**Abstract—The high people concentration at large cities, as well the population aging, makes lost people occurrences (children or elderly) a recurring problem in the everyday life. This article proposes the development of a prototype, integrating Computer Vision and Internet of Things to assist the detection of people using their characteristics (like glasses, hair color, or clothes).** **The proposed solution will allow the implementation of a system with many nodes and a central control, where its differential will be the MQtt protocol use for real time sending of new image recognition models for each node, as well as sending back the captured images at the nodes, each time the target detection is positive.**

Busca de pessoas em tempo real com OpenCV em placa Raspberry PI

Kleber de Mattos Dobrowolski

***Index Terms*—Raspberry, OpenCV, Computational Vision**

*Resumo*—A alta concentração de pessoas nas grandes cidades, bem como o envelhecimento da população em geral, faz com que situações de pessoas perdidas (crianças ou idosos) seja um problema recorrente no cotidiano. Este artigo tem como proposta o desenvolvimento de um protótipo que integra Visão Computacional e Internet das Coisas para auxiliar na busca de pessoas através de características (sejam óculos, cor de cabelo ou peça de vestuário). A solução proposta permitirá a implantação de um sistema com muitos nós e um controle central, onde seu diferencial será a utilização do protocolo MQtt para envio, em tempo real, de novos modelos de reconhecimento de imagem para cada nó, assim como o envio das imagens capturadas pelos nós, a cada vez que a detecção do alvo for positiva.

**Palavras chave—Raspberry, OpenCV, Visão Computacional**

# INTRODUÇÃO

Nos grandes centros urbanos a ocorrência de pessoas perdidas (tais como crianças desgarradas dos pais ou idosos senis que se afastaram do acompanhante) em estabelecimentos de grande área tais como metrô, *shopping centers* e museus não é incomum [1]. Porém, mesmo com a existência de grande quantidade de equipamentos de vigilância, tais como câmeras, a busca prática por estas pessoas não é eficiente, já que estas câmeras no máximo são utilizadas para busca visual manual por operadores humanos. Para efetuar estas buscas, normalmente são utilizadas características físicas simples, tais como: se o sujeito da busca é uma criança ou adulto; cor da roupa; uso de óculos ou laço no cabelo; características físicas como calvície ou cabelos ruivos.

Com os avanços tecnológicos dos anos recentes, tecnologias que antes costumavam ser caras e dependentes de *hardware* dedicado, além de lentas (como o uso de reconhecimento de

Trabalho de Conclusão de Curso apresentado ao Instituto Nacional de Telecomunicações, como parte dos requisitos para a obtenção do Certificado de Pós-Graduação em Internet das Coisas (IoT). Orientador: Prof. Felipe Andery Reis. Trabalho aprovado em julho/2019.

imagens por computador), hoje estão mais acessíveis financeiramente.

Este baixo custo, aliado à miniaturização destes equipamentos, possibilitou a difusão de soluções para Internet das Coisas – IoT, onde diversos equipamentos mais simples são capazes de, operando em conjunto, atingir resultados tão bons quanto o de caros sistemas centralizados. Além disto, as várias técnicas do campo de Inteligência Artificial e Visão Computacional também sofreram avanços e estão mais difundidas, através de bibliotecas de domínio público, que já provêm os principais algoritmos necessários nestas áreas.

O projeto proposto consiste na criação de um sistema de localização de pessoas utilizando abordagem IoT e visão computacional através do reconhecimento de imagens, trazendo como proposta um *hardware* de baixo custo, a comunicação remota em tempo real, e o aproveitamento de processamento em nuvem. O nome sugerido para o sistema é Argos: referência ao monstro grego mitológico com cem olhos. O protótipo terá como foco a integração do *hardware* aos conceitos de IoT, sendo a criação dos modelos de reconhecimento de imagem fora do escopo do trabalho.

Este trabalho será dividido da seguinte forma: a seção II apresenta os trabalhos relacionados, na seção III serão abordadas as tecnologias utilizadas, e na seção na IV será detalhado o desenvolvimento. Na última seção, as conclusões e sugestões para desenvolvimentos futuros serão apresentadas.

# trabalhos relacionados

Serão listados, à seguir, alguns trabalhos utilizados como base para o desenvolvimento deste artigo.

