UNIVERSIDADE DE SÃO PAULO

ESCOLA DE ENGENHARIA DE SÃO CARLOS

DEPARTAMENTO DE ENGENHARIA ELÉTRICA E DE COMPUTAÇÃO

Guilherme Rocha Gonçalves

Projeto de um sistema de monitoramento de ambientes

São Carlos

2017

Guilherme Rocha Gonçalves

Projeto de um sistema de monitoramento de ambientes

Monografia apresentada à Escola de Engenharia de São Carlos, Universidade de São Paulo, como parte dos requisitos para obtenção do título de Engenheiro Eletricista.

Área de Concentração: Processamento de Imagens e Eletrônica Embarcada.

Orientador: Prof. Dr. Evandro Luis Linhari Rodrigues

São Carlos

2017

Gonçalves, Guilherme Rocha
G953p Projeto de um sistema de monitoramento de ambientes / Guilherme Rocha Gonçalves; orientador Evandro Luis Linhari Rodrigues; coorientador José Carlos de Melo Vieira Júnior. São Carlos, 2017.

Monografia (Graduação em Engenharia Elétrica com ênfase em Eletrônica) -- Escola de Engenharia de São Carlos da Universidade de São Paulo, 2017.

1. Internet das Coisas. 2. Visão Computacional. 3. Sistemas Embarcados. 4. Segurança Residencial. 5. Inteligência Artificial. I. Título.

FOLHA DE APROVAÇÃO

Nome: Guilherme Rocha Gonçalves

Título: "Projeto de um sistema de monitoramento de ambientes"

Trabalho de Conclusão de Curso defendido e aprovado em 21/06/17,

com NOTA 9,5 (Nove, cinco), **pela Comissão Julgadora:**

Prof. Associado Evandro Luis Linhari Rodrigues - Orientador - SEL/EESC/USP

Prof. Associado José Carlos de Melo Vieira Júnior - SEL/EESC/USP

Prof. Titular Marco Henrique Terra - SEL/EESC/USP

Coordenador da CoC-Engenharia Elétrica - EESC/USP:
Prof. Associado José Carlos de Melo Vieira Júnior

Resumo

Esta monografia apresenta o projeto de um sistema de monitoramento com foco residencial desenvolvido como Trabalho de Conclusão de Curso. O objetivo do projeto é a implementação de um sistema que detecta qualquer movimento inesperado em algum ambiente. Caso encontre algum movimento o sistema o salva na forma de imagens e vídeos. O desafio do projeto é a inovação em relação aos sistemas já existentes nessa área. Essa inovação está no dinamismo implementado ao que concerne às notificações no celular do usuário através de um aplicativo para Android criado usando o *Android Studio* e o *Firebase* e também à ideia de modularização do sistema de maneira a permitir integração com outros sistemas inteligentes. O processo de aprendizagem de máquina também foi utilizado para identificar a presença de pessoas, utilizando o *Tensorflow*. Foi escolhido como plataforma de desenvolvimento o sistema embarcado *BeagleBone Black*, uma plataforma poderosa e que suporta a instalação de sistemas operacionais, como por exemplo o *Linux* embarcado na distribuição *Debian* usada no projeto. Os objetivos foram alcançados conforme os requisitos definidos para o projeto.

Palavras-Chave: Segurança. Monitoramento. Internet das coisas. Sistemas embarcados. *Linux* embarcado. Visão Computacional.

Abstract

This monograph presents the project of a home surveillance system developed for a undergraduate project. This goal of the project is to implement a system able to detect any unexpected movement in any environment. If it finds any movement, the system saves it in the form of images and videos. The project challenge is to innovate in comparison to existing systems in this area. The desired innovation lies on the dynamic implemented in relation to notifications shown on the user's cell phone through an Android application created by using *Android Studio* and *Firebase* and also the idea of system modularization as a way to allow integration with other inteligent systems. A machine learning process was used to identify the presence of people, by using *Tensorflow*. The *BeagleBone Black* embedded system was chosen as a development platform, a powerful platform that supports the installation of operating systems, such as *Linux* embedded in the Debian distribution used in the project. The objectives were achieved according to the requirements defined for the project.

Keywords: Safety. Surveillance. Internet of Things. Embedded Systems. Embedded *Linux*. Computer Vision.

Lista de Figuras

Lista de Abreviaturas, Siglas e Unidades

- **API** : Application Programming Interface
- **ARM** : Advanced RISC Machine
- **CISC** : Complex Instruction Set Computer
- **CPU** : Central Processing Unit
- **DC** : Direct current
- **DNS** : Domain Name System
- **eMMC** : embedded Multi-Media Controller
- **GB** : Gigabyte
- **GTK** : GIMP Tool Kit
- **GUI** : Graphical User Interface
- **HD** : High-definition
- **HDMI** : High-definition Multimidia Interface
- **HTTP** : Hypertext Transfer Protocol
- **I/O** : Input/Output
- **IoT** : Internet of Things
- **IP** : Internet Protocol
- **JPEG (ou JPG)** : Joint Photographic Experts Group
- **JSON** : JavaScript Object Notation
- **LED** : Light Emitting Diode
- **mA** : miliampère
- **MAC** : Media Access Control
- **MB** : Megabyte
- **MHz** : Megahertz

- **MJPEG** : Motion JPEG
- **OpenGL** : Open Graphics Library
- **OpenCV** : Open Source Computer Vision Library
- **PNG** : Portable Network Graphics
- **RISC** : Reduced Instruction Set Computer
- **SD** : Secure Digital
- **RAM** : Random Access Memory
- **SDRAM** : Synchronous Dynamic Random Access Memory
- **SSH** : Secure Shell
- **URL** : Uniform Resource Locator
- **USB** : Universal Serial Bus
- **V** : Volts

Sumário

Capítulo 1

Introdução

A tecnologia cresce numa taxa tão alta atualmente que um celular comprado no começo do ano já se torna a segunda opção no final do ano. Além disso, um celular de hoje em dia possui mais tecnologia do que a espaçonave Apollo que levou o homem à lua em 1969. Para efeito de comparação, o *smartphone* Samsung Galaxy Note 4 [1] possui um processador Qualcomm Snapdragon 805 com 4 núcleos funcionando a 2,7GHz, 3 GB de memória RAM, 32 GB de memória interna e muitos outros periféricos. A astronave Apollo em contrapartida chegou na lua com apenas um computador de bordo (Apollo Guidance Computer) que pesava 32kg e tinha apenas 2kB de memória RAM, além de um processador com poder de processamento de 2,048MHz [2].

Além disso os *smartphones* têm se tornado cada vez mais comuns entre a população hoje em dia. A quantidade de entregas únicas de *smartphones* no mundo no primeiro trimestre de 2016 foi de 174,3 milhões [3].

Com os celulares da atualidade é possível jogar com gráficos avançados, assistir vídeos em alta qualidade, fazer vídeo chamadas e mil outras funcionalidades. Portanto com uma ferramenta tão poderosa em nossas mãos porque não a utilizar para integrar com outras aplicações inteligentes?

Não só os celulares avançaram em tecnologia mas também a área de sistemas embarcados que se tornaram poderosos o suficiente para executar algoritmos de processamento de imagens e de quaisquer outras aplicações, além de permitir a instalação de sistemas operacionais.

Os primeiros computadores desenvolvidos na década de 1940 eram muitas vezes dedicados a uma única tarefa e eram muito grandes para serem considerados embarcados [4]. O conceito de controlador programável foi desenvolvido em 1968.

O primeiro sistema embarcado reconhecido e já citado acima foi o Apollo Guidance Computer desenvolvido por Charles Stark Draper no MIT, se tratava de um computador de tempo real utilizado como guia. O primeiro sistema embarcado de produção em massa foi o computador guia do míssil

nuclear LGM-30 Míssil Minuteman [5] lançado em 1961. Em 1966 o computador guia foi substituído por outro que possibilitou a redução de custos de circuitos integrados como por exemplo a porta NAND [6].

Desde 1960 portanto os sistemas embarcados vêm reduzindo seu preço, além dos avanços no poder de processamento e tamanho. Em 1978 foi lançada pela National Engineering Manufacturers Association uma norma para microcontroladores programáveis. E em 1980, vários componentes externos foram integrados no mesmo *chip* do processador resultando em circuitos integrados chamados de microcontroladores e na difusão dos sistemas eletrônicos embarcados [7].

Com o custo dos microcontroladores menor que um dólar americano foi possível substituir componentes analógicos de preço elevado por eletrônica digital controlada. Hoje em dia podemos encontrar embarcados bastante poderosos pelo preço de cinco dólares americanos como é o caso do embarcado Raspberry Pi Zero.

Já em 1991, Linus Torvalds, um estudante da Universidade de Helsinki, começou a desenvolver um projeto que se tornaria o núcleo Linux. Sua inspiração foi o sistema operacional Minix que era um pequeno sistema Unix desenvolvido pelo professor Andrew S. Tanenbaum com fins educacionais [8].

Com o objetivo de criar seu próprio sistema e após alguns desacordos com o professor Tanenbaum ele fez um anúncio em um grupo chamado "comp.os.minix"dizendo que seu projeto era apenas um *hobby* e que gostaria de saber sobre o que as pessoas gostavam e desgostavam do sistema Minix.

O Linux então surgiu e contou com participação de muitas pessoas em seu desenvolvimento. Hoje em dia o Linux é utilizado pelos computadores mais poderosos do mundo.

O fato de o Linux ser de código aberto possibilitou que a comunidade tivesse acesso aos códigos fontes do sistema. E no presente podemos ver os resultados desse desenvolvimento nas várias distribuições disponíveis e na velocidade e facilidade nas operações desejadas, por isso o núcleo Linux foi utilizado no projeto.

Poucos anos depois em 1999, surgiu o projeto OpenCV, que foi uma iniciativa da empresa Intel para melhorar aplicações com alto uso de CPU. O maior contribuidor para este projeto foi a Intel russa. Os objetivos do projeto foram a pesquisa em visão computacional, a disseminação de conhecimento ao se providenciar uma melhor infraestrutura na qual os desenvolvedores possam trabalhar e também a possibilidade de fazer aplicações de visão computacional comerciais através da performance otimizada e pelo fato da biblioteca ser disponibilizada gratuitamente.

A primeira versão do OpenCV foi lançada para o público na Conferência de Visão Computacional e Reconhecimento de padrões da IEEE em 2000. E em 2009 a segunda versão foi lançada com grandes avanços e mudanças na interface C++. O OpenCV constitui a biblioteca base da detecção de movimentos deste projeto.

1.1 Histórico e o Estado da Arte

A ideia de sistemas de segurança vem do início dos anos 70 segundo o autor André Vitalis, Ph.D. em ciência política. Os sistemas surgiram devido à necessidade de regulação do tráfego rodoviário e à luta contra o roubo a bancos e a lojas de luxo [9].

Desde o início esse novo tipo de vigilância despertou uma suspeita e no final dos anos 80 na França, a Comissão Nacional de Informática e Liberdades (CNIL) criou as primeiras medidas de proteção. O CNIL é um órgão regulador administrativo francês independente que tem como objetivo garantir que a lei de privacidade dos dados seja aplicada. Uma pesquisa feita em 1996 revela que a aceitação social dos sistemas de vigilância varia de acordo com a aplicação [9]. "Apenas 9% dos entrevistados consideram o uso de câmeras em estacionamentos e lojas como uma invasão de privacidade; No entanto 51% acreditam que a captura de imagens públicas constituem uma grande violação".

Os primeiros sistemas de segurança precisavam funcionar o tempo todo pois não havia uma maneira eficiente de salvar essas informações, isso só se tornou mais viável em 1970 com a tecnologia das fitas VHS [10].

Nos anos 90, a multiplexação digital surgiu e permitiu a gravação de várias câmeras ao mesmo tempo e a gravação apenas dos movimentos, isso causou um aumento do uso de sistemas de monitoramento [11].

A princípio, o monitoramento residencial serviu para vigiar abusos domésticos ou vigiar o cuidado e o serviço de babás e faxineiras. O sucesso destes produtos provocou o desenvolvimento de novas versões de câmeras digitais cada vez menores e melhores.

Algoritmos de reconhecimento facial para rastreamento de crianças desaparecidas e identificação de infratores sexuais registrados surgiram em 2003 em Phoenix, Arizona. Depois com a internet, foi possível acessar remotamente uma câmera de qualquer lugar.

O estado da arte do monitoramento residencial é constituído de sistemas capazes de detectar movimentos, como faz a empresa FLIR [12], que possui câmeras de segurança que detectam movimento em tempo real e possuem também outras funcionalidades. Além desta empresa, outras câmeras de monitoramento existem com interfaces mais amigáveis e portáteis para uso residencial como por exemplo a Ulo [13].

1.2 Objetivos

O objetivo do projeto é o desenvolvimento de um sistema de vigilância de baixo custo, confiável

e diferenciado para uso residencial utilizando *Linux* embarcado em *BeagleBone Black* como base para suportar o processamento de imagens, as configurações do sistema e as informações *on-line* e registradas em base de dados.

1.3 Motivação

Um desafio atual do Brasil é a segurança. Não é incomum o noticiário mostrar novas mortes ou assaltos. Até mesmo em casa é difícil sentir-se seguro, sentimento até que justificável devido ao número de roubos à residências que chega a ser de aproximadamente 2,27% do total de roubos em 2015 em São Paulo [14]. Os dados chegam a ser ainda mais assustadores se levarmos em consideração o número de homicídios dolosos em São Paulo, 3.113 casos registrados em 2015.

Apesar de várias instituições tentarem remediar este problema, sempre é recomendável que se invista em segurança pessoal, seja com alarmes residenciais, sensores de presença na frente das casas, serviços de segurança mensais ou outras soluções. Mas a maioria dos sistemas ainda não proporcionam uma sensação muito grande de segurança. Segundo o Jornal Estadão, apenas 36% dos brasileiros disseram se sentir seguros, ficando entre as 15 piores avaliações [15].

A ideia é que o projeto nos permita sair de casa, ir trabalhar ou viajar sem ter a preocupação com a segurança da casa. A aplicação do projeto não se limita somente a ambientes residenciais, mas também a escritórios, jardins, salas de aula, ambientes restritos e muitos outros ambientes. A vantagem de sua utilização é que saibamos imediatamente caso algum incidente ocorra. Dessa maneira uma providência é tomada com mais agilidade do que alarmes comuns, que dependem da vizinhança, do acaso ou do pagamento de mensalidades a empresas de segurança. Além disso, todas as imagens do que ocorreu durante o incidente estarão armazenadas na nuvem.

Além do interesse na área de segurança e na sensação de sentir-se seguro, outra área de interesse são as casas inteligentes e a "*Internet* das Coisas" que estão cada vez mais comuns hoje em dia. A ideia de automação residencial tem como objetivo a simplificação da vida diária das pessoas. Já existem atualmente sistemas de controle de iluminação, climatização e de segurança.

O projeto tem como inspiração a adição de outros recursos relacionados com segurança e com utilidade em geral, tornando a residência ou local de aplicação conectados à *Internet*. Outros sistemas que podem ser adicionados: sistema de alarme, interfones inteligentes, fechaduras inteligentes, iluminação inteligente etc.

Os sistemas embarcados, que são computadores encapsulados ou dedicados a um dispositivo, também são uma área de interesse. Diferente de computadores de uso geral, os embarcados realizam um conjunto de tarefas predefinidas e com requisitos geralmente específicos.

Por último é importante ressaltar o interesse na área de visão computacional e de aprendizagem de máquina. Vários conceitos de ambas as áreas foram utilizados no projeto.

1.4 Organização do Trabalho

Para melhor organização e entendimento do texto, a monografia foi dividida em 5 capítulos listados a seguir:

- **1. Introdução -** Capítulo atual que apresenta uma introdução histórica do tema, objetivos e motivação.
- **2. Embasamento Teórico -** Capítulo que aborda todos os conceitos básicos necessários para a execução do trabalho, destacando as tecnologias e plataformas usadas para a implementação.
- **3. Materiais e Métodos -** Capítulo que apresenta todas as ferramentas e dispositivos utilizados para o desenvolvimento do trabalho.
- **4. Resultados e Discussões -** É o capítulo que sumariza o resultado das ferramentas e dispositivos utilizados para o desenvolvimento do projeto. Portanto, apresenta o resultado final do trabalho e perspectivas futuras.
- **5. Conclusão -** Este capítulo conclui todo o trabalho em aspectos de viabilidade, resultados, eficiência e confiabilidade do projeto.

Capítulo 2

Embasamento Teórico

Neste capítulo todas as tecnologias utilizadas no projeto são descritas quanto à funcionalidade, além disso uma revisão dos conceitos que foram usados no projeto de cada um dos tópicos é feita.

2.1 Arquitetura ARM

A arquitetura ARM, originalmente *Acorn RISC Machine* e posteriormente *Advanced RISC Machine* é uma família de conjunto de instruções reduzidas (RISC) para processadores, foi criada pela companhia britânica *ARM Holdings*, que atualmente desenvolve novas tecnologias e licencia para outras empresas que criam os seus próprios produtos usando a arquitetura.

Um computador RISC requer menos transistores do que um computador de instruções CISC (conjunto complexo de instruções), isso reflete no custo, peso e consumo reduzidos, justificando sua aplicação em dispositivos como telefones celulares e tablets por exemplo. Em 2014, cerca de 50 bilhões de processadores ARM foram produzidos, além de ser a arquitetura mais utilizada em termos de quantidade produzida [16].

A arquitetura ARM evoluiu ao longo do tempo e a partir da versão sete da arquitetura, três perfis de operação foram definidos: o perfil de aplicações, o perfil de tempo real e o perfil de microcontrolador.

A arquitetura possui nove modos de operação da CPU dependendo da versão. É possível mudar o modo de operação através de comandos ou através de eventos externos.

As principais características da arquitetura ARM são:

- Arquitetura *Load/Store* múltipla: instruções afetam apenas valores de registradores e sempre armazenam os resultados em registradores.

- Instruções fixas de 32 bits, com exceção de instruções *Thumb* (que tem a mesma função das instruções fixas porém são codificadas em 16 *bits*, trata-se de um modo de operação diferente[17]).

- Execução condicional de instruções.
- Controle sobre o deslocamento e sobre a Unidade Lógica Aritmética.
- 15 Registradores de 32 bits para uso geral, incluindo o *Program Counter*, *Link Register* e o *Stack Pointer*.
- Auto incremento e decremento para ciclos de *loop*.
- Baixo consumo de energia.
- Manipulação de periféricos de I/O como dispositivos mapeados na memória com suporte à interrupções.

2.2 Linux

Em termos gerais, o *Linux* é o *kernel* de um sistema operacional desenvolvido sob o modelo de desenvolvimento e distribuição *open-source* e gratuito, o que contribuiu imensamente para sua popularização. O que define esse sistema é o núcleo *Linux*, que foi desenvolvido por Linus Torvalds e lançado em 1991.

Kernel é o código responsável pelo núcleo do sistema operacional e é responsável por controlar tudo no sistema.

O núcleo *Linux* é um dos componentes do sistema operacional que é a base do funcionamento deste. É o programa responsável por fazer a interface direta com o *hardware*, gerenciar a comunicação de periféricos e também a memória do sistema, além de decidir a cada momento qual programa deverá ter acesso à Unidade Central de Processamento do sistema.

Diferente de outros sistemas, o *Windows* por exemplo, o núcleo do *Linux* é modular, ou seja, há independência entre as diferentes partes que o compõe, a queda de uma dessas partes não necessariamente implica no comprometimento dos outros componentes. Esta característica gera maior estabilidade, segurança e grande facilidade em se adaptar o núcleo para funcionar em diferentes arquiteturas e sistemas, o que contribui para a popularização deste *software* [18].

Em relação a sistemas embarcados, tipicamente refere-se ao sistema operacional como um *Linux* embarcado (ou *eLinux*), que nada mais é do que uma versão do programa adaptada para ser executada em um sistema específico (com menos poder de processamento no caso), já levando em consideração pacotes e rotinas necessários para se fazer uso total de todos periféricos e conexões presentes em uma placa de sistema embarcado. Neste projeto, foi utilizado uma versão da distribuição *Debian* disponibilizada pelo fabricante da *BeagleBone Black*.

Esta escolha por uma distribuição baseia-se na versatilidade, recursos disponíveis, atualizações, estabilidade, entre outros fatores que devem ser compatíveis com a aplicação pretendida do sistema embarcado, devendo haver suporte para os diferentes programas a serem utilizados. Atualmente existem diversas distribuições *Linux* para diferentes sistemas, portanto, cabe ao usuário escolher devidamente uma para poder trabalhar.

Um sistema *Linux* é composto resumidamente por três componentes: *Kernel, Bootloader e root filesystem*. O *Kernel* é o que faz a interface entre o *hardware* e as aplicações em execução. As bibliotecas, arquivos, configurações de sistema e de usuário e programas ficam todos armazenados no *root filesystem*. O *bootloader* se localiza na primeira parte da memória e é responsável pela preparação do *hardware* para a execução do *Kernel*.

Existe um conjunto de ferramentas para compilação e depuração de código chamada *toolchain* (cuja tradução seria conjunto de ferramentas). Esse conjunto é utilizado para o desenvolvimento de aplicações ou sistemas complexos para ambientes Linux embarcados em plataformas ARM. Cada *toolchain* possui basicamente 5 elementos:

- **Compilador GCC:** É o Compilador C GNU compatível com diversas linguagens além de ser capaz de gerar código para arquiteturas como x86, ARM, entre outras.
- **Binutils:** É um conjunto de ferramentas capaz de manipular programas binários para uma arquitetura em específico.
- **Biblioteca Padrão C:** É uma biblioteca responsável por fazer a interface do programa com o *Kernel* através de chamadas do sistema. Para embarcados, essa biblioteca utiliza a implementação "uClibc", devido à sua otimização para gerar um código menor.
- **Kernel Headers:** É a lista de chamadas de sistema da versão e é utilizada pela biblioteca padrão C. Essa biblioteca também precisa ter acesso à constantes e estruturas do *Kernel*.
- ***Debugger:*** O *debugger* padrão do *Linux* embarcado é o GDB. É usado para encontrar defeitos de um programa.

