transmitirá os *templates* e a classe de marcação dos atributos:

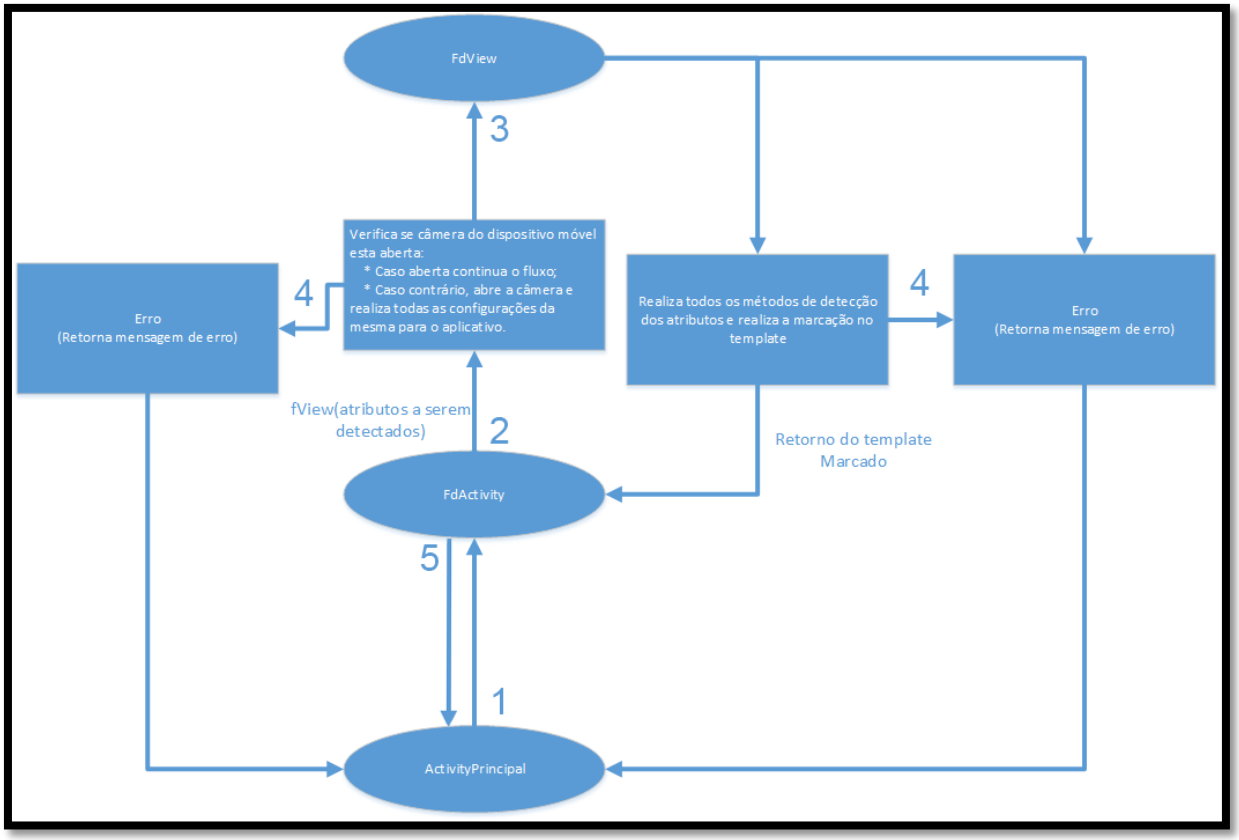

Figura 29 – Fluxo *Activity* x Classe de detecção e marcação

As elipses do fluxo representam as *Activities* ou classe do processo de marcação e detecção facial, sendo que os retângulos representam os métodos que a que a *Activity* executa antes de passar para outro estado (*Activity* ou classe).

No estado 1, é a representação do Menu Principal do aplicativo, em que o usuário tem opções de configurar os atributos a serem marcados e em seguida aciona a detecção, passando ao estado 2 que é a *Activity* responsável em mostrar ao usuário os *templates* já marcados e automaticamente já encaminha-se ao estado 3, executando antes os seguintes passos:

- Verifica se a câmera do dispositivo móvel encontra-se aberta.
    - Caso esteja aberta o fluxo continua;
    - Caso esteja fechada, abre-se a câmera, verifica a compatibilidade com o aplicativo, realiza todas as configurações e segue o fluxo, passando ao estado 3.

No estado 3 é a classe responsável pela marcação dos atributos escolhidos pelo usuário. Após a marcação, a classe retorna ao *Activity* 2 para que o mesmo transmita ao usuário as imagens marcadas. O passo 5, significa a volta ao Menu Principal, quando o usuário aciona o botão voltar do aplicativo.

6.3.3 Utilização de elementos *Haar-Like Features*

Os elementos *Haar-Like-Features* são utilizados não somente para reconhecimento de faces humanas, mas também de objetos.

Os algoritmos classificadores de características em cascata, possuem vários tipos e mutações, ou seja, subdivisões, que são específicas para cada tipo de objeto ou característica em que se deseja encontrar em uma imagem digital.

O método utilizado neste projeto foi um dos mais custosos em poder de processamento e memória, pois o algoritmo terá que varrer a *template* inteira para encontrar as características desejadas pelo usuário.

Já em sistemas biométricos, existe uma alternativa para diminuir o custo de processamento e memória. A saída para o problema é realizar detecções por prioridades.

Partindo da ideia que um sistema biométrico deverá detectar sempre todos os atributos, a saída é começar realizando a detecção facial, caso encontre- passa ao próximo nível por exemplos olhos, assim que realiza a detecção dos olhos passa-se ao nariz e assim por diante. Esta lógica resulta em uma diminuição de processamento e ganho de memória de aproximadamente 60%, pois o único algoritmo que varre a imagem toda é o de detecção facial, os próximos varrerão somente dentro do perímetro facial já previamente detectado. Por exemplo ao detectar os olhos e realizar a marcação, o algoritmo seguinte por exemplo, de detecção do nariz só irá realizar a detecção abaixo dos olhos encontrados, diminuindo o perímetro de varredura.

