**UNIVERSIDADE DO VALE DO RIO DOS SINOS - UNISINOS**

**UNIDADE ACADÊMICA DE GRADUAÇÃO**

**CURSO DE ENGENHARIA DA COMPUTAÇÃO**

**JEAN JOSÉ FLACH**

**VIGILÂNCIA RESIDENCIAL:**

**Teste de Carga em um Sistema de Vigilância Embarcado**

**São Leopoldo**

**2018**

JEAN JOSÉ FLACH

**VIGILÂNCIA RESIDENCIAL:**

**Teste de Carga em um Sistema de Vigilância Embarcado**

Trabalho de Conclusão de Curso apresentado como requisito parcial para obtenção do título de Bacharel em Engenharia da Computação, pelo Curso de Engenharia da Computação da Universidade do Vale do Rio dos Sinos - UNISINOS

Orientador: Prof Me. Lúcio Renê Prade

São Leopoldo

2018

**AGRADECIMENTOS**

Agradeço a todos que contribuíram direta ou indiretamente com a caminhada que converge neste trabalho. Agradeço ao meu orientador por todo o apoio e ajuda na realização do mesmo. Agradeço aos meus amigos que trilharem parcialmente juntos este caminho comigo. Agradeço aos meus familiares que fizeram que esta fosse possível, e agradeço a minha namorada que anda ao meu lado nesta trilha.

**RESUMO**

Este trabalho apresenta um estudo sobre o desenvolvimento de um sistema de vigilância utilizando o sistema embarcado Raspberry Pi e câmeras IP como alternativa para os gravadores tradicionais de CFTV, no qual foi desenvolvido um protótipo de um sistema autônomo na questão de iniciar e encerrar as gravações de vídeo, limitando-se a gravar apenas quando é detectado movimento. O sistema é capaz de realizar o despejo das imagens gravadas em um serviço de armazenamento online, eliminando a necessidade de descarregar o armazenamento interno e facilitando o acesso das gravações para o usuário. Para isso, foram utilizados e comparados algoritmos de detecção de movimento, quanto a sua carga computacional para o sistema embarcado e sua eficiência na detecção positiva de movimentos, com o objetivo de reduzir o volume de gravações, aumentando a autonomia off-line e reduzindo a utilização da internet para o armazenamento online. Também foram realizados testes de carga para averiguar a quantidade de câmeras que sistema é capaz de suportar simultaneamente. Dessa maneira, obteve-se um sistema com custo inferior a 350 reais mais em torno de 100 reais por câmera IP, com capacidade para suportar a partir de 2 e até 5 câmeras, variando em função da complexidade do algoritmo de detecção utilizado. Foi constatado também que a variação da complexidade altera a variação nos resultados de taxas de acertos e de falsos positivos, mas não é possível afirmar que os algoritmos de complexidade maior trouxeram resultados melhores que os de baixa complexidade. Foi, porém, constatado que a variação dos *thresholds* dos algoritmos é capaz de gerar uma grande variação dos resultados, nos quais as taxas de falsos positivos são em maioria proporcionais às taxas de acertos. O sistema é capaz de em cenários específicos resultar em um menor volume de gravações em relação a um sistema de gravação em tempo integral.

**Palavras-chave:** Vigilância. Sistema Embarcado. Raspberry Pi. Teste de Carga. Nuvem.

**LISTA DE FIGURAS**

[Figura 1 - Raspberry Pi 3 Modelo B+ 17](#_Toc529781513)

[Figura 2 - Entradas e saídas do Raspberry Pi 3 Modelo B+ 18](#_Toc529781514)

[Figura 3 - RGB e HSV 22](#_Toc529781515)

[Figura 4 - Conversão entre RGB e Y'CbCr 22](#_Toc529781516)

[Figura 5 - Resultado da aplicação de um filtro gaussiano 23](#_Toc529781517)

[Figura 6 - Antes e depois do filtro de erosão 23](#_Toc529781518)

[Figura 7 - *Frame* 1, *frame* 2 e a diferença entre os *frames* 24](#_Toc529781519)

[Figura 8 - Subtração de uma imagem 25](#_Toc529781520)

[Figura 9 - Entrada e saída de um detector de bordas 26](#_Toc529781521)

[Figura 10 - Imagem original, filtro de Sobel em X e em Y 27](#_Toc529781522)

[Figura 11 - Canny Edge Detector 27](#_Toc529781523)

[Figura 12 - Fluxo óptico 28](#_Toc529781524)

[Figura 13 - Detecção de *Blobs* 29](#_Toc529781525)

[Figura 14 - Bloco DCT e sequência zigue-zague 31](#_Toc529781526)

[Figura 15 - Comando *top* na linha de comando 33](#_Toc529781527)

[Figura 16 - Interface gráfica 34](#_Toc529781528)

[Figura 17 - Visão geral do sistema 35](#_Toc529781529)

[Figura 18 – Fluxo principal do programa 36](#_Toc529781530)

[Figura 19 - Fluxograma da *thread* das câmeras 37](#_Toc529781531)

[Figura 20 - Fluxograma da *thread* do *upload* à nuvem 38](#_Toc529781532)

[Figura 21 – Fluxograma da metodologia 40](#_Toc529781533)

[Figura 22 - *Frames* das sequências do vídeo teste 42](#_Toc529781534)

[Figura 23 - Resultado da operação de subtração de *frames* com bordas 54](#_Toc529781535)

**LISTA DE GRÁFICOS**

[Gráfico 1 - Justificativa para adquirir um sistema de segurança 12](#_Toc529781536)

[Gráfico 2 - Comparação da taxa de acerto para subtração de *frames* com erosão 55](#_Toc529781537)

[Gráfico 3 - Comparação da taxa de acerto para subtração de *frames* com detecção de *blobs* 56](#_Toc529781538)

[Gráfico 4 - Comparação da taxa de acerto para subtração de *background* com erosão 56](#_Toc529781539)

[Gráfico 5 - Comparação da taxa de acerto para subtração de *background* com detecção de *blobs* 57](#_Toc529781540)

[Gráfico 6 - Comparação dos algoritmos testados 58](#_Toc529781541)

[Gráfico 7 – Carga computacional da subtração de *frames* com erosão a cada 0,5 segundo de uma câmera 59](#_Toc529781542)

[Gráfico 8 - Carga computacional da subtração de *frames* com erosão a cada 1 segundo de uma câmera 60](#_Toc529781543)

[Gráfico 9 - Carga computacional da subtração de *frames* com erosão a cada 0,5 segundo de várias câmeras 60](#_Toc529781544)

[Gráfico 10 – Carga computacional da subtração de *frames* com erosão a cada 1 segundo de várias câmeras 61](#_Toc529781545)

[Gráfico 11 - Carga computacional da subtração de *frames* com detecção de *blobs* a cada 0,5 segundo de uma câmera 62](#_Toc529781546)

[Gráfico 12 - Carga computacional da subtração de *frames* com detecção de *blobs* a cada 1 segundo de uma câmera 62](#_Toc529781547)

[Gráfico 13 - Carga computacional da subtração de *frames* com detecção de *blobs* a cada 0,5 segundo de várias câmeras 63](#_Toc529781548)

[Gráfico 14 - Carga computacional da subtração de *frames* com detecção de *blobs* a cada 1 segundo de várias câmeras 64](#_Toc529781549)

[Gráfico 15 – Carga computacional da subtração de *background* com erosão a cada 0,5 segundo de uma câmera 65](#_Toc529781550)

[Gráfico 16 - Carga computacional da subtração de *background* com erosão a cada 1 segundo de uma câmera 65](#_Toc529781551)

[Gráfico 17 - Carga computacional da subtração de *background* com erosão a cada 0,5 segundo de várias câmeras 66](#_Toc529781552)

[Gráfico 18 - Carga computacional da subtração de *background* com erosão a cada 1 segundo de várias câmeras 66](#_Toc529781553)

[Gráfico 19 - Carga computacional da subtração de *background* com detecção de *blobs* a cada 0,5 segundo de uma câmera 67](#_Toc529781554)

[Gráfico 20 - Carga computacional da subtração de *background* com detecção de *blobs* a cada 1 segundo de uma câmera 68](#_Toc529781555)

[Gráfico 21 - Carga computacional da subtração de *background* com detecção de *blobs* a cada 0,5 segundo de várias câmeras 68](#_Toc529781556)

[Gráfico 22 - Carga computacional da subtração de *background* com detecção de *blobs* a cada 1 segundo de várias câmeras 69](#_Toc529781557)

[Gráfico 23 – Utilização da CPU durante o *upload* à nuvem 71](#_Toc529781558)

[Gráfico 24 – Uso de recursos computacionais da gravação em disco em formato MP4 e AVI 72](#_Toc529781559)

[Gráfico 25 – Uso de recursos computacionais durante a execução de detecção de movimento e gravação AVI 73](#_Toc529781560)

**LISTA DE TABELAS**

[Tabela 1 - Descrição do Vídeo de Testes 41](#_Toc529781561)

[Tabela 2 – Subtração de *frames* e erosão com matriz 5x5 46](#_Toc529781562)

[Tabela 3 – Subtração de *frames* e erosão com matriz 7x7 46](#_Toc529781563)

[Tabela 4 – Subtração de *frames* e erosão com matriz 9x9 47](#_Toc529781564)

[Tabela 5 – Subtração de *frames* e detecção de *blobs* de 20 pixels 48](#_Toc529781565)

[Tabela 6 – Subtração de *frames* e detecção de *blobs* de 40 pixels 48](#_Toc529781566)

[Tabela 7 – Subtração de *frames* e detecção de *blobs* de 80 pixels 49](#_Toc529781567)

[Tabela 8 – Subtração de *background* e erosão com matriz 3x3 50](#_Toc529781568)

[Tabela 9 – Subtração de *background* e erosão com matriz 5x5 50](#_Toc529781569)

[Tabela 10 – Subtração de *background* e erosão com matriz 7x7 51](#_Toc529781570)

[Tabela 11 – Subtração de *background* e detecção de *blobs* de 20 pixels 52](#_Toc529781571)

[Tabela 12 – Subtração de *background* e detecção de *blobs* de 40 pixels 53](#_Toc529781572)

[Tabela 13 – Subtração de *background* e detecção de *blobs* de 80 pixels 53](#_Toc529781573)

[Tabela 14 – Resumo dos testes de carga dos algoritmos 70](#_Toc529781574)

[Tabela 15 – Resumo dos testes de carga da gravação de vídeo 72](#_Toc529781575)

[Tabela 16 – Limite de câmeras para cada algoritmo considerando a gravação em disco 74](#_Toc529781576)

[Tabela 17 – Tempo de autonomia offline 75](#_Toc529781577)

[Tabela 18 – Resumo de desempenho dos algoritmos 75](#_Toc529781578)

**LISTA DE SIGLAS**

|  |  |
| --- | --- |
| CFTV | Circuito Fechado de Televisão |
| IP | *Internet Protocol* (Protocolo de Internet) |
| ARM | *Advanced RISC Machine* (Máquina RISC Avançada) |
| RAM | *Random Acess Memory* (Memória de Acesso Aleatório) |
| USB | *Universal Serial Bus* (Barramento Serial Universal) |
| GPIO | *General Purpose Input/Output* (Entrada/Saída de Propósito Geral) |
| SoC | *System-on-Chip* (Sistema em um Chip) |
| HDMI | *High-Definition Multimedia Interface* (Interface Multimídia de Alta Definição) |
| UART | *Universal asynchronous receiver/transmitter* (Receptor/Transmissor Assíncrono Universal) |
| I²C | *Inter-Integrated Circuit* (Circuito Inter-Integrado) |
| SPI | *Serial Peripheral Interface* (Interface Serial Periférica) |
| I²S | *Inter-IC Sound* (Som de Circuito Inter-Integrado) |
| EEPROM | *Electrically-Erasable Programmable Read-Only Memory* (Memória de Apenas Leitura Programável Eletricamente Apagável) |
| BLE | *Bluetooth Low Energy* (Bluetooth de Baixa Energia) |
| PC | *Personal Computer* (Computador Pessoal) |
| SD | *Secure Digital* (Digital Seguro) |
| NOOBS | *New Out Of Box Software* (Novo Software Fora da Caixa) |
| DVR | *Digital Video Recorder* (Gravador de Vídeo Digital) |
| NVR | *Network Video Recorder* (Gravador de Vídeo de Rede) |
| OEM | *Original Equipment Manufacturer* (Fabricante Original do Equipamento) |
| TCP | *Transmission Control Protocol* (Protocolo de Controle de Transmissão) |
| UDP | *User Datagram Protocol* (Protocolo de Datagrama de Usuário) |
| MPEG | *Moving Picture Experts Group* (Grupo de Especialistas em Imagens em Movimento) |
| JPEG | *Joint Photographic Experts Group* (Conjunto do Grupo de Especialistas em Fotografia) |
| RGB | *Red, Green, Blue* (Vermelho, Verde, Azul) |
| HSV | *Hue, Saturation, Value* (Matiz, Saturação e Valor) |
| HSL | *Hue, Saturattion, Lightness* (Matiz, Saturação e Brilho) |
| DCT | *Discrete Cosine Transform* (Transformada Discreta do Cosseno) |
| HTTP | *Hypertext Transfer Protocol* (Protocolo de Transferência de Hipertexto) |
| CPU | *Central Process Unit* (Unidade Central de Processamento) |
| MP4 | *MPEG-4 Part 14* (Versão 4, parte 14 do formato MPEG) |
| AVI | *Audio Video Interleave* (Intercalação de Áudio e Vídeo) |
| AVC | *Advanced Video Coding* (Codificação Avançada de Vídeo) |

**SUMÁRIO**

[1 INTRODUÇÃO 12](#_Toc529781579)

[1.1 Tema 12](#_Toc529781580)

[1.2 Delimitação do Tema 13](#_Toc529781581)

[1.3 Objetivos 13](#_Toc529781582)

[1.3.1 Objetivo Geral 13](#_Toc529781583)

[1.3.2 Objetivos Específicos 14](#_Toc529781584)

[1.4 Justificativa 14](#_Toc529781585)

[2 REVISÃO BIBLIOGRÁFICA 16](#_Toc529781586)

[2.1 Raspberry Pi 16](#_Toc529781587)

[2.1.1 Famílias 16](#_Toc529781588)

[2.1.2 Raspberry Pi 3 Modelo B+ Hardware 17](#_Toc529781589)

[2.1.3 Raspbian OS 18](#_Toc529781590)

[2.2 Câmeras IP 19](#_Toc529781591)

[2.2.1 Protocolo de Transmissão 19](#_Toc529781592)

[2.3 Detecção de Movimento 20](#_Toc529781593)

[2.3.1 Cores 21](#_Toc529781594)

[2.3.2 Processamento das Imagens 23](#_Toc529781595)

[2.3.3 Subtração de Dois *Frames* 24](#_Toc529781596)

[2.3.4 Subtração de *Background* 25](#_Toc529781597)

[2.3.5 Detecção de Bordas 26](#_Toc529781598)

[2.3.5 Fluxo Óptico 28](#_Toc529781599)

[2.3.6 Análise de *Blobs* 28](#_Toc529781600)

[2.3.7 OpenCV 29](#_Toc529781601)

[2.4 Armazenamento 30](#_Toc529781602)

[2.4.1 JPEG e MPEG 30](#_Toc529781603)

[2.4.2 Armazenamento Online 32](#_Toc529781604)

[2.5 Teste de Carga 32](#_Toc529781605)

[2.5.1 Linha de Comando 32](#_Toc529781606)

[2.5.2 Interface Gráfica 33](#_Toc529781607)

[3 Desenvolvimento do sistema de vigilÂncia 35](#_Toc529781608)

[3.1 Programa 36](#_Toc529781609)

[3.2 Detecção de Movimento 39](#_Toc529781610)

[3.2.1 OpenCV 39](#_Toc529781611)

[4 metodologia 40](#_Toc529781612)

[4.1 Algoritmos 41](#_Toc529781613)

[4.2 Capacidade do Sistema 43](#_Toc529781614)

[4.3 Autonomia 43](#_Toc529781615)

[5 Resultados obtidos 45](#_Toc529781616)

[5.1 Algoritmos e Taxas de Acerto 45](#_Toc529781617)

[5.1.1 Subtração de Dois *Frames* e Erosão 45](#_Toc529781618)

[5.1.2 Subtração de Dois *Frames* e Detecção de *Blobs* 47](#_Toc529781619)

[5.1.3 Subtração de *Background* com Erosão 49](#_Toc529781620)

[5.1.4 Subtração de *Background* com Detecção de *Blobs* 52](#_Toc529781621)

[5.1.5 Detecção de Bordas 54](#_Toc529781622)

[5.1.6 Comparação dos Resultados 55](#_Toc529781623)

[5.2 Uso de Recursos Computacionais 58](#_Toc529781624)

[5.2.1 Subtração de *Frames* com Erosão 58](#_Toc529781625)

[5.2.2 Subtração de *Frames* com Detecção de *Blobs* 61](#_Toc529781626)

[5.2.3 Subtração de *Background* com Erosão 64](#_Toc529781627)

[5.2.4 Subtração de *Background* com Detecção de *Blobs* 67](#_Toc529781628)

[5.2.5 Comparação da Carga Computacional dos Algoritmos 69](#_Toc529781629)

[5.2.6 Uso de Recursos Computacionais do *Upload* ao Armazenamento Online 70](#_Toc529781630)

[5.2.7 Uso de Recursos Computacionais do Salvamento Local das Gravações 71](#_Toc529781631)

[5.3 Autonomia Offline 74](#_Toc529781632)

[5.4 Síntese dos Resultados Obtidos 75](#_Toc529781633)

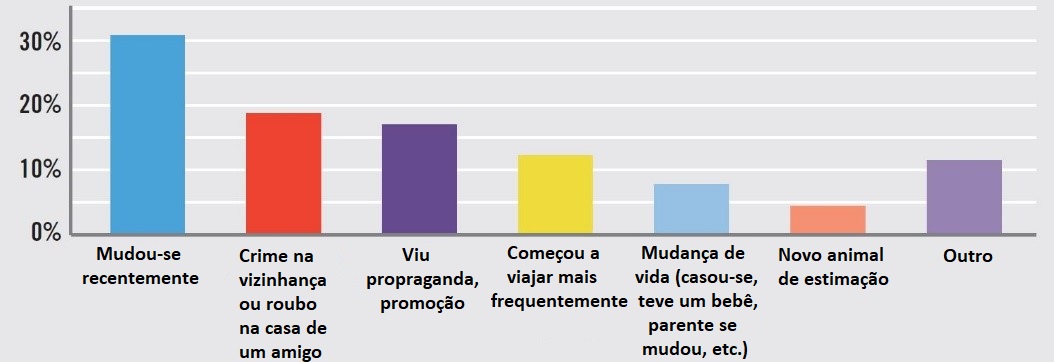
[6 Conclusões 77](#_Toc529781634)

[REFERÊNCIAS 80](#_Toc529781635)

# 1 INTRODUÇÃO

O Brasil é o terceiro país com mais roubos por habitantes da América Latina (UNDP, 2014). Segundo a polícia de São Paulo, em 2015 foram registrados uma média de 11 casos de invasão de residência por dia, no Rio de Janeiro houve crescimento de 16% nos casos de roubo, e no Distrito Federal o aumento foi de 15%. Esses casos tendem a se agravar em períodos de festas e feriados prolongados durante o ano, mas há disponível no mercado uma gama de produtos e serviços que podem mitigar tal risco, como alarmes monitorados e serviços de vigilância (CUNHA, 2016). O gráfico 1 ilustra os principais motivos apontados por consumidores da razão pela qual adquiriram um sistema de segurança recentemente.