2.3 Debian

A distribuição *Debian* é um sistema operacional baseado em *Linux*, composta inteiramente por *software* gratuito, havendo três tipos de versões disponíveis: teste, instável e estável. Sendo que esses tipos foram levados em consideração quando a escolha da distribuição foi feita, uma vez que uma versão pode possuir mais recursos que outra mas também pode ser instável e dificultar o progresso do

desenvolvimento do projeto. Uma versão nova é lançada como instável, após tempo e uso suficiente esta versão é mudada para teste e somente após um nível de maturidade suficiente a versão passa a ser estável.

O Projeto *Debian* se baseia na associação de voluntários para desenvolver um sistema operacional livre e gratuito, novas versões são lançadas com frequência. Este fato aliado à estabilidade, suporte da comunidade na *Internet*, e os mais de 43.000 pacotes pré compilados que acompanham o programa, tornam este sistema operacional atraente para desenvolvedores [19].

Apesar da possibilidade de se utilizar *Debian* com interface gráfica, optou-se por desenvolver todo o projeto através do terminal de comandos pela facilidade de se obter suporte na comunidade *online*, pelo desenvolvimento das habilidades de programação, maior agilidade do sistema e possibilidade de desenvolvimento do projeto a partir de qualquer computador com acesso à rede.

Uma parte fundamental do desenvolvimento do projeto de monitoramento foi o conhecimento dos comandos básicos do *Linux* para poder aproveitar totalmente os recursos que a *BeagleBone* tem a oferecer. Os comandos básicos e suas respectivas funcionalidades estão listados a seguir.

cd : Escolhe diretórios e navega por estes.

ls -al : Lista todos arquivos e diretórios presentes dentro do diretório atual.

sudo : Realiza um comando no modo super usuário.

su : Entra no modo super usuário (requer senha, *temppwd* por padrão do *Debian*).

adduser : Cria um novo usuário.

passwd : Modifica a senha de um usuário.

apt-get : Rotina responsável pela instalação e atualização de pacotes.

kill : Para um processo em andamento.

make : Compila programas.

rm : Deleta arquivos e diretórios especificados (deve-se ter cuidado ao usar esse comando).

mv : Move arquivos ou diretórios.

reboot : Reinicia o sistema.

top : Mostra os processos em andamento no sistema ordenado por uso de CPU.

cp : Faz cópia de arquivos ou diretórios.

fg : Continua um trabalho parado executando-o em primeiro plano.

bg : Continua um trabalho parado executando-o em útlimo plano. [20]

2.4 OpenCV

O OpenCV é uma biblioteca de visão computacional e processamento de imagens. Possui mais de 2500 algoritmos prontos para uso como por exemplo em filtros, redimensionamento, processamento de cores etc. Além disso o OpenCV é compatível com as linguagens C, C++, Java e Python possuindo o melhor desempenho para implementações em C ou C++ devido à realização de operações matemáticas pesadas com paralelismo quando executado em computadores com mais de um núcleo.

Esta biblioteca é modular e possui bibliotecas compartilhadas. Os módulos a seguir estão disponíveis na biblioteca, mas não necessariamente foram utilizados no projeto [21].

- **core -** Módulo principal responsável por definições de estruturas de dados básicas, como matrizes multidimensionais e algumas outras funções usadas por outros módulos.
- **imgproc -** Módulo de processamento de imagens. Inclui filtros, transformações geométricas de imagens, histogramas e outras funções.
- **video -** Módulo que possui algoritmos para processamento de vídeo como rotinas para estimação de movimento, remoção de fundos etc.
- **calib3d -** Módulo para visualização de imagens em três dimensões e rotinas para estimar a posição de objetos e reconstrução 3D.
- **features2d -** Módulo com rotinas para detecção e extração de características e descritores de imagens.
- **objdetect -** Módulo para detecção de objetos, faces, olhos, carros, entre outros.
- **highgui -** Módulo que faz interface para captura de vídeo e imagem, além de *codecs*. Também possui rotinas para criação de interfaces simples para o usuário.
- **gpu -** Módulo para aceleração de algoritmos com execução na GPU.

2.5 Processamento de Imagens

Processamento de Imagens é uma área em crescimento devido à grande necessidade da aproximação da máquina com o Homem. O projeto de monitoramento poderia ser realizado com um vigia trabalhando 24 horas por dia observando as imagens obtidas pela câmera. A vantagem do uso do computador vai além da dispensa do fator humano, mas também deve tornar possível a execução de outras tarefas durante o monitoramento.

As imagens, assim como os sinais, são um suporte físico que carregam em seu interior uma informação. Essa informação pode ser associada à uma medida (fenômeno físico) ou a um nível cognitivo (conhecimento). Ou seja, o processamento de uma imagem pode ter como objetivo o cálculo da distância entre objetos da imagem ou então a identificação de um ato de roubo. Processamento de imagens está relacionado portanto à obtenção da informação nela presente.

O termo *análise* se relaciona com a parte do tratamento da imagem onde existe uma descrição da informação presente na imagem. Pode ser chamada de parametrização e vários parâmetros são usados para descrição de diferentes informações dentro de uma imagem. A análise pode ser tão complexa quanto necessária dependendo do tipo de aplicação.

De modo geral, trabalha-se com imagens em níveis de cinza com o objetivo de aumentar o contraste da imagem ou colocar em evidência uma região de interesse particular. Uma aplicação direta da binarização da imagem é a subtração desta por outra imagem de referência ou então a aplicação de um limite (*threshold*) para destacar regiões de interesse e possibilitar a visualização, correção de deformações espaciais devido à ótica ou à uma variação de inomogeneidade da iluminação de fundo e dessa maneira fazer o tratamento desejado.

É esperado que um sistema de imagens responda a todas as situações possíveis assim como um olho humano responderia. Porém os computadores atuais são eficientes para trabalhar com bases de dados, cálculos numéricos e formais, mas não são eficazes para realização de tarefas consideradas simples para seres biológicos.

As etapas para a construção de um sistema de processamento de imagens pode ser, de maneira geral, descrita por:

1. **Tratamento fora da imagem:** Correção da iluminação, posicionamento da câmera, entre outros tratamentos.

2. **Aquisição da Imagem:** Responsável pela amostragem, armazenamento e compactação da imagem.

3. **Processamento:** É o tratamento inicial que a imagem irá receber. Esse tratamento pode ser o redimensionamento da imagem, a binarização da imagem ou o borramento da imagem, por exemplo.

4. **Segmentação da informação:** Extração dos objetos secundários da imagem.

5. **Parametrização:** Determinação das grandezas relacionadas sobre cada objeto da imagem como por exemplo a área, perímetro, forma, topologia, entre outros.

6. **Reconhecimento:** Classificação dos objetos.

7. **Análise quantitativa:** Associação das grandezas com o problema.

Em geral, as aplicações de processamento de imagem não possuem solução única devido à grande variedade de situações e problemas para a qual o sistema foi projetado. Além do foco do projeto, as limitações de *hardware* levam o sistema a ter ainda mais múltiplas soluções.

Basicamente a entrada de um sistema de processamento de imagem é constituída por uma câmera capaz de captar a imagem de uma cena. Essa imagem é apresentada após uma conversão analógica digital e a mesma é discretizada espacialmente e em luminância (níveis de cinza). A câmera é portanto o sensor de aquisição de imagem e pode ser tão sofisticada quanto se é desejado.

A imagem obtida pela câmera possui três componentes de cor, vermelho, verde e azul, armazenadas separadamente na memória. A binarização é responsável por transformar esses três componentes em apenas uma em escala de cinza. Para isso é necessário um algoritmo que traduza cores para um equivalente na escala de cinza, tal algoritmo existe no OpenCV e é utilizado no próximo capítulo.

Um *pixel* é o elemento básico de uma imagem. Pode ser representado de forma quadrada e frequentemente a organização de uma imagem se dá sob a forma de uma matriz de *pixels*. Cada *pixel* possui 4 vizinhos de borda e 4 vizinhos de diagonal. Isso significa que as distâncias entre um ponto e seus vizinhos não são sempre a mesma dependendo do tipo de vizinho. A distância é igual a 1 para vizinhos de borda e igual a $\sqrt{2}$ para vizinhos na diagonal [22].

2.6 TensorFlow

O *TensorFlow* é uma biblioteca *open source* para aprendizagem de máquina criada pela empresa *Google* para utilização em sistemas capazes de construir e treinar redes neurais para detectar e decifrar padrões e correlações [23].

Para este projeto foi utilizada uma técnica chamada de **transferência de aprendizado** usando um modelo já treinado em outro problema. O processo de transferência de aprendizado é muito mais rápido do que aprendizagem em profundidade, que pode levar dias para ser treinado. A transferência de aprendizado consiste em retreinar apenas uma região específica de uma rede neural convolucional, mantendo as camadas anteriores intactas [24]. O aprendizado em profundidade é baseado em um conjunto de algoritmos que modelam abstrações de alto nível de dados usando um grafo profundo com várias camadas de processamento, compostas de várias transformações lineares e não lineares [25].

Foi utilizada a rede *Inception v3*, que é uma rede neural treinada para o *ImageNet Large Visual Recognition Challenge* de 2012 capaz de diferenciar 1000 classes (por exemplo carro, navio ou cachorro)

diferentes [26] [27]. Para o projeto, essa rede foi retreinada para diferenciar um número pequeno de classes características do problema em questão.

O *ImageNet* é uma base de dados de imagens organizado de acordo com a hierarquia do *WordNet*, onde cada nó da hierarquia é representado por centenas e milhares de imagens [28]. O *WordNet*, por sua vez, é um grande banco de dados da língua inglesa que inclui substantivos, verbos, adjetivos e advérbios em conjuntos de sinônimos [29].

A rede *Inception v3* [30] pode ser simplificada por um diagrama de blocos conforme a Figura 2.1. As primeiras camadas são responsáveis pela detecção de bordas, seguidas pelas camadas centrais que detectam formas. As últimas camadas ficam cada vez mais abstratas e específicas. As duas últimas camadas fazem as conclusões e o reconhecimento da imagem e são elas que serão retreinadas com categorias novas.

Foi necessário ter uma base de dados de imagens que representassem muito bem as categorias que foram inseridas. As categorias que foram escolhidas para fazer parte da transferência de aprendizado foram: ambientes vazios, ambientes ocupados e rostos. A princípio foram reunidas uma série de imagens generalizando o máximo possível as categorias citadas, porém a grande variância entre as imagens provocou uma eficiência muito baixa da rede neural ao identificar imagens. Isso acontece devido ao fato de os ambientes onde a câmera pode ser colocada serem diferentes por natureza. Por exemplo, uma cozinha é completamente diferente de uma garagem, ou um jardim.

A solução do problema da generalização foi criar uma base de dados constituída por imagens específicas do cômodo onde o monitoramento foi feito. Como consequência é necessário que o usuário simule a categoria de ambientes ocupados passando repetidamente no campo de visão da câmera.

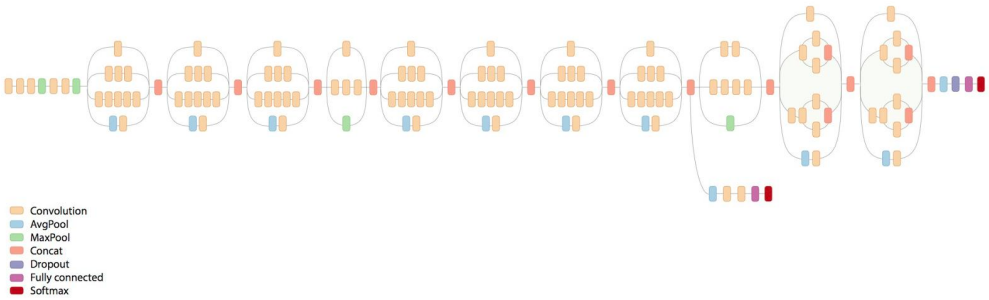

Figura 2.1: Rede neural convolucional do *Inception v3*.

Uma das categorias, a de rostos, apresentou uma eficiência superior a 95% no reconhecimento. Essa categoria é constituída de uma série de fotos de rostos de vários ângulos e foi mantida generalizada, diferente das outras duas categorias (ambientes ocupados e ambientes vazios).

Capítulo 3

Materiais e Métodos

Neste capítulo apresenta-se o sistema embarcado *BeagleBone Black* e seus devidos periféricos juntamente com todas as ferramentas computacionais utilizadas para o desenvolvimento do projeto. Em adição, detalha-se os procedimentos utilizados para compilação e ajuste dos programas.

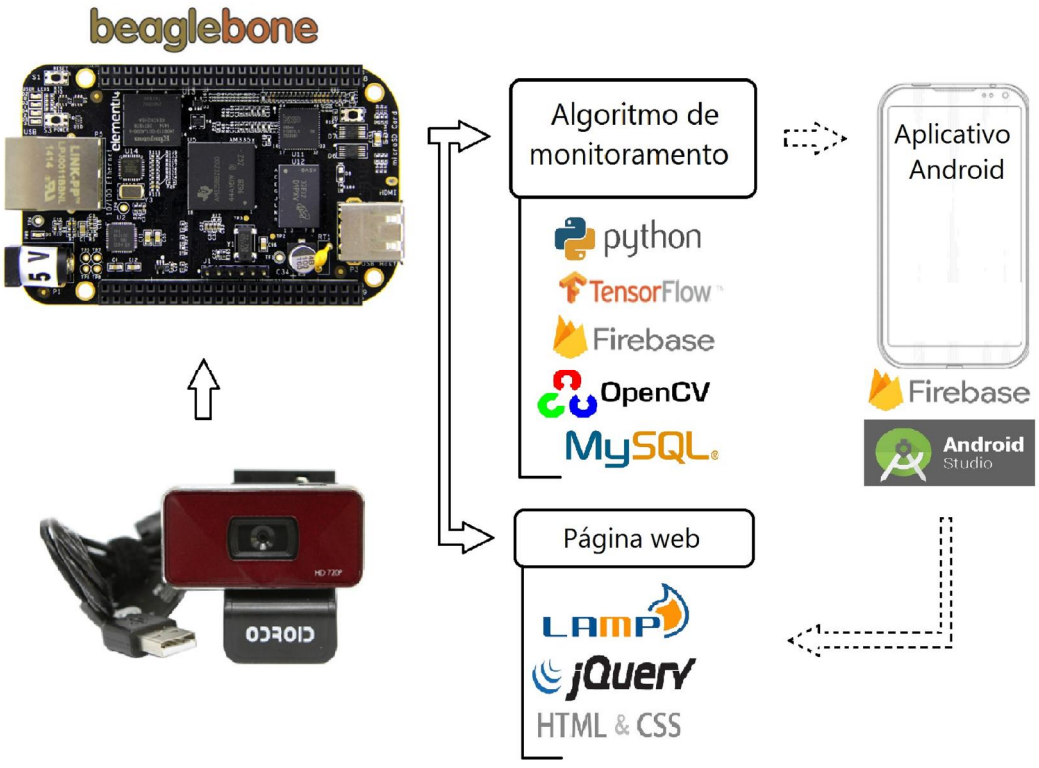

Figura 3.1: Diagrama de blocos do projeto.

A Figura 3.1 mostra um diagrama de blocos de como o projeto foi estruturado. As setas sólidas representam programas e serviços dentro do sistema embarcado, as setas pontilhadas representam comunicação HTTP com a base de dados. Apenas o sistema embarcado *BeagleBone* e a câmera *ODROID* fazem parte do *hardware* do projeto.

3.1 Materiais

3.1.1 BeagleBone Black

A *BeagleBone Black* é um *hardware* aberto (*opensource*) produzido pela *Texas Instruments* em associação com a *Digi-Key* e a *Newark element14*. Ela também foi desenvolvida com *software* aberto (*opensource*). A placa foi desenvolvida por um pequeno grupo de engenheiros com o objetivo de criar uma placa educacional para uso em universidades pelo mundo.

A placa foi escolhida por seu grande poder de processamento e disponibilidade de diversas conexões e portas de comunicação. Todas as características podem ser vistas em detalhes na Figura 3.3 [31]. Também é possível visualizar os componentes e conexões presentes na placa, bem como suas disposições na Figura 3.2. A designação do sistema embarcado também foi motivada pela grande quantidade de fóruns de suporte [32].

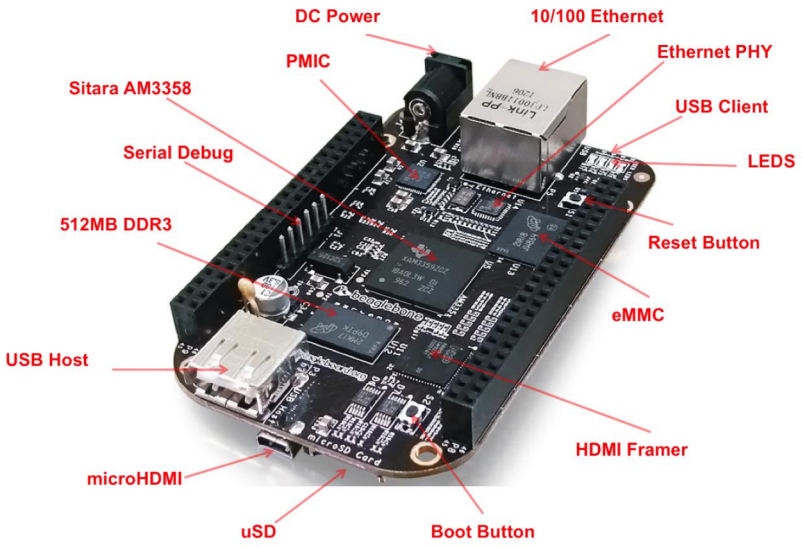

Figura 3.2: *BeagleBone Black* em detalhe com seus componentes nomeados.

Este circuito embarcado possui tamanho próximo ao de um cartão de crédito e funciona como se fosse um computador. Seu custo é de cerca de 50 dólares americanos e mais informações a respeito de seu funcionamento e dúvidas gerais podem ser encontrados em seu site oficial [33].

	Especificação	
Processador	Sitara AM3358BZCZ100 1GHz, 2000 MIPS	
Placa de vídeo	SGX530 3D, 20M Polygons/S	
Memória SDRAM	512MB DDR3L 800MHZ	
Onboard Flash	4GB, 8bit Embedded MMC	
PMIC	TPS65217C PMIC	
Suporte ao Debug	Optional Onboard 20-pin CTI JTAG, Serial Header	
Alimentação	miniUSB USB or DC Jack	5VDC External Via Expansion Header
PCB	3.4" x 2.1"	6 Camadas
Indicadores	1-Power, 2-Ethernet, 4-User Controllable LEDs	
HS USB 2.0 Client Port	Access to USB0, Client mode via miniUSB	
HS USB 2.0 Host Port	USB1, Type A 500mA LS/FS/HS	
Porta Serial	UART0 access via 6 pin 3.3V TTL Header. Header is populated	
Ethernet	10/100, RJ45	
Conector SD/MMC	microSD , 3.3V	
Botões	Botão Reset Botão Boot BotãoPower	
Saída de Vídeo	16b HDMI, 1280x1024 (MAX) 1024x768,1280x720,1440x900 ,1920x1080@24Hz w/EDID Support	
Áudio	Via HDMI Interface, Stereo	
Conectores de Expansão	Power 5V, 3.3V , VDD_ADC(1.8V) 3.3V I/O on all signals McASP0, SPI1, I2C, GPIO(69 max), LCD, GPMC, MMC1, MMC2, 7 AIN(1.8V MAX), 4 Timers, 4 Serial Ports, CAN0, EHRPWM(0,2),XDMA Interrupt, Power button, Expansion Board ID (Up to 4 can be stacked)	
Peso	39.68 grams	

Figura 3.3: Tabela de especificações da *BeagleBone Black* .

3.1.2 Câmera

Uma câmera *ODROID* com interface USB 2.0 foi utilizada no projeto. Capaz de filmar com resolução máxima de 720P HD, formato de vídeo 16:9 *widescreen* e a 30 quadros por segundo, o dispositivo faz uso de um sensor de 1.0 Megapixel para tal tarefa. Em adição, possui um microfone embutido que pode ser acessado pelo conector USB único da câmera, no entanto, este não foi utilizado no projeto.

Possui característica *plug-n-play*, o que possibilita que a câmera seja reconhecida pelo sistema embarcado sem a necessidade de instalação de programas adicionais para isso. Além disso, tem especificação de alimentação DC de 5V e 500mA, o que pode ser fornecido através da conexão USB. Por fim, é importante dizer que o ângulo de visão do dispositivo é de 65 graus [34]. A câmera pode ser vista na Figura 3.4 [34].

Figura 3.4: Câmera *Odroid* utilizada.

3.2 Métodos

3.2.1 Instalando a distribuição Linux

O primeiro passo para se iniciar o desenvolvimento foi a instalação de um sistema operacional baseado no *Unix*. A distribuição escolhida foi o *Debian* 7.9. Essa versão possui uma interface gráfica e mostrou-se bastante eficiente para a aplicação desejada pois não ocupa muita memória, não apresentou problemas durante as configurações iniciais (*Internet, SSH*) e por ser uma versão estável.

Outras distribuições do *Debian* também foram utilizadas, como a versão 8.4 (para SD) e a versão 8.4 com *flasher*. Porém ambas apresentaram problemas na conexão com a *Internet* ou com o espaço ocupado pela distribuição.

A versão 8.4 era relativamente lenta e apresentou grandes dificuldades ao se conectar com a *Internet*. A versão 8.4 com *flasher* é mais rápida por ser mais leve mas apresentou um grande problema com a falta de memória à medida que eram instalados novos pacotes e programas.