Por outro lado, um sistema de reconhecimento facial completo, antes de realizar toda a detecção dos atributos deverá passar por inúmeros filtros de diminuição de ruídos

nas imagens, binarização e muitos outros, custando um poder de processamento enorme.

Os algoritmos utilizados neste trabalho para realizar a detecção dos atributos já possuem um nível baixo de processamento entretanto, resolve o problema proposto.

O trecho de código a seguir demonstra a utilização do algoritmo *haarcascade* no trabalho.

```
package org.opencv.samples.fd;

// ... IMPORTAÇÕES

public class FdView extends SampleCvViewBase {

	// VARIAVEIS CONDICIONAIS
	private Boolean OLHOS = null;
	private Boolean FACE = null;
	private Boolean NARIZ = null;;
	private Boolean BOCA = null;
	private Boolean ZOOM = null;

	// CONSTRUTOR DA CLASSE
	public FdView(Context context, Boolean face, Boolean olhos, Boolean
zoom, Boolean nariz, Boolean boca) {
		super(context);

		FACE = face;
		OLHOS = olhos;
		ZOOM = zoom;
		NARIZ = nariz;
		BOCA = boca;
		// CARREGANDO CLASSIFICADOR FACIAL E CONFIGURANDO DE ACORDO COM
A IMAGEM


		try {

		InputStream is =
context.getResources().openRawResource(R.raw.lbpcascade_frontalface);
File cascadeDir = context.getDir("cascade", Context.MODE_PRIVATE);
mCascadeFile = new File(cascadeDir, "lbpcascade_frontalface.xml");
FileOutputStream os = new FileOutputStream(mCascadeFile);

			byte[] buffer = new byte[4096];
			int bytesRead;
			while ((bytesRead = is.read(buffer)) != -1) {
				os.write(buffer, 0, bytesRead);
			}
			is.close();
			os.close();


// -;-;-;-;-;-;-;-;-;-;-;-;-;-;-;-;-;-;-;-;-;-;-;-;-;-;-;-;-;-;-;-;-;-;-;-;-;-;-;-;-;-;-;-;-;
```

```
// CARREGANDO CLASSIFICADOR DO NARIZ E CONFIGURANDO DE ACORDO COM A IMAGEM
InputStream isnose =
context.getResources().openRawResource(R.raw.haarcascade_mcs_nose);
File cascadeNose = context.getDir("cascade", Context.MODE_PRIVATE);
File cascadeFileNose = new File(cascadeNose,"haarcascade_mcs_nose.xml");
FileOutputStream osnose = new FileOutputStream(cascadeFileNose);

                byte[] bufferNose = new byte[4096];
                int bytesReadNose;
                while ((bytesReadNose = isnose.read(bufferNose)) != -1) {
                      osnose.write(bufferNose, 0, bytesReadNose);
                }
                isnose.close();
                osnose.close();

// -;-;-;-;-;-;-;-;-;-;-;-;-;-;-;-;-;-;-;-;-;-;-;-;-;-;-;-;-;-;-;-;-;-;-;-;-;-;-;-;-;

// CARREGANDO CLASSIFICADOR DA BOCA E CONFIGURANDO DE ACORDO COM A IMAGEM
InputStream ismouth =
context.getResources().openRawResource(R.raw.haarcascade_mcs_mouth);
File cascadeMouth = context.getDir("cascade", Context.MODE_PRIVATE);
File cascadeFileMouth = new File(cascadeNose, "haarcascade_mcs_mouth.xml");
FileOutputStream osmouth = new FileOutputStream(cascadeFileMouth);

                byte[] bufferMouth = new byte[4096];
                int bytesReadMouth;
                while ((bytesReadMouth = ismouth.read(bufferMouth)) != -
1) {
                      osmouth.write(bufferMouth, 0, bytesReadMouth);
                }
                ismouth.close();
                osmouth.close();

// -;-;-;-;-;-;-;-;-;-;-;-;-;-;-;-;-;-;-;-;-;-;-;-;-;-;-;-;-;-;-;-;-;-;-;-;-;-;-;-;-;

// CARREGANDO CLASSIFICADOR DO OLHO ESQUERDO E CONFIGURANDO DE ACORDO COM A
IMAGEM
InputStream iser = context.getResources().openRawResource(
R.raw.haarcascade_lefteye_2splits);
File cascadeDirER = context.getDir("cascadeER", Context.MODE_PRIVATE);
File cascadeFileER = new File(cascadeDirER, "haarcascade_eye_right.xml");
                FileOutputStream oser = new
FileOutputStream(cascadeFileER);

                byte[] bufferER = new byte[4096];
                int bytesReadER;
                while ((bytesReadER = iser.read(bufferER)) != -1) {
                      oser.write(bufferER, 0, bytesReadER);
                }
                iser.close();
                oser.close();

// -;-;-;-;-;-;-;-;-;-;-;-;-;-;-;-;-;-;-;-;-;-;-;-;-;-;-;-;-;-;-;-;-;-;-;-;-;-;-;-;-;

// CARREGANDO CLASSIFICADOR DO OLHO DIREITO E CONFIGURANDO DE ACORDO COM A
IMAGEM
```

```
InputStream isel = context.getResources().openRawResource(
R.raw.haarcascade_lefteye_2splits);
File cascadeDirEL = context.getDir("cascadeEL", Context.MODE_PRIVATE);
File cascadeFileEL = new File(cascadeDirEL, "haarcascade_eye_left.xml");
FileOutputStream osel = new FileOutputStream(cascadeFileEL);

                  byte[] bufferEL = new byte[4096];
                  int bytesReadEL;
                  while ((bytesReadEL = isel.read(bufferEL)) != -1) {
                        osel.write(bufferEL, 0, bytesReadEL);
                  }
                  isel.close();
                  osel.close();

// ...

// MARCAÇÃO FACE - Padrão
if (FACE) {
      for (int i = 0; i < facesArray.length; i++) {
            Rect r = facesArray[i];
            Core.rectangle(mGray, r.tl(), r.br(), new Scalar(0, 255, 0,
                                          255), 3);
            Core.rectangle(mRgba, r.tl(), r.br(), new Scalar(0, 255, 0,
                                          255), 3);
                        }
                  }

// ... DEMAIS MARCAÇÕES E METODOS


}
```