Gráfico 1 - Justificativa para adquirir um sistema de segurança



Fonte: Karyn Hodgson (2018)

É possível notar a partir desses índices que o tema de segurança residencial vem ganhando destaque principalmente se em conjunto com sistemas confiáveis de baixo custo, como apresentado no tópico a seguir.

## 1.1 Tema

Sistemas de vigilância de vídeo são muito difundidos, o CFTV é amplamente utilizado em ambientes comerciais e industriais, e sua utilização vem crescendo no ambiente residencial. Há expectativa de crescimento na área de segurança eletrônica de 8% para 2018 no Brasil (MIGLIORI, 2018).

Novas tecnologias mais acessíveis se tornam úteis para a implementação de novos sistemas de vigilância de menor custo e maior facilidade de instalação e gerenciamento, atraindo novos consumidores para esse mercado, devido também aos crescentes números de roubos no país.

O tema deste trabalho compreende o desenvolvimento de um sistema de vigilância residencial utilizando um sistema embarcado de baixo custo como central de processamento das imagens, capaz de gravar apenas quando é detectado movimento e armazenar as gravações na nuvem. Portanto, foi necessário determinar quantas câmeras um sistema embarcado de baixo custo pode suportar, e qual o melhor algoritmo para a detecção de movimento, considerando necessário a utilização desses para a redução das gravações de vídeo, diminuindo a gravação em disco e a banda de internet para o salvamento online.

## 1.2 Delimitação do Tema

Delimita-se a esse trabalho o desenvolvimento de um sistema de vigilância utilizando um sistema embarcado conectado a uma rede Wi-Fi doméstica, desconsiderando as vulnerabilidades de segurança da mesma e das câmeras IP encontradas no mercado. Esse trabalho limita-se também ao desenvolvimento e comparação de alguns algoritmos consolidados da área de visão computacional, com diferentes configurações de *thresholds*, nas suas características de taxas de acertos e falsos positivos de um vídeo simulando algumas situações de um único cenário de um ambiente externo, produzido com uma única câmera, e também à comparação da utilização de recursos computacionais de uso do processador e da memória RAM utilizando a mesma câmera e *streams* de vídeo duplicados da mesma. Limita-se também à comparação entre dois tipos de compressão para gravação de vídeos em disco quanto ao seu uso computacional e espaço final das gravações.

## 1.3 Objetivos

Para a realização do trabalho foram determinados os seguintes objetivos a seguir.

### 1.3.1 Objetivo Geral

Desenvolver um protótipo de um sistema de vigilância residencial capaz de iniciar e encerrar as gravações automaticamente usando algoritmos de detecção de movimento, também para otimizar o volume de gravações, e capaz de autonomamente descarregar as gravações para um serviço de armazenamento online. Por fim, realizar um teste de carga para verificar quantas câmeras é possível acoplar ao sistema simultaneamente.

### 1.3.2 Objetivos Específicos

Visando realizar o objetivo geral, o mesmo divide-se nos seguintes objetivos específicos:

1. Comparar algoritmos de detecção de movimento quanto ao seu custo computacional e sua eficiência em um sistema embarcado, visando reduzir o volume das gravações;
2. Realizar o salvamento das gravações localmente de forma temporária até o *upload* ao armazenamento na nuvem, disponibilizando para o usuário remotamente;
3. Fazer o teste de carga no sistema para dimensionar quantas câmeras o sistema pode suportar;
4. Determinar a autonomia de tempo de gravações locais para dois tipos de codificação para compressão das gravações;
5. Desenvolver o programa que rode no sistema embarcado utilizando threads para processar múltiplas câmeras, na linguagem C.

## 1.4 Justificativa

Os sistemas de vigilância mais populares atualmente baseiam-se em armazenar localmente as imagens, havendo então um tempo limite de gravações antes de ser necessário esvaziar manualmente o dispositivo de armazenamento físico, já que o princípio de funcionamento deles é se manter gravando constantemente. Para evitar isso, propõe-se esvaziar automaticamente o armazenamento físico fazendo o *upload* das gravações para um armazenamento na nuvem, através de uma conexão com a internet, sendo então possível gerenciar as gravações de qualquer computador conectado à internet, reduzindo o risco de perda de todo o período de gravações, seja por possíveis danos elétricos ou por furto do equipamento. Dessa maneira, reduz-se também o risco de parar de gravar por falta da manutenção para esvaziar o armazenamento local. Portanto, o tópico a seguir irá explorar as tecnologias envolvidas para o desenvolvimento do sistema de vigilância proposto.

# 2 REVISÃO BIBLIOGRÁFICA

O elemento central do sistema de vigilância proposto é a central de processamento dos dados, para a qual foi utilizado o sistema embarcado Raspberry Pi estudado abaixo.

## 2.1 Raspberry Pi

Segundo a descrição do próprio site, é um computador de baixo custo, do tamanho de um cartão de crédito, que pode ser usado com um monitor ou TV, com um mouse e um teclado normal, capaz de diversas aplicações computacionais. A Raspberry Pi Foundation é uma instituição de caridade do Reino Unido que desenvolve e vende o Raspberry Pi com a intenção de prover computadores de alta performance e baixo custo para permitir que todas as pessoas possam aprender e interagir com ferramentas computacionais com o objetivo de resolver problemas (RASPBERRY, 2018).

### 2.1.1 Famílias

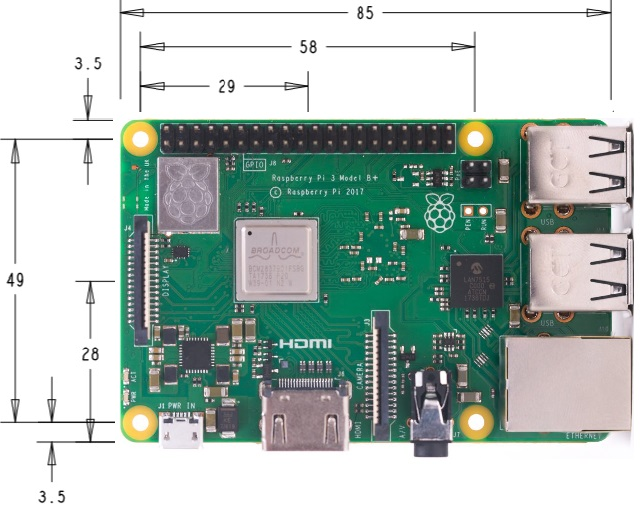
Existem três famílias do Raspberry Pi:

* Raspberry Pi Modelo B: é a família principal, sua primeira revisão foi lançada em março de 2012, possuindo um processador ARM de 700 MHz e 512 MB de memória RAM. A última revisão foi lançada em março de 2018, e possui um processador de 1.4 GHz de quatro núcleos, 1 GB de memória RAM, módulo Wi-Fi integrado e quatro portas USB. É vendido por 35 dólares.
* Raspberry Pi Modelo A: é uma versão reduzida do Modelo B, com um processador menos veloz e apenas uma porta USB para conectividade. Sua última revisão lançada em novembro de 2014 possui 512 MB de memória RAM. É vendido por 20 dólares.
* Raspberry Pi Modelo Zero: A versão com proposta de baixíssimo custo, possui duas revisões no mercado, uma com módulo Wi-Fi e Bluetooth integrado, e outra sem. Ambas possuem um processador de 1 GHz e 512 MB de memória RAM. É vendido por 5 dólares a versão sem Wi-Fi, e 10 dólares a versão com Wi-Fi.
* Raspberry Pi Modelo *Compute*: Versão mais voltada para engenharia, possui processador de 1.4 GHz com quatro núcleos, 1 GB de memória RAM, opção com 4 GB de memória de armazenamento integrado e 46 GPIO’s.

### 2.1.2 Raspberry Pi 3 Modelo B+ Hardware

É a versão mais recente do modelo mais poderoso do Raspberry Pi, o qual foi utilizado para o desenvolvimento do protótipo. É um *SoC* com 4 núcleos de arquitetura ARMv8-A (64/32-bit) com processador ARM Cortex-A53 de 1.4 GHz e 1 GB de memória RAM. Possui 4 portas USB, porta Ethernet 10/100/1000 Mbit/s, Wi-Fi integrado 802.11ac de 2.4 e 5 GHz, Bluetooth 4.2 BLE, saída HDMI para vídeo e áudio, 40 pinos de GPIO’s, distribuídos entre 17 GPIO’s, UART, I²C, SPI, I²S, EEPROM e entrada para cartão MicroSD para armazenamento. Sua alimentação é dada por 5V @ 2.5A, tendo consumo de 450mA até 1.2A. Suas dimensões são de 85.6 mm de comprimento, 56.5 mm de largura e 17 mm de profundidade, pesando 45 gramas (RASPBERRY, 2018). A figura 1 abaixo é a imagem superior do Raspberry Pi 3 Modelo B+, salientando as dimensões descritas.

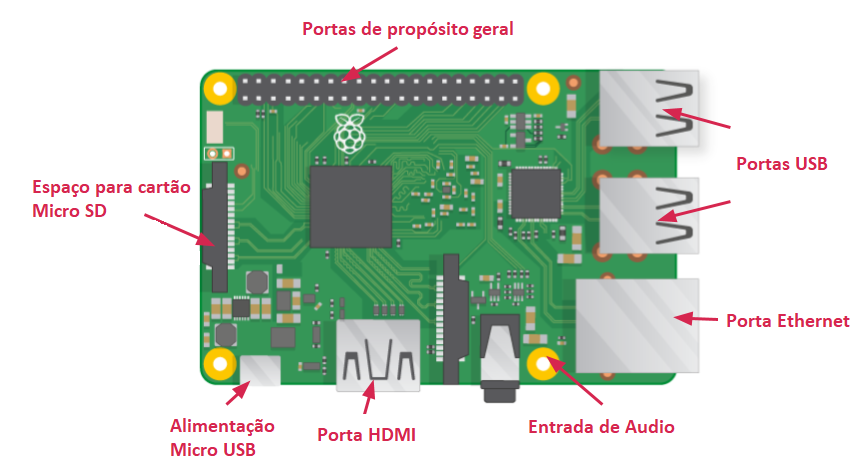
Figura - Raspberry Pi 3 Modelo B+



Fonte: Raspberry Pi Site (2018)

A figura 2 mostra a distribuição das entradas e saídas do Raspberry Pi 3 Modelo B+.

Figura - Entradas e saídas do Raspberry Pi 3 Modelo B+



Fonte: Raspberry Pi Site (2018)

### 2.1.3 Raspbian OS

Existem diversas opções de sistemas operacionais compatíveis com o Raspberry Pi, sendo a grande maioria baseados nas distribuições do Linux. Porém, a Raspberry Pi Foundation tem o Raspbian OS como seu sistema operacional oficial para todos os modelos do Raspberry Pi (RASPBERRY, 2018), e, portanto, esse foi o sistema operacional utilizado para o desenvolvimento do protótipo.

O Raspbian é uma distribuição do Linux baseado no Debian. Atualmente sua versão mais recente é o Raspbian Stretch, versão de *kernel* 4.14, e é disponibilizado de forma gratuita para download no próprio site da Raspberry Pi Foundation. O Raspbian também possui uma versão compatível com PC e Mac disponível para download, permitindo instalar e testar em um computador convencional (RASPBERRY, 2018).

O sistema operacional é instalado diretamente no cartão MicroSD, pois o Raspberry Pi utilizado não possui armazenamento interno, e, portanto, pode ser instalado a partir de um computador no cartão de memória. O processo consiste em formatar o cartão SD que será utilizado, baixar e extrair o instalador fácil do Raspbian, o NOOBS, diretamente no cartão SD. Em seguida, inserir no Raspberry Pi, ligar a placa na alimentação e seguir as instruções (RASPBERRY, 2018).

Por não possuir armazenamento interno e se utilizar um cartão Micro SD, se a velocidade de escrita e leitura do cartão não for rápida que chega, ele pode se tornar um gargalo do sistema, portanto deve se dar preferência para cartões de alta velocidade. O tamanho mínimo recomendado para a instalação do sistema operacional é de 4 GB, portanto a Raspberry Pi Foundation recomenda um cartão de 8 GB de tamanho e classe 6 (6 MB/s de escrita) (RASPBERRY, 2018).

O próximo tópico irá estudar a diferença entre uma câmera convencional e uma câmera IP e seu funcionamento.

## 2.2 Câmeras IP

Enquanto as câmeras convencionais funcionam transmitindo analogicamente o sinal de vídeo através de cabos coaxiais até o dispositivo de processamento central, como um DVR, as câmeras IP funcionam transmitindo um *stream* de dados dentro de uma rede para um computador ou um NVR (ALMEIDA, 2016).

A conexão da câmera com a rede pode ser feita via cabo de rede ou via Wi-Fi. O processo de configuração da câmera varia de modelo a modelo. Existem centenas de marcas diferentes de câmeras IP no mercado devido ao fato do modelo de negócios *White-Label* ser predominante nessa indústria, onde uma empresa fabrica um produto e o vende para empresas terceiras que então customizam esse produto e o vendem como de sua marca. No caso das câmeras IP’s, grande parte das OEM’s são chinesas, e as empresas que compram suas câmeras adicionam software customizado. Então, apesar de serem vendidas como câmeras diferentes, elas compartilham o mesmo princípio de funcionamento interno (PIERRE, 2017). Quanto a maneira em que as câmeras transmitem os dados, o tópico abaixo aborda esse assunto.

### 2.2.1 Protocolo de Transmissão

Existem dois protocolos de transmissão utilizados no *stream* de vídeo das câmeras IP, o protocolo TCP em que não há perdas, e o protocolo UDP que dá prioridade para o desempenho (UNIFORE, 2017).

O protocolo TCP utiliza uma banda maior para transmissão de dados auxiliares que são utilizados para conferir a integridade dos dados que estão sendo transmitidos e recebidos, a fim de garantir que todos os pacotes são recebidos e tratados (UNIFORE, 2017). No contexto de *stream* de vídeo das câmeras, isso significa que todos os *frames* gerados pela câmera serão recebidos e processados pela central de processamento, a um custo de uma maior banda de tráfego de dados. Como a rede utilizada é uma rede local em um ambiente residencial onde não é exigido uma grande carga, não será um gargalo desde que a resolução das imagens da câmera não seja alta.

O protocolo UDP não utiliza os mesmos mecanismos do protocolo TCP para garantir a recepção de todos os *frames*, o que implica um pacote de dados mais enxuto, porém com menor garantia de que será processado (UNIFORE, 2017), sendo uma melhor opção para câmeras de maior resolução. No contexto de processamento de imagens e gravação, isso pode resultar em alguns *frames* danificados, podendo denegrir a qualidade das imagens, prejudicando o processo de detecção de movimento.

A partir das imagens da câmera, é necessário determinar o momento de iniciar a gravação das imagens das câmeras. O tópico a seguir abordará todo o processo envolvido na determinação de movimento.

## 2.3 Detecção de Movimento

A detecção do movimento é responsável por identificar uma mudança brusca no fluxo das imagens, indicando um evento de interesse, na qual será realizado o salvamento das imagens. Existem vários algoritmos de detecção de movimento, variando de baixa complexidade até de alta complexidade. Quanto maior a complexidade, espera-se menor taxa de erros, porém maior a carga computacional para serem executados.