Várias versões de distribuições Linux (*Debian* em particular) podem ser encontradas na página *web* oficial do sistema embarcado *BeagleBone*, inclusive a versão utilizada no projeto. Para evitar o problema de falta de memória, foi decidido a instalação da distribuição diretamente no cartão SD. Logo, os passos seguidos foram:

1. O cartão SD foi formatado usando o *SDCard Formatter* [35].

2. A imagem da distribuição foi gravada no cartão SD usando o *Win32 Disk Imager* [36].

3. O cartão SD foi inserido na placa e bastou, então, aguardar a instalação terminar.

Antes de prosseguir para o próximo passo, é necessário redimensionar o tamanho da partição do cartão SD, pois a instalação redefine o tamanho da partição como sendo igual ao tamanho do sistema operacional. Para isso foi utilizado o programa *gparted* [37], que é um gerenciador de partições de

disco. Basta conectar o cartão SD e redimensionar para o tamanho da partição desejada. Em seguida foi inserido na placa o cartão SD, um cabo HDMI (para visualização em um monitor), um cabo de *ethernet* no conector RJ45, um teclado na porta USB e então a placa foi ligada.

3.2.2 Configurando a rede

Através de um monitor foi possível o acesso da interface gráfica da distribuição. A primeira configuração feita na placa é a configuração da *Internet* assim como mostra a Figura 3.5.

```
  GNU nano 2.2.6             File: /etc/network/interfaces

# This file describes the network interfaces available on your system
# and how to activate them. For more information, see interfaces(5).

# The loopback network interface
auto lo
iface lo inet loopback
auto eth0
iface eth0 inet static




# The primary network interface
#auto eth0
#iface eth0 inet dhcp
# Example to keep MAC address between reboots
#hwaddress ether DE:AD:BE:EF:CA:FE

# The secondary network interface
#auto eth1
#iface eth1 inet dhcp

# WiFi Example
#auto wlan0
#iface wlan0 inet dhcp
#    wpa-ssid "essid"
#    wpa-psk  "password"

# Ethernet/RNDIS gadget (g_ether)
# Used by: /opt/scripts/boot/autoconfigure_usb0.sh
#iface usb0 inet static
    address 10.235.10.46
    netmask 255.255.255.0
    network 10.235.10.0
    gateway 10.235.10.1
    broadcast 10.235.10.255
dns-nameservers 8.8.8.8
dns-nameservers 8.8.4.4


^G Get Help    ^O WriteOut    ^R Read File   ^Y Prev Page   ^K Cut Text    ^C Cur Pos
^X Exit        ^J Justify     ^W Where Is    ^V Next Page   ^U UnCut Text  ^T To Spell
```

Figura 3.5: Configuração do arquivo *interfaces*.

O primeiro arquivo a ser editado para a conexão com a *Internet* é o *interfaces*. Para isso foi aberto um terminal no *Debian* e utilizado o editor de texto *nano*. Para a edição de um arquivo de qualquer extensão o comando a seguir foi utilizado:

$ sudo nano <local do arquivo>

Os arquivos de configuração da internet estão, na distribuição Debian 7.9, nos diretórios abaixo:

/etc/network/interfaces

/etc/ssh/sshd_config

/etc/resolv.conf

Em seguida foi alterada a porta da conexão de 22 para 22000 no arquivo *sshd_config* assim como a Figura 3.6.

```
  GNU nano 2.2.6                File: /etc/ssh/sshd_config

# Package generated configuration file
# See the sshd_config(5) manpage for details

# What ports, IPs and protocols we listen for
Port 22000
# Use these options to restrict which interfaces/protocols sshd will bind to
#ListenAddress ::
#ListenAddress 0.0.0.0
Protocol 2
# HostKeys for protocol version 2
HostKey /etc/ssh/ssh_host_rsa_key
HostKey /etc/ssh/ssh_host_dsa_key
HostKey /etc/ssh/ssh_host_ecdsa_key
#Privilege Separation is turned on for security
UsePrivilegeSeparation yes
```

Figura 3.6: Configuração do arquivo *sshd_config*

Por último foram adicionadas duas linhas de comando para especificar os endereços DNS para a rede. DNS, ou *Domain Name System*, é um protocolo hierárquico que armazena e divulga a relação entre endereço IPs e domínios [**?**]. Isso é feito no arquivo *resolv.conf*, que pode ser visto na Figura 3.7.

```
  GNU nano 2.2.6                File: /etc/resolv.conf

# Dynamic resolv.conf(5) file for glibc resolver(3) generated by resolvconf(8)
#     DO NOT EDIT THIS FILE BY HAND -- YOUR CHANGES WILL BE OVERWRITTEN
nameserver 8.8.8.8
nameserver 8.8.4.4
nameserver 127.0.0.1
```

Figura 3.7: Configuração do arquivo *resolv.conf*.

Apenas mudar e adicionar linhas de código nos arquivos *interfaces, sshd_config e resolv.conf* não faz com que o *Debian* entre automaticamente na rede. Foi preciso reiniciar o processo para aplicar as configurações, isso foi feito utilizando o seguinte comando:

$ sudo /etc/init.d/networking restart

Não é possível mudar o endereço MAC da *BeagleBone*.

Caso a interface *eth0* não ligue automaticamente é possível forçar a inicialização:

$ sudo ifconfig eth0 10.235.10.46 netmask 255.255.255.0

Foi possível verificar a conexão com a *Internet* utilizando o comando:

$ ifconfig

Foi também possível verificar se as configurações de DNS funcionam com o comando:

$ ping 8.8.8.8

ou

$ ping www.google.com

Depois dessas configurações é possível acessar o terminal via SSH (*secure shell*), que é um protocolo de rede que permite a conexão com outro computador na rede de forma a autorizar comandos através uma unidade remota. O SSH possui também criptografia na conexão entre o cliente e o servidor. Um exemplo de programa para conexão SSH é o *PuTTy* [38].

3.2.3 Usuários e senhas

Foi preciso trocar as senhas de acesso para que o acesso remoto à placa seja restrito. Após instalar a distribuição *Debian* na placa e configurar a *internet*, o usuário pôde se conectar remotamente via SSH no usuário raiz (*root*) ou no usuário *debian*, ambos pela senha padrão *temppwd*. Isso significa que qualquer pessoa que tenha o IP da placa pode se conectar remotamente.

Para adicionar um novo usuário o comando **adduser** foi utilizado. E para mudar a senha de algum usuário o comando **passwd <user>** foi efetuado. Foi alterada a senha de todos os usuários da placa para evitar uma possível invasão.

$ sudo adduser <usuário>

$ sudo passwd <usuário> <senha>

Em seguida, vários programas foram instalados utilizando o usuário *root*, pois este usuário possui permissão para qualquer operação no sistema, o comando **sudo** (*super-user do*) no início de algum comando executado por este usuário é redundante. A vantagem do uso do **sudo** é a limitação de permissão, pois é necessário adicioná-lo em todos os comandos quando se trabalha em qualquer usuário se não o *root* e isso permite que o usuário tenha maior atenção ao executar cada um deles.

Foi possível também adicionar um usuário ao grupo de **sudo**, isso irá garantir que o usuário tenha permissão como as do usuário *root*, para isso:

$ sudo adduser <usuário existente> sudo

3.2.4 Instalando o LAMP

Antes de instalar os pacotes foi importante manter a distribuição atualizada utilizando os dois primeiros comandos a seguir. O *Debian* está em constante desenvolvimento, portanto atualizações ocorrem com muita frequência e é importante manter os arquivos atualizados. O primeiro dos comandos atualiza a lista de pacotes e versões disponíveis (mas não instala nem atualiza nenhum pacote), o segundo comando instala todos os pacotes novos obtidos pelo primeiro comando. Por este motivo é necessário que a ordem dos comandos seja respeitada.

$ sudo apt-get update

$ sudo apt-get upgrade

O pacote *LAMP* é um conjunto de *softwares* de código aberto usados para hospedar servidores. Esse acrônimo significa *Linux*, *Apache*, *MySQL* e *PHP* [39].

O primeiro pacote que foi instalado é o *Apache*, que é um servidor de código de fonte aberta usado em pelo menos 40% dos servidores do mundo [40]. Para instalar o *Apache* basta executar o comando *apt-get*, que é o gerenciador de pacotes do *Debian*. Esse comando é responsável pela instalação de pacotes pré-compilados para o sistema operacional utilizado.

$ sudo apt-get install apache2

Pode-se verificar se a instalação foi bem sucedida utilizando um navegador qualquer e utilizando como URL o IP 143.107.235.46. Uma página contendo *It works!* deve aparecer.

Em seguida foi instalado o *MySQL*, que é um *software* de gerenciamento de banco de dados usado para organizar e recuperar dados. Isso pôde ser feito utilizando o seguinte comando:

$ sudo apt-get install mysql-server

Durante a instalação uma senha foi requerida. Assim que a instalação terminou foi necessário testá-la e criar um banco de dados para esta. Para testar a instalação foi necessário inserir o comando:

$ mysql -u root -p

A senha definida foi inserida e em seguida é possível visualizar quais os bancos de dados disponíveis com:

$ show databases;

Algo similar à Figura 3.8 apareceu no terminal.

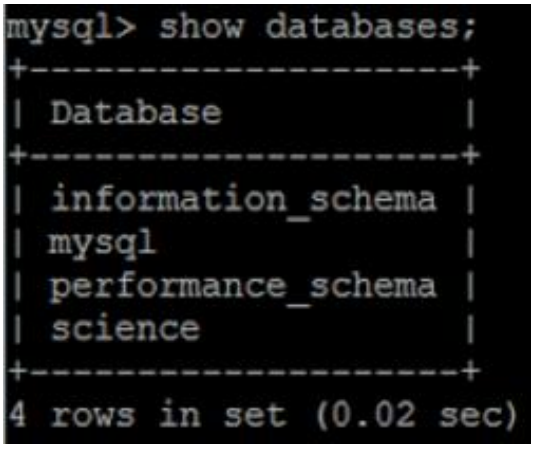

Figura 3.8: Bases de dados do *MySQL*.

Para criar o banco de dados os comandos abaixo foram executados:

» create database science;

» create user scienceadmin@localhost;

» set password for scienceadmin@localhost=password(“<senha>”);

» grant all privileges on science.* to scienceadmin@localhost identified by ’<senha>’;

» flush privileges;

» exit

Assim criou-se o banco de dados, que foi (não na versão final) usado na instalação do *Wordpress*. Porém a utilização do *Wordpress* foi descontinuada neste trabalho por se tratar de um *framework* muito pesado para o *hardware* disponível. Em substituição, foi desenvolvida uma página em *html, css, php e jQuery*. Este assunto está mais esclarecido na seção seguinte.

Em seguida foi preciso fazer a instalação do *PHP* usando o comando a seguir. O *PHP* é uma linguagem interpretada livre usada apenas para aplicações atuantes no lado do servidor.

$ sudo apt-get install php5 php-pear php5-mysql

A maneira mais simples de se testar a instalação do *PHP* foi criar um arquivo na pasta **/var/www**, que é o diretório base do servidor *apache*, com extensão **.php**. O conteúdo do arquivo foi **<?php phpinfo(); ?>**. O serviço do *Apache* foi reiniciado, para isso:

$ sudo service apache2 restart

Basta portanto usar um navegador qualquer e navegar para **143.107.235.46/<nome>.php**. Uma página cujo conteúdo são todas as configurações do servidor, do *site* e do *PHP* apareceu.

Um problema que ocorreu foi nada aparecer ao tentar acessar o endereço **(143.107.235.46)**. Em algumas versões do *Apache2* o diretório de trabalho não é o **/var/www**. Para corrigir este problema foi necessário mudar para a pasta **/var/www** em qualquer um dos arquivos de configuração do *Apache*

localizados nos diretórios a seguir:

- **nano /etc/apache2/sites-enabled/000-default -> DocumentRoot /var/www**
- **nano /etc/apache2/sites-available/default -> DocumentRoot /var/www**
- **nano /etc/apache2/sites-available/default-ssl -> DocumentRoot /var/www**

Outro problema foi a porta do servidor Apache estar em 8080, dessa maneira o endereço da página deve levar a porta no final (**143.107.235.46:8080**), o que é indesejado. A porta padrão é a 80, ao usá-la não é necessário especificar a porta no endereço do navegador. Para isso a porta foi mudada para 80 nos arquivos abaixo, onde estavam diferentes:

- **nano /etc/apache2/ports.conf -> NameVirtualHost *:80 | Listen 80**
- **nano /etc/apache2/sites-enabled/000-default -> <VirtualHost *: 80>**
- **nano /etc/apache2/sites-available/default -> <VirtualHost *: 80>**

No caso da distribuição *Debian* 7.9 fornecida na página da *BeagleBone* também ocorreu um erro de ao se tentar acessar a página de teste, a página padrão da *BeagleBone* abrir. Isso aconteceu pois nesta distribuição em específico existem serviços responsáveis por manter esta página no ar. Os comandos responsáveis por desativá-los são portanto:

$ systemctl disable bonescript.service

$ systemctl disable bonescript.socket

$ systemctl disable bonescript-autorun.service

No *Fedora 25*, a distribuição escolhida para testes mais rápidos em um computador, o módulo que ofereceu problemas de permissão é o SELinux [41], que é uma implementação da arquitetura MAC (Mandatory Access Control) para o *Linux*. Nele todos os processos do sistema estão previstos em uma política de segurança que baseia suas decisões em regras pré estabelecidas com várias informações relevantes à segurança. As permissões MAC podem ser aplicadas até mesmo no usuário *root*, limitando suas ações. Desabilitar esta segurança adicional não é uma boa ideia, mas a nível de testes de desenvolvimento o desligamento do SELinux não é um problema.

3.2.5 Wordpress

O *Wordpress* é um aplicativo de sistema de gerenciamento de conteúdo para a rede. Foi escrito em *PHP* com banco de dados *MySQL*. É uma ferramenta que possibilita a criação de *blogs* e páginas com grande facilidade, além de possuir uma aparência agradável [42].

Este *framework* foi a princípio escolhido pelo fato de ser muito simples de ser implementado. Porém sua implementação utilizando o *hardware* da *BeagleBone Black* se mostrou lenta e pesada, especialmente enquanto o monitoramento estava habilitado. Como o algoritmo responsável pelo monitoramento ocupa boa parte da CPU da *BeagleBone* o *Wordpress* foi abandonado e foi criada uma página utilizando *html, css, jQuery e PHP*, que mostrou um desempenho muito superior. A página utilizada será discutida na seção seguinte e apesar de o *Wordpress* ter sido descontinuado é interessante ressaltar brevemente como foi feita sua instalação e seu uso.

Para instalar o *Wordpress* foi necessário baixar sua última versão usando o comando:

$ sudo wget -O wordpress.tar.gz http://wordpress.org/latest.tar.gz

$ sudo tar -xvpzf wordpress.tar.gz

Em seguida os arquivos do *Wordpress* foram copiados para o diretório de trabalho do *Apache* com o comando **$ sudo cp -r wordpress/* /var/www/**. O próximo passo foi entrar no endereço **143.107.235.46/wp-admin/** e clicar em **Criar Arquivo de Configuração**.Para prosseguir foi necessário navegar para a pasta **/var/** e utilizar o seguinte comando:

$ sudo chown -R www-data:www-data www

O *Apache* foi instalado como raiz, como consequência disso, as pastas criadas pelo instalador são propriedade do usuário raiz. Porém o *Apache* funciona sobre um usuário chamado **www-data**, que não possui as mesmas permissões que o *root*. O comando acima muda a posse da pasta **www** e todas as outras pastas abaixo dela para o usuário *www-data* do *Apache*. Caso isso não seja feito um erro pode ocorrer na instalação.

Em seguida foi necessário seguir os passos da instalação. Assim que o *Wordpress* foi instalado ele pôde ser configurado através do endereço **143.107.235.46/wp-login**. Neste endereço foi possível personalizar o site para deixá-lo conforme desejado de maneira bastante intuitiva.

3.2.6 Página web

Depois de testar o *Wordpress* e verificar sua ineficácia para a aplicação em questão, uma página em HTML (abreviação para a expressão inglesa HyperText Markup Lenguage) foi criada. O HTML é uma linguagem de marcação para criação de páginas da Web, se trata de uma linguagem interpretada por navegadores.

O objetivo foi criar uma única página dinâmica contendo *login*, informações sobre o projeto, configurações do projeto e uma transmissão ao vivo do monitoramento da câmera. Para criar uma página dinâmica, uma linguagem e uma biblioteca foram utilizadas: o PHP e o jQuery. O PHP, como já foi mencionado, é uma linguagem interpretada pelo servidor e que também gera a página web que

será visualizada pelo cliente. O jQuery é uma biblioteca de código aberto que contém várias funções *JavaScript* que interagem com o HTML. Essa biblioteca foi criada para simplificar a navegação do documento HTML, criar animações e eventos, aplicações AJAX (JavaScript e XML Assíncrono) e outras aplicações.

Para estilizar a página web foi utilizado o CSS (Cascading Style Sheets) que é um tipo de arquivo que contém todas as configurações de estilo da página sem que seja necessário fazer isso no programa HTML.

Para a página principal foi criado um menu de navegação. A aba início contém um *slogan* do projeto, a aba "Desenvolvedor"contém informações acerca do projeto e na aba *login* o usuário tem acesso ao sistema.

Para o processo de *login*, uma base de dados foi criada no *MySQL* através do aplicativo web *phpMyAdmin*, que permite gerenciar bases de dados, criar, remover ou alterar tabelas e qualquer tipo de manipulação no banco de dados. Para acessar este recurso basta instalar [43] e acessar através de um navegador utilizando o endereço de IP terminado por "/phpMyAdmin".

A base de dados criada levou o nome de *devgrgoncal* e possui três tabelas. A primeira tabela, chamada de *users*, contém as informações dos usuários que podem acessar o *site*. As outras duas tabelas são usadas em outros programas e serão explicadas em suas respectivas seções.

O objetivo do projeto é que ele seja desenvolvido futuramente como um produto. Isso significa que cada sistema de monitoramento possui sua própria base de dados e que a página de determinado sistema não pode acessar outro sistema diferente.

O processo de *login* nada mais é do que uma comparação entre os dados que o usuário inseriu com os dados salvos na base de dados. Isso é feito através de um *POST*, que envia dados para serem processados no servidor. O arquivo que recebe o POST foi nomeado *login.php* e é responsável por acessar o banco de dados e comparar com os dados recebidos através do POST.

O PHP possui uma função chamada *session_start()*, que cria uma sessão ou resume a sessão anterior baseado em um ID de sessão acessado via POST ou GET (ou via cookie). Essa função é chamada ou então é executada quando uma sessão é iniciada automaticamente e chama dois manipuladores (internos do PHP), o *open* e o *read*. O *read* recupera qualquer informação de sessão existente e preenche a super global *$_SESSION*, se os dados existirem [44]. Então no início do código de todas as páginas esse comando deve ser utilizado.

Além disso, utilizando o comando descrito no parágrafo anterior é possível usar a variável super global *$_SESSION* para esconder ou mostrar conteúdo da página web.

Para efetuar *logout*, ou seja, sair do usuário atual o servidor executa o comando *session_destroy()*, que destrói os dados registrados em uma seção.

O método de *login* implementado possui segurança média e será alvo de futuros estudos do projeto. Várias medidas de segurança adicionais podem ser tomadas, como por exemplo limitar a quantidades de tentativas de *login*. O assunto de segurança é indispensável para o projeto visto como um produto, afinal um dispositivo de segurança que não possui segurança digital não é muito útil. Porém como este é um tópico bastante complexo e se estenderia demais, apenas a segurança básica foi adotada por ora.

Ao entrar no sistema, três abas aparecem no lugar da aba "login": Operações, Cadastro e Sair. A aba Sair executa o comando *session_destroy()* já citado acima; a aba Operações é responsável por ligar e desligar o sistema como um todo e também gerenciar configurações, como por exemplo gravar vídeo, usar monitoramento inteligente (usando aprendizagem de máquina) e o *Dropbox*; a aba Cadastro é responsável por adicionar usuários ao sistema e talvez seja descontinuada.

Para a aba de Operações o *site* acessa o banco de dados e verifica o estado de cada uma das configurações para que o usuário saiba quais configurações estão atualmente ligadas. As configurações foram estilizadas em botões deslizantes da maneira mais intuitiva possível. Quando um desses botões é ativado, a biblioteca jQuery se encarrega de gerar um POST para o código chamado *changeConf.php*, que recebe o pedido HTTP e acessa o *database* MySQL e aplica as mudanças. A Figura 4.6 mostra o resultado da implementação.

O jQuery foi utilizado em diversas seções da página para customização e dinamismo.

3.2.7 Instalando a OpenCV e dependências

A *OpenCV* (*Open Source Computer Vision Library*) foi originalmente desenvolvida pela *Intel* em 2000 e é uma biblioteca multiplataforma livre para o uso acadêmico e comercial. É útil para o desenvolvimento de aplicativos na área de visão computacional pois possui módulos de processamento de imagens, vídeo I/O, estrutura de dados, álgebra linear, GUI (Interface Gráfica ao Usuário), além de muitos outros algoritmos de visão computacional [45].

A *OpenCV* foi desenvolvida em C/C++ e possui suporte para *Java*, *Python* e Visual Basic. Em seguida, será especificada como a instalação do *OpenCV* foi feita. Para os procedimentos em seguida o usuário usado foi o *root*, pelo fato de possuir permissões.

O primeiro passo foi instalar as ferramentas de desenvolvedor. O pacote *pkg-config* costuma vir instalado com a distribuição mas é útil sua inclusão mesmo assim. Os outros pacotes são o *git*, para poder baixar a OpenCV pelos repositórios do *GitHub* (*GitHub* é um serviço de servidores de rede compartilhada para projetos que usam o controle de versionamento *Git*) e o *cmake*, usado para configurar uma *build*. Para isso foi necessário o comando:

$ sudo apt-get install build-essential cmake git pkg-config

Foi preciso em seguida instalar os diferentes tipos de formatos de imagens que a *OpenCV* usa

como, por exemplo, as extensões JPEG, PNG etc. Para isso precisa-se dos pacotes de imagem I/O:

$ sudo apt-get install libjpeg8-dev libtiff4-dev libjasper-dev libpng12-dev

Com essa biblioteca instalada a *OpenCV* já é capaz de carregar uma imagem da memória, mas não é capaz de mostrar esta imagem na tela. Para isso foi usada a biblioteca de desenvolvimento *GTK*, do qual o módulo *highgui* da *OpenCV* depende para associar a sua Interface Gráfica ao Usuário (GUI). Bastou portanto:

$ sudo apt-get install libgtk2.0-dev

Para processar transmissão de vídeo e acessar quadros (*frames*) individuais o comando abaixo foi executado. Além dele, o comando consecutivo possui bibliotecas que otimizam várias rotinas internas do *OpenCV*.