O trecho demonstra parte do processo do carregamento do arquivo *XML* dos padrões de cascata utilizados no protótipo (face, olhos, nariz, boca).

Nota-se que o construtor da classe recebe a imagem e os padrões que devem ser detectados. Em seguida são carregados os padrões *haarcascade* para serem configurados.

A Figura 30 demonstrará fluxo de utilização dos padrões *cascades*:

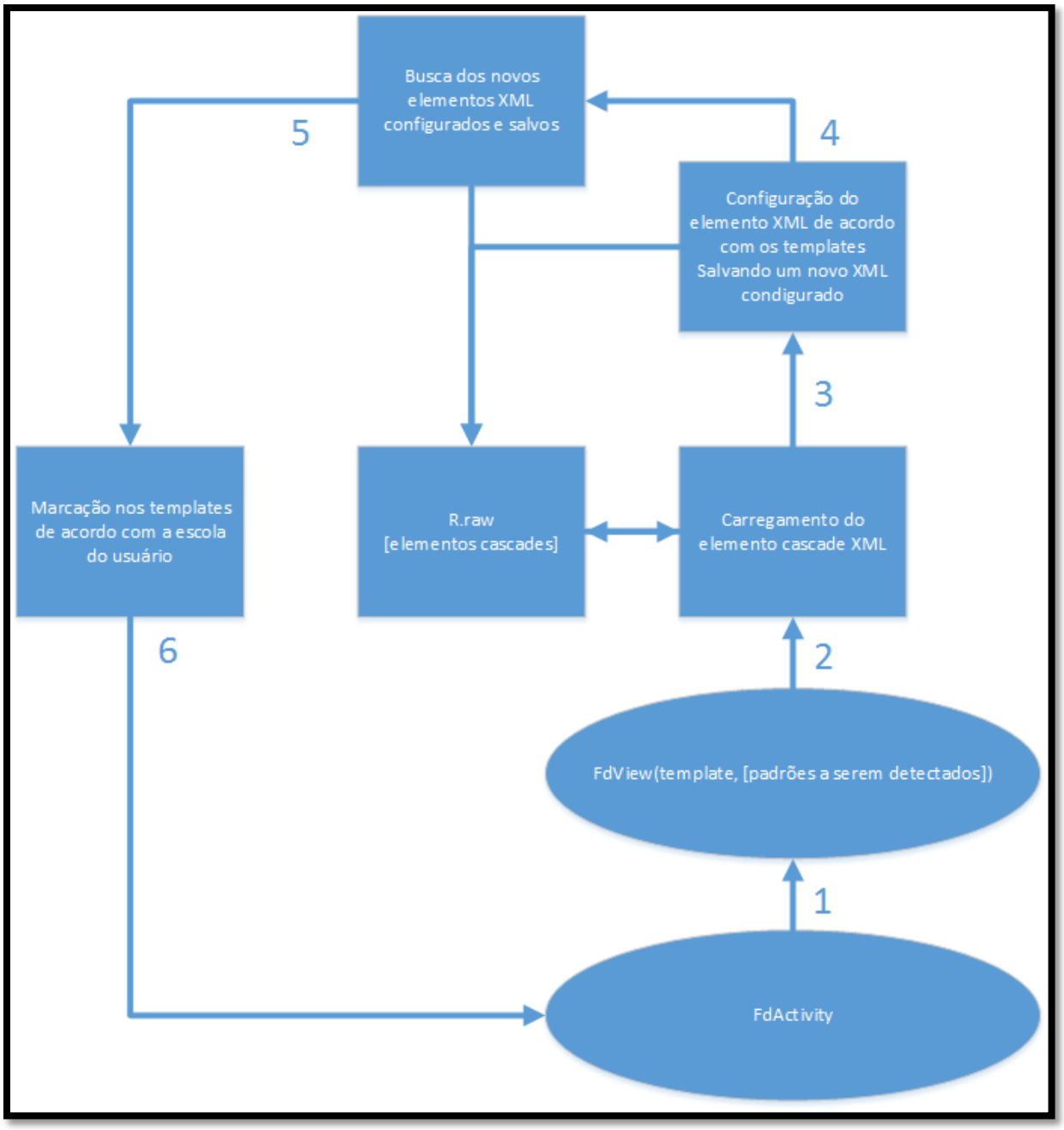

Figura 30 – Fluxo *HaarCascade*

A lógica do fluxo de marcação inicia-se na invocação da *Activity* responsável por apresentar ao usuário a imagem marcada.

No estado 1, encontra-se a invocação do da classe responsável pela marcação, passando por parâmetros quais os atributos que devem ser detectados.

Já o estado 2 é responsável pelo carregamento dos arquivos *XML* que ficam localizados na pasta *res/raw* do projeto. Estes arquivos estão disponíveis no site da biblioteca *OpenCV*.

O estado 3, é responsável por configurar o arquivo *XML* carregado de acordo com o *template,* em seguida é criado um novo arquivo de acordo com as configurações realizadas. Este processo é realizado com todos os arquivos *XML.*

Após todos os arquivos configurados e salvos, os mesmos são carregados no estado 5 para serem utilizados no próximo estado, onde será realizado a detecção dos atributos escolhidos e a marcação dos mesmos. Após a marcação dos atributos o *template* é retornado a *Activity* chamadora.