A abordagem de utilização de equipamento de baixo custo para reconhecimento facial já foi apresentada por Chen *at al.* [2] e por Nguyen *at al.*[3], portanto o problema de busca de pessoas utilizando procedimento similar parece ser uma tarefa factível, dado que a complexidade do problema proposto é bem menor (foco em detecção de formas e adereços, do que em um rosto específico).

Em relação às diversas bibliotecas de processamento digital de imagens existentes, tais como a Dlib (criada por Davis E. King) e a *Open Source Computer Vision Library* (OpenCV - originalmente desenvolvida pela Intel), onde a segunda apresenta melhor performance, segundo análise de Boyko e Shakhovska [4]. É importante ressaltar que a biblioteca OpenCV permite o aproveitamento da Unidade de Processamento Gráfico (GPU) da fabricante NVIDIA, com aumento notável na velocidade de processamento das imagens [5]. No campo dos algoritmos para detecção de faces e também rastreamento de alvos há diversos trabalhos, como o de Alcantara [6] que aborda o uso do algoritmo *Haar-like*, já implementado dentro da OpenCV. Uma abordagem concorrente é a biblioteca *TensorFlow* da Google, que tem sido utilizada para identificação de objetos usando algoritmos de redes neurais [7].

Por último, Nuratch apresenta um guia para utilização de protocolo MQtt em sistemas embarcados [8].

# tecnologias utilizadas

Uma breve descrição das tecnologias que foram empregadas neste projeto serão apresentadas nesta seção.

O sistema proposto foi composto por diversos nós, cada um destes contendo uma placa Raspberry PI 3 [9][8], acoplada à sua câmera modular (sensor V2, de 8 MP de resolução), utilizando comunicação por rede WiFi e protocolo MQtt para controlar a atualização dinâmica do algoritmo de busca. O processamento de imagens é realizado utilizando as funções da biblioteca OpenCV [10]. Na Figura 1, um esquema de como estas tecnologias interagem entre si.

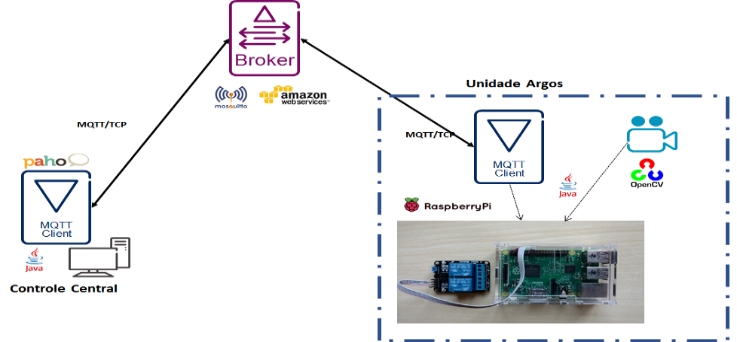


Fig. 1. Esquema de uma unidade (nó)

## JAVA e PAHO MQtt

A decisão pela linguagem de programação Java foi baseada em sua ótima portabilidade, bem como maturidade e facilidades de *debug* no computador *desktop* e posterior implantação no *hardware* IoT. A existência de rotinas para paralelização de *software* nativas também foram um fator decisivo. Conjuntamente, foi escolhida a biblioteca PaHo MQtt [11] em Java para comunicação leve entre os nós do sistema. Ressalta-se que esta biblioteca também permite a integração deste projeto com ambientes de nuvem comerciais, tais como o Amazon AWS [18].

## Hardware IoT

Foi utilizada a placa de desenvolvimento *Raspberry* PI 3, amplamente em uso em projetos de IoT similares. Apenas de pequena e portátil, esta placa tem poder computacional similar ao de computadores *desktop*. Porém, nota-se que a escolha da linguagem Java permite a implementação deste projeto em processadores IoT similares, tais como a placa *SnapDragon* da *Qualcomm* [12]ou a placa *Jetson Nano* da Nvidia [13].

A câmera acoplada também permite flexibilidade: podem ser utilizadas desde câmeras USB genéricas até módulos específicos para *hardware* IoT, como a câmera *Raspberry* PI V2, ou mesmo câmeras com capacidade para visão noturna.