$ sudo apt-get install libavcodec-dev libavformat-dev libswscale-dev libv4l-dev

$ sudo apt-get install libatlas-base-dev gfortran

O *pip* é um sistema de gerenciamento de pacotes usado para gerenciar e instalar pacotes escritos na linguagem *Python*. O *pip* possui comandos simples do tipo **$ pip install/uninstall <pacote>**. Para sua instalação foi utilizado o comando **wget**, que é um programa livre que permite baixar dados da rede. Podem ocorrer avisos de certificado que impedem que o arquivo seja baixado, para ignorar estes avisos o argumento **–no-check-certificate** foi utilizado.

$ wget –no-check-certificate https://bootstrap.pypa.io/get-pip.py

$ sudo python get-pip.py

Em seguida foi feita a instalação do *virtualenv* e o *virtualenvwrapper* que são dois pacotes que permitem a criação de ambientes *Python* individuais para cada projeto que está sendo trabalhado.

$ sudo pip install virtualenv virtualenvwrapper

$ sudo rm -rf /.cache/pip

Depois de instalar estes pacotes foi atualizado o arquivo **/.bashrc**, que é um interpretador de comandos. O **bashrc** é como se fosse um tradutor entre o sistema operacional e o usuário. Os comandos deste arquivo são executados no momento que o usuário inicia um *shell* que não requer autenticação.

Para que as mudanças feitas no arquivo **/.bashrc** sejam aplicadas foi preciso recarregar o seu conteúdo. Isso foi feito com o terceiro comando apresentado a seguir.

$ export WORKON_HOME=$HOME/.virtualenvs

$ source /usr/local/bin/virtualenvwrapper.sh

$ source /.bashrc

Foi então criado um ambiente virtual chamado *cv* onde a instalação continuará.

$ mkvirtualenv cv

Como já foi citado anteriormente a *OpenCV* oferece suporte para a linguagem *Python* e além

disso, os programas de monitoramento foram escritos nesta linguagem devido às suas facilidades e o seu alto nível. Portanto foi preciso instalar o *Python*:

$ sudo apt-get install python2.7-dev

O *OpenCV* representa imagens como vetores multidimensionais *NumPy*, que é um pacote para linguagem *Python* que adiciona suporte para vetores grandes e multi dimensionais além de outras funções. O comando seguinte foi realizado fora do ambiente *cv* para que em um arquivo *Python* qualquer fora do ambiente virtual esse pacote possa ser também importado.

$ pip install numpy

Finalmente foi feito o *download* do *OpenCV* a partir do *home* e sua versão foi verificada. Foi preciso executar os próximos comandos dentro do ambiente virtual *cv*.

$ workon cv

$ env GIT_SSL_NO_VERIFY=true git clone https://github.com/Itseez/opencv.git

$ cd /opencv

$ git checkout 3.0.0

$ cd ..

Também foi preciso fazer o *download* do arquivo OpenCV_contrib, pois sem esse repositório não é possível ter acesso a algumas funções que eram disponíveis na versão 2.4 da *OpenCV*. Note que a versão é a mesma da anterior.

$ env GIT_SSL_NO_VERIFY=true git clone https://github.com/Itseez/opencv_contrib.git

$ cd /opencv_contrib

$ git checkout 3.0.0

$ cd ..

Para continuar a instalação foi criada uma pasta para compilar a *OpenCV*.

$ cd /opencv

$ mkdir build

$ cd build

$ cmake -D CMAKE_BUILD_TYPE=RELEASE

-D CMAKE_INSTALL_PREFIX=/usr/local

-D INSTALL_C_EXAMPLES=OFF

-D INSTALL_PYTHON_EXAMPLES=ON

-D OPENCV_EXTRA_MODULES_PATH= /opencv_contrib/modules

-D BUILD_EXAMPLES=ON ..

Antes de compilar a *OpenCV* foi preciso corrigir um *bug* que causa erros na compilação [46]. O erro está relacionado com o *FFMPEG* e pode ser resolvido substituindo **AVCodecID** por **CV_CODEC_ID**

em todas as ocorrências com exceção do **#define** nas linhas 1174, 1546 e 1556 do arquivo **opencv-3.0.0/modules/videoio/src/cap_ffmpeg_impl.hpp** .

Finalmente foi possível compilar a *OpenCV*. A compilação consome bastante tempo para terminar e após terminar foram utilizados outros dois comandos para instalar.

$ sudo make

$ sudo make install

$ sudo ldconfig

O *OpenCV* foi instalado em **/usr/local/lib/python2.7/dist-packages**, porém o ambiente virtual ainda está localizado no **home**, portanto foi necessário criar um *link* simbólico para este diretório:

$ cd /.virtualenvs/cv/lib/python2.7/site-packages/

$ ln -s /usr/local/lib/python2.7/site-packages/cv2.so cv2.so

Ocorreu um erro ao tentar entrar no ambiente virtual *cv* e foi necessário editar o arquivo **.bashrc** adicionando as seguintes linhas no final do arquivo, assim como a Figura 3.9.

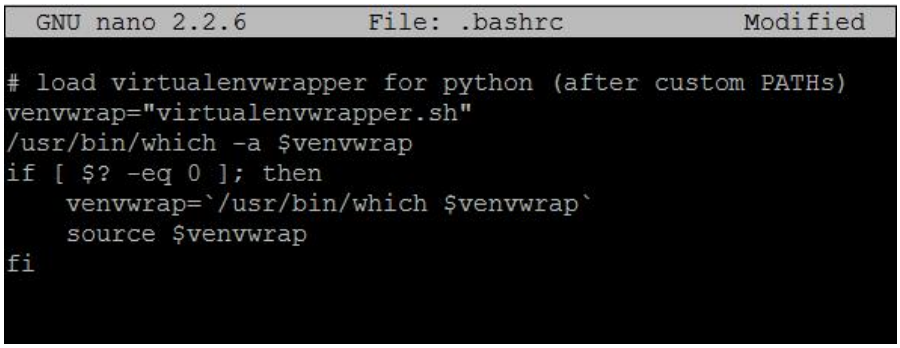

Figura 3.9: Correção de erros do ambiente virtual no **.bashrc**

$ source /.bashrc

Para confirmar o local de instalação do *OpenCV* foi utilizado o comando a seguir:

$ find / -name "cv2.so"

Foi necessário utilizar a localização encontrada com o último comando executado no comando a seguir para adicionar a biblioteca **cv2.so** no caminho do *Python*, caso o *link* simbólico feito não funcione (como foi o caso).

$ export PYTHONPATH=$PYTHONPATH:/usr/local/lib/python2.7/dist-packages

Para confirmar a instalação foi necessário entrar novamente no ambiente virtual, importar a biblioteca do *OpenCV* e verificar a versão usada.

$ workon cv

$ python

» import cv2

» cv2.__version__

'3.0.0'

Não foi necessário implementar o programa em *Python* pelo ambiente *cv* pois já foram instaladas todas as bibliotecas necessárias. O ambiente virtual iria evitar a desorganização do sistema caso o *Debian* não possuísse as bibliotecas necessárias.

3.2.8 Detecção de movimento

Depois de instalado a *OpenCV* foi dado continuação ao projeto. Foi intalado o pacote *imutils*[47] que inclui várias funções convenientes da *OpenCV* que realiza tarefas básicas como translação, rotação, redimensionamento e esqueletização de imagens e outro pacote relacionado com o API do *Dropbox*, que permitiu o *upload* das fotos na nuvem.

$ pip install imutils

$ pip install dropbox

Durante o primeiro desenvolvimento, o programa de detecção carregava um arquivo de configurações do tipo JSON, que é um acrônimo para *JavaScript Object Notation*. Esse arquivo tinha importantes configurações para o programa de detecção:

- **use_dropbox** : Assume valores booleanos e ativa ou desativa o *upload* de fotos para o *Dropbox*.
- **min_upload_seconds** : Tempo mínimo entre os *uploads* de imagens no *Dropbox*. Para evitar vários *uploads* consecutivos.
- **min_area** : A área mínima em *pixels* para que uma região da imagem seja considerada movimento.
- **dropbox_key** : A chave pública do *Dropbox*.
- **delta_thresh** : Valor mínimo da diferença absoluta entre o *frame* atual e o *frame* de fundo para que seja detectado o movimento. Valores menores fazem com que mais movimentos sejam detectados e valores maiores fazem com que menos movimentos sejam detectados.
- **dropbox_base_path** : O nome da pasta onde o *Dropbox* irá armazenar as fotos.
- **dropbox_secret** : A chave privada do *Dropbox*.
- **status_monitoramento** : Responsável por ligar e desligar o monitoramento.

- **min_motion_frames** : Número mínimo de *frames* consecutivos contendo movimento antes de um *upload* no *Dropbox*.
- **use_tensorflow** : Habilita ou desabilita o uso de aprendizagem de máquina para tomar conta da decisão de atualizar o plano de fundo.
- **record_video** : Habilita ou desabilita a gravação de vídeo quando um movimento é detectado.
- **accessToken** : É o código do Dropbox usado para se conectar ao API do Dropbox sem ter que passar pelo processo de autorização.
- **apiKey** : É a identificação do API do Firebase utilizado para enviar notificações nos dispositivos móveis dos usuários.

O arquivo era chamado de **conf.json** e tinha a seguinte estrutura (alguma das configurações listadas acima não constam no JSON abaixo pois foram adicionadas depois):

```
{       "use_dropbox": true,
        "min_upload_seconds": 3.0,
        "min_area": 5000,
        "dropbox_key": "g319zf46qjh46qn",
        "delta_thresh": 7,
        "dropbox_base_path": "/Aplicativos",
        "dropbox_secret": "5kz546o7omyvd2u",
        "show_video": true,
        "status_monitoramento": 0,
        "min_motion_frames": 4
}
```

Porém durante o desenvolvimento do aplicativo para Android o arquivo JSON foi descontinuado e implementado na forma de uma tabela do MySQL. A tabela recebeu o nome *conf* e pôde ser alterada através de pedidos HTTP (POST) no arquivo *changeConf.php*, ela também pode ser lida através do arquivo *showConf.php* através do mesmo pedido.

Para obter as informações das configurações **dropbox_key** e **dropbox_secret** foi criado um aplicativo no *Dropbox API v2* [48]. Na página de desenvolvedores do Dropbox foi gerado um *Access Token* para evitar a necessidade de acesso implícito toda vez que é feita a conexão com o API do *Dropbox* no programa de monitoramento. Este processo de criação é simples e está demonstrado no Apêndice B.

O algoritmo de detecção de movimento desenvolvido teve como base uma aplicação com mesmo intuito desenvolvida para *Raspberry Pi* [49]. A inspiração se deu pela maneira que o programa detecta movimento. A ideia básica é capturar um plano de fundo e defini-lo como a representação de uma sala vazia e então comparar novas imagens do cômodo com a imagem de referência. Aqui se encontra o primeiro requerimento que o algoritimo exige: é preciso inicializar o sistema usando um ambiente vazio. Caso isso seja ignorado o cômodo ocupado por uma ou mais pessoas será definido como referência (plano de fundo) e o programa irá encontrar movimento quando as pessoas se moverem ou até mesmo se deixarem o cômodo.

Um segundo requerimento do sistema é que a câmera esteja sujeita ao mínimo de movimento possível. Então é necessário evitar posicionar a câmera em locais onde há muita vibração ou locais sujeitos a condições ambientais como por exemplo fortes ventos. Isso é necessário pelo fato de o algoritmo ser sensível. Para o caso onde a câmera foi movida intencionalmente (para mudar o local de atuação ou o ângulo de vista) o algoritmo responde atualizando o plano de fundo (a referência) para se adaptar ao novo ângulo.

Antes de prosseguir para a interpretação do algoritmo é importante ressaltar que vislumbra-se a possibilidade de transformar no futuro, o projeto em um produto. Isso significa que todo sistema de monitoramento terá sua própria base de dados independente e vinculada com a conta específica do *Dropbox* do cliente e ao aplicativo do *Firebase* (que é uma plataforma de desenvolvimento para aplicativos de celular; será discutido na seção seguinte) [50].

Nas primeiras linhas são importadas várias bibliotecas que são utilizadas no programa. Os *imports* representam respectivamente as bibliotecas da API do *Dropbox*, do *OpenCV*, do *NumPy*, de tempo, de aprendizagem de máquina, de gerenciamento de arquivos, do MySQL e da API responsável por gerar notificações para o aplicativo de dispositivos móveis.

IMPORTS

1. from dropbox.client import DropboxOAuth2FlowNoRedirect
2. from dropbox.client import DropboxClient
3. import cv2
4. import numpy as np
5. import imutils
6. import time
7. import datetime
8. import tensorflow as tf,sys
9. import os, shutil

```
10. import MySQLdb
11. from pyfcm import FCMNotification
```

Após importar todas as bibliotecas necessárias, duas funções são definidas: a função *predictions()* e a função *cleanTrainingPhotos(folder)*. A primeira delas utiliza os grafos de aprendizagem de máquina para identificar se o cômodo está ocupado ou não. Essa função será melhor explicada (devido a sua complexidade) em uma seção separada, por ora apenas sua funcionalidade geral será comentada. A segunda função limpa o diretório de treinamento do *TensorFlow* (também será explicada em outra seção).

Os primeiros comandos carregam os dados da tabela *conf* do banco de dados do MySQL. No código a seguir, cujo cabeçalho é "LOADING CONFIGURATIONS", a linha 5 recebe um vetor bidimensional de configurações e foi lido diversas vezes utilizando *results[0][coluna da configuração]*. Para facilitar a navegação neste vetor, foram atribuídos os nomes de cada configuração para cada índice de coluna da tabela *conf*, isso é feito da linha 6 até a 18. O vetor *results* é bidimensional pois geralmente as tabelas do MySQL possuem mais de uma linha, no caso deste projeto nenhuma configuração é adicionada, mas sim modificada. Então basta usar o índice 0 para as linhas do vetor.

```
#### LOADING CONFIGURATIONS ##################################
1. db = MySQLdb.connect("localhost","root","senha","devgrgoncal")
2. cursor = db.cursor()
3. sql = "SELECT * FROM conf"
4. cursor.execute(sql)
5. results = cursor.fetchall()
6. use_dropbox = 0
7. min_upload_seconds = 1
8. min_area = 2
9. dropbox_key = 3
10. delta_thresh = 4
11. dropbox_base_path = 5
12. dropbox_secret = 6
13. status_monitoramento = 7
14. min_motion_frames = 8
15. use_tensorflow = 9
16. record_video = 10
```

17. accessToken = 11
18. apiKey = 12

O banco de dados *devgrgoncal* possui 3 tabelas. A tabela *users* usada para listar os usuários do sistema, a tabela *conf* que contém todas as configurações do monitoramento e a última delas, a tabela *notifications* que contém uma lista de identificações de usuários que devem receber notificações quando um movimento é detectado. Essas identificações são únicas para cada dispositivo cadastrado no sistema através do aplicativo Android. Dessa vez, o vetor *results2* pode ter mais que uma linha, diferente do caso anterior.

Portanto para acessar o banco de dados desta tabela o código do cabeçalho "LOADING NOTIFICATIONS"foi utilizado.

LOADING NOTIFICATIONS

1. cursor2 = db.cursor()
2. sql2 = "SELECT * FROM notifications"
3. cursor2.execute(sql2)
4. results2 = cursor2.fetchall()
5. APIKey = results[0][apiKey]

Em seguida várias variáveis que são usadas no programa são declaradas com seus respectivos valores iniciais. A câmera é definida como sendo a padrão do dispositivo e sua resolução foi ajustada para 640x480. Além disso a função *cleanTrainingPhotos(folder)* é chamada duas vezes para esvaziar dois diretórios de treinamento. A parte de declarações não será inserida aqui por ser trivial.

Para a conexão com o *Dropbox* os comandos da seção "ENTRANDO NO DROPBOX"são executados. Para automatizar o processo de conexão o *accessToken* obtido durante a criação da API do Dropbox foi utilizado (processo descrito no Apêndice B).

ENTRANDO NO DROPBOX

1. flow = DropboxOAuth2FlowNoRedirect(results[0][dropbox_key],results[0][dropbox_secret])
2. client = DropboxClient(results[0][accessToken])
3. print "[DROPBOX] Account linked: "+ results[0][accessToken]

Em seguida o programa entra em um *loop* infinito que depende do **status_monitoramento**. O programa lê um quadro da câmera e define um texto que será escrito no *frame* (quadro) como "Deso-

cupado"(como pré-suposição).

O quadro obtido é então redimensionado para 500 *pixels* de largura utilizando o *imutils*, é convertido para escala de cinza e borrado nos comandos das linhas 7 e 8 do código de cabeçalho "MAIN"adiante. Se o quadro de fundo ainda não tiver sido definido, ele é atualizado para o quadro atual. Para a utilização dos comandos da OpenCV, foi utilizada a documentação oficial várias vezes [45].

Para exemplificar a conversão da imagem para escala de cinza a Figura 3.10 representa uma imagem onde um rosto foi mantido no campo de visão da câmera. Isso significa que o rosto é um dos elementos do quadro de fundo da imagem e caso esse rosto se mova ou se ausente do campo de visão da câmera, o algoritmo entenderá esta mudança como parte de um movimento.

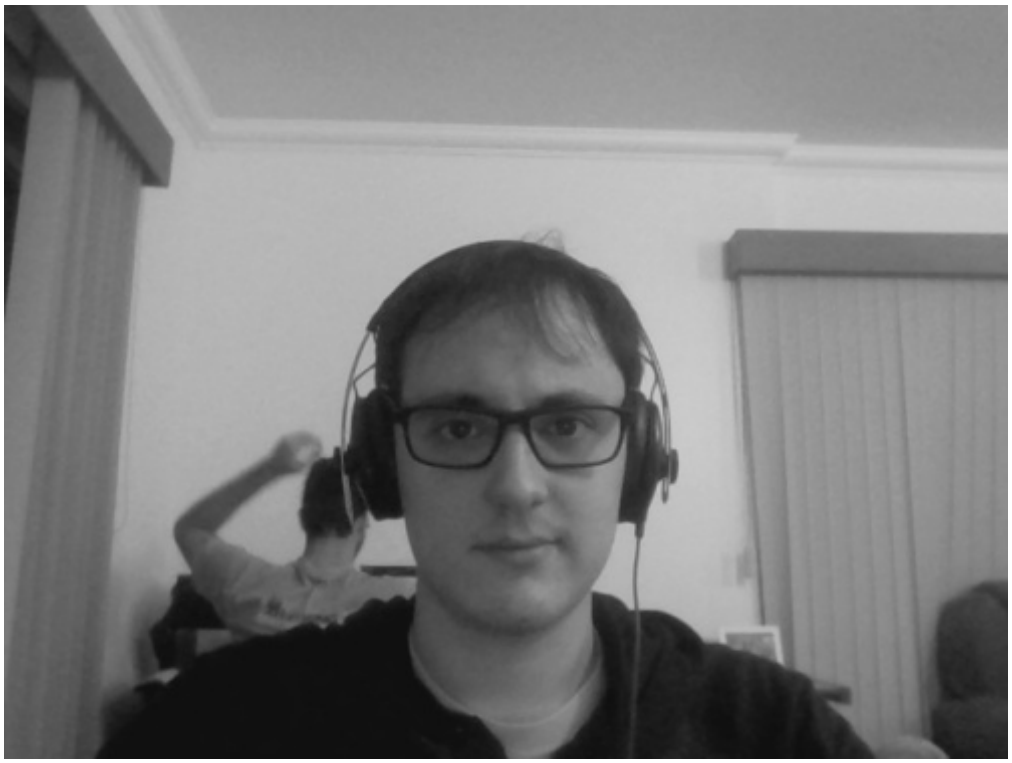

Figura 3.10: Transformação da imagem para escala de cinza.

A câmera não tira fotos idênticas. Mesmo que a câmera esteja submetida a condições ideais sempre haverá *pixels* diferentes entre duas fotos. Essa diferença não é trivial ao analisar a foto, porém ela pode gerar um erro muito grande para o algoritmo. Por exemplo, se em uma região da foto há muita variação nos *pixels*, a operação de limite (função *threshold*, que em breve será abordada) encontrará vários pontos que sugerem um movimento, resultando portanto em um falso positivo. A Figura 3.11 mostra a mesma imagem após o comando da linha 8 ser aplicado.

Figura 3.11: Imagem borrada.

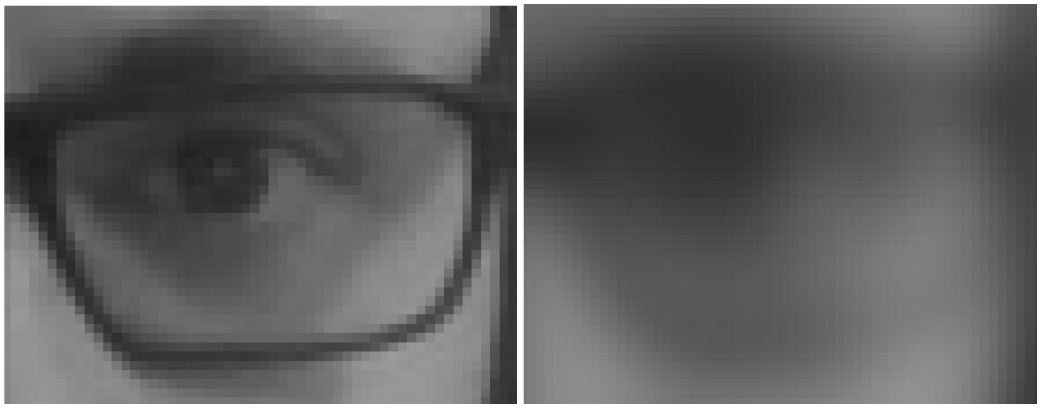

Figura 3.12: Comparativo entre os *pixels* de uma região antes e depois de borrar.