Em seguida será demonstrado um trecho do código *XML haarcascade,* utilizado para a detecção facial.

```
<?xml version="1.0"?>
<!--
number of positive samples 3000
number of negative samples 1500
-->
<opencv_storage>

<cascade type_id="opencv-cascade-classifier">
  <stageType>BOOST</stageType>
  <featureType>LBP</featureType>
  <height>24</height>
  <width>24</width>
  <stageParams>

    <boostType>GAB</boostType>
    <minHitRate>0.9950000047683716</minHitRate>
    <maxFalseAlarm>0.5000000000000000</maxFalseAlarm>
    <weightTrimRate>0.9500000000000000</weightTrimRate>
    <maxDepth>1</maxDepth>
    <maxWeakCount>100</maxWeakCount></stageParams>

  <featureParams>

    <maxCatCount>256</maxCatCount></featureParams>

  <stageNum>20</stageNum>

  <stages>
    <!-- stage 0 -->
    <_>
      <maxWeakCount>3</maxWeakCount>
      <stageThreshold>-0.7520892024040222</stageThreshold>
      <weakClassifiers>
        <!-- tree 0 -->
        <_>
```

```
          <internalNodes>
            0 -1 46 -67130709 -21569 -1426120013 -1275125205 -21585
            -16385 587145899 -24005</internalNodes>

          <leafValues>

            -0.6543210148811340 0.8888888955116272</leafValues></_>

        <!-- tree 1 -->
        <_>

          <internalNodes>
            0 -1 13 -163512766 -769593758 -10027009 -262145 -514457854
            -193593353 -524289 -1</internalNodes>

          <leafValues>
            -0.7739216089248657 0.7278633713722229</leafValues></_>


<!-- ...  -->


</opencv_storage>
```

O código do arquivo *XML* de cada atributo é muito extenso, chegando a 1600 linhas cada arquivo. A configuração realizada de acordo com o *template* ajusta a resolução, profundidade e outras minúcias para melhor detecção e performance do aplicativo.

6.4 INTERFACES DA APLICAÇÃO

Nesta seção serão apresentadas todas as interfaces do aplicativo proposto, bem como suas funcionalidades.

A Figura 31 apresenta a tela inicial do aplicativo e, a partir deste menu, o usuário poderá usufruir de todo aplicativo, configurando os atributos a serem detectados, em seguida realizando a detecção e marcação das imagens.

Além de tudo, a partir desta interface, o usuário terá acesso a uma breve explicação sobre o que se trata o aplicativo, obtendo informações do desenvolvedor e da fundação educacional.

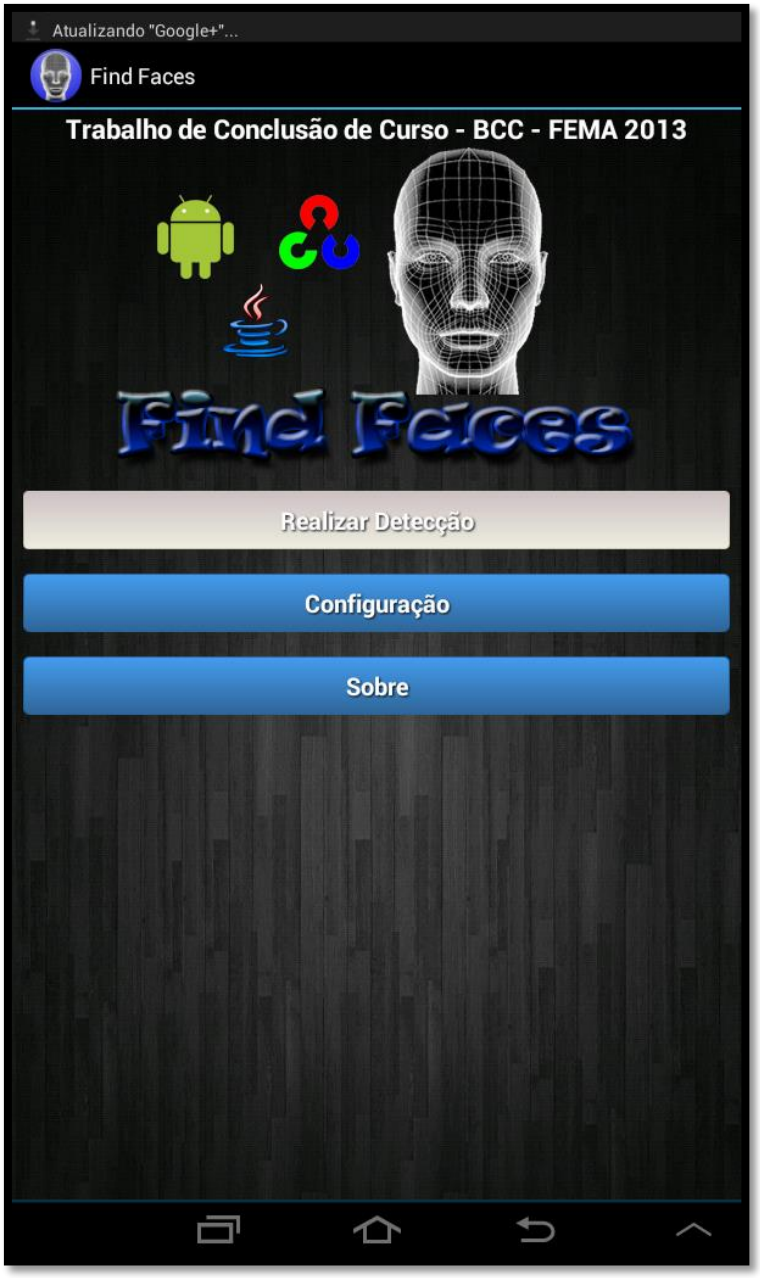

Figura 31 – Interface de Menu

O nome dado ao protótipo é *Find Faces,* que no português significa encontrar faces.