No contexto de vigilância residencial, uma detecção de movimento em falso não irá resultar em alguma consequência grave, apenas o salvamento de um vídeo onde nada interessante acontece, consumindo também carga computacional e banda de dados para o armazenamento online dos vídeos. Algo não desejado, porém a não identificação de um movimento de interesse pode acarretar na perda importante de dados, sendo então em alguns casos melhor ocorrer um falso positivo do que não identificar um movimento de interesse no vídeo.

Os cenários mais comuns de falso positivos em vigilância residencial são a movimentação de objetos dinâmicos, como arvores se movendo devido ao vento, ou a mudança de iluminação causado pelas nuvens.

Existem várias aproximações para detecção de movimento em um *stream* contínuo de dados. Todos se baseiam na comparação do *frame* atual com um anterior ou com algo chamado de *background* (KIRILLOV, 2007). O primeiro passo para compreender como funciona um algoritmo de detecção de movimento é entender como as imagens são armazenadas e manipuladas digitalmente, iniciando pelos sistemas de cores utilizados em computadores, assunto abordado pelo tópico a seguir.

### 2.3.1 Cores

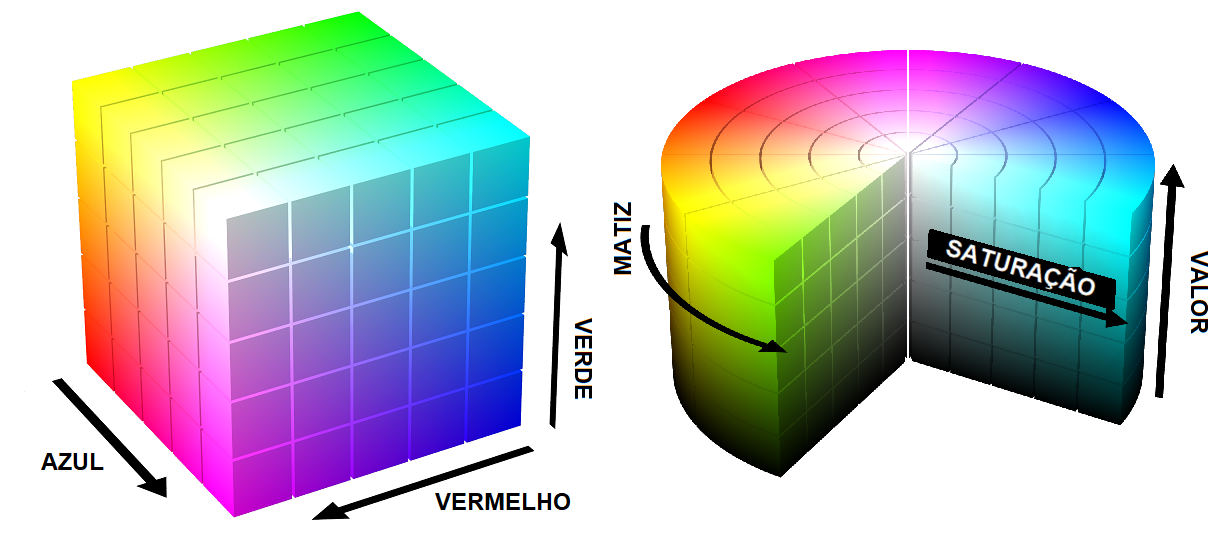
O fluxo de dados que uma câmera envia pode ser em vários formatos, como o MPEG e o JPEG, os quais utilizam o sistema de cores Y’CbCr (MARJANOVIC, 2003). Existem diversos sistemas de cores, como RGB, o HSV e um amplamente utilizado nos algoritmos de detecção, a escala de cinza*.*

O RGB é o mais simples, no qual a cor é representada pela soma de porções das três cores primárias: vermelho, verde e azul. No sistema RGB de 8 bits, cada cor varia de 0 até 255, sendo os três no valor 0 para preto e os três no valor 255 para branco (MARJANOVIC, 2003). Pode ser representado como um cubo, e a distância de uma cor para outra é um vetor de um ponto a outro dentro do cubo, no qual o módulo representa o quão longe uma cor está da outra.

O HSV usa um cilindro de cores, onde o ângulo do círculo representa a tonalidade, a distância para o centro a saturação e a altura o brilho. Dessa maneira, o centro do cilindro é uma escala de cinza. Este esquema de cores é interessante para a utilização em algoritmos de detecção pois apenas a tonalidade de cor ou o valor do brilho podem ser usados para comparação de imagens, sendo menor o custo computacional usando apenas um canal já existente.

A figura 3 mostra uma representação visual para comparação entre esses dois sistemas.

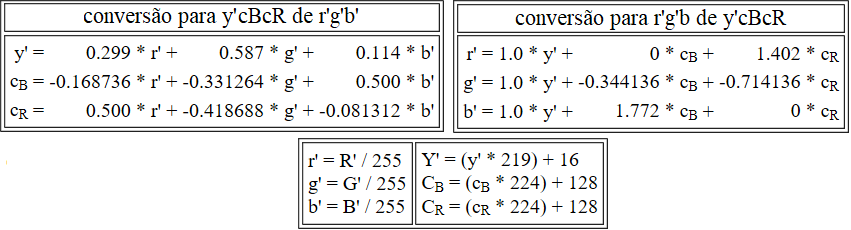
Figura - RGB e HSV



Fonte: Isometricland (2017)

O YCbCr é o esquema de cores padrão para processamento digital de vídeo (ITU-R, 2011). O *Y’* é referente ao brilho percebido do pixel, o qual é independente da cor, e toda a informação da tonalidade da cor está presente no Cb e no Cr, que são chamados de canais de diferença de cores. O motivo de utilização desse sistema se deve a que o olho humano é menos sensível a mudança de cor do que de brilho, podendo ser amostrado numa resolução menor sem perda de qualidade de imagem (MARJANOVIC, 2003). A figura 4 especifica como realizar a conversão entre o sistema RGB e Y’CbCr.

Figura - Conversão entre RGB e Y'CbCr



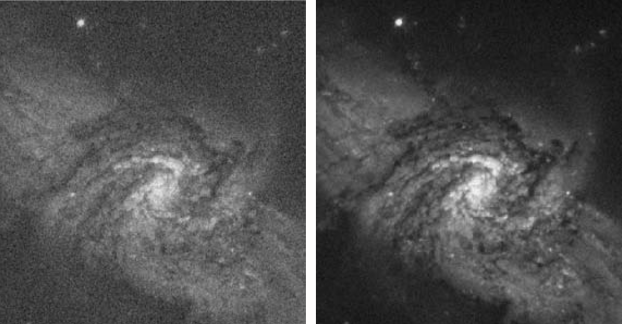
Fonte: Marjanovic (2003)

A escala de cinza, como o nome sugere, é um sistema onde há apenas um valor para cada pixel. Num sistema de 8 bits, o 0 representa o preto, e o 255 o branco (GONZALEZ, WOODS, 2007). É muito utilizado em algoritmos de detecção de movimento por ser apenas um valor a ser comparado com os outros pixels. O tópico a seguir irá estudar a manipulação dessas imagens digitais para realçar ou mudar certas características dessas imagens digitais, que é comumente a primeira etapa nos algoritmos de detecção de movimento.

### 2.3.2 Processamento das Imagens

Devido a captação das imagens ser feita através da luz incidente no sensor da câmera depois de refletido dos objetos da cena, é comum haver ruído nas imagens geradas. Por isso, para suavizar as imagens e reduzir esse ruído, é comum realizar o pré-processamento delas com a utilização de filtros. Existem diversos filtros, como o filtro gaussiano, utilizado para borrar a imagem, criando um efeito de imagem embaçada. O filtro gaussiano é capaz de remover ruído gaussiano das imagens (GONZALEZ, WOODS, 2007), como a figura 5 demonstra.

Figura - Resultado da aplicação de um filtro gaussiano



Fonte: Gonzalez, Woods (2007)

Na figura 5, a imagem da esquerda possui ruído gaussiano, e após aplicado o filtro gaussiano, obtém-se a imagem da direita. Outros filtros podem ser utilizados, como utilizando a operação de erosão como filtro, usado em imagens binárias ou de escala de cinza, que é capaz de remover pequenos grupos de pixels sem massa significativa (LEVIN, 2018). Este filtro é útil para descartar pequenas variações nas imagens após a operação de subtração de dois *frames*. A figura 6 demonstra o resultado da aplicação desse filtro.

Figura - Antes e depois do filtro de erosão



Fonte: Levin (2018)

A figura 6 mostra na esquerda uma imagem com pequenos pixels brancos que não representam nenhum objeto significativo, e na direita o resultado da aplicação do filtro de erosão, removendo a grande maioria desses pixels.

A técnica de erosão utiliza um elemento estruturante como termo para realizar a operação, que basicamente é uma matriz usada para verificar pixel a pixel a imagem, e enquanto os pixels vizinhos não forem totalmente preenchidos pela matriz do elemento estruturante, estes são descartados (FISHER, 2003).

Após a etapa de processamento em um algoritmo de detecção de movimento, chega o momento de realizar alguma operação com as imagens a fim de realçar certos objetos ou diferenças entre as imagens. A seguir serão estudados algumas dessas operações.

### 2.3.3 Subtração de Dois *Frames*

A subtração de imagens é frequentemente aplicada para destacar a diferença entre duas imagens. A técnica de subtração de dois *frames* consecutivos consiste na subtração pixel a pixel de dois *frames* num intervalo de tempo pequeno para realçar a diferença entre eles (GONZALEZ, WOODS, 2007), como demonstrado na figura 7.

Figura - *Frame* 1, *frame* 2 e a diferença entre os *frames*



Fonte: Castro, Silva, Pazoti e Artero (2015)

Na figura 7, a imagem da direita representa a diferença da subtração pixel a pixel da imagem da esquerda e do centro. Nesta técnica, quanto maior o intervalo de tempo entre os dois frames subtraídos, maior será a diferença entre eles, considerando um objeto se movimentando com direção fixa. No tópico a seguir, a subtração de *background* é descrita.

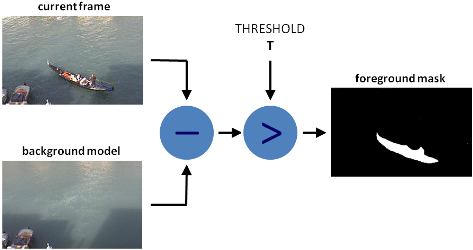
### 2.3.4 Subtração de *Background*

Similar a subtração de dois *frames*, a subtração de *background* utiliza os conceitos de *background* e *foreground* para identificar a entrada ou saída de objetos da cena (SOBRAL, VACAVANT, 2014).

O *background* é um *frame* em que não há objetos de interesse ou objetos se movendo, apenas o plano de fundo da imagem, portanto pode ser usada como referência para identificar se há objetos estranhos em outros *frames*. O problema se encontra em determinar automaticamente qual o *frame* se encontra apenas o *background*, pois este pode se modificar ao longo do tempo, devido a mudança de luminosidade ou entrada e saída de objetos estáticos na cena (SOBRAL, VACAVANT, 2014).

O *foreground* é o resultado de um *frame* atual subtraído o *background*, resultando apenas nos objetos móveis. A subtração é realizada pixel a pixel, e se o resultado for menor que um determinado limite, o resultado é um pixel preto, e se for maior que o limite, é um pixel branco. A figura 8 a seguir ilustra o processo de subtração de *background*.

Figura - Subtração de uma imagem



Fonte: OpenCV (2018)

Nota-se na figura 8 que a diferença entre o *frame* atual e o *background* é destacado através da operação de subtração. No tópico a seguir, outro método de identificação de objetos é apresentado: o de detecção de bordas.

### 2.3.5 Detecção de Bordas

A detecção de bordas (*Edge Detection*) se propõe a identificar mudanças abruptas de cores nas imagens, as quais podem ser interpretadas como os contornos dos objetos dentro da imagem. Essas informações são importantes porque a partir delas podemos dizer onde se encontram os objetos, além do seu tamanho e formato (PHILLIPS, 2000).

Numa escala de cinza, uma borda se encontra onde há uma mudança de um tom escuro para um tom claro, ou vice-versa. A borda se encontra no centro dessa mudança (PHILLIPS, 2000). O retorno de um detector de borda é uma imagem em escala de cinza onde as bordas são pontos brancos, e o restante da imagem é totalmente preta, como mostra a figura 9, onde a imagem original se encontra na esquerda, e a imagem na direita é o resultado da operação de detecção de bordas.

Figura - Entrada e saída de um detector de bordas



Fonte: MATLAB (2013)

Para verificar a mudança de tons em todas as direções, é utilizado uma técnica de convolução de imagens com máscaras, onde é pego um ponto da imagem e multiplicado por uma matriz de números ponto a ponto e colocado o resultado no centro da imagem (PHILLIPS, 2000). Semelhante a como é feito o filtro gaussiano, existem diversas máscaras diferentes com resultados diferentes.

Um algoritmo popular que utiliza a detecção de bordas é o *Canny Edge Detector*. Desenvolvido por John F. Canny em 1986, é um algoritmo de vários estágios, no qual a imagem é primeiro filtrada para reduzir os ruídos, seguido pela aplicação do filtro de Sobel. O filtro de Sobel utiliza duas máscaras para a detecção de gradientes, uma para a orientação vertical e outra para a horizontal (GREEN, 2002). A figura 10 mostra a imagem original (à esquerda) ao lado do resultado do filtro em X (ao centro) e Y (à direita).

Figura - Imagem original, filtro de Sobel em X e em Y



Fonte: OpenCV (2014)

Com os gradientes em X e Y, a intensidade das bordas em um determinado pixel é dada pela raiz dos quadrados de X e Y, e os ângulos das bordas podem ser encontradas através do inverso da tangente da divisão entre os gradientes X e Y. Com os ângulos, em seguida é utilizado um algoritmo de supressão dos valores não-máximos para remover bordas duplicadas, eliminando traços de bordas duplicados e ficando apenas com traços finos para as bordas. Para realizar isso, são verificados os pixels vizinhos a um determinado pixel, e se o mesmo possui mesma direção que o primeiro, o com maior intensidade de gradiente é transformado num pixel branco, e o menor num pixel preto. O passo seguinte consiste em verificar quais são realmente bordas, através de comparação com valores limiares para borda e possível borda. Por fim, são verificados os possíveis pixels de borda, verificando os pixels conectados a eles, sendo marcados como verdadeiras bordas casos estejam conectados em pixel de bordas com gradientes fortes (GREEN, 2002). A figura 11 demonstra a aplicação desse algoritmo.

Figura - Canny Edge Detector



Fonte: Green (2002)

Na figura 11, a imagem do meio apresenta bordas grossas antes da remoção de bordas duplicadas, com o resultado final na imagem da direita. Outro método para detecção de movimentos em um fluxo de imagens é a utilização do fluxo óptico, descrito no tópico a seguir.

### 2.3.5 Fluxo Óptico

Fluxo óptico é o reconhecimento do padrão de que há pixels em movimento entre dois *frames* consecutivos, causado pela movimentação do objeto ou da câmera. Esta movimentação pode ser traduzida em vetores, e funciona se apoiando nas premissas de que as intensidades dos pixels em *frames* consecutivos não mudam e de que os pixels vizinhos realizam um movimento similar (OPENCV, 2017). A figura 12 mostra o resultado inserindo pontos persistentes na imagem baseado no fluxo dos veículos.

Figura - Fluxo óptico



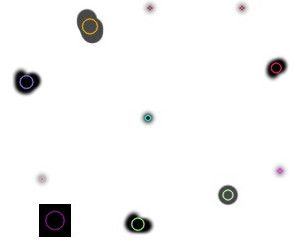
Fonte: OpenCV (2017)

Apesar de ser possível monitorar a movimentação de objetos na cena, este método se torna oneroso devido ao fato de ser matematicamente complexo para ser executado, além de que para detecção de movimentos não é necessário acompanhar o movimento dos objetos. Portanto, descarta-se a utilização deste método. Outro método de análise de imagens é a análise de *blobs*, descrito no tópico abaixo.

### 2.3.6 Análise de *Blobs*

Um *Blob* é um grupo de pixels conectados em uma imagem que compartilha uma propriedade em comum (MALLICK, 2015). A análise de uma imagem a procura de *blobs* é útil após alguma operação, como por exemplo a de subtração de imagens, para procurar se dentro do resultado existe algum conjunto de pixels que representem algum objeto maior. A figura 13 abaixo mostra uma imagem na qual os *blobs* foram detectados e realçados com círculos.

Figura - Detecção de *Blobs*



Fonte: Mallick (2015)

Os *blobs* podem ser filtrados por tamanho, descartando assim qualquer agrupamento de pixels pequenos que não representam movimentos significativos. Para a utilização de todos esses algoritmos de operação com imagens é utilizado uma biblioteca, esta descrita no tópico a seguir.

### 2.3.7 OpenCV

O *Open Source Computer Vision Library* é uma biblioteca *open-source* que inclui centenas de algoritmos de visão computacional, disponíveis através de uma API C++. Dentre suas funcionalidades estão a capacidade de analisar vídeo, incluindo detecção de movimento, subtração de *background* e algoritmos de rastreamento de objetos (OpenCV, 2018).