## Processamento de imagens

A biblioteca OpenCV foi utilizada para o processamento de imagens e controle da câmera, embora seja possível o uso de bibliotecas concorrentes tal como *Tensor Flow* [14]. Além do pré-processamento básico aplicado nas imagens recebidas pela câmera (conversão para tons de cinza, ajustes de contraste, entre outros), a OpenCV também implementa técnicas de *Machine Learning* chamadas de classificadores Haar Cascade [15]. Estes classificadores podem ser utilizados na detecção dos mais variados tipos de objetos dentro de imagens, tais como aviões ou automóveis. Neste projeto, utilizou-se os modelos já existentes na biblioteca, voltados para detecção de características humanas: rostos, olhos, óculos e até mesmo barba. A OpenCV disponibiliza chamadas para a criação de novos modelos, as chamadas *Cascade Classifier Training*. Assim, utilizando um conjunto de imagens para treinamento (já previamente rotuladas: vestido, mochilas e óculos, por exemplo) e após considerável tempo de processamento, a saída resultante é um arquivo .XML, o qual, utilizado em conjunto com as chamadas de execução da OpenCV, permitem dizer se uma imagem de entrada contém ou não os elementos desejados. Justamente devido ao alto custo computacional para gerar novos modelos .XML específicos, se propõe o uso de computação em nuvem, enquanto que a execução dos modelos é facilmente possível dentro de CPUs mais simples, como uma Raspberry PI.

Convém notar que até mesmo a criação de um modelo .XML para um rosto específico, em tempo real, é possível, embora esta abordagem esteja fora do escopo do presente projeto por sua elevada complexidade. O foco aqui é a utilização dos modelos genéricos já existentes na biblioteca.

# Desenvolvimento

Nesta seção será ilustrado um exemplo de como seria a operação do sistema no mundo real, além de uma descrição dos comandos básicos criados para operação dos nós e recepção de imagens capturadas, além de detalhes sobre a flexibilidade do protocolo MQtt, e como o mesmo pode ser utilizado para segmentar os nós em diversas regiões distintas (exemplo: *hall* de entrada do estabelecimento, estacionamento e praça de alimentação). Por último será apresentada uma interface simplificada (GUI) que foi criada para demonstração do sistema.

## Cenário De Operação

O projeto proposto é flexível o bastante para atender muitos cenários operacionais (monitoração de metrô, sistemas distribuídos em ônibus, e outros). Para um exemplo mais concreto, segue uma descrição de operação hipotética em um *shopping center*, onde:

⇨ Há diversos nós distribuídos pelos departamentos do *shopping center*, sejam acoplados à câmeras de segurança já existentes ou substituindo as mesmas (Figura 2). Estes nós comunicam-se utilizando a rede TCP/IP preexistente. Na Central de Controle (CC) existe um *hardware* mais poderoso, para a operação centralizada dos nós e, opcionalmente, com ferramentas mais poderosas de *machine learning*, para criar novos perfis baseados em fotos, em poucos minutos.

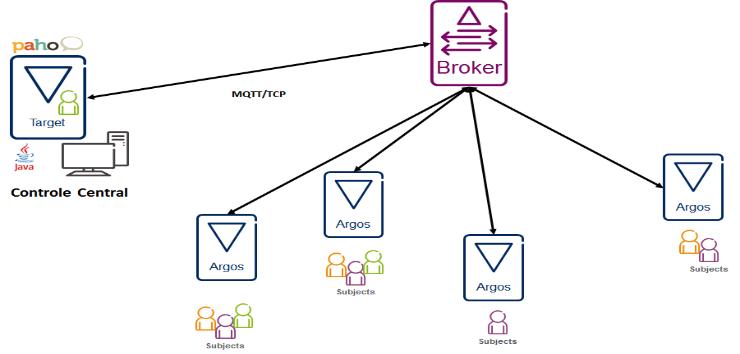


Fig. 2. Diversos nós Argos, sob um Controle Central

⇨ A criação dos modelos de busca, através de técnicas de *machine learning*, geralmente é realizado a priori, utilizando o poder computacional de processamento em nuvem, sendo o resultado obtido em um arquivo XML, acoplado a uma classe Java para detecção. Este conjunto pode então ser injetado em tempo real nos nós, que passarão a buscar o novo sujeito, bastando para isto apenas executar o modelo atualizado (o que exige baixo poder de processamento), enviando os resultados das buscas para uma Central de Controle (CC), tal como exemplificado na Figura 3.