Na imagem de logo, na parte superior da interface encontram-se os símbolos das tecnologias estudadas durante o processo deste trabalho: *Google Android, OpenCV* e *Java*, e a face preta e branca significando a biometria, tema também pesquisado neste trabalho.

Já na Figura 32 ilustra a interface de explicação sobre o trabalho:

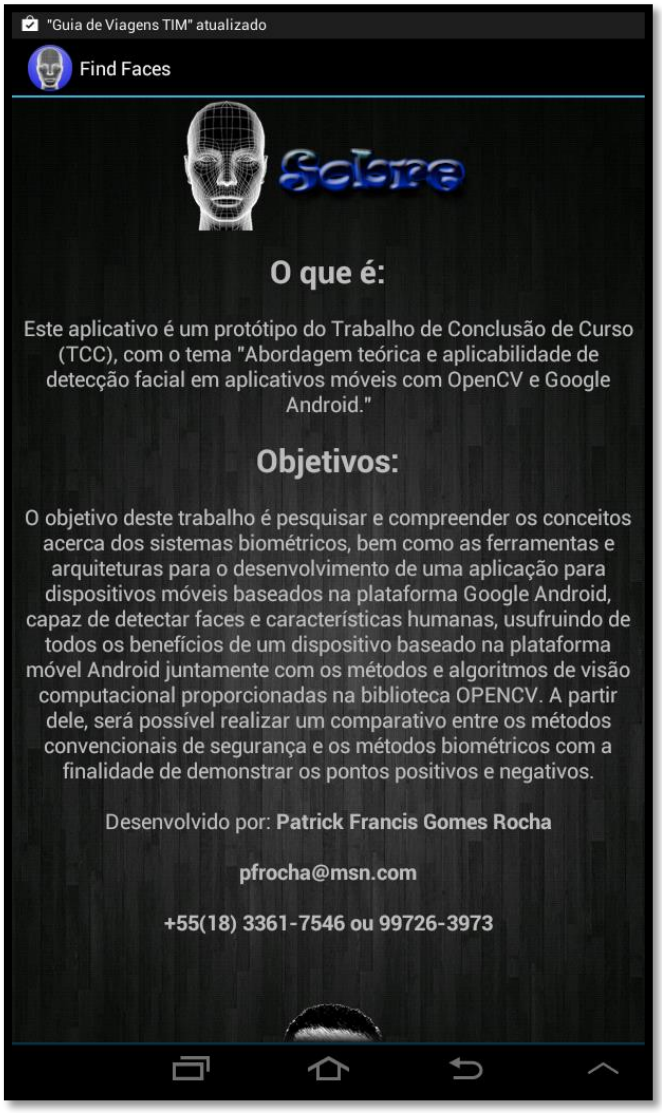

Figura 32 – Interface Sobre

A interface apresentada na Figura 33 demonstra as configurações em que o usuário tem por opções para realizar a detecção dos atributos.

Um ponto relevante é na escolha da detecção dos olhos, em que ao escolher este atributo, o usuário terá opção realizar um *zoom* nos olhos no momento da detecção.

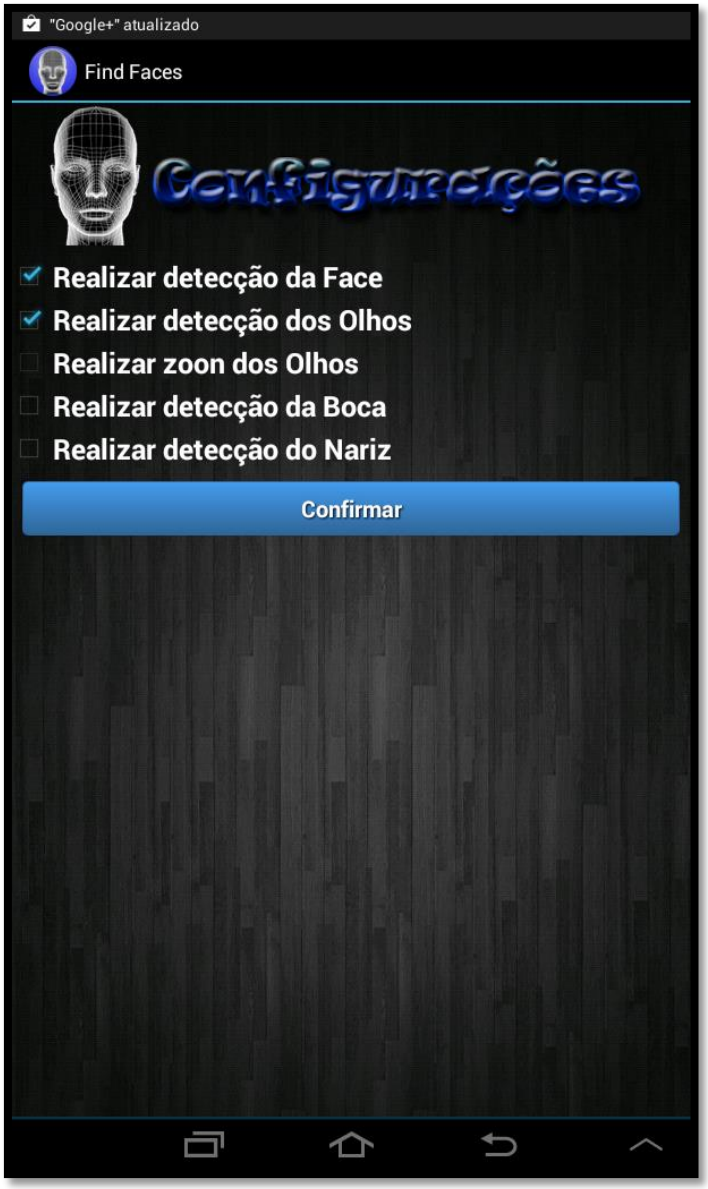

Figura 33 – Interface Configurações

Todos os atributos devem ser confirmados para que a escolha do usuário seja apresentada no momento da detecção. A interface ilustrada na Figura 34 demonstra este processo.