Pode-se notar que os pixels da imagem da Figura 3.11 apresentam uma característica homogênea e isso fica muito mais claro na Figura 3.12, onde os *pixels* de ambas as imagens podem ser comparados.

```
#### MAIN ##############################
1. while(results[0][status_monitoramento]==1):
2.     ret,frame=camera.read()
3.     text = "Desocupado"
4.     timestamp = datetime.datetime.now()
5.     frame = imutils.resize(frame,width=500)
6.     rawframe = frame
7.     gray = cv2.cvtColor(frame,cv2.COLOR_BGR2GRAY)
8.     gray = cv2.GaussianBlur(gray,(21,21),0)
9.
10.    if avg is None:
11.        avg = gray.copy().astype("float")
12.        continue
```

A próxima parte do código é essencial e é responsável por atualizar o quadro de fundo. A atualização do quadro de fundo é feita utilizando o comando da linha 4 ou 29 do código de cabeçalho "BACKGROUND REFRESH": *cv2.accumulateWeighted(src,dst,alpha[,mask])*. Este comando faz uma média entre duas imagens. Os parâmetros desse comando são a imagem de entrada, a imagem que receberá o resultado (com o mesmo número de canais da imagem de entrada), o peso que a imagem de entrada terá no cálculo da média e uma máscara de operação opcional (que não foi usado no projeto), respectivamente.

Calcular a média dos quadros sem movimento foi o método escolhido para detectar movimento. É possível também detectar movimento comparando quadro a quadro.

Em teoria seria possível utilizar apenas uma imagem como referência, sem necessidade de atualizar o fundo, porém durante o dia a iluminação pode variar ou então uma brisa pode fazer as folhas de uma planta do cômodo se movimentarem e causarem em ambos os casos, um falso evento. Por causa desse tipo de situação, o fundo é atualizado constantemente.

Durante o desenvolvimento, o algoritmo foi implementado de maneira a atualizar o fundo (fazer a média através do comando *cv2.accumulateWeighted*) sempre. Como consequência, se alguma pessoa entrasse no campo de visão da câmera e ficasse parada no mesmo lugar por muito tempo, ela passaria a ser parte do quadro de fundo e o sistema deixaria de identificar movimento. Para resolver esse problema foi criada uma solução simples: o plano de fundo só atualiza se não houver movimento. O plano de fundo atualiza a cada 6 segundos caso não tenha ocorrido nenhum movimento nos últimos 6 segundos. Essa condição pode ser vista na linha 28 do código de cabeçalho "MAIN".

Além das condições de atualização de fundo descritas no parágrafo anterior, o fundo também é atualizado nos primeiros 30 quadros (seguidos) da câmera. Esse é um procedimento que faz com que o quadro de fundo faça várias médias do cômodo vazio para que na próxima parte do código, quando for feita a diferença entre o quadro atual e a média, pequenas variações de foto para foto (mesmo que sejam alguns *pixels*) não sejam detectadas como parte de um movimento. Isso é um pouco contra intuitivo porque a imagem foi borrada na linha 7 do código de cabeçalho "MAIN", então era de se esperar que a variação dos *pixels* fosse automaticamente retirada, mas não é. Quando isso aconteceu nos testes, o quadro todo foi marcado como sendo um movimento (como se tudo tivesse trocado de lugar), esse também seria um problema a ser tratado no caso do monitoramento quadro a quadro (sem fazer a média). Então não basta que o fundo seja a média de um quadro borrado pelo comando *cv2.GaussianBlur*: é necessário fazer mais do que uma média.

Ainda na linha 28 é possível notar que ainda existem mais duas condições. Essas condições se referem ao uso do *TensorFlow* para determinar quando atualizar o fundo. O "monitoramento inteligente", assim como foi chamado no projeto o ato de usar aprendizagem de máquina no monitoramento, é um opção que pode ser ativada ou não. Caso não esteja ativada, o quadro de fundo atualiza a cada 6 segundos que não ocorre movimento. Porém, caso ela esteja ativada é preciso treinar a rede neural para que o *TensorFlow* possa identificar o que tem na foto. Para isso é necessário construir uma base de treinamento utilizando fotos com e sem pessoas do ambiente e esse processo leva algum tempo, portanto a atualização de fundo continua operando a cada 6 segundos que não ocorre movimento. Assim que a base de dados para treinamento é completada e o treinamento é feito, o programa deixa de atualizar constantemente o fundo e passa a atualizar o fundo com base na análise de imagem do *TensorFlow*. Esse processo será descrito com detalhes na seção do *TensorFLow* para não tirar o foco do monitoramento.

Suponhamos que alguém invada o cômodo onde o monitoramento está ocorrendo e roube uma televisão, ou um rádio, ou qualquer coisa que esteja no campo de visão da câmera. Se o tipo de atualização de fundo utilizada for o da linha 28 (com a opção de "monitoramento inteligente"desativada) o roubo do objeto será sem dúvidas registrado, porém após o roubo a ausência do objeto na imagem será identificada como um movimento e irá persistir nos quadros seguintes. Para resolver esse problema seria necessário atualizar o fundo sempre e então o problema de alguém entrar e ficar parado voltaria a acontecer. Ou então, para resolver esse problema o *TensorFlow* poderia ser utilizado para reconhecer o que está presente na imagem e determinar se é seguro atualizar o fundo ou não. Esse tipo de atualização de plano de fundo traz uma confiabilidade maior para o sistema em termos de quando atualizar o fundo e evitar falsos positivos. Esse método também permite que seja detectado por software se a câmera foi movida, essa solução será descrita nesta seção mais adiante.

Então nas linhas 1 até 26 o algoritmo analisa o que tem no quadro atual e toma as devidas decisões. O algoritmo também salva o quadro atual no diretório de treinamento e a hora exata da última atualização de fundo (usada em várias condicionais).

```
#### BACKGROUND REFRESH ############################
1. if (time.time() - lastAccumulated)>30 and lastTraining>0 and results[0][use_tensorflow]==1:
2.     statePrevision = predictions()
3.     if "empty"in statePrevision:
4.         temp = cv2.accumulateWeighted(gray,avg,0.4)
5.         print "[BACKGROUND] Background Model Updated."
6.         print "[PREVISION] Room empty."
7.         if len(next(os.walk("/tf_files/photos/Empty"))[2]) < 250:
8.             cv2.imwrite("/tf_files/photos/Empty/empty"+ str(lastEmpty) + ".jpg",rawframe)
9.             lastEmpty = lastEmpty + 1
10.        else:
11.            lastEmpty = 1
12.        if "faces"in statePrevision:
13.        print "[PREVISION] Face detected."
14.        if len(next(os.walk("/tf_files/photos/Faces"))[2]) < 250:
15.            cv2.imwrite("/tf_files/photos/Faces/faces"+ str(lastFace) + ".jpg",rawframe)
16.            lastFace = lastFace + 1
17.        else:
18.            lastFace = 1
19.        if "bodies"in statePrevision:
20.        print "[PREVISION] Body detected."
21.        if len(next(os.walk("/tf_files/photos/Bodies"))[2]) < 250:
22.            cv2.imwrite("/tf_files/photos/Bodies/body"+ str(lastBody) + ".jpg",rawframe)
23.            lastBody = lastBody + 1
24.        else:
25.            lastBody = 1
26.        lastAccumulated = time.time()
27.
28. elif ((results[0][use_tensorflow] == 0 or lastTraining == 0) and (time.time() - lastAccumulated > 6 and time.time() - lastMovement > 6)) or firstRun < 30:
```

```
29.    temp = cv2.accumulateWeighted(gray,avg,0.4)
30.    lastAccumulated = time.time()
31.    firstRun = firstRun + 1
```

Finalmente o cerne da detecção é a subtração entre o quadro de referência (que representa o fundo padrão) e o quadro atual. Isso é feito com o comando da linha 1 do código com cabeçalho "DETECTION": *cv2.absdiff*. O comando *cv2.convertScaleAbs* é usado para escalar, calcular o valor absoluto e converter o resultado para 8 *bit*.

A linha 2 contém o comando *cv2.threshold* e aplica um *threshold* (limite) para cada elemento de um vetor (no caso é a imagem). Isso é aplicado na imagem resultante da diferença calculada na linha 1. O tipo de limite aplicado é o **THRESH_BINARY**, que define a saída em 255 (por se tratar de uma imagem de 8 bits) se houver qualquer valor maior que 7 no vetor de entrada, caso contrário a saída é 0. A saída representa então uma imagem binária de movimento. Essa configuração é chamada de *delta_thresh* e foi definida em 7. Quanto maior for o valor desta configuração, mais o sistema ignora movimentos pequenos. Por exemplo a movimentação de uma planta no cômodo por causa de alguma corrente de ar seria ignorada para valores mais elevados da configuração e detectada como movimento para valores pequenos. A disponibilidade do acesso desta configuração ao usuário não foi oferecida e por ora ficará como opção de desenvolvimento futuro: configurações avançadas para usuários.

Considerando novamente uma situação onde um rosto constitui parte do quadro de fundo de um ambiente, a Figura 3.13 representa a diferença entre o quadro de fundo e um quadro onde o rosto não está mais presente no campo de visão da câmera. Toda a região em preto da foto representa o fundo do quadro que não apresentou mudanças, de maneira análoga a região branca representa a ausência do rosto (caracterizando um movimento). Neste exemplo o movimento foi caracterizado pela retirada de um dos elementos da foto (o rosto) porém a mesma coisa ocorre quando um corpo entra no campo de visão da câmera.

Ao aplicar o limite na mesma imagem de exemplo o resultado é o da Figura 3.14. A região onde o rosto deveria estar apresenta uma diferença maior que o limite estabelecido pela configuração *delta_thresh* e é destacada em branco.

Figura 3.13: Exemplo da diferença entre um quadro e o fundo.

Figura 3.14: Exemplo da binarização da diferença entre um quadro e o fundo.

Após realizar a segmentação da imagem atual é preciso encontrar os pontos onde houve movimento. O comando *cv2.findContours* encontra contornos em uma imagem binária. O parâmetro **cv2.RETR_EXTERNAL** é responsável por determinar que o contorno obtido é apenas o contorno mais externo de um grupo de pontos. É possível também obter outros tipos de contorno. O último parâmetro **cv2.CHAIN_APPROX_SIMPLE** comprime segmentos verticais, horizontais e diagonais deixando apenas os pontos finais; um retângulo, por exemplo, é descrito por apenas 4 pontos com esse parâmetro.

```
#### DETECTION ############################
1. frameDelta = cv2.absdiff(gray,cv2.convertScaleAbs(avg))
2. thresh = cv2.threshold(frameDelta,results[0][delta_thresh],255,cv2.THRESH_BINARY)[1]
3. cnts = cv2.findContours(thresh.copy(),cv2.RETR_EXTERNAL,cv2.CHAIN_APPROX_SIMPLE)[-
2]
```

Após encontrar o contorno na imagem atual, o comando *cv2.contourArea* é utilizado para determinar a área dos contornos calculados. A configuração "min_area"é usada para definir qual a área mínima que o contorno deve ter para ser considerado um movimento. O padrão usado nesta configuração foi 6000 e também poderia ter sido utilizada como sendo uma configuração avançada para os usuários pois assim seria possível determinar o tamanho mínimo para que algo seja considerado um movimento. Uma aplicação disso para usuários que possuem animais de estimação como peixes, gatos ou cachorros é que seria possível ignorar a presença dos animais, contanto que eles não possam se aproximar muito da câmera.

Se houve um contorno com área maior que 6000 $pixels^2$ então os comandos abaixo são executados para desenhá-lo. O comando *cv2.boundingRect* calcula e retorna o retângulo mínimo de limites para os pontos, por isso essa função retorna 4 pontos.

Usando os pontos calculados na linha 2, a linha 3 desenha-os no quadro atual usando a cor verde (arbitrária) com uma espessura de 2 *pixels*. Essas linhas são referentes ao código de cabeçalho "CONTOURS".

```
#### CONTOURS ############################
1. for c in cnts:
2.     (x,y,w,h) = cv2.boundingRect(c)
3.     cv2.rectangle(frame,(x,y),(x+w,y+h),(0,255,0),2)
```

Outras operações feitas são a adição do estado atual do cômodo no canto superior esquerdo do quadro e da data no canto inferior esquerdo, ambos na cor vermelha. Isso é usado com o comando *cv2.putText*. Sempre que um movimento ocorre, a hora exata que o movimento ocorreu é gravada, esto é utilizado em vários pontos do algoritmo para fazer condicionais.

Como para o projeto, nenhum tipo de *hardware* adicional foi utilizado (e constará como implementação futura), não é simples detectar quando a câmera foi movida. Isso seria uma tarefa bastante mais simples se a câmera tivesse um acelerômetro, por exemplo, mas na ausência, a solução foi implementada via *software*. Para que este mecanismo funcione, a configuração de "monitoramento inteligente"deve estar habilitada e é obrigatório que a rede neural já tenha sido treinada.

Para detectar se a câmera foi movida, foram feitas duas etapas de verificação. A primeira delas é a condição básica para que a câmera tenha sido movida e é baseada numa consequência de a câmera ter sido movida. Quando a câmera muda seu campo de visão, a subtração entre o quadro definido como referência e o quadro atual irá resultar em vários pontos com valores maiores que a configuração de *threshold* igual a 7. Dessa maneira o cálculo dos contornos do quadro obtido depois de mover a câmera resulta na própria resolução da foto, fazendo com que o quadro inteiro seja detectado como sendo um movimento. Então se o quadro inteiro foi detectado como movimento, a área de contorno tem valor de aproximadamente 240000 $pixels^2$.

Portanto a primeira condição para verificar se a câmera foi movida é checar se a área de contorno excede um valor muito grande (definido como 200000), se já houve um treinamento da rede neural (que é um requisito para a funcionalidade) e se o algoritmo continua por algum tempo detectando o quadro todo como um movimento (se uma mosca passar bem na frente da câmera, o quadro todo é considerado um movimento mas por pouco tempo). A Figura 3.15 é um exemplo do que acontece quando a câmera é ligeiramente movida para cima.

Caso todos os requisitos descritos no parágrafo anterior forem atendidos por alguns frames seguidos, então existe um indicativo de que a câmera foi movida. É preciso agora aplicar um segundo teste para se verificar se a câmera foi mexida: verificar utilizando o *TensorFlow* se existe algum rosto na imagem. Se um rosto foi identificado na imagem significa que o rosto está bastante perto da câmera e é a origem da área de contorno no limite e neste caso a câmera ou não foi movida ou então o sistema acabou de detectar uma invasão. Caso existam corpos na imagem ou se o cômodo estiver vazio, o algoritmo irá considerar que a câmera foi movida.

Figura 3.15: Exemplo da saída do algoritmo quando a câmera é movida.

Porém, duas situações podem ocorrer. Se o algoritmo detectar que existem corpos na imagem, isso pode significar que o dono do sistema mudou a câmera de lugar porque queria mudar o campo de visão da câmera. Porém isso também pode significar que alguém invadiu o cômodo e mudou a câmera de lugar. Se o algoritmo detectar que não existem corpos na imagem, isso pode significar ou que a câmera mudou para uma posição que cancela totalmente seu campo de visão (comprometendo o reconhecimento de imagem) ou que o movimento da câmera foi feito com muita agilidade ou indiretamente. O código responsável por determinar se a câmera foi movida está a seguir com cabeçalho "CAMERA MOVED".

```
#### CAMERA MOVED ##################################
1. if cv2.contourArea(c) > 200000 and lastTraining > 0 and (time.time() - lastMovement < 4):
2.     cameraMovedIndicator = cameraMovedIndicator + 1
3.     if cameraMovedIndicator > 20:
4.         statePrevison = predictions()
5.         if "faces"in statePrevision:
6.                 cameraMovedIndicator = -10
7.         else:
```

```
8.                  cleanTrainingPhotos("tf_files/photos/Empty")
9.                  cleanTrainingPhotos("tf_files/photos/Bodies")
10.                 firstRun = 5
11.                 lastTrainingTime = -1000000
12.                 lastTraining = 0
13.                 print "[UPDATE] Camera moved, starting new background model."
14. if (time.time() - lastMovement > 4):
15.     cameraMovedIndicator = -10
```

A resposta do programa após concluir que a câmera mudou de posição é recalcular o quadro de fundo e apagar os diretórios de treinamento para que uma nova rede neural seja treinada. Dessa maneira o sistema é reestabelecido.

A solução via *software* é útil para evitar que, caso alguma coisa cause um movimento na câmera o sistema volte a operar normalmente sem a necessidade de alguma pessoa ter que reiniciá-lo. Porém, o sistema não tem como saber se a causa do movimento foi proposital ou não. A adição de algum *hardware* permitiria uma decisão mais precisa. Se um sensor capacitivo fosse implementado na câmera, o sistema saberia imediatamente que a câmera foi tocada, por exemplo. Vários outros melhoramentos foram pensados para este problema e constam nas implementações futuras do projeto.

Quando um movimento é detectado, uma configuração chamada "min_motion_frames", que determina a quantidade de quadros que a detecção precisa persistir para realmente gerar um alerta, é verificada para assegurar de que um movimento realmente ocorreu. Isso é usado para evitar que algum objeto caindo, ou atravessando o campo de visão da câmera em alta velocidade seja detectado como movimento e deverá ser futuramente mais uma configuração avançada para o usuário que quer detectar movimentos mais rápidos.

A primeira ação que o programa toma ao detectar movimento é enviar um alerta na forma de notificação para o dispositivo móvel dos usuários. Se o movimento no campo de visão da câmera persistir, notificações são enviadas de 5 em 5 segundos. Essa é uma forma de assegurar que o usuário considere a situação como uma potencial ameaça. Isso é feito no código abaixo através da API do *Firebase*, que será objeto de discussão em breve.

NOTIFICATION

```
1. if time.time() - lastNotification > 5:
2.     message_body = "A movement on: "+ time.asctime(time.localtime(time.time()))
3.     push_service = FCMNotification(api_key=APIKey)
```

4. for index in results2:

5. registration_id = index[1]

6. result = push_service.notify_single_device(registration_id = registration_id, message_title = message_title, message_body = message_body)

7. lastNotification = time.time()

Depois de enviar notificações para o grupo de usuários a imagem atual é enviada para o *Dropbox* (se a configuração estiver ativada).

As tabelas *conf* e *notifications* são carregadas no início do programa e precisam ser recarregadas para verificar se houve mudança em alguma configuração do sistema ou no grupo de usuários. Isso é feito a cada 40 frames.

Por último o programa também é capaz de gerar vídeos quando um movimento é detectado. Um vídeo gerado tem duração igual a do movimento mais 3 segundos e tem extensão ".avi". Se um movimento ocorrer depois dos 3 segundos finais um vídeo diferente é gerado. Isso é feito porque não necessariamente os movimentos ocorrem em sequência. Os vídeos têm como objetivo condensar todos os acontecimentos, evitando que o usuário tenha que assistir uma gravação inteira procurando as ocorrências.

3.2.9 TensorFlow

Como já mencionado diversas vezes o *TensorFlow* é uma biblioteca de código aberto para computação numérica especializada em aplicações de aprendizagem de máquina [51].

Para o reconhecimento de imagens, uma categoria de redes neurais chamada de Redes Neurais Convolucionais (Convolutional Neural Networks, CNN) se mostrou muito eficaz ao longo do tempo. Resumidamente, essas redes podem ser consideradas como sendo um bloco fechado que montam características que se não fosse pela rede neural, teriam que ser montadas uma a uma. As características que as CNN criam em seu treinamento são abstratas e generalizadas, tornando possível seu uso em reconhecimento de imagens [52].

Para o trabalho, uma CNN pré treinada chamada *inception v3* foi utilizada [30]. Essa rede neural foi treinada pelo Google usando 100 mil imagens e 1000 categorias. Porém no projeto deseja-se que o algoritmo seja capaz de identificar categorias diferentes como cômodos vazios, cômodos ocupados e rostos. Para reutilizar a mesma rede neural para classificar novas categorias o conceito de transferência de aprendizado foi utilizada.

O modelo de treinamento da rede neural *inception* faz várias operações com as imagens de treina-

mento. As primeiras camadas do modelo são responsáveis em aprender a detectar bordas nas imagens, em seguida nas camadas centrais, por detectar a forma das imagens e assim por diante cada vez com categorias mais abstratas. As últimas camadas são responsáveis por detecções de objetos inteiros e representam as camadas de mais alto nível. Para o processo de transferência de aprendizado essas últimas camadas são retreinadas usando as imagens que representam as categorias que serão adicionadas.

Portanto a primeira preocupação para treinar a rede neural é obter imagens de treinamento que representem muito bem as categorias que a rede deve aprender. Três categorias foram criadas: cômodos vazios, cômodos ocupados e rostos.

Durante os primeiros testes a ideia foi utilizar um mesmo conjunto de imagens que represente bem cada um desses casos. Então uma base de dados de imagens de vários cômodos ocupados ou não foi montada, uma base de dados para rostos também foi montada. As imagens foram retiradas da internet e de fotografias de cômodos e vistas que a câmera poderia ter. Porém o resultado desses testes foi pouco satisfatório devido ao fato de poucas imagens serem utilizadas no treinamento e da grande variância que as fotos apresentam, fator que pode ter confundido a rede neural. Como o objetivo do projeto é poder utilizar a câmera para monitorar qualquer cômodo, as imagens geradas pela câmera podem conter praticamente qualquer coisa, então com essa diferença muito grande de local para local fica impossível para o algoritmo identificar um padrão para as categorias de cômodo vazio e ocupado.