Fig. 3. Resultado ilustrativo de busca por **criança** vestida de cor-de-rosa

⇨ A CC recebe uma demanda: uma criança de aproximadamente 4 anos, vestida de vermelho, perdeu-se da mãe. Uma foto da criança é repassada ao CC, através do celular de um dos seguranças do *shopping*. O operador decide qual a melhor abordagem: criar um modelo ‘haarcascade’ novo (consistindo de uma classe Java + arquivo xml) personalizado com a foto da criança, ou apenas utilizar um dos modelos padrão pré-existentes, porém ajustado (‘criança’ + ‘roupa vermelha’).

⇨ Imediatamente, os nós recebem a nova classe Java através de MQtt, compilam e carregam na memória, passando a executá-la. As câmeras que detectarem crianças, e que se enquadrem nas novas características, começarão a enviar imagens periodicamente (arquivos no formato .JPG) com a localização do setor e a hora, para o CO.

⇨ Como, provavelmente, existirão algumas detecções falso-positivo, um operador do CO fará a inspeção visual final, para confirmar se a criança foi realmente encontrada. Se sim, enviará a foto da detecção para o segurança mais próximo do setor, que poderá abordar a criança e trazê-la com segurança ao CO (ou acompanha-la diretamente ao encontro de sua mãe).

## Preparação da placa Raspberry

O modelo utilizado na construção do protótipo foi uma placa Raspberry PI 3, modelo B, com cartão de memória de 16 Gb. Foi instalada a imagem do sistema operacional *Raspbian Stretch Lite*, e em seguida baixados os softwares adicionais, e a biblioteca OpenCV, que foi compilada localmente. A compilação da biblioteca OpenCV levou aproximadamente 6 horas.

## Comandos para Controle dos nós

O protocolo MQtt é bem simples e leve, consistindo basicamente de dois campos principais, um campo para o tópico das mensagens enviadas/recebidas (*topic*) e outro para os dados em si (*payload*). Neste projeto foram implementados alguns comandos bem simples, através do campo *topic* do protocolo, fazendo o *parsing* deste campo para identificar diretamente o comando recebido, e utilizando o campo *payload* quando necessário.

• ***exit*** e ***restart***: desligam e reiniciam, respectivamente, o programa Java. Não necessitam do campo *payload*.

• ***read*** e ***write***: permitem ler e escrever parâmetros do arquivo de configuração (chamado argos.properties). Exemplos destes parâmetros: endereço IP do *broker* MQtt, ID individual do nó, nome da classe Java OpenCV utilizada na detecção de imagens. A mudança de alguns destes parâmetros demandam reinício do programa, tal como mudança do endereço IP do *broker*. Isto é realizado através do comando *restart*, acima. O novo conteúdo dos parâmetros é recebido diretamente na parte *payload* do protocolo MQtt.

• ***snapshot***: força o nó individual a enviar uma foto instantânea, com *upload* através de protocolo MQtt, dentro do campo de *payload* (observação: pode existir limitação de tamanho da mensagem, dependendo do *broker* utilizado).

• ***image***: o nó Argos utiliza este *subject* para enviar uma imagem .JPG de um alvo detectado, cujos *bytes* são enviados diretamente dentro do campo *payload*. Este comando também é utilizado como resposta, quando uma requisição de ***snapshot*** chega ao nó.

• ***class***: comando para receber um novo modelo OpenCV, implementado como uma classe Java. Este arquivo será imediatamente compilado dentro da placa *Raspberry* PI, e imediatamente carregado através da técnica de Java *Reflexion* [16]. Assim, em questão de segundos, o nó já mudará o alvo a ser detectado.

• ***status*** e ***error***: são enviados pelos nós Argos, com informações como o início de operação de um nó, a troca da classe de detecção da câmera e também para reportar erros graves (como a chegada de uma classe Java defeituosa, por exemplo).

## Escopo dos comandos MQtt

A utilização do protocolo MQtt permite grande flexibilidade no envio de comandos: pode-se definir escopos como nós individuais, grupos de nós, ou todos os nós. Para enviar comandos para um nó específico usa-se o Id do mesmo (/região\_1/zona\_1/nó\_2/restart). Também é possível enviar comandos para todos os nós acoplados ao sistema (/região\_1/snapshot), bastando para isto escolher o tópico MQtt de nível mais genérico, conforme pode-se ver na Figura 4.