Caso o usuário opte por não confirmar a interface fica disponível para ajustes, isto é, para que possa ser modificado os padrões que desejam ser detectados. Caso o usuário confirme a interface inicial (Menu) voltará, apta a realizar a detecção conforme as configurações pré-determinadas.

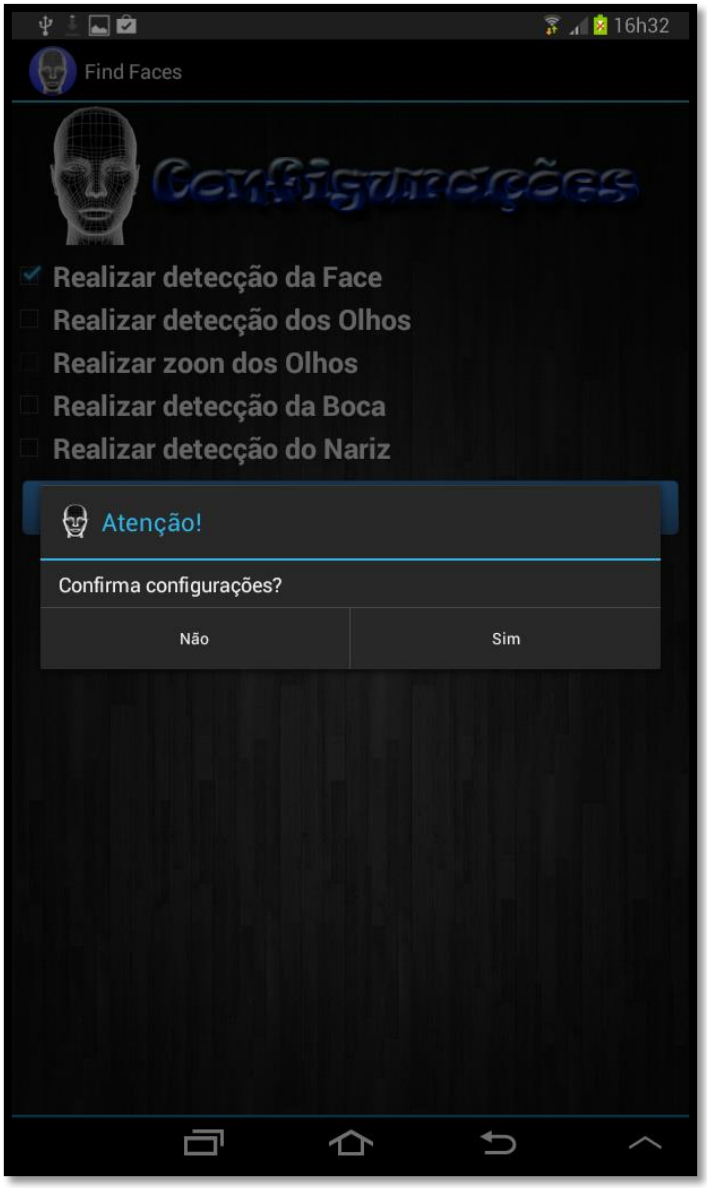

Figura 34 – Confirmação das configurações

É importante lembrar que caso o usuário volte a interface não captará os atributos desejados e consequentemente a interface Principal não estará apta em realizar a detecção.

Por fim, a Figura 35 demonstra a interface onde é apresentado ao usuário toda a detecção dos atributos escolhidos e configurados:

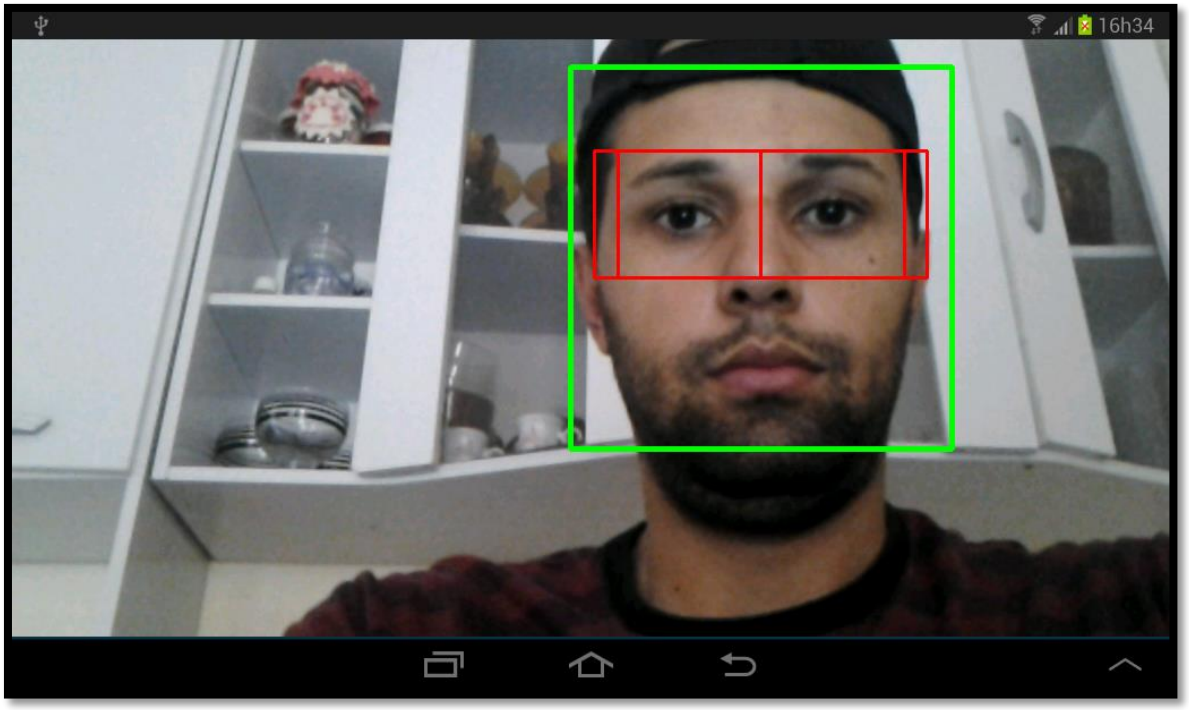

Figura 35 – Interface detecção

Neste exemplo, vale ressaltar que foram detectados os atributos da face e olhos, pois na interface de configurações foi feito esta escolha, consequentemente caso ser feito outros atributos, a interface de detecção, marcará o que o usuário optou.

7 – CONCLUSÃO

Neste capítulo serão descritos as considerações finais e dificuldades deste projeto, bem como propostas para trabalhos futuros tendo este projeto como base, para seguimento nas pesquisas e implementações para um sistema que detecte e realize reconhecimentos faciais.

7.1 CONSIDERAÇÕES FINAIS

Como resposta às preocupações quanto a segurança das informações, acessos a lugares restritos, fraudes em pontos eletrônicos, a biometria atende de maneira eficaz e eficiente todos os quesitos. Muitos bancos já fazem uso de algum padrão biométrico para aumentar ainda mais a segurança em meio a acessos a contas bancaria.

Outro exemplo bem intenso e presente é o da justiça eleitoral do país optar por tecnologias biométricas para autenticação no período das eleições.

Apesar de possuir vários pontos favoráveis, como a tecnologia ainda é recente, o investimento é alto para adquirir ou adequar o sistema ao local que se deseja aplica-lo. Ainda há muita pesquisa que deverá ser feita para que estes algoritmos tornem-se cada vez mais eficazes exigindo um poder computacional mais reduzido.

Acredita-se que o uso desta tecnologia na Copa do Mundo de 2014 no país e nos jogos Olímpicos de 2016 irão alavancar as pesquisas sobre a tecnologia.

A integração *Google Android* e *OpenCV* também é um estudo recente, por esta razão fora feito a escolha deste tema no trabalho, onde também encontrou-se algumas dificuldades, devido ao pouco material existente.

Outra grande dificuldade foi no ponto em que foi realizado a configuração, pois existem muitas versões da biblioteca *OpenCV* e com frequência atualizações são feitas no *Google Play* dificultando ainda mais a compatibilidade.

Um ponto favorável sobre a realização deste trabalho, foi poder usufruir do conhecimento adquirido no Projeto de Iniciação Cientifica (PIC), desenvolvido em dois

anos nesta Instituição, podendo ter base de conhecimento sobre padrões biométricos e visão computacional, que foram ferramentas de metodologia deste trabalho.

Por fim, com o desenvolvimento deste projeto, pode-se perceber que em um futuro próximo, a biometria contribuirá de uma forma muito forte em métodos alternativos de segurança, visto que, um sistema bem desenvolvido, contendo todos os alicerces necessários, substituirá definitivamente os métodos convencionais como senhas, cartões magnéticos, entre outros, trazendo a empresa ou negócio um poder de segurança muito elevado.

7.2 TRABALHOS FUTUROS

A partir deste projeto é possível dar continuidade, desenvolvendo um sistema completo que realize o reconhecimento facial, visto que os primeiros passos para realização foram realizados neste trabalho.

Outra contribuição deste trabalho é a facilidade de troca de atributos, a partir da ideia que o algoritmo desenvolvido realiza as detecções a partir dos arquivos *XML* carregados, ou seja, fornecendo outros tipos de arquivos de padrões o algoritmo também detectara e realizará as marcações, sendo possível, a partir deste protótipo, dar continuidade, focando não somente em um sistema biométrico convencional, mas sim, podendo mudar os tipos dos padrões e logica de autenticação do sistema.

Conclui-se, portanto, que o projeto apresentou uma maneira alternativa de realizar os procedimentos dos padrões biométricos aplicando o melhor das plataformas de *Android e* OpenCV para fornecer maior estabilidade e confiabilidade nas diferentes etapas da detecção facial, assim, sendo base e modelo para trabalhos futuros, fazendo com que o pesquisador possa ter várias formas de implementar um sistema biométrico.

REFERÊNCIAS

ALBUQUERQUE, Márcio Portes de; ALBUQUERQUE, Marcelo Portes de. **Processamento de Imagens: Métodos e Análises.** Rio de Janeiro: Centro Brasileiro de Pesquisas Físicas – CBPF/MCT, 2009. 12 p.

AQUINO, Juliana França Santos. **Introdução à Computação Móvel.** 2007. 14 f. Monografia – Programa Pós Graduação em Informática, Departamento de Informática, Pontifícia Universidade Católica do Rio de Janeiro, Rio de Janeiro, 2007.

AVILA, Luiz Fernando (Ed.). **O Globo:** Tecnologia. Disponível em: <http://oglobo.globo.com/tecnologia/mundo-tem-6-bilhoes-de-celulares-6401960>. Acesso em: 28 maio 2013.

BARROS, Thiago. **Android Studio é o programa do Google para desenvolver apps para Android.** Disponível em: <http://www.techtudo.com.br/tudo-sobre/s/android-studio.html>. Acesso em: 11 jun. 2013.