Apesar dos resultados serem insatisfatórios, a categoria rostos apresentou uma taxa de acerto acima de 95% todas as vezes, então o diretório de treinamento para rostos foi mantido.

Para resolver o problema da grande variação de cômodo para cômodo o algoritmo usa as fotos do cômodo atual para treinar a rede neural. Para o cômodo vazio basta salvar as fotos onde nenhum movimento foi detectado, mas para o cômodo ocupado são necessárias fotos com movimento. Então é necessário que o usuário entre no campo de visão da câmera e faça com que o sistema registre fotos do cômodo ocupado.

Dessa maneira a rede neural é treinada para o cômodo em específico e apresentou resultados com fidelidade mínima de 80%. Isso traz uma desvantagem quando a câmera muda de posição. Caso isso aconteça é necessário que um novo diretório de treinamento seja criado, uma vez que a solução é aplicada para o cômodo em específico, se isso não for feito não existirá nenhuma garantia a respeito da confiabilidade da previsão.

Voltando no algoritmo usado para detectar movimento. Logo quando o programa inicia uma das primeiras ações é a limpeza dos diretórios de treinamento. Isso é feito utilizando as bibliotecas *os* e *shutil* já mencionadas anteriormente.

Durante algum tempo o algoritmo começa a armazenar imagens para serem futuramente usadas para treinar a rede neural. A rede só é treinada caso haja um número mínimo de imagens em cada

categoria, isso porque quanto maior for o diretório de treinamento, maior é a confiabilidade. Depois que a rede neural é treinada a atualização do quadro de fundo passa a operar dependendo da análise da imagem feita pela rede neural.

Para retreinar a última camada da CNN *inception* é utilizado o algoritmo *retrain.py*, que não será explicado neste texto por ser bastante extenso e complexo [51]. O comando com cabeçalho "TRAINING"a seguir possui os parâmetros necessários para o treinamento da última camada da CNN. Para economizar tempo de treinamento apenas 100 iterações de treinamento são executadas.

TRAINING

1. python retrain.py \
   –bottleneck_dir=/tf_files/bottlenecks \
   –how_many_training_steps=100 \
   –model_dir=/tf_files/inception \
   –output_graph=/tf_files/retrained_graph.pb \
   –output_labels=/tf_files/retrained_labels.txt \
   –image_dir=/tf_files/photos

O treinamento tem duração de 2 ou 3 minutos (dependendo do tamanho da base de dados de treinamento) e pode ser feito utilizando a GPU (causando redução desse tempo). Para a utilização da rede neural treinada o *TensorFlow* é utilizado.

A primeira linha do programa de cabeçalho "RECOGNIZE"é responsável por ler a imagem. A segunda, por carregar as categorias inseridas. Em seguida da linha 4 a 7 a rede é carregada e importada.

Por fim, criou-se uma sessão do *TensorFlow*, criando assim um ambiente para fazer operações como a da linha 10. A primeira coisa feita na sessão é criar um *softmax tensor* para a última camada do modelo. Esse tensor é usado para transformar dados de entrada em probabilidades de uma saída esperada, isso é feito na linha 10 e 11. A última linha organiza as previsões em ordem de confiança.

RECOGNIZE

1. image_data = tf.gfile.FastGFile("output.jpg", ’rb’).read()
2. label_lines = [line.rstrip() for line
3. in tf.gfile.GFile("retrained_labels.txt")]
4. with tf.gfile.FastGFile("retrained_graph.pb", ’rb’) as f:
5. graph_def = tf.GraphDef()

```
6.    graph_def.ParseFromString(f.read())
7.    _ = tf.import_graph_def(graph_def, name='')
8.
9. with tf.Session() as sess:
10.    softmax_tensor = sess.graph.get_tensor_by_name('final_result:0')
11.    predictions = sess.run(softmax_tensor,
          'DecodeJpeg/contents:0': image_data)
12.    top_k = predictions[0].argsort()[-len(predictions[0]):][::-1]
```

3.2.10 Android Studio e Firebase

O *Android Studio* [53] é o ambiente de desenvolvimento oficial para a plataforma Android. Foi anunciado em 2013 pelo Google e foi escrito em Java, portanto o aplicativo todo foi escrito em Java. É um programa de alto nível, possui um emulador interno que permite testar a aplicação, possui um editor de código bastante avançado, possui muitas bibliotecas prontas e permite a adição de qualquer outra biblioteca.

O *Firebase* [50] por sua vez é um *back-end* completo para aplicações móveis, ou seja, o *Firebase* cuida das operações do lado do servidor para aplicativos de celular. O *Firebase* evoluiu em 2011 de um projeto chamado *Envolve* criado por Andrew Lee e James Tambplin. O *Envolve* era uma API que integrava *chats* em *sites*, permitindo a comunicação entre desenvolvedores e clientes. Porém esse aplicativo começou a ser utilizado para trocar dados que não eram mensagens comuns, sincronizando dessa forma os dados com o aplicativo. Em 2014 o *Firebase* foi comprado pela empresa Google e a partir daí o projeto cresceu muito.

Possui diversas funcionalidades como por exemplo a integração com alguns serviços da empresa Google como o *AdMob* e o *AdWords*. As principais funções que foram utilizadas no projeto foram a autenticação, o *database* de tempo real(NoSQL para dados JSON) e as notificações para usuários.

A ideia foi criar um aplicativo capaz de acessar qualquer sistema de monitoramento através de um cadastro e de uma autentificação.

O cadastro feito através do aplicativo é o mesmo que será utilizado na página web criada. A criação de usuários foi definida como um processo que só pode ser feito pelo aplicativo. Para unificar cada sistema, cada um dos sistemas foi associado a um número de série (que constará na embalagem do produto). Então na criação de usuário um endereço de e-mail, uma senha e um número de série são requeridos. Caso o número de serial exista e o e-mail seja válido uma conta é criada no *Firebase*.

Para permitir que esta mesma conta acesse a página web o e-mail e a senha do cadastro são salvas no banco de dados do MySQL do sistema que o usuário possui (na tabela *users*).

Além dessas informações a respeito da conta do usuário, o aplicativo também salva a identificação do usuário na tabela *notifications* da base de dados para que seja possível o envio de notificações para o dispositivo móvel do usuário. Cada dispositivo móvel possui uma identificação única.

Cada número de série foi vinculado ao endereço de IP da placa. Quando um usuário é criado as informações do usuário (e-mail, número de série e IP) também são salvas na base de dados do *Firebase*.

O maior problema encontrado no gerenciamento da base de dados do *Firebase* foi o fato de as funções de acesso à base de dados serem assíncronas. Isso impossibilita que qualquer variável mude dentro dessas funções, mesmo que a variável seja global ou processada como um método.

Após a autentificação do usuário o aplicativo possui 4 funcionalidades principais: configurações de conta, sair, configurações do sistema e uma transmissão ao vivo.

Nas configurações de conta é possível excluir a conta atual. Isso é um procedimento simples porém requer que o usuário da tabela *conf* e a identificação do usuário da tabela *notifications* também sejam removidos, isso é feito utilizando um POST em códigos feitos em PHP. O usuário salvo na base de dados do Firebase também é removido.

A segunda funcionalidade é a mais simples e é responsável por sair da conta atual.

As configurações de conta acessam a tabela *conf* da base de dados do sistema e verificam o estado de cada configuração. Isso é feito para que as configurações apareçam na tela conforme o estado atual delas, ou seja: se o monitoramento estiver habilitado, o seletor que habilita o monitoramento fica na posição que representa o monitoramento ativo.

Da mesma maneira, quando um desses seletores for mudado de posição a configuração é alterada na base de dados. O programa utiliza POST usando requerimentos HTTP para acessar e mudar as configurações. Os códigos que recebem o POST são ambos feitos em PHP.

Por último, a transmissão das imagens do monitoramento em tempo "real"(quase real devido à latência) pode ser feita. Para isso uma biblioteca para exibição de imagens e de vídeos chamada *Picasso* foi utilizada [54]. Essa biblioteca é capaz de carregar uma imagem a partir de um endereço, então a imagem atual do monitoramento é colocada no diretório raiz do Apache. Como a imagem é sobrescrita a cada instante o efeito resultante é a de transmissão de vídeo.

Os códigos em Java são bastante extensos e constam no Apêndice A desta monografia.

3.2.11 Transmissão de vídeo

A princípio, em paralelo com o programa de detecção de movimento, um programa de transmissão de vídeo também era executado. Um programa também foi feito em *Python* e utiliza *Flask* como servidor.

O programa fazia a leitura da imagem armazenada e transmitia para a página na porta 8082. Como desvantagem o programa depende da eficiência de execução do programa de detecção de movimentos, logo se o programa demora a ser executado (que é o caso), a taxa de atualização de quadros do transmissor também é baixa.

Para não sobrecarregar a *BeagleBone* esse código foi desencorajado e para transmitir o estado atual do monitoramento, o quadro atual do monitoramento é salvo diretamente no diretório raiz do *Apache*. Quando o algoritmo descrito acima é utilizado a câmera pode perder sincronia com o algoritmo de monitoramento, então dessa forma esse problema também é evitado.

3.2.12 Data e hora

A sincronização da data e da hora da *BeagleBone* torna-se importante a partir do momento que se deseja utilizar o *Dropbox*, pois para fazer um *upload* ou até mesmo para se conectar com o aplicativo a data e a hora precisam estar sincronizadas.

Esse controle pôde ser feito manualmente através dos comandos:

$ date –set YYYY:MM:DD

$ date –set HH:MM:SS

Mas esse processo foi automatizado para que o horário seja atualizado durante o *boot* do sistema. Para isso foram instalados pacotes para atualizar o tempo automaticamente. O *ntp* é um dos programas que automatizam esse processo, mas apresentou problemas durante a execução, portanto o programa usado foi o *rtime*.

$ apt-get install rtime

Portanto para atualizar a hora foi executado o comando abaixo:

$ rdate -s time-a.nist.gov.

3.2.13 Tarefas agendadas

O *cron* é um serviço do *Linux* que é carregado durante o processo de *boot* do sistema. É uma ferramenta que permite programar a execução de programas e processos de maneira repetitiva ou apenas uma única vez [55].

Para executar tarefas o *cron* usa uma espécie de tabela conhecida como *crontab*. Para acessar esse arquivo basta acessar com um editor:

$ nano /etc/crontab

As seguintes funções foram adicionadas para que sejam executadas.

@reboot root rdate -s time-a.nist.gov

***/10 * * * * root python /home/grgoncal/deteccao/monitor_python.sh**

@daily root rdate -s time-a.nist.gov

De 10 em 10 minutos o *crontab* executa um programa que monitora o programa de detecção de movimento caso algum erro ou *bug* encerre ele. Além disso a hora da BeagleBone é atualizada ao ligar e diariamente. Alguns comandos do *crontab* estão na Figura 3.16 [56].

O programa responsável pelo monitoramento do programa de deteção é um *shell* bastante simples que quando executado verifica se existe alguma aplicação *Python* em execução e reestabelece o monitoramento caso não exista [57].

Keyword	Equivalent
@yearly	0 0 1 1 *
@daily	0 0 * * *
@hourly	0 * * * *
@reboot	Run at startup.

Figura 3.16: Comandos do *crontab*

Capítulo 4

Resultados e Discussões

O objetivo principal do aplicativo foi alcançado, que foi o desenvolvimento de um sistema de segurança. A página de *login* e a página de cadastro podem ser vistas na Figura 4.1.

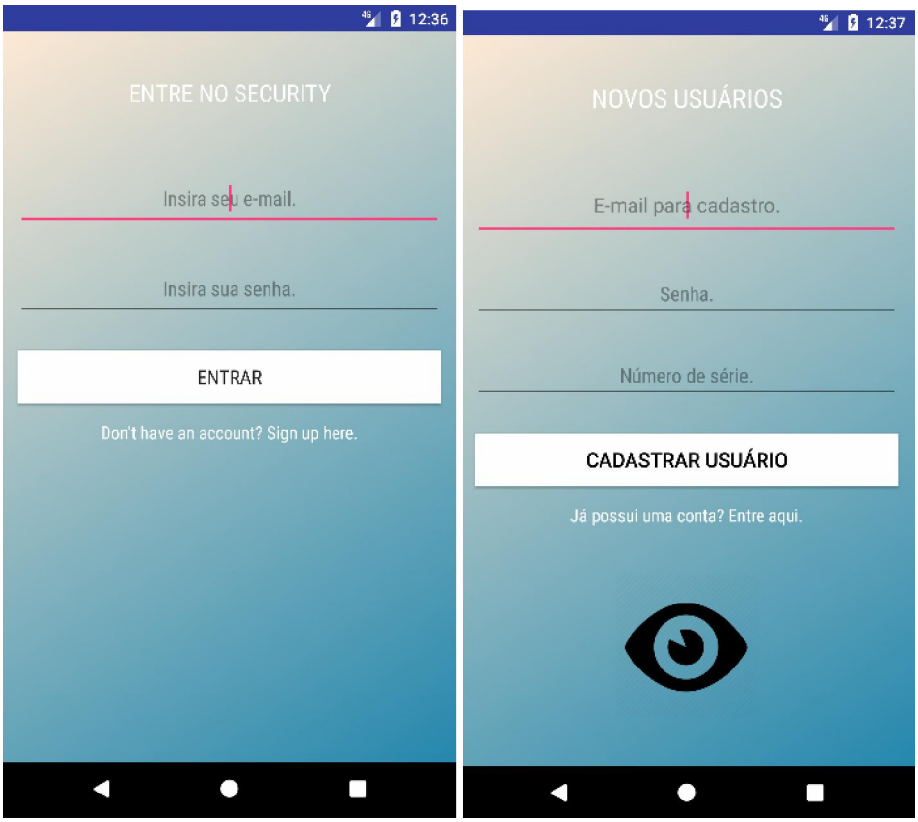

Figura 4.1: Página de *login* e de cadastro do aplicativo

Quando o usuário entra no aplicativo ele tem acesso às configurações do sistema, às configurações

de conta e a uma transmissão ao vivo do monitoramento. Essas funcionalidades podem ser vistas nas Figuras 4.2 e 4.3.

Outras funcionalidades podem ser adicionadas futuramente, além de mudanças estéticas. A opção de excluir usuário é oculta e aparece apenas quando o botão excluir conta é pressionado, atuando dessa maneira como um ato de confirmação de exclusão.

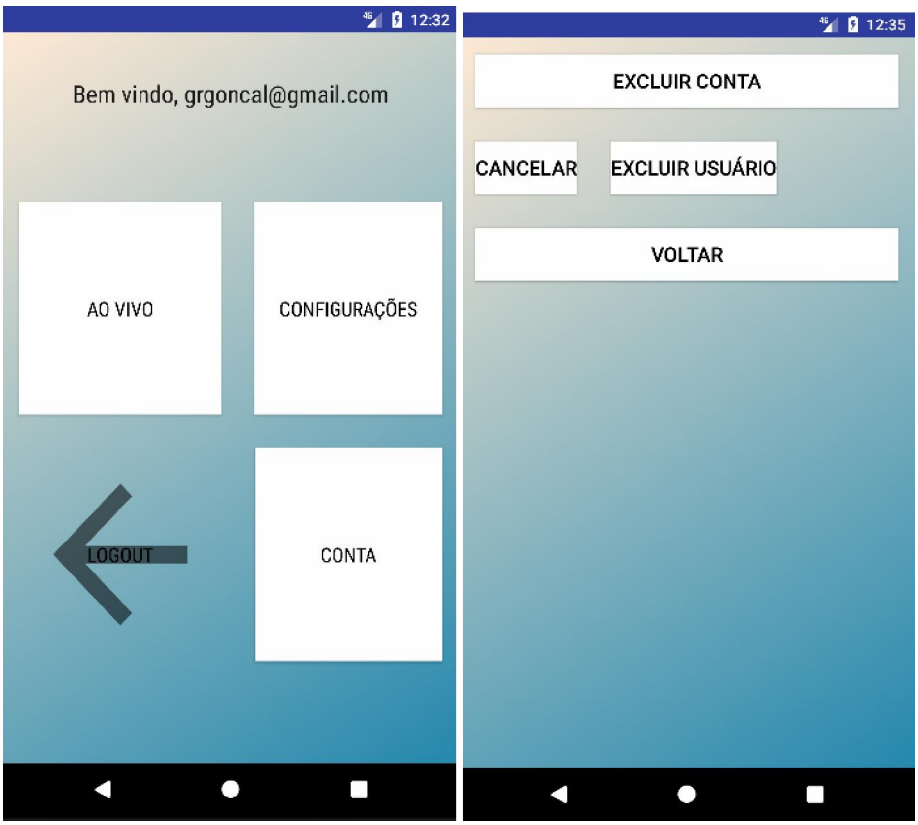

Figura 4.2: Página de menu e de configuração de conta

A transmissão de vídeo transmite a imagem diretamente do servidor web. As configurações do sistema são carregadas diretamente da base de dados e ficam da cor rosa quando estão ligadas.

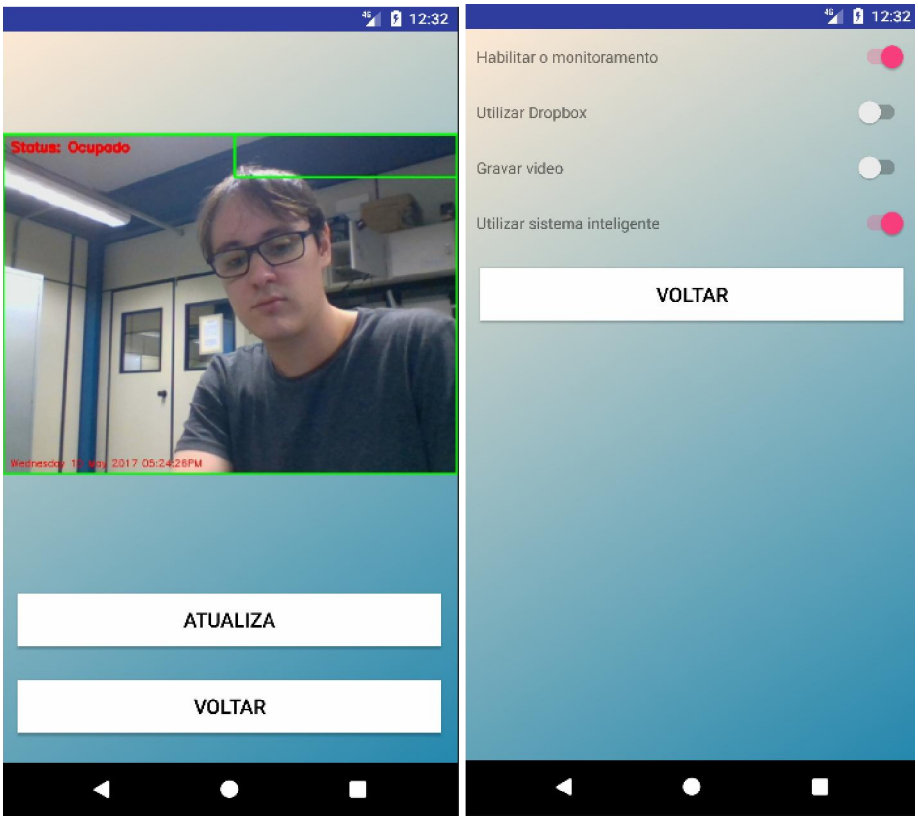

Figura 4.3: Página de transmissão e de configurações

O aplicativo e o sistema como um todo requerem conexão com a *internet*. Se o *smartphone* do usuário não estiver conectado a *internet*, não será possível entrar no sistema ou executar qualquer operação.

Para a página web do sistema a parte estética foi mais trabalhada. O site foi desenvolvido de maneira dinâmica em apenas uma página. A página inicial é a da Figura 4.4.

Figura 4.4: Página inicial do site

Figura 4.5: Página do desenvolvedor

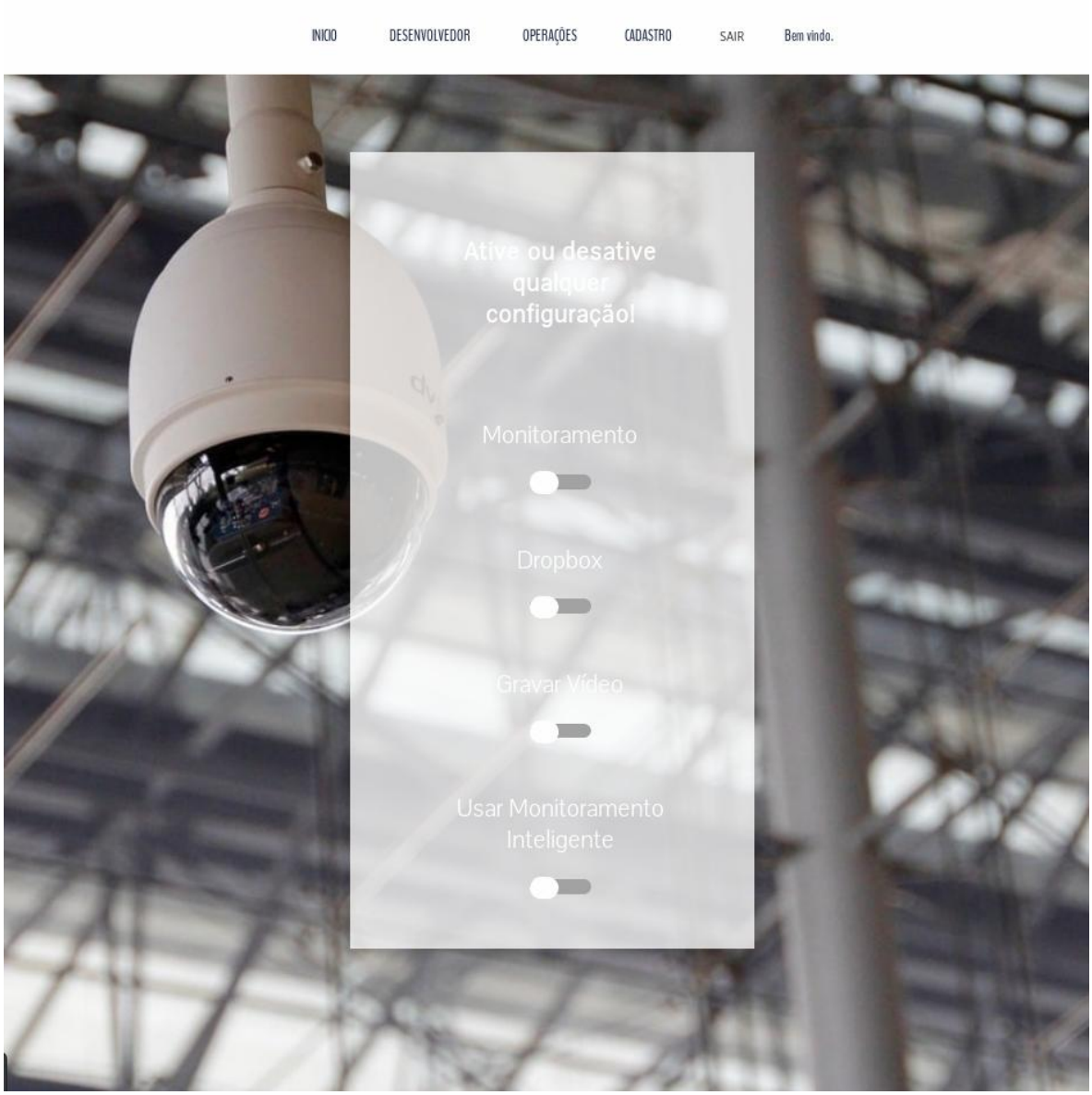

Figura 4.6: Página de configurações

Finalmente foram observadas as saídas do programa. A partir do quadro de fundo, qualquer objeto ou corpo que esteja dentro do campo de visão da câmera e que seja retirado (no caso do objeto) ou se mova (no caso de um corpo) será considerado como um movimento. Como consequência, caso a câmera seja movida o sistema indicará um falso positivo. A reposta do sistema é rápida (poucos segundos) e depende da conexão com a *internet*.