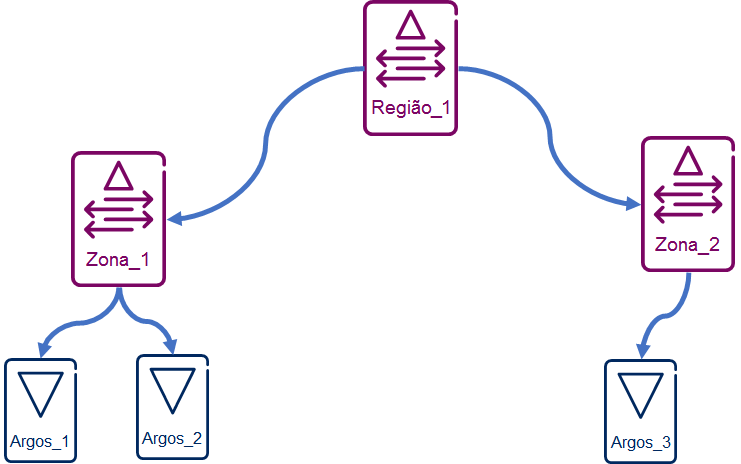


Fig. 4. Esquema de tópicos, para divisão por regiões

## Descrição da interface de usuário

Uma interface gráfica rudimentar foi criada, apenas para possibilitar a visualização de fotos recebidas pelos nós, de maneira rápida. Ela também permite o envio de classes Java, para serem compiladas e entrar em operação imediata dentro dos nós, conforme pode ser visto na Figura 5.

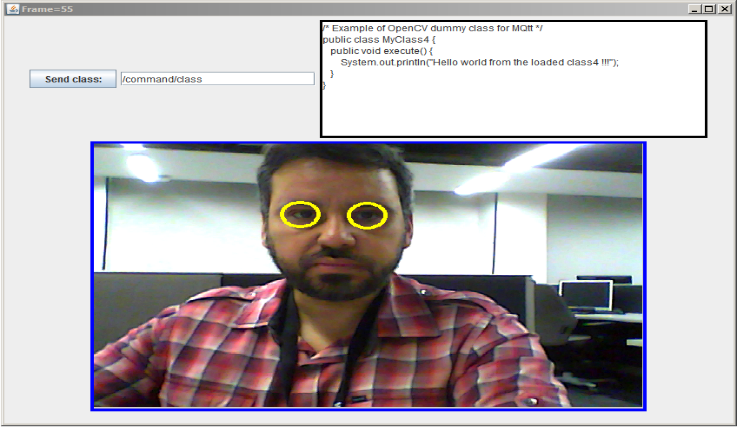
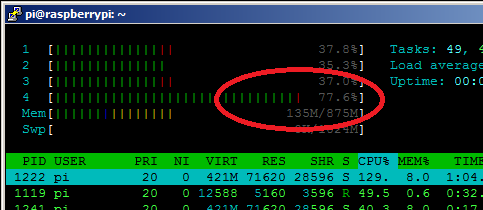


Fig. interface gráfica do projeto, com modelo de detecção de olhos

## Utilização das CPUs

Durante os testes pôde-se constatar, através do comando ‘*top*’ do Linux, que o sistema utiliza apenas 1 dos 4 núcleos disponíveis na placa Raspberry PI (Figura 6). Embora o programa já faça uso de *threads* para possibilitar o uso paralelo dos componentes, tal como o suporte MQtt desacoplado da rotina de visão computacional, esta abordagem ainda é considerada tímida e existe espaço para melhorias.



*Fig. 6. Uso intensivo de apenas uma única CPU*

# Conclusão e Desenvolvimentos futuros

Conforme as expectativas, foi possível demonstrar a viabilidade da utilização de *hardware* IoT, em conjunto com bibliotecas de processamento gráfico de uso livre, para a criação deste sistema compacto de buscas. A compilação e carga da classe Java, em tempo real, foi possível. E a placa Raspberry conseguiu um desempenho satisfatório, sem problemas como sobreaquecimento ou travamentos, embora a instalação e compilação da biblioteca OpenCV tenha se mostrado uma tarefa bastante desafiadora.