Certos algoritmos permitem customização dos *threshold*, que são os valores utilizados como padrão para determinar se o resultado de uma operação chegou a um ponto onde uma decisão deve ser feita, como por exemplo determinar se foi um movimento ou não. Mudando esses valores, é possível tornar o algoritmo mais ou menos sensível. Para a vigilância residencial, estes valores devem idealmente fazer com que a detecção de movimento nunca retorne negativo quando de fato há um movimento de interesse nas imagens das câmeras, porém um falso positivo é aceitável devido ao fato que irá resultar apenas em uma gravação armazenada localmente. Apesar disso, não deverá ser muito sensível ao ponto de fazer com que o sistema permaneça constantemente gravando, não sendo possível descarregar as gravações para a nuvem na mesma velocidade, ou esgotando o armazenamento local muito rapidamente em caso de indisponibilidade da internet. O gerenciamento do armazenamento local e online será estudado no tópico a seguir.

## 2.4 Armazenamento

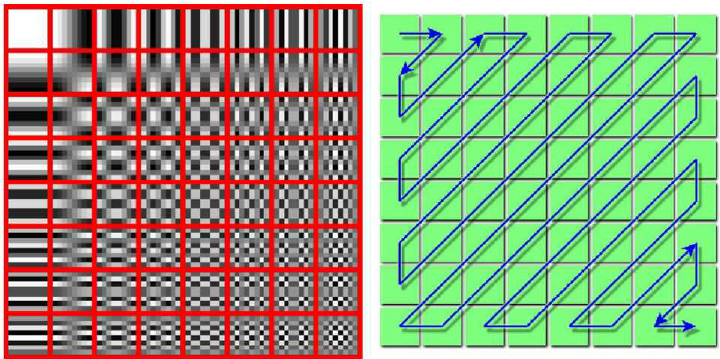
O formato dos dados de imagem que são recebidos, processados ou armazenados são importantes. O *stream* de vídeo enviado por câmeras digitais tende a ser no formato MPEG, que é uma coletânea de imagens em formato JPEG, como estudado no tópico abaixo.

### 2.4.1 JPEG e MPEG

O JPEG foi criado pela Joint Photografic Experts Group, o grupo que dá nome ao formato, e foi lançado em 1992. É um formato de compreensão de dados no qual há perda de informações, muito usado em imagens fotográficas geradas digitalmente. O grau de compressão pode ser ajustado, permitindo trocar qualidade de imagem por tamanho do arquivo. O formato JPEG pode atingir uma compressão de até 10 para 1 sem perda perceptível de qualidade (HAINES, CHUANG, 1992), utilizando um formato de compressão baseado na transformada discreta de cossenos, a DCT.

O objetivo da transformada discreta de cossenos na compressão do JPEG é representar um bloco bidimensional de tamanho 8x8 utilizando uma soma de 64 cossenos, representando qualquer imagem desse tamanho sem perda perceptível de qualidade. O que se percebe em fotografia digital é que enquanto os cossenos de menor frequência representam a maior parte da imagem, cossenos de maior frequência tem menor influência sobre a imagem final (RABIE, KAMEL, 2015). Após aplicada a DCT, a matriz resultante pode ser dividida por diferentes matrizes de quantização com diferentes valores de perda de informação, e seus valores arredondados, dessa maneira os cossenos de maior frequência são zerados. A figura 14 ajuda a explicar esse processo.

Figura - Bloco DCT e sequência zigue-zague



Fonte: Rabie, Kamel (2015)

O passo seguinte é ordenar os valores na sequência de zigue-zague mostrado, como mostrado na figura 14, fazendo com que os valores das altas frequências zeradas estejam todos agrupados no final, facilitando ainda mais a compressão dos dados, como por exemplo utilizando a codificação de Huffman, a qual organiza os dados mais frequentes em grupos de bits menores não repetíveis (RABIE, KARMEL, 2015).

Como o JPEG utiliza o esquema de cores YCbCr, esta operação é realizada nos três canais, sendo que nos canais de crominância podem ser utilizados matrizes de quantização com maior perda sem que haja perda de qualidade aparente, pois o olho humano é mais sensível a luminosidade do que as cores.

O MJPEG é a compressão de cada *frame* em formato JPEG, amplamente utilizado em sistemas de captura de vídeo. Quando utilizado para *stream* de vídeo, cada *frame* é encapsulado individualmente no formato de uma resposta HTTP. Em resposta para uma requisição de ‘GET’ de um *stream* MJPEG, uma conexão TCP não é encerrada enquanto ambos o cliente e o servidor queiram receber e enviar dados, respectivamente. A diferença entre o MJPEG e o MPEG é que o MJPEG trata da compressão individual em cada *frame*, enquanto o MPEG faz a compressão dos *frames* coletivamente.

Com o formato dos dados definido, o tópico abaixo irá abordar a etapa de *upload* dos dados para a nuvem.

### 2.4.2 Armazenamento Online

Serviços de armazenamento online permitem que seja possível salvar arquivos na nuvem e liberar espaço no armazenamento local. Atualmente, o serviço de armazenamento online da Google, o Google Drive, permite o armazenamento de imagens e vídeos ilimitado, segundo seu próprio site, caso seja aceito comprimir para um tamanho menor, ainda mantendo alta qualidade (GOOGLE, 2018).

Segundo o seu site, a API de desenvolvimento do Google Drive não possui suporte para as linguagens C ou C++ (GOOGLE, 2018). Existe, porém, um programa de linha de comando compatível com Linux e diversos serviços de armazenamento, o Rclone, permitindo que seja feito um código em qualquer linguagem que interaja com a linha de comando e realize o *upload* dos arquivos. Pode ser instalado e configurado diretamente da linha de comando (RCLONE, 2018). A etapa final então é testar o sistema para determinar a sua capacidade em termos de câmeras paralelas, tópico abordado abaixo.

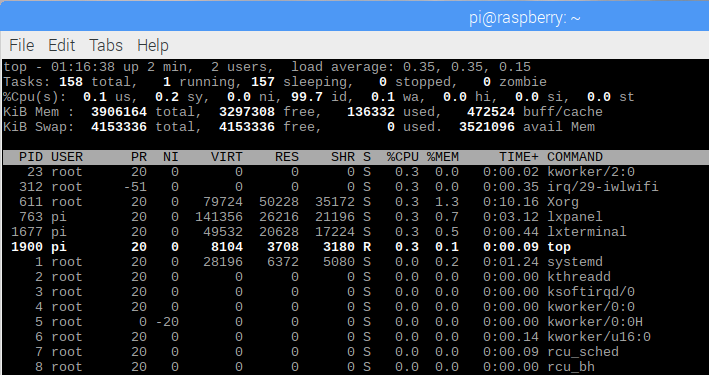
## 2.5 Teste de Carga

O teste de carga é importante para determinar quantas câmeras o sistema poderá suportar. O Raspbian OS possui duas ferramentas nativas para verificarmos o uso geral da CPU e memória e de processos individuais, sendo o primeiro em linha de comando, como visto no tópico abaixo.

### 2.5.1 Linha de Comando

O comando *top* exibe uma tela atualizada com informações básicas do uso da CPU e da memória junto com uma lista de todos os processos e seus respectivos usos individuais do processador e da memória (BISWAS, 2018). A figura 15 mostra a sua utilização na linha de comando.

Figura - Comando *top* na linha de comando



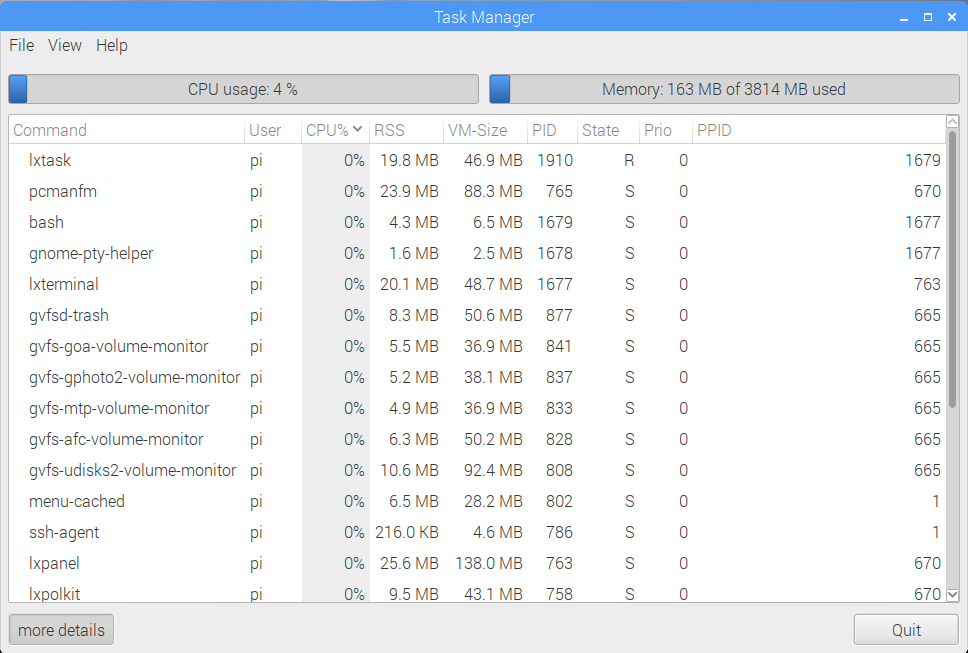
Fonte: Autor

Na figura 15 é possível observar na linha “%Cpu(s):” a porcentagem de uso do processador executando tarefas de usuário (*us*), de sistema (*sy*), tempo ocioso (*id*), entre outros. Na coluna “%CPU” nos processos é possível verificar o uso individual percentualmente do processador, e na coluna ao lado de “%MEM” o uso individual de cada processo da memória do sistema (BISWAS, 2018). É possível monitorar processos individualmente utilizando o modificador *grep* para o comando *top*. O tópico abaixo demonstra a outra ferramenta nativa, com interface gráfica.

### 2.5.2 Interface Gráfica

Também é possível verificar o uso geral e individual de memória e uso de CPU através da interface gráfica, acessível através do comando *lxtask.* A figura 16 mostra a interface gráfica dessa ferramenta.

Figura - Interface gráfica



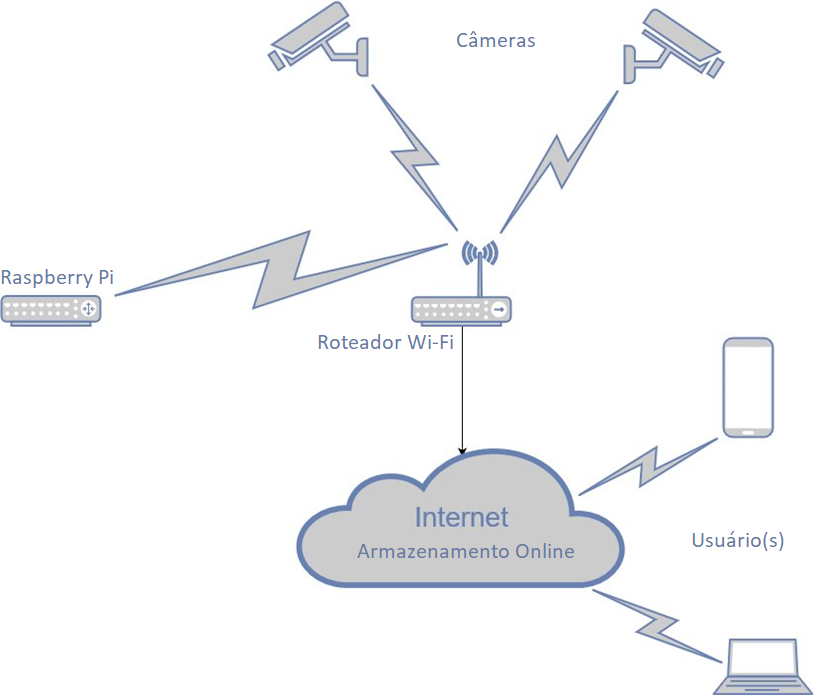
Fonte: Autor

De maneira similar ao comando *top*, o comando *lxtask* provê informações gerais do sistema quanto ao uso da CPU e da memória e informações individuais dos processos. O próximo tópico irá abordar o desenvolvimento do sistema de vigilância proposto.

# 3 Desenvolvimento do sistema de vigilÂncia

O desenvolvimento do sistema de vigilância proposto está detalhado neste capítulo e seguiu a visão geral da figura 17.

Figura - Visão geral do sistema



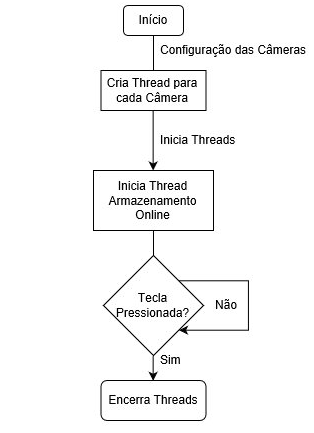
Fonte: Autor

O Raspberry Pi 3 Modelo B+ como processador central, recebe as imagens das câmeras via Wi-Fi na rede local, processa, e caso detectado movimento grava as imagens localmente até carregar os vídeos para o armazenamento on-line pela internet. O roteador Wi-Fi apenas fornece a rede na qual as câmeras IP e o Raspberry Pi se conectam e utilizam para comunicar entre si. Para que seja possível realizar o *upload* das gravações no armazenamento online, a rede deve ter acesso a internet. No Raspberry Pi foi instalado o Raspbian OS Stretch e foi compilado a versão 4.0 do OpenCV, além de instalado a ferramenta Rclone para ser possível realizar o upload das gravações para a nuvem. A câmera IP utilizada foi uma genérica com resolução VGA de 640x480. No tópico a seguir encontra-se os detalhes do programa desenvolvido.

## 3.1 Programa

Para realizar a monitoração do ambiente, é necessário obter as imagens das câmeras. Para cada câmera é criado uma *thread* que é responsável pela leitura, processamento e gravação das imagens, e uma *thread* independente responsável pelo *upload* das imagens na nuvem, conforme o fluxograma da figura 18.

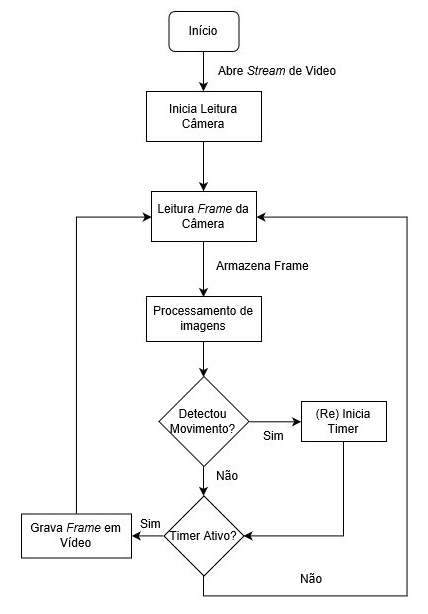
Figura – Fluxo principal do programa



Fonte: Autor

O programa principal apenas inicia as *threads* para cada câmera e a *thread* do armazenamento online, e em seguida fica aguardando alguma tecla pressionada para encerrar sua execução. A *thread* de cada câmera é responsável por abrir a comunicação com a câmera, ler os *frames*, processá-los, e em caso de detecção de movimento, gravá-los em arquivos. Os detalhes de como foram implementados e ajustados os algoritmos de detecção de movimento se encontram no tópico 3.2. O fluxograma da *thread* das câmeras se encontra na figura 19.

Figura - Fluxograma da *thread* das câmeras

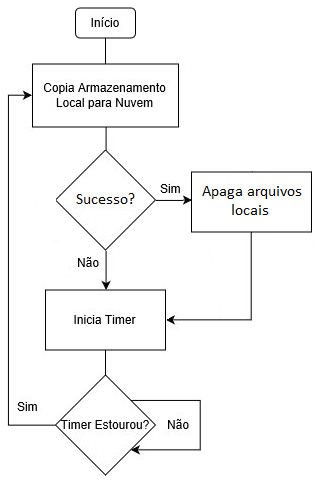


Fonte: Autor

Ao iniciar, a *thread* da câmera abre uma comunicação com a câmera e em seguida entra em seu *loop* principal, onde lê um *frame* e em seguida realiza o processamento do mesmo para detectar algum possível movimento. Se detectar movimento, inicia um *timer* durante o qual permanece gravando as imagens, e é renovado cada vez que é detectado movimento, gerando então continuidade da gravação de movimentos detectados em pequenos espaços de tempo.

Para o armazenamento das gravações foi utilizado uma técnica de armazenar uma quantidade predeterminada de *frames* anteriores para caso houver uma detecção de movimento, a gravação mostrar os instantes anteriores ao momento em que foi detectado o movimento. Para o fim da gravação, foi utilizado a mesma técnica e são gravados um número predeterminado de *frames* após a última detecção. Desta maneira, intervalos pequenos nos quais não há detecção de movimento são mesclados no mesmo vídeo onde ocorrem movimentos, criando um elo de relação entre o movimento de um objeto que atravessa a visão da câmera. O intervalo de tempo utilizado para os *frames* anteriores e posteriores ao da detecção de movimento foi de dois segundos. O salvamento local dos *frames* é efetuado no cartão SD utilizando o OpenCV, e podem ser feitos tanto no formato MP4 quanto no formato AVI, os quais são comparados no subcapítulo 5.2.7. Após as gravações locais, é necessário realizar o seu *upload* para o armazenamento em nuvem. O fluxograma da *thread* responsável por esse processo se encontra na figura 20.

Figura - Fluxograma da *thread* do *upload* à nuvem



Fonte: Autor

A *thread* do armazenamento online apenas realiza o *upload* e o apagamento local das imagens periodicamente. Para a sincronização das gravações com o armazenamento online, a *thread* realiza uma operação de cópia para o armazenamento online, na qual todos os arquivos que estão no armazenamento local que não estão no armazenamento online são copiados, testando por tamanho e data de modificação para comparar arquivos de mesmo nome. No tópico a seguir é detalhado o processo de detecção de movimento.