Duas das imagens obtidas podem ser vistas na Figura 4.7 e na Figura 4.9. Essas imagens representam movimentos detectados pela câmera.

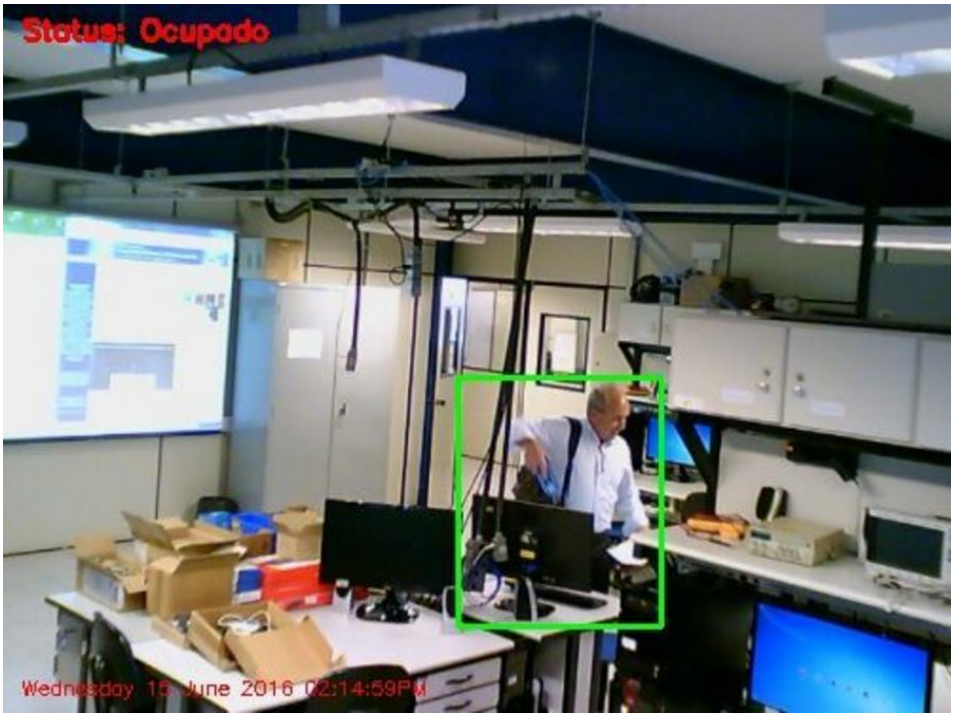

Figura 4.7: Detecção de movimento no laboratório

Figura 4.8: Detecção de movimento em ambiente com pouca luz

Figura 4.9: Detecção de movimento em ambiente com pouca luz

Figura 4.10: Erro de detecção devido à sombras

Outro exemplo obtido é a Figura 4.9. Nesse caso a iluminação do cômodo interfere bastante na detecção devido ao fato de a câmera ter maior variação de pixels no escuro, mesmo assim o algoritmo

foi capaz de identificar o movimento. A sombra causada por alguém pode interferir no resultado principalmente nos casos onde a iluminação é ruim, como pode ser visto nas Figuras 4.11 e 4.10.

Para exemplificar como era feita a atualização do quadro de fundo podemos ver na imagem 4.11 onde quatro pessoas estão na imagem, mas apenas uma pessoa foi detectada. Isso acontece porque as outras três pessoas agora fazem parte do quadro de fundo da imagem e só serão detectadas quando voltarem a se mexer ou se saírem da visão da câmera muito devagar.

Figura 4.11: Erro de detecção devido à imobilidade

Para demonstrar o funcionamento da atualização de fundo utilizando o *TensorFlow*, uma cadeira que compunha o plano de fundo do monitoramento foi retirada do local. Assim que foi retirada, o sistema começou a detectar a área onde a cadeira deveria estar como parte de um movimento. Porém, cerca de um minuto depois, o *Tensorflow* não identificou nenhum corpo na imagem e atualizou o plano de fundo. Isso pode ser visto nas Imagens 4.12 e 4.13.

Se o Tensorflow não estivesse habilitado, o fundo nunca seria atualizado e a ausência da cadeira seria detectada como movimento até que o usuário desligue o sistema.

Figura 4.12: Fundo adaptativo utilizando Tensorflow

Figura 4.13: Fundo adaptativo utilizando Tensorflow

O projeto apresenta uma capacidade de armazenamento limitada para os vídeos gerados. Os vídeos possuem durações variadas e têm uma média de 0,52MB por segundo, dessa maneira um vídeo de 1 minuto tem aproximadamente 31MB. No projeto foi utilizado um cartão SD de 2GB, a ROM interna

da *BeagleBone Black* não foi utilizada. O sistema operacional e as aplicações ocupam cerca de 1,2GB deste espaço, sobrando cerca de 800MB para uso geral. Usando a média aproximada dos vídeos é possível gravar no total 25 minutos. Esse tempo pode ser otimizado utilizando uma distribuição menor e aumentando o tamanho do cartão SD utilizado. Um cartão de 16GB possibilitaria mais de 7 horas de gravação.

Caso a conexão com a *internet* cesse antes do início da execução do algoritmo, não será possível fazer o *upload* de fotos no Dropbox e também não será possível enviar notificações para o usuário. Porém os vídeos do movimento são registrados. A conexão com a *internet* é recomendada pois a falta dela impede o funcionamento de vários módulos.

O poder de processamento da *BeagleBone Black* é pouco comparado a um computador pessoal. O algoritmo responsável pela vigilância e detecção de movimento é bastante pesado e precisa atuar todo o tempo ocupando, dessa maneira, cerca de 90% da CPU. Dessa maneira o *hardware* disponível mantém uma taxa de quadros baixa com aproximadamente 2 quadros por segundo, um número baixo. Além disso o Tensorflow também ocupa grande parte do poder de processamento e durante o seu uso interrompe o algoritmo de monitoramento. Apesar de a taxa de quadros ser baixa, a utilização da GPU da *BeagleBone* em ambos os casos poderá aumentar esse número. Essa implementação não foi feita e consta como a implementação futura mais importante.

Capítulo 5

Conclusão

A ideia do projeto foi baseada no conceito de *Internet* das Coisas e deseja trazer mais segurança para as pessoas.

A transformação do projeto em um produto foi mantida com clareza e levada em consideração nas implementações futuras. O diferencial do projeto é o custo baixo e a baixa manutenção que o projeto deverá apresentar.

O projeto deu oportunidade para o estudo na área de sistemas embarcados e de sistemas operacionais baseados em *Linux*. Além do estudo de diversas outras áreas e linguagens de programação tais como o Python, Java, PHP, SQL, HTML, jQuery, CSS e o OpenCV. Várias dessas áreas foram vistas pela primeira vez durante o desenvolvimento, assim como vários *frameworks* e API's utilizados.

O algoritmo de monitoramento teve resultados dentro do esperado apesar de usar boa parte do poder de processamento da CPU da *BeagleBone*. Boa parte dos resultados foram obtidos através de um computador para acelerar o desenvolvimento. Várias funcionalidades foram adicionadas ao longo do desenvolvimento e todas se mostraram úteis para o projeto. O uso do TensorFlow para identificação do estado atual do quarto foi um salto muito grande no desenvolvimento e representa uma solução que está cada vez mais presente nas aplicações em geral.

A página web foi desenvolvida por inteiro sem o uso de nenhum modelo pronto e apresentou resultados ótimos tanto na funcionalidade como esteticamente. Se comparada com os resultados obtidos com o *framework* Wordpress os resultados podem ser considerados excelentes.

O aplicativo desenvolvido para *smartphone* também atendeu às expectativas e funcionou bem para a aplicação. Muito ainda pode ser feito no aplicativo assim como será descrito nas implementações futuras.

A *BeagleBone Black* mostrou ser um circuito embarcado eficiente para a aplicação apesar de seu

poder de processamento limitado. Finalmente o projeto foi desenvolvido e apresentou bons resultados acompanhados de um aprendizado muito grande.

5.0.1 Implementações futuras

A implementação futura mais importante é a adaptação do algoritmo de monitoramento utilizando OpenCL (Open Computing Lenguage), que é um *framework* (biblioteca de classes que suportam uma funcionalidade) para escrever programas que são executados através de plataformas heterogêneas como por exemplo uma GPU (a BeagleBone possui uma GPU, a SGX530 3D). Essa adaptação irá proporcionar uma aceleração no tempo de execução do algoritmo de monitoramento causando uma taxa de quadros maior (maior fluidez dos movimentos detectados).

O algoritmo de aprendizagem de máquina também deve ser processado pela GPU da BeagleBone e será alvo de estudos futuros.

A interface com o usuário também deve sofrer grandes alterações estéticas e funcionais tanto para a página web quanto para o aplicativo Android. As alterações funcionais a serem adicionadas são a adição do acesso a configurações avançadas do sistema (por exemplo a área mínima para que o algoritmo detecte), a adição de mais configurações de conta do usuário e a adição de maiores medidas de segurança, como por exemplo a verificação de e-mail após o cadastro.

A questão de segurança digital é o segundo ponto mais importante do desenvolvimento do projeto. De nada adianta um sistema de segurança não possuir segurança em sua implementação. Medidas como rastreamento de IP, número limitado de tentativas de login e talvez até a adição de um reCAPTCHA [58], que protege ataques automáticos.

O programa de detecção de movimentos é sempre alvo de futuros estudos em relação ao desempenho e aos *bugs* em termos de melhoramento contínuo.

Toda a funcionalidade deverá ser programada de forma a permitir a adição de módulos extras para o projeto. Esses módulos podem ser outros projetos como por exemplo interfones inteligentes, controle de iluminação ou de som, ou qualquer outro tipo de aplicação IOT e até mesmo *offline*.

Uma opção importante a ser analisada e testada é o uso de hardware em algumas partes do projeto. Uma delas é a adição de um sensor de presença para evitar que a BeagleBone fique o tempo todo sobre carregada. Essa adição permitiria que a CPU efetuasse várias outras operações enquanto nada acontece no campo de visão da câmera. Essas várias operações poderiam ser por exemplo outras extensões do monitoramento, uma transmissão ao vivo sem processar a imagem (resultando em uma taxa de quadros muito maior) ou até mesmo outras funcionalidades completas como o controle de fechaduras eletrônicas, interfones eletrônicos, iluminação ou qualquer outra função.

Outra adição de hardware que deverá ser testada é um sensor capacitivo ou um acelerômetro na câmera. Isso permitiria uma informação a mais a respeito de quando a câmera foi movida, além do que seria uma informação com muito mais confiabilidade do que as obtidas via software.

Há um problema bastante grave que ainda não foi resolvido no projeto. Esse problema está no fato de o endereço de IP de cada um dos sistemas variar de rede para rede. Para resolver esse problema será necessário que exista um servidor DNS responsável por todos os sistemas e representa o próximo desenvolvimento que o projeto terá. O servidor DNS irá fazer a comunicação entre todos os sistemas e irá fazer com que cada um deles tenha um endereço de IP único.

O uso do "monitoramento inteligente"será analisado e testado uma maneira de simular movimento no cômodo de maneira a poupar o usuário de ter que se movimentar no cômodo, além de salvar tempo dele. Isso poderá ser feito adicionando corpos via software nas fotos do cômodo vazio, simulando dessa maneira um local ocupado.

O *upload* de vídeos no Dropbox também será feito para segurança dos dados. Além disso o algoritmo de detecção sofrerá adaptações para não perder informações caso haja uma queda na conexão com a *internet*.

Referências

[1] Especificações do *smartphone Samsung Galaxy Note 4*. www.samsung.com/pt/promotions/galaxynote4/spec/. Acesso em 26 jun. 2016.

[2] Portal Terra. Há 43 anos, homem chegava à lua com computador de 2kb de RAM. http://noticias.terra.com.br/ciencia/ha-43-anos-homem-chegava-a-lua-com-computador-de-2-kb-de-ram,582a8116492da310VgnCLD200000bbcceb0aRCRD.html. Acesso em 26 jun. 2016.

[3] Teleco: *Smartphones* no Mundo. http://www.teleco.com.br/Smartphone_mundo.asp. Acesso em 26 jun. 2016.

[4] A. Monteiro Mario. *Introdução à Organização de Computadores*. Editora LTC. 5ª Edição, 2012.

[5] The Minuteman III ICBM. http://nuclearweaponarchive.org/Usa/Weapons/Mmiii.html. Acesso em 26 jun. 2016.

[6] Netcampos. Você sabe o que são Sistemas Embarcados? http://www.gruponetcampos.com.br/2011/05/voce-sabe-o-que-sao-sistemas-embarcados/. Acesso em 26 jun. 2016.

[7] Universidade Federal do Paraná. O que são Sistemas Eletrônicos Embarcados?]. www.eletrica.ufpr.br/graduacao/noturno/embarcados.html. Acesso em 26 jun. 2016.

[8] Machtelt garrels. introduction to linux: A hands on guide. 1.27 edition, 2008. http://tldp.org/LDP/intro-linux/html/sect_01_01.html. Acessado em 24 junho 2017.

[9] Vitalis, andré. le regard omniprésent de la vidéosurveillance. https://www.monde-diplomatique.fr/1998/03/VITALIS/3586. Acessado em 31 maio 2017.

[10] CCTV Surveillance: Video Practices and Technology. Kruegle, Herman. Segunda edição.

[11] História da vídeo vigilância e das câmeras de segurança. http://www.wecusurveillance.com/cctvhistory. Acessado em 31 maio 2017.

[12] FLIR. Câmeras de monitoramento. https://fx.flir.com/flirus/home. Acesso em 26 jun. 2016.

[13] Empresa ULO. Câmeras de monitoramento. https://www.kickstarter.com/projects/vivienmuller/ulo/description. Acesso em 26 jun. 2016.

[14] Dados estatísticos do Estado de São Paulo. Governo do Estado de São Paulo, Secretaria da Segurança Pública. www.ssp.sp.gov.br/novaestatistica/mapas.aspx. Acesso em 20 jun. 2016.

[15] *Website* do jornal Estadão de 25 Setembro 2015. Resk, Felipe. "Sensação de segurança no Brasil é equivalente à do Afeganistão". Acesso em 20 jun. 2016.

[16] Kerry McGuire Balanza (11 May 2010), ARM from zero to billions in 25 short years, ARM Holdings, 2012.

[17] Processadores ii. apostila: Arquitetura arm-cortex. http://www.feng.pucrs.br/~stemmer/proc2stm/cortex-m3.html. Acessado em 24 junho 2017.

[18] Informações pertinentes relacionadas ao Linux. http://www.linuxfoundation.org/what-is-linux. Acessado em 16 jun. 2016.

[19] Site oficial da distribuição Debian. https://www.debian.org/intro/about. Acessado em 16 jun. 2016.

[20] Documentação disponível para a distribuição *Debian*. https://www.debian.org/doc/manuals/user/ch6.html. Acessado em 16 jun. 2016.

[21] Documentação oficial do opencv. módulos da biblioteca. http://docs.opencv.org/3.1.0/. Acessado em 24 junho 2017.

[22] R. E. Gonzalez, R. C.; Woods. *Processamento Digital de Imagens*. Editora Pearson. 3ª Edição, 2010.

[23] Página oficial do *Tensorflow* e informações sobre a biblioteca. https://www.tensorflow.org/. Acessado em 24 junho 2017.

[24] Holger R. Roth; Mingchen Gao; Le Lu; Ziyue Xu; Isabella Nogues; Jianhua Yao; Daniel Mollura; Ronald M. Summers* Hoo-Chang, Shin;. *Deep Convolutional Neural Networks for Computer-Aided Detection: CNN Architectures, Dataset Characteristics and Transfer Learning. 2016.*

[25] Bengio; Geoffrey Hinton Yann, LeCun; Yoshua. *Deep Learning*. Macmillan Publishers Limited. 2015.

[26] Desafio em larga escala de reconhecimento de imagens (*Large Scale Visual Recognition Challenge*). http://www.image-net.org/challenges/LSVRC/. Acessado em 31 maio 2017.

[27] Documentação oficial de reconhecimento de imagens do tensorflow. https://www.tensorflow.org/tutorials/image_recognition. Acessado em 31 maio 2017.

[28] Site oficial do ImageNet. http://image-net.org/. Acessado em 16 maio 2017.

[29] Site oficial do WordNet. http://wordnet.princeton.edu/. Acessado em 16 maio 2017.

[30] Liu; Yangqing Jia; Pierre Sermanet; Scott Reed; Dragomir Anguelov; Dumitru Erhan; Vincent Vanhoucke; Andrew Rabinovich; Google Inc; University of North Carolina; Chapel Hill; University of Michigan; Ann Arbor; Magic Leap Inc. Christian, Szegedy; Wei. *Going Deeper with Convolutions*. CVPR Open Access version by Computer Vision Foundation. 2015.

[31] Documentação completa da *BeagleBone*. http://elinux.org/Beagleboard:BeagleBoneBlack. Acessado em 16 jun. 2016.

[32] Instalação da distibuição *Debian* no sistema embarcado *BeagleBone*. http://www.decom.ufop.br/imobilis/instalando-o-ubuntu-na-beaglebone-black-em-sd-card/. Acessado em 16 jun. 2016.

[33] Site oficial do sistema embarcado *BeagleBone Black*. https://beagleboard.org/black. Acessado em 16 jun. 2016.

[34] Dados do fabricante da câmera ODROID. http://www.hardkernel.com/main/products/prdt_info.php?g_code=G137517754892. Acessado em 16 jun. 2016.

[35] Página do *Software* usado para formatar cartões sd, *SDCardFormatter*. https://www.sdcard.org/downloads/formatter_4/. Acessado em 24 junho 2017.

[36] Página do *Software* usado para instalar distribuições linux em cartões sd, *Win32 Disk Imager*. https://wiki.ubuntu.com/Win32DiskImager. Acessado em 24 junho 2017.

[37] Editor de partições, *Gparted*. http://gparted.org/. Acessado em 24 junho 2017.

[38] Página do *Software* de emulação de terminal com suporte a ssh, o *Putty*. http://www.putty.org/. Acessado em 24 junho 2017.

[39] Montando um servidor lamp. http://lamphowto.com/. Acessado em 24 junho 2017.

[40] Pesquisa de servidores web feita em Janeiro de 2015. http://news.netcraft.com/archives/2015/01/15/january-2015-web-server-survey.html. Acessado em 16 jun. 2016.

[41] Site oficial do SELinux. http://selinuxproject.org/page/Main_Page. Acessado em 16 maio 2017.

[42] *Wordpress*, um aplicativo de sistema de gerenciamento de conteudo para *web*. https://br.wordpress.com/. Acessado em 24 junho 2017.

[43] Instalação do phpMyAdmin para o Sistema Operacional Debian. https://www.digitalocean.com/community/tutorials/how-to-install-and-secure-phpmyadmin-on-debian-7. Acessado em 16 maio 2017.

[44] Documentação do comando *session_start* do PHP. https://www.digitalocean.com/community/tutorials/how-to-install-and-secure-phpmyadmin-on-debian-7. Acessado em 16 maio 2017.

[45] Documentação oficial do opencv. http://docs.opencv.org/2.4/index.html. Acessado em 18 maio 2017.

[46] Solução do *bug* do FFMPEG durante a instalação do OpenCV. http://stackoverflow.com/questions/31663498/opencv-3-0-0-make-error-with-ffmpeg/31818445#31818445. Acessado em 16 jun. 2016.

[47] I just open sourced my personal imutils package: A series of OpenCV convenience functions. http://www.pyimagesearch.com/2015/02/02/just-open-sourced-personal-imutils-package-series-opencv-convenience-functions/. Acessado em 18 maio 2017.

[48] Página do *Dropbox* para desenvolvedores. https://www.dropbox.com/developers. Acessado em 16 jun. 2016.

[49] Modelo de referência utilizado para o desenvolvimento do programa de detecção de movimentos. http://www.pyimagesearch.com/2015/06/01/home-surveillance-and-motion-detection-with-the-raspberry-pi-python-and-opencv/. Acessado em 16 jun. 2016.