Algumas sugestões foram enumeradas para futura continuação deste projeto, algumas que demandarão maior esforço, porém, algumas pequenas e pontuais o bastante para serem implementadas em curto espaço de tempo.

### Melhorias na GUI

Um dos principais avanços para o atual projeto será a criação de uma interface gráfica de usuário mais amigável e intuitiva, capaz de ser utilizada por pessoas comuns. Hoje, mesmo um programador experiente levará algum tempo até entender a interface e poder operá-la de maneira ágil. Para um projeto real, tal como monitoramento de uma estação de metrô, seria necessário o uso de diversos monitores, bem como maneiras mais simples de introduzir fotos e alterar características básicas (cor, objetos de uso comum como óculos, bolsa, adereços no cabelo, vestimentas).

### Integração com nuvem/GPU

O projeto atual é bastante simplificado, utilizando exemplos genéricos Java disponíveis em projetos *open source*, tais como o *haarcascade\_eye.xml* para detecção de olhos. Porém a gama de ferramentas e ambientes em nuvem está expandindo-se a cada ano. Vislumbra-se uma integração deste sistema com ambientes profissionais em nuvem para, por exemplo, se enviar uma fotografia de uma pessoa, e o seu rosto ser imediatamente treinado em uma rede neural, gerando em poucos minutos um novo modelo .xml totalmente personalizado. Além disto, os ambientes comerciais, tais como *Amazon* AWS, já permitem o uso de GPU *cluster*[17], para acelerar a performance destes treinamentos.

Já do lado do *hardware* IoT, convém relembrar que a biblioteca OpenCV permite a utilização da GPU para aumentar o desempenho das operações, bastando que isto seja implementado no *software*. Isto tanto pode ser aproveitado para reduzir o consumo de energia dos nós, quanto para aumentar a quantidade de alvos detectados ou a precisão e qualidade da detecção.

### Processamento paralelo e empilhamento de classes

É bastante factível que o programa possa ser paralelizado, em busca de melhor aproveitamento do *hardware* da placa Raspberry que, apesar de tamanho reduzido, é de fato bastante poderosa. O esforço de programação não será grande, já que a linguagem Java já embute diversas rotinas auxiliares para processamento paralelo. Embora a captura das imagens pela câmera não seja uma tarefa paralelizável, as rotinas da biblioteca OpenCV podem ser utilizadas para processar *frames* em paralelo, simultaneamente.

Assim, seria possível a criação de uma pilha de classes Java, cada uma dedicada a um reconhecimento de imagem distinto, podendo tanto ser executadas em diferentes CPUs, quanto compartilhar o tempo de processamento em modo *time sharing*. Isto abre a possibilidade do sistema executar reconhecimentos como ‘busca criança perdida’, ‘busca idoso de óculos’, ‘contagem de pessoas com sacola de comprar’, todos ao mesmo tempo. Os modelos de reconhecimento seriam empilhados, utilizados, e em seguida removidos da pilha quando não mais necessário, através de comandos MQtt.

Convém notar que mais testes serão necessários para descobrir os verdadeiros limites da placa Raspberry, já que o uso intensivo da CPU causará efeitos como aumento do consumo de energia, além de aumento da temperatura dos núcleos.

### Limitações do MQtt

O protocolo MQtt, sendo originalmente voltado para pequenos processadores embarcados, tem algumas limitações, sendo a principal o tamanho máximo do *payload*. Para *brokers* como o *Mosquitto*, este tamanho é bastante confortável, de aproximadamente 255 MB[18]. Já para o *broker* da *Amazon AWS* existe um limite bastante pequeno de apenas 128 kB [19]. Este limite vai exigir que as fotos coletadas pelos nós do sistema sejam divididas em blocos menores, e reintegradas na interface gráfica ou no controle central. As classes Java ou modelos .xml podem também necessitar este tipo de tratamento. A integração com protocolos de dados como *ftp* também pode ser utilizada como estratégia para salvar as imagens.