## 3.2 Detecção de Movimento

Para a realização da operação de detecção de movimento, foram testadas algumas variações de algoritmos de detecção, utilizando a biblioteca OpenCV, como abordado a seguir.

### 3.2.1 OpenCV

Com o OpenCV foram realizadas as operações de aplicar filtros, operação de erosão, subtração de imagens, subtração de *background* dinâmico, detector de bordas e detector de *blobs*, em combinações diferentes. Visando um algoritmo enxuto considerando o ambiente embarcado em que ele é executado, foi proposto um algoritmo dividido em três etapas, as quais são executadas em um intervalo predeterminado de *frames*. Na primeira etapa, o *frame* é convertido para escala de cinza e então aplicado um filtro gaussiano para amenizar os ruídos decorrentes da aquisição das imagens. A segunda etapa realiza uma operação para encontrar os candidatos a um possível movimento, usando os algoritmos abaixo:

* Subtração de dois *frames*;
* Subtração de *background* dinâmico;
* Detecção de Bordas.

Na terceira etapa, para analisarmos o resultado, são utilizados os algoritmos abaixo e então determinado se há movimento ou não.

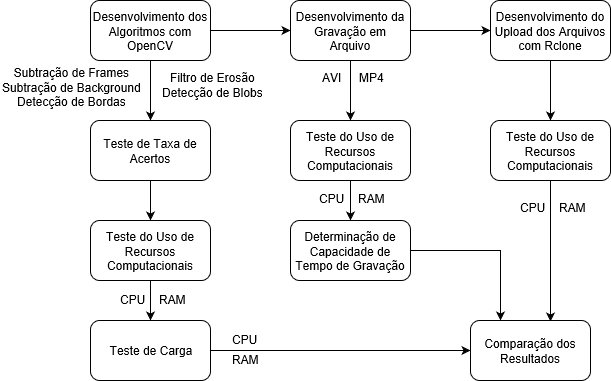
* Filtro de Erosão;
* Análise de *Blobs*;
* Subtração (apenas com o uso de detecção de bordas).

Como resultado, aplicando diferentes *thresholds* foi possível ajustar a sensibilidade para determinar se foi um movimento de interesse ou não. Mais detalhes sobre os resultados e ajustes dos algoritmos de detecção estão presentes no tópico 5.1. O capítulo a seguir apresenta a metodologia abordada para os testes e análises dos algoritmos e do sistema.

# 4 metodologia

Neste tópico são discutidas as metodologias seguidas para o desenvolvimento do trabalho. A visão geral segue o fluxograma descrito na figura 21.

Figura – Fluxograma da metodologia



Fonte: Autor

Inicialmente, os algoritmos de detecção por subtração de *frames* com erosão e detecção de *blobs*, subtração de *background* com erosão e detecção de *blobs*, e detecção de bordas com subtração foram desenvolvidos utilizando o OpenCV para serem testados quanto às suas taxas de acerto e consumo de processador e memória do sistema embarcado, para então serem submetidos ao teste de carga com múltiplos *streams*, visando determinar quantas câmeras o sistema é capaz de suportar com cada algoritmo desenvolvido. Em seguida, foi desenvolvido a gravação em disco das imagens utilizando os formatos AVI e MP4, para serem testados quanto aos seus consumos computacionais durante a gravação em disco do vídeo e o tamanho dos arquivos gravados, para determinar a capacidade do sistema em termos de tempo de gravação. Depois foi desenvolvido o *upload* das gravações para a nuvem e testado o seu consumo computacional. Por fim, os resultados foram cruzados e comparados, visando avaliar os resultados obtidos. A seguir a metodologia utilizada para a avaliação dos algoritmos é detalhada.

## 4.1 Algoritmos

Para a avaliação dos algoritmos no que se refere a taxa de acertos, esses foram executados sobre as mesmas sequencias de *frames*, utilizando um vídeo gravado em que determinados movimentos são executados em sequência, os quais devem ser identificados pelos algoritmos de detecção. Este vídeo consiste de uma pessoa se movendo em diferentes contextos e direções, e os algoritmos foram ajustados visando encontrar um ponto central no qual há uma taxa satisfatória de detecção de movimentos pequenos, e então ajustados com uma configuração de maior sensibilidade e uma de menor sensibilidade em relação a esse ponto central. O vídeo utilizado para ajustes conta com oito sequências de movimentos relevantes, e uma sem movimentação utilizada para testar falsos positivos, todas enumeradas na tabela 1, junto com uma intensidade arbitrária para cada.

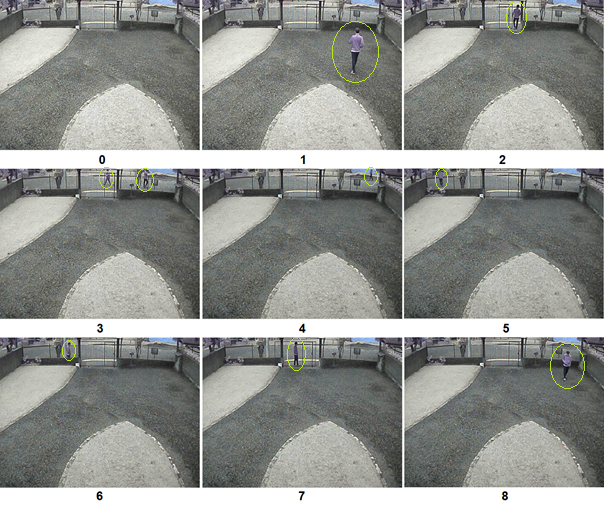
Tabela 1 - Descrição do Vídeo de Testes

|  |  |  |
| --- | --- | --- |
| **Sequência do vídeo** | **Descrição do movimento** | **Intensidade do movimento** |
| 0 | *Background* sem movimentação, duração de 60 segundos | Nula |
| 1 | Pessoa se distanciando da câmera, em direção diagonal | Alta |
| 2 | Portão se abrindo, pessoa saindo, portão se fechando | Média |
| 3 | Pessoas andando horizontalmente | Alta |
| 4 | Pessoa andando lentamente horizontalmente, atravessando atrás de obstáculos | Media |
| 5 | Pessoa andando lentamente em direção a câmera | Baixa |
| 6 | Pessoa andando até o portão | Media |
| 7 | Portão se abrindo, pessoa entrando, portão se fechando | Media |
| 8 | Pessoa andando lentamente, perto da câmera | Alta |

Fonte: Autor

Na figura 22 são apresentados um *frame* de cada sequência de movimentos descritas na tabela 1. Em todas as sequências, foram anotados o menor e o maior valor encontrado pelo algoritmo, e anotado também o resultado de cada processamento de detecção de movimento, a fim de encontrar a taxa de acerto de cada algoritmo para cada sequência de movimentos. As taxas de falsos positivos são as porcentagens de erros de cada algoritmo na sequência de movimento que não há movimentos de interesse, apenas o background. Quanto maior o intervalo entre os dois *frames* a serem subtraídos, maior será a diferença de um objeto se este se manter em movimento constante, portanto espera-se maior taxa de acerto para movimentos com direção constante, além de menor consumo de recursos computacionais. Visando isso, utilizou-se os valores de 0,5 segundos e 1 segundo como o intervalo de tempo a cada processamento de detecção de movimento.

Figura - *Frames* das sequências do vídeo teste



Fonte: Autor

Também foram mensurados a utilização dos recursos computacionais durante a execução desses testes, sendo este o assunto do tópico em sequência.

## 4.2 Capacidade do Sistema

Para a avaliação do consumo computacional, foram testados cada uma das combinações de algoritmos de detecção de movimento separadamente da gravação e de upload, visando isolar o custo computacional de cada componente do programa, sendo a única execução em comum em todos a recepção dos *frames* das câmeras, necessário para testar os demais componentes.

Primeiramente foi testado cada um dos algoritmos com uma câmera, e em seguida foram sendo adicionados um *stream* de vídeo a cada 15 segundos, para dessa forma determinar com quantas câmeras o sistema pode realizar a detecção de movimento sem apresentar queda de desempenho, e a partir de quantos *streams* o sistema começa a ficar instável. Em seguida, foram testados o custo computacional do *upload* das gravações para o armazenamento online e o da gravação em disco de uma câmera, este que foi efetuado utilizando dois *codecs* diferentes de codificação e decodificação para as gravações, para serem comparados quanto ao tamanho do vídeo produzido e o custo computacional para a codificação dos *frames* e gravação em disco. O primeiro sendo o *codec* MPEG para salvamento em formato AVI e o segundo sendo o *codec* AVC para salvamento em MP4.

Os testes foram realizados utilizando o comando *top* na linha de comando com uma taxa de atualização de 2 décimos de segundo, anotando o uso da CPU e da memória RAM, durante um período de 20 segundos para cada algoritmo. O tópico a seguir abordará como foi feito a análise da autonomia offline do sistema.

## 4.3 Autonomia

A questão da autonomia tem importância caso haja queda na disponibilidade de internet, um risco real no ambiente residencial. Nesse caso, a capacidade de armazenar localmente as gravações será proporcional ao tamanho do cartão SD e a frequência de movimentação diante as câmeras. Para quantificar essa capacidade, foi calculado a média de espaço local usado por segundo de gravação durante os testes de carga, e estimado quanto tempo seria possível se manter gravando novos vídeos até chegar ao ponto em que o armazenamento local se esgotasse, considerando algumas situações hipotéticas, incluindo o pior caso, no qual o sistema se manteria gravando constantemente devido à alta movimentação diante das câmeras. Outros casos que foram considerados foi a de gravação durante 50% do tempo e de 25% do tempo. Foi considerado o espaço disponível no cartão SD após a instalação do sistema operacional, do OpenCV e do programa, para ser projetado com tamanhos de cartão SD de 16, 32 e 64 GB, para determinar quantas horas de gravação o sistema seria capaz de armazenar em cada um desses antes de esgotar o armazenamento local. O capítulo a seguir trata dos resultados obtidos.

# 5 Resultados obtidos

Este tópico discute os resultados obtidos dos algoritmos testados, da capacidade de carga do sistema e da autonomia offline*.*

## 5.1 Algoritmos e Taxas de Acerto

Neste subcapítulo encontram-se os detalhes dos algoritmos testados, os ajustes realizados e os resultados obtidos. Todos os algoritmos testados contam com um filtro gaussiano após a aquisição das imagens com objetivo de reduzir ruídos decorrente da mesma. O vídeo teste foi inicialmente utilizado para testar o algoritmo mais simples, detalhado no tópico a seguir.

### 5.1.1 Subtração de Dois *Frames* e Erosão

O algoritmo de subtração de dois *frames* com o pós-processamento de erosão tem por objetivo obter uma imagem binária com informação da diferença de dois *frames* espaçados no tempo, e em seguida eliminar diferenças pequenas através da erosão, dessa forma obtendo apenas a diferença entre objetos significativos. Para o elemento estruturante da operação de erosão, utilizou-se matrizes retangulares de 5x5, 7x7 e 9x9 totalmente preenchidas. Após a operação de erosão, contou-se o número de pixels não pretos para quantificar o resultado, e anotado o menor e o maior valor de pixels encontrados em cada sequência, assim como a quantidade de acertos. Tais dados estão nas tabelas 2, 3 e 4 disponíveis a seguir. Será considerado detectado movimento qualquer resultado maior que zero.

Tabela 2 – Subtração de *frames* e erosão com matriz 5x5

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | **Intervalo de 0,5 segundo** | | | **Intervalo de 1 segundo** | | |
| **Sequência do Vídeo** | **Mínimo de Pixels** | **Máximo de Pixels** | **Taxa de Acertos** | **Mínimo de Pixels** | **Máximo de Pixels** | **Taxa de Acertos** |
| 0 | 0 | 55 | 85% | 0 | 7204 | 76,7% |
| 1 | 12 | 4383 | 100% | 393 | 4488 | 100% |
| 2 | 11 | 488 | 100% | 5 | 457 | 100% |
| 3 | 103 | 636 | 100% | 100 | 775 | 100% |
| 4 | 9 | 176 | 95,8% | 15 | 194 | 100% |
| 5 | 2 | 122 | 100% | 6 | 136 | 100% |
| 6 | 21 | 458 | 100% | 56 | 606 | 100% |
| 7 | 13 | 331 | 100% | 156 | 843 | 100% |
| 8 | 90 | 1077 | 100% | 270 | 1237 | 100% |

Fonte: Autor

Utilizando uma matriz 5x5, independente do intervalo utilizado entre os *frames*, em todas as sequências são sempre identificados movimentos, porém nota-se também que são detectados falsos positivos diversas vezes. Abaixo a tabela 3 com os resultados da utilização de erosão com matriz de elemento estruturante 7x7.

Tabela 3 – Subtração de *frames* e erosão com matriz 7x7

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | **Intervalo de 0,5 segundo** | | | **Intervalo de 1 segundo** | | |
| **Sequência do Vídeo** | **Mínimo de Pixels** | **Máximo de Pixels** | **Taxa de Acertos** | **Mínimo de Pixels** | **Máximo de Pixels** | **Taxa de Acertos** |
| 0 | 0 | 129 | 95% | 0 | 1536 | 91,7% |
| 1 | 45 | 411 | 100% | 171 | 3344 | 100% |
| 2 | 0 | 157 | 80% | 0 | 216 | 80% |
| 3 | 29 | 282 | 100% | 2 | 455 | 100% |
| 4 | 0 | 129 | 97,9% | 1 | 101 | 100% |
| 5 | 0 | 60 | 57,7% | 0 | 60 | 69,2% |
| 6 | 0 | 310 | 93,3% | 5 | 396 | 100% |
| 7 | 20 | 441 | 100% | 37 | 540 | 100% |
| 8 | 22 | 675 | 100% | 102 | 867 | 100% |

Fonte: Autor

Com a matriz 7x7, o número de falsos positivos diminui consideravelmente, porém durante as sequências certos movimentos não são detectados, apesar de ainda manter um bom número de acertos, permitindo uma gravação com boa consistência. A seguir a tabela 4 com os resultados da utilização de erosão com matriz de elemento estruturante 9x9.

Tabela 4 – Subtração de *frames* e erosão com matriz 9x9

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | **Intervalo de 0,5 segundo** | | | **Intervalo de 1 segundo** | | |
| **Sequência do Vídeo** | **Mínimo de Pixels** | **Máximo de Pixels** | **Taxa de Acertos** | **Mínimo de Pixels** | **Máximo de Pixels** | **Taxa de Acertos** |
| 0 | 0 | 69 | 97,5% | 0 | 288 | 90% |
| 1 | 262 | 2190 | 100% | 70 | 2506 | 100% |
| 2 | 0 | 54 | 50% | 0 | 84 | 70% |
| 3 | 0 | 127 | 85% | 5 | 264 | 100% |
| 4 | 0 | 34 | 66,7% | 0 | 137 | 87,5% |
| 5 | 0 | 29 | 30% | 0 | 17 | 15% |
| 6 | 0 | 178 | 66,7% | 0 | 245 | 75% |
| 7 | 0 | 211 | 70% | 2 | 327 | 100% |
| 8 | 0 | 359 | 87,5% | 22 | 559 | 100% |

Fonte: Autor

A matriz 9x9 reduz os falsos positivos, porém reduz também a identificação de movimentos pequenos, comprometendo a consistência das gravações em sequências com menor intensidade de movimentos.

Sobre o intervalo entre os *frames*, ambos mostraram desempenho similares, sendo melhor então optar, visando desempenho, o intervalo de 1 segundo. Quanto a sensibilidade, o algoritmo com a matriz de erosão 7x7 se mostra o melhor balanço entre acerto de negativos e positivos, sem prejudicar a plenitude de gravações. No tópico a seguir serão apresentados os resultados utilizando o mesmo algoritmo de subtração de *frames*, porém com pós-processamento de detecção de *blobs*.

### 5.1.2 Subtração de Dois *Frames* e Detecção de *Blobs*

Este algoritmo utiliza o mesmo operador de subtração entre dois *frames* intercalados, porém para a determinação de movimento utiliza-se um detector de *blobs*, onde filtra-se por tamanho do *blob* para desconsiderar variações muito pequenas entre os *frames*. Serão testados três valores mínimos de área para os *blobs*: 20, 40 e 80 pixels. A diferença entre detecção de *blobs* e filtro de erosão é que o princípio de funcionamento da detecção de *blobs* é encontrar agrupamento de pixels, ajudando a desconsiderar pequenas diferenças detectadas entre os *frames*. Os resultados podem ser encontrados nas tabelas 5, 6 e 7 a seguir, começando pela detecção com área mínima de 20 pixels da tabela 5.

Tabela 5 – Subtração de *frames* e detecção de *blobs* de 20 pixels

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | **Intervalo de 0,5 segundo** | | | **Intervalo de 1 segundo** | | |
| **Sequência do Vídeo** | **Mínimo de *Blobs*** | **Máximo de *Blobs*** | **Taxa de Acertos** | **Mínimo de *Blobs*** | **Máximo de *Blobs*** | **Taxa de Acertos** |
| 0 | 0 | 1 | 98,3% | 0 | 1 | 99,2% |
| 1 | 1 | 10 | 100% | 2 | 9 | 100% |
| 2 | 1 | 7 | 100% | 3 | 5 | 100% |
| 3 | 2 | 7 | 100% | 1 | 6 | 100% |
| 4 | 0 | 2 | 97,9% | 1 | 2 | 100% |
| 5 | 0 | 3 | 61,5% | 0 | 2 | 76,9% |
| 6 | 1 | 5 | 100% | 2 | 4 | 100% |
| 7 | 2 | 7 | 100% | 2 | 7 | 100% |
| 8 | 3 | 9 | 100% | 3 | 6 | 100% |

Fonte: Autor

Utilizando uma área mínima de 20 pixels, há uma taxa de acerto satisfatória, principalmente no intervalo de 1 segundo. Também apresentou uma boa taxa de falsos positivos. Na tabela 6 consta os resultados utilizando área mínima de 40 pixels.