BATISTA, Mônica de Lourdes Souza et al. Processamento digital de imagens para a detecção e classificação de nódulos em mamografias. **Revista Eletrônica da Faculdade Metodista Granbery**, Juiz de Fora - MG, 10 maio 2013.

BRADSKI, Gary. **The OpenCV Library.** Disponível em: <http://www.drdobbs.com/open-source/the-opencv-library/184404319?pgno=1>. Acesso em: 08 de maio de 2013.

CANEDO, José Alberto. **Visão geral de um Sistema Biométrico:** Fórum Biometria. Disponível em: <http://www.forumbiometria.com/fundamentos-de-biometria/129-visao-geral-de-um-sistema-biometrico.html>. Acesso em: 14 maio 2013.

CAVALCANTE, Adalberto Luiz Sobral; BACCI, Márcio Demetrio; HOKAMA, Marçal De Lima. **Assinatura de Documentos Digitais através da Biometria no Exército Brasileiro.** Rio de Janeiro: Editora, 2005.

CHIACHIA, Giovani; PENTEADO, Bruno Elias; MARANA, Aparecido Nilceu. **Fusão de Métodos de Reconhecimento Facial através da Otimização por Enxame de Partículas.** Bauru - Sp: UNESP – Faculdade de Ciências – Departamento de Computação, 2011. 4 p.

DANTAS, George Felipe de Lima, **Sistemas Biométricos de Identificação pela Imagem Facial**. Disponível em: <www.logicengenharia.com.br/mcamara/alunos/Bio metria.pdf>. Acesso em: 12 de Fevereiro de 2013.

FARIA, Alessandro de Oliveira. Programe seu androide. **Linux Magazine**, Brazil, 5 p. 28 maio 2010.

FARTO, Guilherme de Cleva. **Abordagem Orientada a Serviços para implementação de um aplicativo Google Android.** Trabalho de Conclusão de Curso Assis: Fundação Educacional do Município de Assis - FEMA, 2010. 83 p.

HASEMAN, Chris. **Android Essentials.** New York: Firstpress, 2008. 116 p.

JAIN, Anil K.; DUIN, Robert P.W.; MAO, Jianchang. **Statistical Pattern Recognition: A Review**. IEEE Transactions on Pattern Analysis and machine Intelligence, vol. 22, nº 1, 2000.

LECHETA, Ricardo R. **Google Android:** Aprenda a criar aplicações para dispositivos móveis com Android SDK. 3. ed. São Paulo: Novatec, 2013. 576 p.

LOPES, Eduardo Costa. **Detecção de Faces e Características Faciais.** Rio Grande do Sul: PUCRS - Pontifícia Universidade Católica do Rio Grande do Sul, 2009. 44 p.

MARENGONI, Maurício; STRINGHINI, Denise. Tutorial: Introdução à Visão Computacional usando OpenCV. **Revista de Informática Teórica e Aplicada**: RITA, Cidade, v. 1, p.1-36, 01 fev. 2009.

MEDNIEKS, Zigurd et al. **Programando Android:** Programando Java para a nova geração de dispositivos móveis. 2. ed. São Paulo: Novatec, 2012. 116 p.

MILANO, Danilo de; HONORATO, Luciano Barrozo. **Visão Computacional.** Campinas: UNICAMP – UNIVERSIDADE ESTADUAL DE CAMPINAS, 2011. 7 p.

MÜLLER, Daniel Nehme; DARONCO, Everaldo Luis. **Operações matemáticas em Imagens.** Porto Alegre: Universidade Federal Do Rio Grande Do Sul - Programa De Pós-graduação Em Computação, 2000. 13 p.

NASCIMENTO, Andréia Vieira; GONZAGA, Adilson. **Detecção de faces humanas em Imagens digitais: Um algoritmo baseado em Lógica Nebulosa**, Departamento de Engenharia Elétrica - USP, 2008.

NETO, Hugo Vieira; FILHO, Oge Marques. **Processamento Digital de Imagens.** Rio de Janeiro: Brasport, 1999. 330 p.

OPENCV: **OpenSource Computer Vision**. Disponível em: <http://opencv.org/>. Acesso em: 08 maio 2013.

OPEN Handset Alliance Disponível em: <http://www.openhandsetalliance.com/index. html>. Acesso em: 31 maio 2013.

PEREIRA, Lucio Camilo de Oliveira; SILVA, Michel Lourenço da. **Android para desenvolvedores.** Rio de Janeiro: Brasport, 2009. 221 p.

ROCHA, Patrick Francis Gomes; NITTO Marisa Atsuko. **Aplicativo para Armazenamento de Tratamento de imagens de faces humanas** – IV Fórum de Ciência e Tecnologia, FEMA - IMESA, Assis, SP, 2011.

ROCHA, Patrick Francis Gomes; NITTO Marisa Atsuko. **Aplicativo para Reconhecimento de Faces Humanas** – V Fórum de Ciência e Tecnologia, FEMA - IMESA, Assis, SP, 2012.

SERRANO, Tiago Barquilha. **Padrões Biométricos Para Identificação**, Trabalho de Conclusão de Curso, Bacharelado em Ciência da Computação, FEMA, 2010.

SILVA, Antonio Idelvane Santana; CARVALHO, Ney Paranaguá de; SANTOS NETO, Pedro de Alcântara Dos. **Uma aplicao de biometria na web voltada para planos de saúde.** 4. ed. Rio de Janeiro: Reciis - Revista Eletronica de Comunicação Informação e Inovação em Saúde, 2010. (5).

SILVA, Clevertom et al. **A segurança através da Biometria.**: SEGeT – Simpósio de Excelência em Gestão e Tecnologia, 2012. 8 p.

VASCONCELOS, Clayton de Medeiros. **Introdução ao Processamento Digital de Imagens.** Disponível em: <www.cbpf.br/cat/download/seminarios/internos/Clayton.pp t>. Acesso em: 10 maio 2013.