Referências

1. J. Sie, S. Yang, Z. Hong, C. Liu, J. Chen and S. C. Li, "Integrating Cloud Computing, Internet-of-Things (IoT), and Community to Support Long-Term Care and Lost Elderly Searching," *2016 International Computer Symposium (ICS)*. [Chiayi, 2016, pp. 452-457]
2. Chen, Yong-Ping at al. “*Low-Cost Face Recognition System Based on Extended Local Binary Pattern*”, in International Conference on Automatic Control Conference, p. 13–18, novembro. 2016.
3. Nguyen. Huu-Quoc at al. Low Cost Real-Time System Monitoring Using Raspberry Pi. Em 2015 Seventh International Conference on Ubiquitous and Future Networks, 7-10 July 2015Aaba
4. N. Boyko, O. Basystiuk and N. Shakhovska, “Performance Evaluation and Comparison of Software for Face Recognition, Based on Dlib and Opencv Library” *IEEE Second International Conference on Data Stream Mining & Processing (DSMP)*, Lviv, pp. 478-482, 2018
5. K. Preethi and K. S. Vishvaksenan, "Gaussian Filtering Implementation and Performance Analysis on GPU," *2018 International Conference on Inventive Research in Computing Applications* (ICIRCA), [Coimbatore, 2018, pp. 936-939].
6. G. K. L. Alcantara, I. D. J. Evangelista, J. V. B. Malinao, O. B. Ong, R. S. D. Rivera and E. L. U. Ambata, "Head Detection and Tracking Using OpenCV," 2018 *IEEE 10th International Conference on Humanoid, Nanotechnology, Information Technology,Communication and Control, Environment and Management (HNICEM)*, [Baguio City, Philippines, 2018, pp. 1-5.]
7. R. Phadnis, J. Mishra and S. Bendale, "Objects Talk - Object Detection and Pattern Tracking Using TensorFlow," *2018 Second International Conference on Inventive Communication and Computational Technologies (ICICCT)*, [Coimbatore, 2018, pp. 1216-1219]
8. S. Nuratch, "Applying the MQTT Protocol on Embedded System for Smart Sensors/Actuators and IoT Applications," *2018 15th International Conference on Electrical Engineering/Electronics, Computer, Telecommunications and Information Technology (ECTI-CON)*, [Chiang Rai, Thailand, 2018, pp. 628-631].
9. Raspberry Pi 3 Model B specification [Online]. Disponível: https://www.raspberrypi.org/products/raspberry-pi-3-model-b/
10. OpenCV (Open Source Computer Vision Library) [Online]. Disponível: https://opencv.org/about/
11. Eclipse Paho Java Client guide [Online]. Disponível: https://www.eclipse.org/paho/clients/java/
12. Qualcomm Snapdragon 8 Series Mobile Platform [Online]. Disponível: https://www.qualcomm.com/snapdragon
13. Nvidia Jetson Nano [Online]. Disponível: https://www.nvidia.com/en-us/autonomous-machines/embedded-systems/jetson-nano/.
14. Introduction to TensorFlow [Online]. Disponível: https://www.tensorflow.org/learn
15. Haar Feature-based Cascade Classifiers [Online]. Disponível: https://docs.opencv.org/3.4.1/d7/d8b/tutorial\_py\_face\_detection.html
16. Using Java Reflection [Online]. Disponível: https://www.oracle.com/technetwork/articles/java/javareflection-1536171.html
17. V. Campos, F. Sastre, M. Yagües, J. Torres and X. Giró-I-Nieto, "Scaling a Convolutional Neural Network for Classification of Adjective Noun Pairs with TensorFlow on GPU Clusters" *IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGRID)*, Madrid, pp. 677-682, 2017.
18. Eclipse Mosquitto™ manual and details [Online]. Disponível: <https://mosquitto.org/man/mosquitto-conf-5.html>
19. Amazon Web Services, General Reference (Version 1.0). Disponível: <https://docs.aws.amazon.com/general/latest/gr/aws_service_limits.html#limits_iot>

AUTOR

**Kleber de Mattos Dobrowolski** nasceu em Taubaté, SP, em 03 de novembro de 1975. Possui os títulos de Bacharel em Computação Científica (UNITAU, 2000) e Mestrado em Computação Aplicada (Instituto Nacional de Pesquisas Espaciais - INPE, 2004).

De 2003 a 2006 foi analista de sistemas no INPE, na Divisão de Desenvolvimento de Sistemas Solo - DSS. Trabalha desde 2006 na Ericsson Brasil, em desenvolvimento de software para bilhetagem de aparelhos celulares. Tem interesse nas áreas de Redes Neurais, Computação Aplicada e Internet das Coisas.