Tabela 6 – Subtração de *frames* e detecção de *blobs* de 40 pixels

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | **Intervalo de 0,5 segundo** | | | **Intervalo de 1 segundo** | | |
| **Sequência do Vídeo** | **Mínimo de *Blobs*** | **Máximo de *Blobs*** | **Taxa de Acertos** | **Mínimo de *Blobs*** | **Máximo de *Blobs*** | **Taxa de Acertos** |
| 0 | 0 | 0 | 100% | 0 | 0 | 100% |
| 1 | 2 | 7 | 100% | 1 | 8 | 100% |
| 2 | 1 | 5 | 100% | 2 | 4 | 100% |
| 3 | 1 | 5 | 100% | 1 | 4 | 100% |
| 4 | 0 | 2 | 97,9% | 1 | 2 | 100% |
| 5 | 0 | 2 | 38,4% | 0 | 2 | 53,8% |
| 6 | 0 | 5 | 93,3% | 0 | 4 | 87,5% |
| 7 | 2 | 6 | 100% | 2 | 5 | 100% |
| 8 | 2 | 6 | 100% | 0 | 4 | 91,6% |

Fonte: Autor

O algoritmo com detecção de *blobs* de tamanho mínimo de 40 pixels, como esperado, possui menor taxa de falsos positivos, porém também possuí uma baixa taxa de acerto na sequência 5, apesar de um pouco mitigada pelo intervalo de 1 segundo. Abaixo a tabela 7 com os resultados utilizando 80 pixels de área mínima.

Tabela 7 – Subtração de *frames* e detecção de *blobs* de 80 pixels

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | **Intervalo de 0,5 segundo** | | | **Intervalo de 1 segundo** | | |
| **Sequência do Vídeo** | **Mínimo de *Blobs*** | **Máximo de *Blobs*** | **Taxa de Acertos** | **Mínimo de *Blobs*** | **Máximo de *Blobs*** | **Taxa de Acertos** |
| 0 | 0 | 0 | 100% | 0 | 0 | 100% |
| 1 | 2 | 5 | 100% | 1 | 4 | 100% |
| 2 | 1 | 3 | 100% | 1 | 3 | 100% |
| 3 | 0 | 3 | 90% | 1 | 2 | 100% |
| 4 | 0 | 2 | 95,8% | 0 | 2 | 91,6% |
| 5 | 0 | 1 | 7,7% | 0 | 2 | 7,7% |
| 6 | 0 | 5 | 60% | 0 | 2 | 87,5% |
| 7 | 1 | 4 | 100% | 1 | 4 | 100% |
| 8 | 0 | 4 | 95,8% | 0 | 2 | 91,6% |

Fonte: Autor

Como esperado, o detector de *blobs* com menor sensibilidade possui alta taxa de erro na sequência onde há menor intensidade de movimentos, o que resulta também em uma baixa taxa de falsos positivos.

Sobre o intervalo entre *frames*, tanto o de 0,5 quanto o de 1 segundo tiveram taxa de acertos similares em relação as sequências de movimentos de média e alta intensidade, porém o de 1 segundo teve taxa de acerto superior na sequência de movimento de baixa intensidade. A utilização de área mínima de 80 pixels torna o algoritmo incapaz de detectar consistentemente movimentos de baixa intensidade, mas com uma boa taxa de falsos positivos. O próximo tópico irá detalhar os resultados de um algoritmo usando outra técnica de diferenciação, o de subtração de *background* dinâmico.

### 5.1.3 Subtração de *Background* com Erosão

Este algoritmo utiliza a técnica de manter um *frame* com *background* atualizado dinamicamente, usado para comparar o *frame* atual para encontrar diferenças. Devido a este algoritmo ter apresentado resultados onde há menor sensibilidade e as diferenças serem menores, foram utilizados como elementos estruturantes para a operação de erosão matrizes retangulares de 3x3, 5x5 e 7x7, mantendo os intervalos testados de 0,5 e 1 segundo. Abaixo a tabela 8 com a matriz 3x3.

Tabela 8 – Subtração de *background* e erosão com matriz 3x3

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | **Intervalo de 0,5 segundo** | | | **Intervalo de 1 segundo** | | |
| **Sequência do Vídeo** | **Mínimo de Pixels** | **Máximo de Pixels** | **Taxa de Acertos** | **Mínimo de Pixels** | **Máximo de Pixels** | **Taxa de Acertos** |
| 0 | 0 | 190402 | 67,5% | 0 | 230247 | 60% |
| 1 | 286 | 3858 | 100% | 205 | 3544 | 100% |
| 2 | 136 | 907 | 100% | 29 | 1115 | 100% |
| 3 | 7 | 771 | 100% | 41 | 673 | 100% |
| 4 | 15 | 240 | 100% | 55 | 254 | 100% |
| 5 | 0 | 182 | 88,4% | 0 | 230 | 84,6% |
| 6 | 75 | 784 | 100% | 226 | 875 | 100% |
| 7 | 375 | 949 | 100% | 541 | 943 | 100% |
| 8 | 374 | 1255 | 100% | 219 | 1170 | 100% |

Fonte: Autor

Com a matriz 3x3, o algoritmo obteve bom resultado na detecção de todas as sequências, porém apontou diversos falsos positivos, devido também a detecção falsa em variações de iluminação. Abaixo a tabela 9 com os resultados da matriz 5x5.

Tabela 9 – Subtração de *background* e erosão com matriz 5x5

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | **Intervalo de 0,5 segundo** | | | **Intervalo de 1 segundo** | | |
| **Sequência do Vídeo** | **Mínimo de Pixels** | **Máximo de Pixels** | **Taxa de Acertos** | **Mínimo de Pixels** | **Máximo de Pixels** | **Taxa de Acertos** |
| 0 | 0 | 165849 | 94,2% | 0 | 207829 | 91,7% |
| 1 | 109 | 2553 | 100% | 148 | 2385 | 100% |
| 2 | 0 | 395 | 85% | 12 | 312 | 100% |
| 3 | 4 | 418 | 100% | 2 | 295 | 100% |
| 4 | 31 | 133 | 100% | 1 | 117 | 100% |
| 5 | 0 | 95 | 46,1% | 0 | 13 | 53,8% |
| 6 | 2 | 550 | 100% | 44 | 614 | 100% |
| 7 | 14 | 416 | 100% | 0 | 471 | 80% |
| 8 | 75 | 604 | 100% | 138 | 1025 | 100% |

Fonte: Autor

Com a matriz 5x5, a taxa de falsos positivos diminui, juntamente com a taxa de acerto, porém ainda mantem bom acerto nos movimentos de média e alta intensidade. Abaixo a tabela 10 com os resultados da matriz 7x7.

Tabela 10 – Subtração de *background* e erosão com matriz 7x7

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | **Intervalo de 0,5 segundo** | | | **Intervalo de 1 segundo** | | |
| **Sequência do Vídeo** | **Mínimo de Pixels** | **Máximo de Pixels** | **Taxa de Acertos** | **Mínimo de Pixels** | **Máximo de Pixels** | **Taxa de Acertos** |
| 0 | 0 | 147497 | 96,7% | 0 | 188817 | 95% |
| 1 | 23 | 1804 | 100% | 62 | 1613 | 100% |
| 2 | 0 | 118 | 60% | 0 | 115 | 80% |
| 3 | 0 | 250 | 90% | 0 | 109 | 90% |
| 4 | 0 | 63 | 91,6% | 0 | 68 | 91,6% |
| 5 | 0 | 0 | 0% | 0 | 0 | 0% |
| 6 | 0 | 352 | 93,3% | 57 | 391 | 100% |
| 7 | 0 | 261 | 90% | 0 | 203 | 80% |
| 8 | 45 | 499 | 100% | 29 | 694 | 100% |

Fonte: Autor

Com a matriz 7x7, o algoritmo tem dificuldades de detectar movimentos de baixa intensidade, e como esperado possui baixa taxa de falsos positivos. Ideal para detectar apenas objetos grandes ou movimentos de maior intensidade.

A subtração de *background* com erosão 3x3 demonstrou alta taxa de acerto de baixo movimentos, porém a técnica de subtração de *background* também demonstrou alta sensibilidade a mudanças de iluminação, sendo uma das principais causas de falsos positivos nesta combinação de algoritmo, que ao utilizar matrizes maiores para a erosão foram reduzidos, porém ainda ocorrentes.

O intervalo de 1 segundo tende a ter melhor taxa de acertos, como nos demais algoritmos. Esse intervalo de tempo maior entre os *frames* processados permite uma maior diferença entre o *background* dinâmico e o *frame* atual, ao custo de uma ocorrência maior de falsos positivos devido a mudança de iluminação. A seguir estão os resultados do algoritmo de subtração de *background* utilizando detecção de *blobs.*

### 5.1.4 Subtração de *Background* com Detecção de *Blobs*

A utilização de ambas as técnicas de subtração de *background* e a detecção de *blobs* já testadas em combinações diferentes foram testadas e os resultados se encontram nas tabelas 11, 12 e 13 a seguir.

Tabela 11 – Subtração de *background* e detecção de *blobs* de 20 pixels

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | **Intervalo de 0,5 segundo** | | | **Intervalo de 1 segundo** | | |
| **Sequência do Vídeo** | **Mínimo de *Blobs*** | **Máximo de *Blobs*** | **Taxa de Acertos** | **Mínimo de *Blobs*** | **Máximo de *Blobs*** | **Taxa de Acertos** |
| 0 | 0 | 231 | 94,2% | 0 | 160 | 78,3% |
| 1 | 2 | 13 | 100% | 1 | 10 | 100% |
| 2 | 1 | 13 | 100% | 1 | 4 | 100% |
| 3 | 1 | 6 | 100% | 1 | 7 | 100% |
| 4 | 1 | 2 | 100% | 1 | 2 | 100% |
| 5 | 0 | 3 | 65,3% | 0 | 2 | 76% |
| 6 | 1 | 3 | 100% | 1 | 3 | 100% |
| 7 | 2 | 7 | 100% | 1 | 11 | 100% |
| 8 | 1 | 7 | 100% | 3 | 7 | 100% |

Fonte: Autor

Este algoritmo apresentou bons resultados tanto na detecção de movimentos pequenos quanto na detecção de movimentos grandes, porém teve uma taxa considerável de falsos positivos, principalmente decorrente de mudanças de iluminação. A seguir a tabela 12 com o teste de área mínima de 40 pixels.

Tabela 12 – Subtração de *background* e detecção de *blobs* de 40 pixels

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | **Intervalo de 0,5 segundo** | | | **Intervalo de 1 segundo** | | |
| **Sequência do Vídeo** | **Mínimo de *Blobs*** | **Máximo de *Blobs*** | **Taxa de Acertos** | **Mínimo de *Blobs*** | **Máximo de *Blobs*** | **Taxa de Acertos** |
| 0 | 0 | 109 | 96,6% | 0 | 66 | 93,3% |
| 1 | 2 | 8 | 100% | 1 | 8 | 100% |
| 2 | 1 | 7 | 100% | 0 | 5 | 80% |
| 3 | 1 | 4 | 100% | 1 | 3 | 100% |
| 4 | 1 | 2 | 100% | 1 | 1 | 100% |
| 5 | 0 | 2 | 46,1% | 0 | 2 | 69,2% |
| 6 | 1 | 3 | 100% | 0 | 2 | 87,5% |
| 7 | 1 | 6 | 100% | 1 | 6 | 100% |
| 8 | 1 | 7 | 100% | 1 | 3 | 100% |

Fonte: Autor

Como esperado, quanto maior a área mínima de detecção, menor a taxa de acertos para movimentos pequenos. Abaixo a tabela 13 com os resultados do teste com área mínima de 80 pixels.

Tabela 13 – Subtração de *background* e detecção de *blobs* de 80 pixels

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | **Intervalo de 0,5 segundo** | | | **Intervalo de 1 segundo** | | |
| **Sequência do Vídeo** | **Mínimo de *Blobs*** | **Máximo de *Blobs*** | **Taxa de Acertos** | **Mínimo de *Blobs*** | **Máximo de *Blobs*** | **Taxa de Acertos** |
| 0 | 0 | 35 | 96,6% | 0 | 22 | 98,3% |
| 1 | 2 | 5 | 100% | 1 | 3 | 100% |
| 2 | 1 | 3 | 100% | 0 | 5 | 90% |
| 3 | 1 | 3 | 100% | 1 | 3 | 100% |
| 4 | 1 | 1 | 100% | 1 | 1 | 100% |
| 5 | 0 | 1 | 7,7% | 0 | 2 | 46,1% |
| 6 | 0 | 2 | 93,3% | 1 | 2 | 100% |
| 7 | 1 | 5 | 100% | 1 | 3 | 100% |
| 8 | 1 | 3 | 100% | 1 | 3 | 100% |

Fonte: Autor

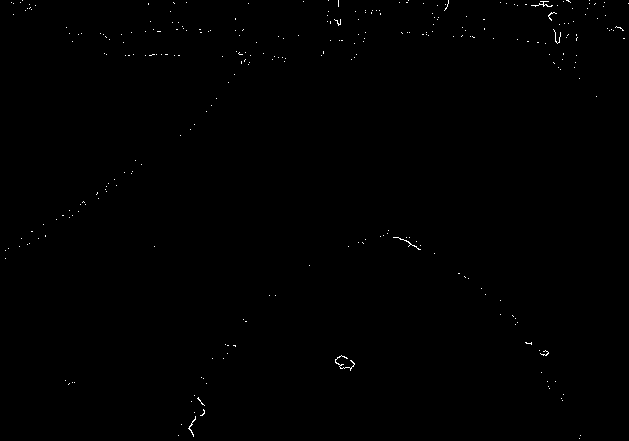
Com área mínima de 80 pixels, os resultados da taxa de acerto caem principalmente nos movimentos pequenos, se tornando incapaz de detectar consistentemente na sequência de movimentos de baixa intensidade.

No geral, a combinação de detecção de *background* com detector de *blobs* possui boa taxa de acerto para movimentos de média e alta intensidade, e com área mínima de 20 pixels é capaz de detectar a maioria dos movimentos de baixa intensidade, principalmente com intervalo de 1 segundo entre o processamento, além de manter uma baixa taxa de falsos positivos. Com área mínima de 80 pixels, há baixa taxa de acertos em cenas de baixa intensidade. A detecção com subtração de *background* se mostrou sensível a mudanças de iluminação, principalmente com a área mínima de 20 pixels, o qual fica evidente na comparação do intervalo de 0,5 para 1 segundo, devido ao tempo necessário para o *background* dinâmico se adaptar a essas mudanças, onde seria necessário um intervalo menor de processamento para alimentar o *background* e torna-lo menos sensível.

### 5.1.5 Detecção de Bordas

O algoritmo que utiliza a detecção de bordas tinha por objetivo a subtração de dois *frames* com as bordas dos objetos para realçar as bordas dos objetos se movendo, entretanto este não detectou consistentemente em nenhuma das situações do vídeo de teste, com nenhuma combinação de *thresholds*. Acredita-se que isso se deve ao fato de que qualquer diferença entre um *frame* e outro gera bordas diferentes, ocasionando na imprecisão da subtração entre os mesmos. Seria necessário então mais uma etapa de pós-processamento, o que foi descartado devido ao sistema ter por objetivo o baixo consumo computacional. Abaixo a figura 23 ilustra o ruído restante da operação de subtração.

Figura - Resultado da operação de subtração de *frames* com bordas



Fonte: Autor

Após todos os algoritmos testados, é necessário comparar os resultados para determinar os pontos positivos e negativos de cada um. Esse assunto é abordado no tópico a seguir.

### 5.1.6 Comparação dos Resultados

No total, 4 algoritmos de detecção foram testados, cada um com 6 variações quanto ao intervalo de processamento e os *thresholds* de suas respectivas técnicas. Os resultados dos testes efetuados foram sumarizados em gráficos para facilitar a comparação entre eles, estes disponíveis a seguir, começando pelo gráfico 2 de subtração de dois *frames* com erosão.

Gráfico - Comparação da taxa de acerto para subtração de *frames* com erosão

Fonte: Autor

A subtração de dois *frames* com erosão apresenta bons índices de falsos positivos, sendo consideravelmente sensível ao aumento da matriz usada como elemento estruturante da operação de erosão, resultando em baixas taxas de acerto inclusive para movimentos de intensidade média. A seguir o gráfico 3 com os resultados da subtração de dois *frames* com detecção de *blobs*.

Gráfico - Comparação da taxa de acerto para subtração de *frames* com detecção de *blobs*

Fonte: Autor

A subtração de dois *frames* com detecção de *blobs* apresenta resultados mais estáveis em relação ao com erosão no que se refere a movimentos de intensidade média, e menor taxa de falsos positivos, porém mantém uma taxa de acerto similar em movimentos de baixa intensidade. A seguir o gráfico 4 com a taxa de acerto da subtração de *background* com erosão.