[50] *Firebase*, *backend* de aplicativos para *smartphones*. https://firebase.google.com/?hl=pt-br. Acessado em 24 junho 2017.

[51] Tutorial do google developers codelabs do tensorflow for poets. https://codelabs.developers.google.com/codelabs/tensorflow-for-poets/?utm_campaign=chrome_series_machinelearning_063016&utm_source=gdev&utm_medium=yt-desc#0. Acessado em 18 maio 2017.

[52] Jeff Heaton. *Artificial Intelligence for Humans, Volume 3: Deep Learning and Neural Networks*. Createspace Independent Publishing Platform. 2015.

[53] *Android Studio*, plataforma para criação de aplicativos android. https://developer.android.com/studio/index.html. Acessado em 24 junho 2017.

[54] Biblioteca *Picasso* para o *download* e *caching* de imagens para android. http://square.github.io/picasso/. Acessado em 21 maio 2017.

[55] Raphaël Hertzog; Roland Mas ; Freexian SARL. *O Manual do Administrador Debian*. Livro *online* licenciado sob Licença Creative Commons Attribution-ShareAlike 3.0 Unported. 2003-2015. Capítulo 9.7, disponível em https://debian-handbook.info/browse/pt-BR/stable/sect.task-scheduling-cron-atd.html. Acessado em 24 junho 2017.

[56] Exemplos de uso do Crontab. http://www.thegeekstuff.com/2009/06/15-practical-crontab-examples/. Acessado em 16 jun. 2016.

[57] Uso do comando condicional *if* para programas *shell*. http://tldp.org/LDP/Bash-Beginners-Guide/html/sect_07_01.html. Acessado em 16 jun. 2016.

[58] Google *reCAPTCHA*, para proteção anti *bots*. https://www.google.com/recaptcha/intro/android.html. Acessado em 21 maio 2017.

Apêndice A

Monitoramento completo

Para que o trabalho não se extenda muito mais os programas utilizados foram todos disponibilizados no Git Hub, uma plataforma de hospedagem de código para controle de versão e colaboração. O link de acesso é:

https://github.com/grgoncal/monitoramento

O único código que será disponibilizado neste papel é o algoritmo de monitoramento.

```
#!/usr/local/lib/python2.7
#### IMPORTS ########################################
from dropbox.client import DropboxOAuth2FlowNoRedirect
from dropbox.client import DropboxClient
import cv2
import warnings
import numpy as np
import imutils
import time
import datetime
import tensorflow as tf,sys
import os, shutil
import MySQLdb
from pyfcm import FCMNotification
#### PREDICTIONS ####################################
def predictions():
```

```
        image_data = tf.gfile.FastGFile("output.jpg", 'rb').read()
        label_lines = [line.rstrip() for line
                in tf.gfile.GFile("retrained_labels.txt")]
        with tf.gfile.FastGFile("retrained_graph.pb", 'rb') as f:
            graph_def = tf.GraphDef()
            graph_def.ParseFromString(f.read())
            _ = tf.import_graph_def(graph_def, name='')
        with tf.Session() as sess:
            softmax_tensor = sess.graph.get_tensor_by_name('final_result:0')
            predictions = sess.run(softmax_tensor,
                    'DecodeJpeg/contents:0': image_data)
            top_k = predictions[0].argsort()[-len(predictions[0]):][::-1]
            for node_id in top_k:
                human_string = label_lines[node_id]
                score = predictions[0][node_id]
                return "%s %.2f"% (human_string, score)
#### TRAINING PHOTOS CLEANER ########################################
def cleanTrainingPhotos(folder):
    for the_file in os.listdir(folder):
        file_path = os.path.join(folder, the_file)
        try:
            if os.path.isfile(file_path):
                os.unlink(file_path)
            elif os.path.isdir(file_path): shutil.rmtree(file_path)
        except Exception as e:
            print(e)
#### LOADING CONFIGURATIONS ##########################################
db = MySQLdb.connect("localhost","root",«senha>","devgrgoncal")
cursor = db.cursor()
sql = "SELECT * FROM conf"
cursor.execute(sql)
results = cursor.fetchall()
use_dropbox = 0
min_upload_seconds = 1
```

```
min_area = 2
dropbox_key = 3
delta_thresh = 4
dropbox_base_path = 5
dropbox_secret = 6
status_monitoramento = 7
min_motion_frames = 8
use_tensorflow = 9
record_video = 10
accessToken = 11
apiKey = 12
#### LOADING NOTIFICATIONS ########################################
cursor2 = db.cursor()
sql2 = "SELECT * FROM notifications"
cursor2.execute(sql2)
results2 = cursor2.fetchall()
APIKey = results[0][apiKey]
#### CONFIGURACOES DA CAMERA E DE FLAGS ############################################
camera = cv2.VideoCapture(0)
camera.set(3,640)
camera.set(4,480)
camera.set(5,10)
avg = None
LastUploaded = datetime.datetime.now()
firstRun = 0
lastAccumulated = time.time()
motionCounter = 0
loadJson = 0
lastMovement = -1000
lastVideo = 1
lastFace = 1
lastBody = 1
lastEmpty = 1
lastEmptyUpdate = time.time()
```

```
    lastBodyUpdate = time.time()
    cameraMovedIndicator = -15
    lastTraining = 0
    lastTrainingTime = -10000000
    recording = 0
    fourcc = cv2.VideoWriter_fourcc(*'MJPG')
    cleanTrainingPhotos("tf_files/photos/Empty")
    cleanTrainingPhotos("tf_files/photos/Bodies")
    print "[CLEANING] Photos Directory Cleaned"
    message_title = "[ALERT] Security"
    message_body = "A movement was detected on the system."
    lastNotification = time.time()
    #### ENTRANDO NO DROPBOX ########################################
    if results[0][use_dropbox]:
        flow = DropboxOAuth2FlowNoRedirect(results[0][dropbox_key],results[0][dropbox_secret])
        client = DropboxClient(results[0][accessToken])
        print "[DROPBOX] Account linked: "+ results[0][accessToken]
    #### MAIN ########################################
    while(results[0][status_monitoramento]==1):
        ret,frame=camera.read()
        text = "Desocupado"
        timestamp = datetime.datetime.now()
        frame = imutils.resize(frame,width=500)
        rawframe = frame
        gray = cv2.cvtColor(frame,cv2.COLOR_BGR2GRAY)
        gray = cv2.GaussianBlur(gray,(21,21),0)
        if avg is None:
            print "[INFO] Starting Background Model."
            avg = gray.copy().astype("float")
            continue
        if (time.time() - lastAccumulated) > 30 and lastTraining > 0 and results[0][use_tensorflow] ==
1:
            statePrevision = predictions()
            if "empty"in statePrevision:
```

```
                temp = cv2.accumulateWeighted(gray,avg,0.4)
                print "[BACKGROUND] Background Model Updated."
                print "[PREVISION] Room empty."
                if len(next(os.walk("tf_files/photos/Empty"))[2]) < 250:
                    cv2.imwrite("tf_files/photos/Empty/empty"+ str(lastEmpty) + ".jpg",rawframe)
                    lastEmpty = lastEmpty + 1
                else:
                    lastEmpty = 1
            if "faces"in statePrevision:
                print "[PREVISION] Face detected."
                if len(next(os.walk("tf_files/photos/Faces"))[2]) < 250:
                    cv2.imwrite("tf_files/photos/Faces/faces"+ str(lastFace) + ".jpg",rawframe)
                    lastFace = lastFace + 1
                else:
                    lastFace = 1
            if "bodies"in statePrevision:
                print "[PREVISION] Body detected."
                if len(next(os.walk("tf_files/photos/Bodies"))[2]) < 250:
                    cv2.imwrite("tf_files/photos/Bodies/body"+ str(lastBody) + ".jpg",rawframe)
                    lastBody = lastBody + 1
                else:
                    lastBody = 1
            lastAccumulated = time.time()
        elif ((results[0][use_tensorflow] == 0 or lastTraining == 0) and (time.time() - lastAccumulated
> 6 and time.time() - lastMovement > 6)) or firstRun < 30:
            temp = cv2.accumulateWeighted(gray,avg,0.4)
            lastAccumulated = time.time()
            firstRun = firstRun + 1
        frameDelta = cv2.absdiff(gray,cv2.convertScaleAbs(avg))
        thresh = cv2.threshold(frameDelta,results[0][delta_thresh],255,cv2.THRESH_BINARY)[1]
        cnts = cv2.findContours(thresh.copy(),cv2.RETR_EXTERNAL,cv2.CHAIN_APPROX_SIMPLE)[-
2]
        for c in cnts:
            if cv2.contourArea(c) < results[0][min_area]:
```

```
                continue
            text = "Ocupado"
    #### CAMERA MOVED ########################################
            if cv2.contourArea(c) > 200000 and lastTraining > 0 and (time.time() - lastMovement < 4):
                cameraMovedIndicator = cameraMovedIndicator + 1
                if cameraMovedIndicator > 20:
                    statePrevison = predictions()
                    if "faces"in statePrevision:
                        cameraMovedIndicator = -10
                    else:
                        cleanTrainingPhotos("tf_files/photos/Empty")
                        cleanTrainingPhotos("tf_files/photos/Bodies")
                        firstRun = 5
                        lastTrainingTime = -1000000
                        lastTraining = 0
                       print "[UPDATE] Camera moved, starting new background model.  Stay clear
from the camera."
            if (time.time() - lastMovement > 4):
                cameraMovedIndicator = -10
            (x,y,w,h) = cv2.boundingRect(c)
            cv2.rectangle(frame,(x,y),(x+w,y+h),(0,255,0),2)
            lastMovement = time.time()
        ts = timestamp.strftime("%A %d %B %Y %I:%M:%S%p")
        cv2.putText(frame,"Status: ".format(text),(10,20),cv2.FONT_HERSHEY_SIMPLEX,0.5,(0,0,255),2)
        cv2.putText(frame,ts,(10,frame.shape[0]-10),cv2.FONT_HERSHEY_SIMPLEX,0.35,(0,0,255),1)
        cv2.imwrite("//output.jpg",frame)
        cv2.imwrite("/var/www/html/out1.jpg",frame)
        if text=="Ocupado":
            if(timestamp - LastUploaded).seconds>= results[0][min_upload_seconds]:
               motionCounter = motionCounter + 1
                if motionCounter >= results[0][min_motion_frames]:
    #### NOTIFICATION ########################################
                    if time.time() - lastNotification > 5:
                        message_body = "A movement on: "+ time.asctime(time.localtime(time.time()))
```

```
                    push_service = FCMNotification(api_key=APIKey)
                    for index in results2:
                        registration_id = index[1]
                        result = push_service.notify_single_device(registration_id = registration_id,
message_title = message_title, message_body = message_body)
                    lastNotification = time.time()
    #### DROPBOX ########################################
                if (results[0][use_dropbox]==1):
                    path="{base_path}/{timestamp}.jpg".format(base_path=results[0][dropbox_base_path],timestamp
                    client.put_file(path,open("output.jpg","rb"))
                    print "[UPLOAD] ".format(ts)
                lastUploaded = timestamp
                motionCounter = 0
    #### OBTAINING FIRST OCCUPIED TRAINING IMAGES ########################################
                if len(next(os.walk("tf_files/photos/Bodies"))[2]) < 200 and lastTraining == 0 and
time.time() - lastBodyUpdate > 1.2:
                    cv2.imwrite("tf_files/photos/Bodies/bodies"+ str(lastBody) + ".jpg",rawframe)
                    lastBody = lastBody + 1
                    lastBodyUpdate = time.time()
                    print "[TRAIN DATABASE] A body was added to the train database."
        else:
            motionCounter = 0
    #### OBTAINING FIRST EMPTY TRAINING IMAGES ########################################
            if len(next(os.walk("tf_files/photos/Empty"))[2]) < 200 and time.time() - lastEmptyUpdate
> 1.2 and lastTraining == 0 and text == "Desocupado":
                cv2.imwrite("tf_files/photos/Empty/empty"+ str(lastEmpty) + ".jpg",rawframe)
                lastEmpty = lastEmpty + 1
                lastEmptyUpdate = time.time()
                print "[TRAIN DATABASE] An empty room was added to the train database."
    #### RELOADING JSON FILE ########################################
        loadJson = loadJson + 1
        if(loadJson == 40):
            loadJson = 0
            cursor.execute(sql)
```

```
            results = cursor.fetchall()
            sql = "SELECT * FROM conf"
            cursor.execute(sql)
            results = cursor.fetchall()
            sql2 = "SELECT * FROM notifications"
            cursor2.execute(sql2)
            results2 = cursor2.fetchall()
    #### PICTURE NAME FLAGS RESET ########################################
        if len(next(os.walk("tf_files/photos/Empty"))[2]) >= 250:
            lastEmpty = 1
            cleanTrainingPhotos("tf_files/photos/Empty")
        if len(next(os.walk("tf_files/photos/Bodies"))[2]) >= 250:
            cleanTrainingPhotos("tf_files/photos/Bodies")
            lastBody = 1
    #### VIDEO RECORDING ########################################
        if time.time() - lastMovement < 3 and results[0][record_video] == 1:
            if recording == 0:
                size = frame.shape[1], frame.shape[0]
                out = cv2.VideoWriter(str(lastVideo)+'.avi',fourcc, 10.0, size,True)
            out.write(frame)
            recording = 1
        elif time.time() - lastMovement > 2.5 and recording == 1:
            recording = 0
            lastVideo = lastVideo + 1
            out.release()
    #### TENSORFLOW TRAINING ########################################
        if results[0][use_tensorflow] == 1 and len(next(os.walk("tf_files/photos/Empty"))[2]) >= 60
and len(next(os.walk("tf_files/photos/Faces"))[2]) > 100 and len(next(os.walk("tf_files/photos/Bodies"))[2])
>= 50 and time.time() - lastTrainingTime > 86400:
            cleanTrainingPhotos("tf_files/bottlenecks")
            os.system("python retrain.py –bottleneck_dir=tf_files/bottlenecks –how_many_training_steps
100 –model_dir=tf_files/inception –output_graph=tf_files/retrained_graph.pb –output_labels=tf_files/retrained_labels.t
–image_dir tf_files/photos")
            lastTrainingTime = time.time()
```

```
shutil.copyfile("tf_files/retrained_graph.pb","retrained_graph.pb")
shutil.copyfile("tf_files/retrained_labels.txt","retrained_labels.txt")
 lastTraining = lastTraining + 1
print "[TENSORFLOW] Model retrained."
```

Apêndice B

Criando uma pasta no API do Dropbox

A criação da API do Dropbox é bastante simples e pode ser feita em poucos passos. Basta entrar na página de Desenvolvedores do Dropbox e criar um novo *app*.

Figura B.1: Criando o API do Dropbox parte 1

Em seguida só é necessário definir o aplicativo como uma pasta e nomeá-la como desejado.

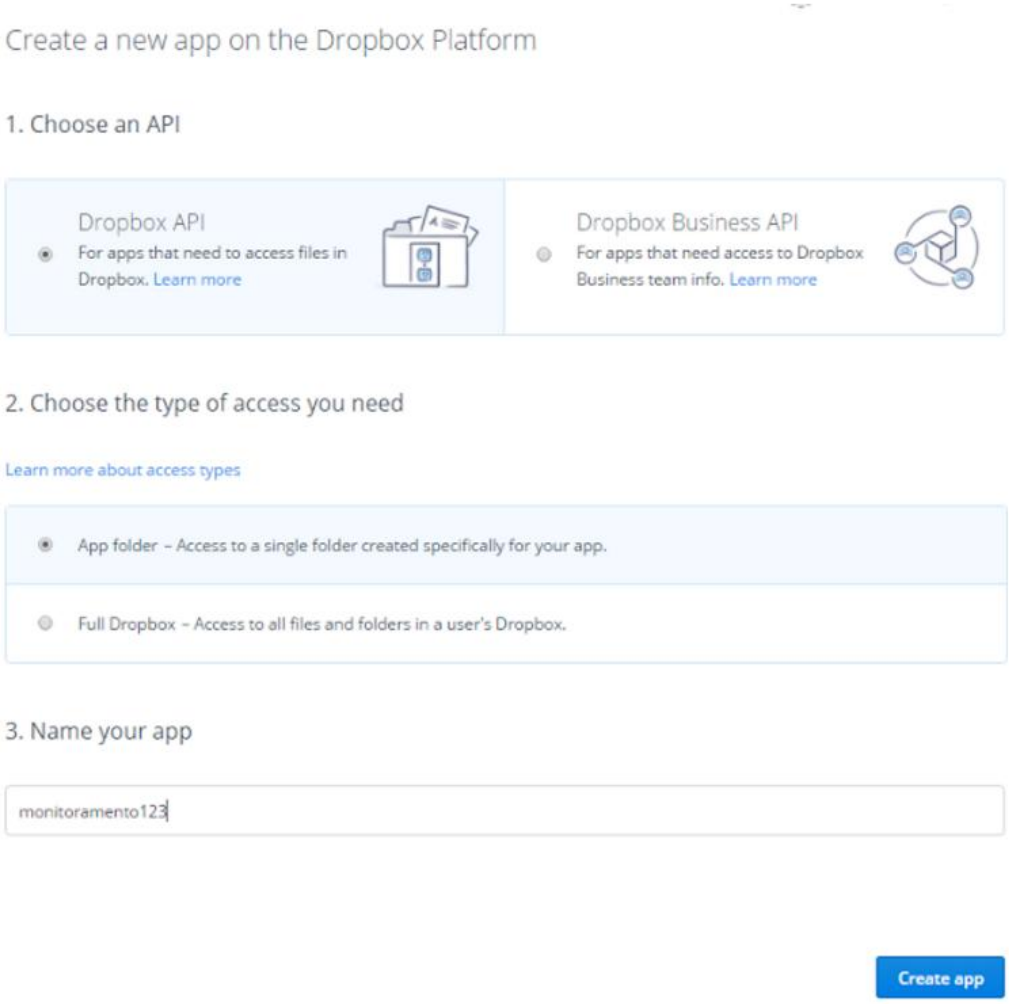

Figura B.2: Criando o API do Dropbox parte 2

Algumas informações importantes podem ser acessadas na aba de configurações do aplicativo, tais como a chave do aplicativo, a senha e o endereço de acesso remoto.

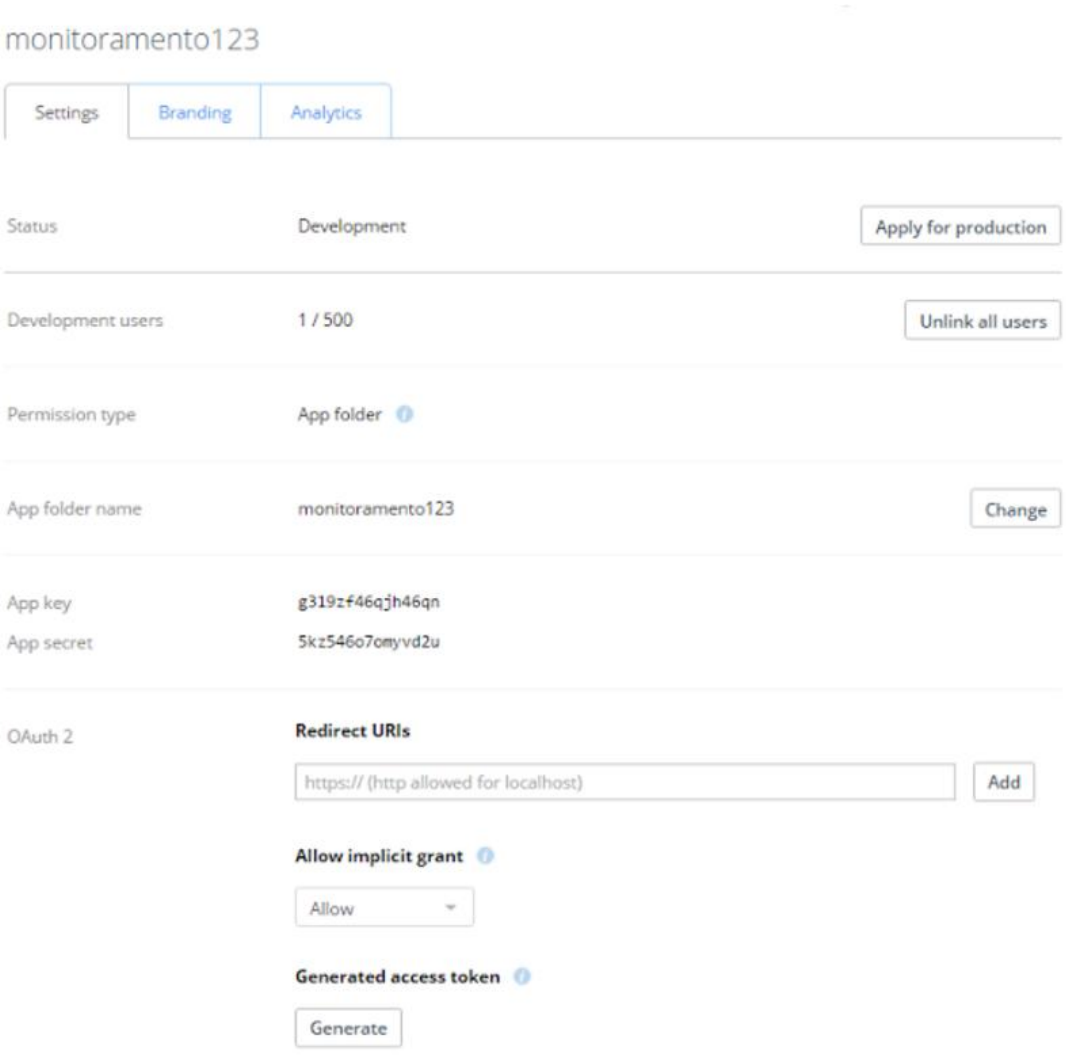

Figura B.3: Criando o API do Dropbox parte 3