Gráfico - Comparação da taxa de acerto para subtração de *background* com erosão

Fonte: Autor

Este algoritmo apresentou uma relação forte entre uma menor taxa de falsos positivos com uma menor taxa de acerto para movimentos de baixa intensidade, em conjunto com uma menor taxa de acerto para movimentos de média intensidade em relação aos comparados anteriormente, e a menor taxa de acertos de movimentos de baixa intensidade para uma baixa taxa de falsos positivos. Abaixo o gráfico 5 com a comparação dos resultados do algoritmo de subtração de *background* com detecção de *blobs*.

Gráfico - Comparação da taxa de acerto para subtração de *background* com detecção de *blobs*

Fonte: Autor

A subtração de *backgroud* com detecção de *blobs* apresentou resultados similares ao de subtração de dois *frames* com detecção de *blobs*, tendo uma menor taxa de acertos e maior taxa de falsos positivos. A seguir no gráfico 6 se encontra a comparação da média da taxa de acerto de todos os algoritmos.

Gráfico - Comparação dos algoritmos testados

Fonte: Autor

Este gráfico mostra as principais características de cada algoritmo, sendo o de subtração de dois *frames* com detecção de *blobs* o que apresentou, na média, melhores resultados na não detecção de falsos positivos, enquanto o de subtração de dois *frames* com erosão se mostrou, na média, o mais instável, o que pode estar relacionado com sua melhor detecção de movimentos pequenos. Os algoritmos com detecção de *blobs* foram os que apresentaram menor taxa de falsos positivos comparado com os que utilizaram erosão. A próxima etapa na análise dos resultados deste trabalho compreende o teste de carga do sistema, tema do tópico a seguir.

## 5.2 Uso de Recursos Computacionais

Quanto a utilização de recursos computacionais, primeiro foram testados os algoritmos usando o comando “top” a cada 0.2 segundo e anotado os valores percentuais de uso da CPU e de memória RAM durante a execução. Iniciando pelo algoritmo de detecção por subtração de *frames* e filtro de erosão, detalhado no tópico abaixo.

### 5.2.1 Subtração de *Frames* com Erosão

Os resultados do algoritmo de Subtração de *frames* com erosão para uma câmerase encontram nos gráficos 7 e 8 a seguir.

Gráfico – Carga computacional da subtração de *frames* com erosão a cada 0,5 segundo de uma câmera

Fonte: Autor

Percebe-se dois momentos distintos durante a execução do programa: a execução do recebimento dos *frames* e o processamento do algoritmo de detecção, onde há um pico de uso do processador, podendo chegar perto de 30% do uso da capacidade de processamento do Raspberry Pi. Durante a execução do recebimento dos *frames*, o uso da CPU fica em torno de 4%. A média do uso do processador durante todo o período é de 10,4%. O uso da memória RAM não varia depois que chega a seu valor nominal após o início da execução do programa, e em média é de 12,9%. A seguir o gráfico 8 com os resultados do intervalo de 1 segundo a cada processamento do algoritmo de detecção.

Gráfico - Carga computacional da subtração de *frames* com erosão a cada 1 segundo de uma câmera

Fonte: Autor

A média do uso do processador com intervalo de 1 segundo cai para 7,3%, como esperado devido a haver um intervalo maior entre cada processamento do algoritmo de detecção. O uso da memória RAM mantém-se em 12,9%. Abaixo se encontram os gráficos do uso médio da CPU e memória RAM utilizando várias câmeras, começando pelo gráfico 9 com intervalo de 0,5 segundo.

Gráfico - Carga computacional da subtração de *frames* com erosão a cada 0,5 segundo de várias câmeras

Fonte: Autor

Nos testes realizados, o algoritmo de subtração de *frames* com erosão a cada 0,5 segundos não apresentou engasgos com até 6 câmeras. Com 8 câmeras o sistema para de responder, devido ao alto consumo de memória RAM. Abaixo o gráfico 10 com o resultado do processamento a cada 1 segundo.

Gráfico – Carga computacional da subtração de *frames* com erosão a cada 1 segundo de várias câmeras

Fonte: Autor

Com o algoritmo de subtração de *frames* com erosão a cada 1 segundo, o uso da CPU é menor, se tornando a memória RAM o limitador, chegando ao colapso com 9 câmeras. O sistema funciona sem queda de desempenho com até 7 câmeras. O tópico a seguir apresenta os resultados para o algoritmo de subtração de dois *frames* com detecção de *blobs*.

### 5.2.2 Subtração de *Frames* com Detecção de *Blobs*

Os gráficos 11 e 12 apresentam os resultados para o algoritmo de subtração de *frames* com detecção de *blobs* para uma câmera*.*

Gráfico - Carga computacional da subtração de *frames* com detecção de *blobs* a cada 0,5 segundo de uma câmera

Fonte: Autor

Com o uso da detecção de *blobs*, nota-se um aumento do uso do processador durante os picos de processamento, chegando em 52,4%. Na média, o uso do processador fica em 22,4% e o uso da memória RAM em 13%. Abaixo o gráfico 12 com os resultados utilizando intervalo de 1 segundo.

Gráfico - Carga computacional da subtração de *frames* com detecção de *blobs* a cada 1 segundo de uma câmera

Fonte: Autor

Utilizando intervalo de 1 segundo, uso da CPU cai em torno de 40%. O maior pico de processamento é de 53,1%, com média geral de 13,2% de uso de processador, menor devido ao intervalo entre cada processamento. O uso de memória RAM é de 13%. Abaixo se encontram os gráficos do uso médio da CPU e memória RAM utilizando várias câmeras, iniciando pelo gráfico 13 com o intervalo de 0,5 segundo.

Gráfico - Carga computacional da subtração de *frames* com detecção de *blobs* a cada 0,5 segundo de várias câmeras

Fonte: Autor

O algoritmo de subtração de *frames* com detecção de *blobs* a cada 0,5 segundo possui alto uso da CPU, e apesar da memória RAM permitir, a partir de 6 câmeras o sistema já não é capaz de suportar a carga computacional, sendo apenas com 4 câmeras que não apresenta engasgos. Abaixo o gráfico 14 com o processamento a cada 1 segundo.

Gráfico - Carga computacional da subtração de *frames* com detecção de *blobs* a cada 1 segundo de várias câmeras

Fonte: Autor

Com intervalo de 1 segundo, o algoritmo de subtração de *frames* com detecção de *blobs* é capaz de processar até 6 câmeras sem engasgos, sendo 8 o ponto de ruptura. Agora serão apresentados os resultados do algoritmo de detecção de *background* com erosão.

### 5.2.3 Subtração de *Background* com Erosão

Neste tópico serão apresentados os detalhes de uso de recursos computacionais do algoritmo de subtração de *background* com erosão, começando pelo intervalo de meio segundo no gráfico 15.

Gráfico – Carga computacional da subtração de *background* com erosão a cada 0,5 segundo de uma câmera

Fonte: Autor

O uso do processador nesse algoritmo apresenta picos do uso da CPU de até 31%, com média geral de 15,1%. O uso da memória RAM se mantém em torno da média de 14,8%. Abaixo o gráfico 16 com intervalo de 1 segundo.

Gráfico - Carga computacional da subtração de *background* com erosão a cada 1 segundo de uma câmera

Fonte: Autor

Utilizando intervalo de 1 segundo, a média do uso do processador diminui para 9,9%, com picos de até 38%. O uso da memória RAM fica similar ao anterior, com média de 14,8%. Abaixo se encontram os resultados utilizando várias câmeras, iniciando pelo intervalo de 0.5 segundo no gráfico 17.

Gráfico - Carga computacional da subtração de *background* com erosão a cada 0,5 segundo de várias câmeras

Fonte: Autor

Com o algoritmo de subtração de *background* com erosão, o uso da CPU limita o sistema a operar sem engasgos com até 5 câmeras, e com 6 o sistema começa a ficar instável. Abaixo o gráfico 18 com o intervalo de 1 segundo.

Gráfico - Carga computacional da subtração de *background* com erosão a cada 1 segundo de várias câmeras

Fonte: Autor

Utilizando intervalo de 1 segundo, o algoritmo de subtração de *background* com erosão é capaz de processar até 6 câmeras sem engasgos. Com 8 câmeras o sistema falha por falta de memória RAM, uma situação similar ao que ocorre com 7 câmeras. No próximo tópico estão os resultados do teste do algoritmo de subtração de *background* com detecção de *blobs*.

### 5.2.4 Subtração de *Background* com Detecção de *Blobs*

Este algoritmo de subtração de *background* com detecção de *blobs* tem carga computacional detalhada nos gráficos apresentados neste subcapítulo, iniciando pelo intervalo de 0,5 segundo.

Gráfico - Carga computacional da subtração de *background* com detecção de *blobs* a cada 0,5 segundo de uma câmera

Fonte: Autor

Este algoritmo em média consumiu 26,6% da capacidade do processador, chegando a picos de 62,5%, com uso de memória RAM de 14,9%. Este algoritmo apresenta o maior consumo de recursos computacionais dos algoritmos testados. O gráfico 20 apresenta os resultados com intervalo de 1 segundo.

Gráfico - Carga computacional da subtração de *background* com detecção de *blobs* a cada 1 segundo de uma câmera

Fonte: Autor

Com intervalo de 1 segundo, o consumo médio do processador é de 15,8%, com picos de até 63,3%, com uso médio de memória RAM de 14,9%. Abaixo são apresentados os resultados de múltiplas câmeras, no gráfico 21.

Gráfico - Carga computacional da subtração de *background* com detecção de *blobs* a cada 0,5 segundo de várias câmeras

Fonte: Autor

O algoritmo de subtração de *background* com detecção de *blobs*, o algoritmo mais oneroso, é capaz de processar sem engasgos apenas 3 câmeras, devido ao alto uso da CPU. Abaixo o gráfico 22 com o resultado do intervalo de 1 segundo.

Gráfico - Carga computacional da subtração de *background* com detecção de *blobs* a cada 1 segundo de várias câmeras

Fonte: Autor

Com intervalo de 1 segundo, o algoritmo de subtração de *background* com detecção de *blobs* é capaz de processar até 4 câmeras sem engasgos. No tópico a seguir os resultados do uso dos recursos computacionais são comparados entre todos os algoritmos testados.

### 5.2.5 Comparação da Carga Computacional dos Algoritmos

Neste tópico os resultados anteriores são discutidos. O uso de subtração de *background* é mais complexo computacionalmente que a subtração de *frames*, e a detecção de *blobs* mais complexa que o filtro de erosão, o que justifica este ser o algoritmo mais oneroso para o sistema enquanto o de subtração de *frames* com erosão ser o menos oneroso. A seguir a tabela 14 contém os resultados dos testes de carga resumidos de cada algoritmos.

Tabela 14 – Resumo dos testes de carga dos algoritmos

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Algoritmo** | **CPU com 1 Câmera** | **Memória com 1 Câmera** | **Máximo de Câmeras** | **Limitador** |
| *Frames* Erosão 1 s | 7,3% | 12,9% | 7 | Memória |
| *Frames* Erosão 0,5 s | 10,4% | 12,9% | 6 | Memória |
| Frames *Blobs* 1 s | 13,2% | 13% | 6 | CPU e Memória |
| Frames *Blobs* 0,5 s | 22,4% | 13% | 4 | CPU |
| *Background* Erosão 1 s | 9,9% | 14,8% | 6 | Memória |
| *Background* Erosão 0,5 s | 15,1% | 14,8% | 5 | CPU e Memória |
| *Background* *Blobs* 1 s | 15,8% | 14,9% | 4 | CPU |
| *Background* *Blobs* 0,5 s | 26,6% | 14,9% | 3 | CPU |

Fonte: Autor

Utilizando o algoritmo de detecção por subtração de *frames* e erosão com intervalo de 1 segundo, o sistema é capaz de processar até 7 câmeras sem apresentar queda de desempenho, mais que o dobro que o algoritmo de detecção por subtração de *background* com detecção de *blobs,* que é capaz de processar 3 câmeras. A escolha de qual o melhor algoritmo a ser utilizado deve se embasar na necessidade do sistema, levando em consideração não apenas da taxa de acerto, mas na necessidade de número de câmeras. Essas informações são cruzadas no subcapítulo 5.4, lembrando que o consumo de CPU e memória foram testados isoladamente do *upload* das gravações para o armazenamento online e da gravação local, estes testes que são apresentados no tópico a seguir.

### 5.2.6 Uso de Recursos Computacionais do *Upload* ao Armazenamento Online

Neste tópico são apresentados os resultados do teste de carga do *upload* à nuvem, em conjunto com a recepção de *frames*, a qual está sempre executando no programa. A seguir o gráfico 23 com o uso da CPU e da memória RAM durante o *upload* dos arquivos.

Gráfico – Utilização da CPU durante o *upload* à nuvem

Fonte: Autor

O *upload* das gravações ao serviço de armazenamento online possui baixo uso do processador, não acrescendo carga significativa à recepção dos *frames*. No teste é iniciado a sincronização online no segundo 4, aumentando a média do uso da CPU de 3,5% para 5%. Não foi notado acréscimo de uso da memória RAM durante o *upload*. No tópico a seguir são apresentados os resultados do teste da gravação local da câmera.

### 5.2.7 Uso de Recursos Computacionais do Salvamento Local das Gravações

Este tópico apresenta os resultados do teste de uso de recursos computacionais da gravação de um vídeo no armazenamento local do Raspberry Pi, utilizando dois *codecs* diferentes, o MPEG e o AVC, durante a gravação de um arquivo de vídeo com 4 segundos de duração. A seguir se encontra o gráfico 24 com os resultados da comparação de ambos.

Gráfico – Uso de recursos computacionais da gravação em disco em formato MP4 e AVI

Fonte: Autor

O gráfico 24 mostra o momento em que o programa determina que deve parar de gravar as imagens recebidas pela câmera e então inicia o processo do sistema realizar a gravação do arquivo no cartão SD. O gráfico apresenta apenas a utilização da CPU e da memória desse processo isolado. É possível notar uma alta utilização do processador durante a gravação de vídeo em formato MP4 com *codec* AVC, que se inicia no segundo 1 e vai até o segundo 4, e um pequeno acréscimo na utilização da memória. O alto uso do processador durante a gravação pode vir a prejudicar a resposta do sistema durante a execução do algoritmo de detecção de movimento, principalmente com múltiplas câmeras. O uso do processador durante a gravação em formato AVI com *codec* MPEGse mantém consideravelmente mais baixo, sendo também mais rápido. A diferença do tamanho dos arquivos é apresentada na tabela 15 abaixo.

Tabela 15 – Resumo dos testes de carga da gravação de vídeo

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Formato** | **Tamanho (Kb)** | **Tempo (Segundos)** | **Tamanho/segundo (Kb)** | **Uso médio da CPU** |
| AVI | 849,91 | 4 | 212,47 | 18,3% |
| MP4 | 414,36 | 4 | 103,59 | 74,5% |

Fonte: Autor

Apesar de gerar arquivos de vídeo menores, o custo computacional da compressão é maior, reduzindo a capacidade do sistema de processar diversas câmeras. O gráfico 25 mostra a utilização de recursos computacionais durante todo o processo detecção de movimentos do algoritmo de subtração de *frames* com erosão a cada 1 segundo em conjunto com a gravação do arquivo no armazenamento local utilizando o formato AVI.

Gráfico – Uso de recursos computacionais durante a execução de detecção de movimento e gravação AVI

Fonte: Autor

No gráfico, cerca de 3 gravações são efetuadas, com pequeno intervalo sem movimentação entre as cenas com movimentação. A média do uso da CPU é de 14,7%, com pico da utilização da memória em 13,5%. Em comparação do processo sem a gravação, há um aumento no uso total da CPU de 7,4% e um aumento no uso total da memória de 0,6%. Recalculando o limite de câmeras que o sistema suporta para cada algoritmo utilizando a gravação com *codec* MPEG, temos a tabela 16 a seguir.

Tabela 16 – Limite de câmeras para cada algoritmo considerando a gravação em disco

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Algoritmo** | **CPU com 1 Câmera** | **Memoria com 1 Câmera** | **Máximo de Câmeras** | **Limitador** |
| *Frames* Erosão 1 s | 14,7% | 13,5% | 5 | CPU |
| *Frames* Erosão 0,5 s | 17,8% | 13,5% | 4 | CPU |
| *Frames* *Blobs* 1 s | 20,6% | 13,6% | 4 | CPU |
| *Frames* *Blobs* 0,5 s | 29,8% | 13,6% | 3 | CPU |
| *Background* Erosão 1 s | 17,3% | 15,4% | 4 | CPU |
| *Background* Erosão 0,5 s | 22,5% | 15,4% | 3 | CPU |
| *Background Blobs* 1 s | 23,2% | 15,5% | 3 | CPU |
| *Background Blobs* 0,5 s | 34,0% | 15,5% | 2 | CPU |

Fonte: Autor

Nota-se que a utilização da CPU se torna o limite para todos os algoritmos. No tópico a seguir a capacidade de autonomia offline do sistema em caso de falta de internet é calculada.

## 5.3 Autonomia Offline

A autonomia offline se refere a capacidade do sistema, na falta de internet, armazenar as gravações localmente até que seja possível descarrega-las ao serviço de armazenamento online, e depende da capacidade de armazenamento local e da frequência com que são gravados vídeos, portanto também do número de câmeras e do tamanho de cada gravação. No Raspberry Pi foi constatado um espaço livre de 18,4 GB em um cartão de 32 GB com o sistema operacional e a biblioteca do OpenCV instalada, o que sugere um espaço necessário de até 14 GB para o funcionamento do sistema. Prevendo este mesmo espaço utilizado para cartões de diferentes tamanhos, foram calculados quanto tempo de gravação cartões de 16, 32 e 64 GB seriam capazes de armazenar utilizando as informações da tabela 15 de tamanho por segundo de gravação dos dois *codecs* de gravação em disco testados. Os resultados são apresentados na tabela 17.

Tabela 17 – Tempo de autonomia offline

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Cartão** | **Formato** | **Autonomia gravando 100% do tempo (horas)** | **Autonomia gravando 50% do tempo (horas)** | **Autonomia gravando 25% do tempo (horas)** |
| 16 GB | AVI | 3,3 | 6,6 | 13,2 |
| MP4 | 6,7 | 13,5 | 27,0 |
| 32 GB | AVI | 25,2 | 50,4 | 100,9 |
| MP4 | 51,7 | 103,5 | 206,9 |
| 64 GB | AVI | 69,1 | 138,2 | 276,4 |
| MP4 | 141,7 | 283,4 | 566,9 |

Fonte: Autor

Utilizando um cartão de 16 GB de armazenamento, há pouco espaço para as gravações, não sendo recomendado a utilização de múltiplas câmeras, pois qualquer indisponibilidade de internet poderia resultar em perda de informações. Caso seja necessário utilizar múltiplas câmeras, o cartão de 32 GB já possui armazenamento para algumas horas de indisponibilidade. Caso a disponibilidade de internet seja um problema constante, um cartão de 64 GB é mais adequado. No subcapítulo a seguir, as informações de todos os testes realizados são cruzadas.

## 5.4 Síntese dos Resultados Obtidos

Neste subcapítulo, os resultados dos testes realizados anteriormente serão associados para prover uma visão geral de cada algoritmo testado, disponíveis na tabela 18.

Tabela 18 – Resumo de desempenho dos algoritmos

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Algoritmo** | | **Intervalo (segundos)** | **Taxa de Acertos**  **(Sequência 5)** | | **Taxa de Falsos Positivos (Sequência 0)** | | **Uso da CPU/Câmera** | **Uso da RAM/Câmera** | **Limite Câmeras (AVI)** |
| **Pior** | **Melhor** | **Pior** | **Melhor** |
| **Subtração de *Frames*** | **Filtro de Erosão** | 0,5 | 30,0% | 100% | 15,0% | 2,5% | 7,3% | 12,9% | 4 |
| 1 | 15,0% | 100% | 23,3% | 8,3% | 10,4% | 12,9% | 5 |
| **Detecção de *Blobs*** | 0,5 | 7,7% | 61,5% | 1,7% | 0% | 22,4% | 13,0% | 3 |
| 1 | 7,7% | 76,9% | 0,8% | 0% | 13,2% | 13,0% | 4 |
| **Subtração de *Background*** | **Filtro de Erosão** | 0,5 | 0% | 88.4% | 32,5% | 3,3% | 15,1% | 14,8% | 3 |
| 1 | 0% | 84,6% | 40,0% | 5,0% | 9,9% | 14,8% | 4 |
| **Detecção de *Blobs*** | 0,5 | 7,7% | 65,3% | 5,8% | 3,4% | 26,6% | 14,9% | 2 |
| 1 | 46,1% | 76,0% | 21,7% | 1,7% | 15,8% | 14,9% | 3 |

Fonte: Autor

Considera-se a sequência 5 do vídeo teste a mais relevante devido a ter sido nesta que as taxas de acertos mais variaram, por ser a sequência com movimentos de interesse de menor intensidade. Dos 4 algoritmos testados em todas as suas 24 configurações, variando os *thresholds* e os intervalos de processamento, em apenas 3 configurações obteve-se uma taxa de acerto nas sequências de movimentos de média intensidade abaixo de 80%, e apenas 1 configuração apresentou taxa de acerto inferior a 90% nas sequências de movimentos de alta intensidade. O uso do processador e da memória RAM na tabela 18 refletem o consumo computacional individual dos algoritmos de detecção de movimento, e no limite de câmeras considera-se a gravação em formato AVI, pois a gravação em formato MP4 onera o processador a ponto de não ser possível utilizar mais de uma câmera de maneira confiável caso ocorram frequentes gravações.

# 6 Conclusões

Os algoritmos testados apresentaram resultados diversos entre a variedade de configurações diferentes de cada um. Nenhum deles se provou o melhor absoluto considerando as taxas de acertos com a carga computacional. O algoritmo de detecção de subtração de dois *frames* com filtro de erosão teve 100% de taxa de acerto na sequência de movimentos de baixa intensidade utilizando a matriz de elemento estruturante 5x5, sendo ideal para utilização quando é necessária alta confiabilidade da gravação de movimentos pequenos ou de objetos longe da câmera, tendo como preço a taxa de falsos positivos entre 15% e 25%. Foi também o algoritmo testado capaz de processar mais câmeras simultaneamente, permitindo a utilização de até 5 câmeras no sistema implementado.

O algoritmo de detecção de dois *frames* com detecção de *blobs* foi o que apresentou as menores taxa de falsos positivos em todas as configurações testadas, com média de 0,48%, e com *blob* de área mínima de 20 pixels e intervalo de processamento de 1 segundo ainda foi capaz de ter uma taxa de acerto de 76,9% na sequência de movimentos de baixa intensidade. Portanto este algoritmo poderia ser usado em um cenário onde existem bastantes focos de falsos positivos na cena, mas que apenas os movimentos de média ou alta intensidade são de interesse, ou ainda quando é de interesse gravar apenas objetos grandes ou perto da câmera, tendo capacidade para até 4 câmeras no sistema implementado.

O algoritmo de detecção de subtração de *background* com filtro de erosão apresentou, entre suas configurações de melhores taxas de acerto, entre 85% e 90% na sequência de baixa intensidade, e uma taxa de falsos positivos entre 30% e 40%, estes decorrentes principalmente devido a mudança de iluminação causado pelas nuvens. Para o cenário testado, este não teria utilização por não apresentar nenhuma vantagem em relação aos com subtração de dois *frames*, possuindo ainda maior custo computacional. É possível que este algoritmo apresentasse resultados melhores num ambiente fechado.

O algoritmo de detecção de subtração de *background* com detecção de *blobs*, apesar de apresentar resultados melhores que o com filtro de erosão,também sofreu com as mudanças de iluminação do ambiente externo, porém ao custo de ter sido o algoritmo mais oneroso, sendo capaz de suportar no máximo 3 câmeras simultaneamente, não encontrando nenhuma utilização com base nos testes realizados.

O cenário que foi utilizado para testes foi o pátio de uma residência, ambiente externo, com área privada e com distância para uma área pública, a rua, que pode vir a gerar alta frequência de movimentos, além de haver mudança de iluminação e movimentação de árvores, o que mostraram ser as principais fontes de falsos positivos. Para este cenário específico, existem duas opções: é possível considerar a área de interesse apenas a área privada, que se encontra perto da câmera, o que resultaria em todos os movimento de interesse sendo de objetos médios e grandes e de intensidade de movimento considerável, portanto o algoritmo detecção de dois *frames* com detecção de *blobs* se enquadraria bem devido as suas características de baixa taxa de falsos positivos e boas taxas de acertos. A outra opção seria considerar a rua como área de interesse, portanto sendo necessário a gravação dos movimentos que ocorrem longe da câmera, e o algoritmo de subtração de dois *frames* com filtro de erosão seria o mais indicado por possuir boa taxa de acerto de movimentos de baixa intensidade.

Para um ambiente interno, poderia ser utilizado um algoritmo com sensibilidade ainda menor, já que os objetos tendem a estar próximos da câmera, resultando em maiores mudanças entre os *frames*. Nessa situação, acredita-se que o volume de gravações seria consideravelmente menor que de um sistema CFTV convencional, por haver menos fontes de possíveis falsos positivos. Para um ambiente externo ou público, a redução no volume de gravações tende a ser menor, já que existe uma probabilidade de frequência maior de movimentos de interesse acontecendo.

O sistema facilita também o acesso do usuário para as gravações, já que as mesmas são disponibilizadas no Google Drive, permitindo acesso e gerenciamento para o usuário de qualquer local que ele esteja, sem necessidade de contato direto com o hardware do sistema. Porém há a restrição da necessidade de disponibilidade constante de acesso à internet para que o sistema realize o *upload* das gravações. Em caso de sua indisponibilidade, outro fator a ser considerado é a autonomia offline, esta que varia diretamente com o número de câmeras e da frequência com que ocorrem gravações, e, portanto, dos falsos positivos também.

A autonomia offline depende também da codificação com que as gravações são salvas, isto que também afeta diretamente a capacidade do Raspberry Pi processar várias câmeras. Dependendo da complexidade do algoritmo, é possível acoplar sem engasgos até 5 câmeras, usando o formato AVI que possui taxa de compressão satisfatório para um cartão SD de 32 GB ou mais. Para um cartão de capacidade de 16 GB, seria necessária uma compressão maior das gravações, o que necessitaria de uma codificação das imagens com maior perda de dados, e que não fosse tão oneroso quanto o formato MP4 testado, o qual durante a gravação ocupava a CPU em sua totalidade, tornando inviável a utilização de mais de uma câmera.

Portanto, sente-se a o espaço para estudar, em trabalhos futuros, as taxas de compressão e o custo computacional da gravação em arquivos de vídeo em um sistema embarcado, buscando o melhor custo benefício, além dos dois extremos de menor custo computacional e maior compressão do vídeo, já que estes influenciam diretamente a capacidade do sistema de processar várias câmeras e a capacidade de armazenamento de horas de gravação.

Destaca-se, enfim, que o sistema desenvolvido com o Raspberry Pi 3 Modelo B+, dependendo do cenário e do algoritmo utilizado, é capaz de reduzir o volume de gravações geradas e gravar consistentemente todos os movimentos de interesse, disponibilizando as gravações para acesso remoto ao usuário. O sistema suporta até 5 câmeras, tendo um custo inicial em torno de 450 reais com uma câmera IP e entre 100 e 150 reais por câmera adicional.

# REFERÊNCIAS

MIGLIORI, S. ABESE. **Associação Brasileira das Empresas de Sistemas Eletrônicos de Segurança**, 20 fev. 2018. Disponível em http://www.abese.org.br. Acesso em: 23/04/18.

UNDP. Site do Programa de Desenvolvimento das Nações Unidas. **Relatório Regional de Desenvolvimento Humano 2013-2014**, 2014. Disponível em: www.pg.undp.org/content/dam/rblac/docs/Research%20and%20Publications/IDH/UNDP-RBLAC-ResumoExecPt-2014.pdf. Acesso em: 23/04/18.

CUNHA, G. **Como lidar com as falhas de segurança pública**, 2016. Disponível em: http://revistasecurity.com.br/invasao-a-residencias-e-comercios-como-lidar-com-as-falhas-da-seguranca-publica. Acesso em 23/04/2018.

HODGSON, K. SDM Magazine. **State of the Market: Security & Monitoring 2018**, 2018. Disponível em: https://www.sdmmag.com/articles/94896-state-of-the-market-security-monitoring-2018. Acesso em 23/04/18.

Raspberry Pi Site. Site oficial do Rasberry Pi. **What is a Raspberry Pi**, 2018. Disponível em:https://www.raspberrypi.org/help/what-%20is-a-raspberry-pi. Acesso em 30/04/18.

Raspberry Pi Site. Site oficial do Rasberry Pi. **About us**, 2018. Disponível em:https://www.raspberrypi.org/about. Acesso em 30/04/18.

Raspian OS Site. Site oficial do Raspian OS. **Raspbian FAQ**, 2018. Disponível em: https://www.raspbian.org/RaspbianFAQ. Acesso em 30/04/18.

Raspberry Pi Site. Site oficial do Rasberry Pi. **SD CARDS**, 2018. Disponível em: https://www.raspberrypi.org/documentation/installation/sd-cards.md. Acesso em 30/04/18.

ALMEIDA, C. Instituto CFTV. **Câmeras IP**, 2016. Disponível em: www.institutocftv.com.br/cameras-ip-2.html. Acesso em 04/05/18.

KIM, P. Github Blog. **IT Security Research**, 2017.Disponível em: https://pierrekim.github.io/blog/2017-03-08-camera-goahead-0day.html. Acesso em 04/05/18.

Unifore Site. Blog da Unifore. **TCP vs UDP, Why use UDP for IP Camera's Connection?** 2017. Disponível em: https://www.unifore.net/ip-video-surveillance/tcp-vs-udp-why-use-dup-for-ip-camera-s-connection.html. Acesso em 04/05/18.

KIRILLOV, A. Code Project. **Motion Detection Algorithms**, 2007. Disponível em: https://www.codeproject.com/Articles/10248/Motion-Detection-Algorithms. Acesso em 04/05/18.

HORVATH, M. **PROV-Ray Projects**, 2017. Disponível em: http://isometricland.net/povray/povray.php. Acesso em 17/05/2018.

ITU-R Site. Site da União Internacional de Telecomunicações. **Recommendation ITU-R BT.601-7**, 2011. Disponível em:https://www.itu.int/dms\_pubrec/itu-r/rec/bt/R-REC-BT.601-7-201103-I!!PDF-E.pdf. Acesso em 17/05/18.

MARJANOVIC, M*.* **What is Y'CBCR**? 2003. Disponível em: http://www.mir.com/DMG/ycbcr.html. Acesso em 17/05/18.

FISHER, R; PERKINS, S; WALKER, A; WOLFART, E. **Structuring Elements.** 2003. Disponível em: https://homepages.inf.ed.ac.uk/rbf/HIPR2/strctel.htm. Acesso em 09/10/18.

GONZALEZ, R; WOODS, R. **Digital Image Processing**. 3ª Edição. Pearson, New Jersey, 2007.

LEVIN, G. **Image Processing and Computer Vision**, 2018. Disponível em: https://openframeworks.cc/ofBook/chapters/image\_processing\_computer\_vision.htm. Acesso em 18/05/18.

CASTRO, F; SILVA, F; PAZOTI, M; ARTERO, A. **Identificação e Rastreamento de Veículos Utilizando Fluxo Óptico**, 2015. Disponível em: https://www.researchgate.net/publication/293013645\_IDENTIFICACAO\_E\_RASTREAMENTO\_DE\_VEICULOS\_UTILIZANDO\_FLUXO\_OPTICO. Acesso em 20/05/18.

Open Source Compution Vision. Site do OpenCV. **How to Use Background Subtraction Methods**, 2018. Disponível em: https://docs.opencv.org/3.4/d1/dc5/tutorial\_background\_subtraction.html. Acesso em 18/05/18.

MathWorks Site. Site da central MATLAB. **Canny Edge Detector**, 2013. Disponível em: https://www.mathworks.com/matlabcentral/fileexchange/40737-canny-edge-detector. Acesso em 17/05/18.

PHILLIPS, D. **Image Processing in C**. 2º Edição. R & D Publications, Kansas, 2010. Livro Eletrônico.

MALLICK, S. **Blob Detection Using OpenCV**, 2015. Disponível em: https://www.learnopencv.com/blob-detection-using-opencv-python-c. Acesso em 07/06/18.

Open Source Computer Vision. Site do OpenCV. **Image Gradients**, 2014. Disponível em: https://docs.opencv.org/3.0-beta/doc/py\_tutorials/py\_imgproc/py\_gradients/py\_gradients.html. Acesso em 18/05/18.

GREEN, B. **Canny Edge Detection Tutorial**, 2002. Disponível em: http://dasl.unlv.edu/daslDrexel/alumni/bGreen/www.pages.drexel.edu/\_weg22/can\_tut.html. Acesso em 18/05/18.

BLACK, M. **Optical Flow**, 2014. Disponível em: https://www.youtube.com/watch?v=tIwpDuqJqcE. Acesso em 22/05/18.

HAINES, R; CHUANG, L. **The effects of video compression on acceptability of images for monitoring life sciences experiments**, 1992. Disponível em: https://ntrs.nasa.gov/archive/nasa/casi.ntrs.nasa.gov/19920024689.pdf. Acesso em 24/05/2018.

RABIE, T; KAMEL, I. **On the Embedding Limits of the Discrete Cosine Transform**, 2015. Disponível em: https://www.researchgate.net/publication/ 275641616\_On\_the\_embedding\_limits\_of\_the\_discrete\_cosine\_transform. Acesso em 24/05/18.

POUND, M; RILEY, S. **JPEG DCT, Discrete Cosine Transform (JPEG Pt2)- Computerphile,** 2015. Disponível em: https://www.youtube.com/watch?v= Q2aEzeMDHMA. Acesso em 25/05/18.

PENNEBAKER, W; MITCHELL, J. **JPEG: Still Image Compression Standard**. 1ª Edição, Massachusetts, 1992. p. 107-108.

Site de suporte do Google Drive. Site oficial de suporte aos produtos Google. **Clear Google Drive space & increase storage**, 2018. Disponível em: https://support.google.com/drive/answer/6374270?hl=en. Acesso em 24/05/18.

Site do Rclone. **Rclone – rsync for cloud storage**, 2017. Disponível em: https://rclone.org. Acesso em 24/05/18.

BISWAS, S. **A Guide to the Linux “Top” Command**, 2018. Disponível em: https://www.booleanworld.com/guide-linux-top-command. Acesso em 24/05/18.

ALAVI, S. **Comparison of Some Motion Detection Methods in cases of Single and Multiple Moving Objects**, 2012. Disponível em: https://www.researchgate.net/publication/306357420\_Comparison\_of\_Some\_Motion\_Detection\_Methods\_in\_cases\_of\_Single\_and\_Multiple\_Moving\_Objects. Acesso em 24/05/18.