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

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

Kleber de Mattos Dobrowolski

*Resumo*—Este artigo descreve um projeto integrando visão computacional através da biblioteca OpenCV, junto a características do mundo IoT: hardware de baixo custo (placa Raspberry PI) e um protocolo de comunicação leve (MQtt). O baixo custo da solução permite a implantação de um sistema com muitos nós e um controle central. Seu diferencial é 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.

# 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 (buscar ref). Porém, mesmo com a existência de grande quantidade de equipamentos de vigilância, tais como câmeras, isto não auxilia na busca prática destas pessoas, 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.

O projeto proposto tem nome sugerido de Argos (referência ao monstro grego mitológico com cem olhos) e consiste na criação de um sistema de busca de pessoas utilizando abordagem IoT (*Internet of Things*): hardware de baixo custo, comunicação remota em tempo real, e aproveitamento de processamento em nuvem. O sistema será composto por diversos nós, cada um contendo uma placa *Raspberry* PI 3, acoplada à sua câmera modular (sensor OV5647, de 5 MP de resolução), utilizando comunicação por rede *WiFi* e protocolo *MQTT* para controlar a atualização dinâmica do algoritmo de busca (modelo). O processamento de imagens é realizado utilizando as funções da biblioteca de domínio público OpenCV (*Open Source Computer Vision Library*). A abordagem de utilização de hardware de baixo custo para reconhecimento facial já foi apresentada anteriormente por *Chen at al.* [1] e por *Nguyen at al.* [2], portanto o problema de busca pessoas utilizando processo similar parece ser uma tarefa factível, dado que a complexidade do problema proposto é bem menor (já que estamos mais interessados em formas e adereços, do que em um rostos específico).

Convém notar que o foco deste projeto é a integração do *harware* aos conceitos de IoT, sendo a criação dos modelos de reconhecimento de imagem em si, fora do escopo do trabalho. Porém, observe-se que a criação destes modelos, frequentemente através de técnicas de *machine learning*, poderá ser feita utilizando o poder computacional de processamento em nuvem, podendo o resultado obtido (modelo XML) ser injetado em tempo real nos nós, que passarão a buscar o novo sujeito do modelo atualizado, enviando os resultados das buscas para uma Central de Controle (CC).



Fig. 1. Resultado ilustrativo de busca por criança vestida de cor-de-rosa

# Cenário De Operação

O projeto é flexível o bastante para atender muitos cenários operacionais (monitoração de metrô, sistemas distribuídos em ônibus, etc). Para um exemplo mais concreto, descreveremos a seguir uma operação hipotética em um shopping center, onde:

⇨ Há diversos nós distribuídos pelos departamentos do shopping center, sejam acoplados a câmeras de segurança já existentes ou substituindo as mesmas. Estes nós comunicam-se utilizando a rede TCP/IP preexistente. Na Central de Controle (CC) existe um *harware* 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.

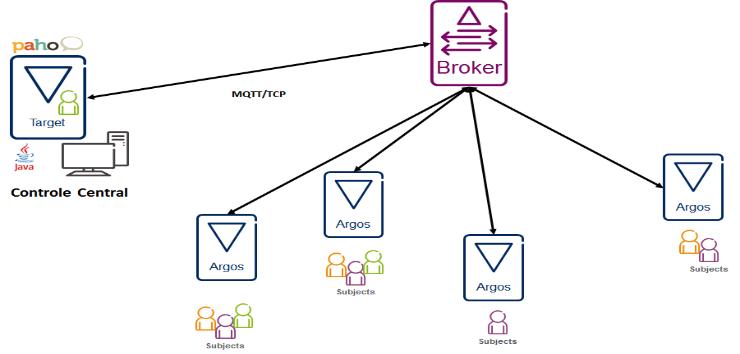


Fig. 2. Diversos nós Argos, e um Controle Central

⇨ 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 de 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).

1. DESENVOLVIMENTO
   1. *Escolhas na implementação*

A decisão pela linguagem de programação Java foi baseada em sua fácil portabilidade, bem como maturidade e facilidades de *debug* no computador *desktop* e posterior implantação no hardware IoT. Rotinas para paralelização de software nativas também foram um fator decisivo.Conjuntamente, foram escolhidas as biblioteca PaHo MQtt para comunicação, tal como a biblioteca OpenCV para o processamento de imagens e controle da câmera, embora seja possível o uso de bibliotecas concorrentes como *Tensor Flow*. O *hardware* utilizado foi a placa *Raspberry* PI 3, porém note-se que a escolha da linguagem Java permite a utilização deste projeto em dispositivos IoT similares, tais como a placa *SnapDragon* da *Qualcomm*. A câmera acoplada também permite flexibilidade: podemos utilizar desde câmeras USB genéricas, até mesmo módulos específicos para hardware IoT, como a Câmera *Raspberry* PI V2 ou mesmo câmeras com visão noturna.

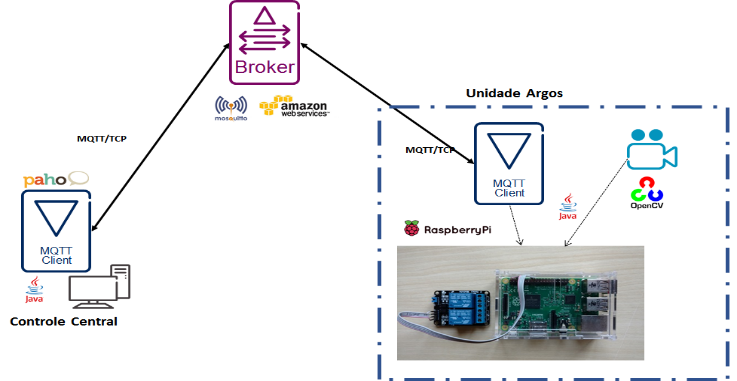


Fig. 3. Esquema de uma unidade (nó)

* 1. *Comandos para Controle dos nós*

Alguns comandos simples foram implementados utilizando o protocolo MQtt, sendo os mesmos recebidos através do campo *topic* do protocolo. Esta escolha foi devido à simplicidade da abordagem, bastando fazer o *parsing* do campo *topic* e utilizando o campo *payload* diretamente, 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*. Assim, em questão de segundos o nó já mudará o alvo a ser detectado.

• ***status*** e ***error***: são enviados pelo 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).

* 1. *Escopo dos comandos e GUI*

A utilização do protocolo MQtt permite grande flexibilidade no envio de comandos: podemos definir escopos como nós individuais, grupos de nós, ou todos os nós. Para enviar comandos para um nó específico podemos usar o Id do mesmo (/região\_1/zona\_1/nó\_2/restart). Também é possível enviar comandos para todas os nós acoplados ao sistema (/região\_1/snapshot), bastando para isto escolher o tópico MQtt de nivel mais genérico.

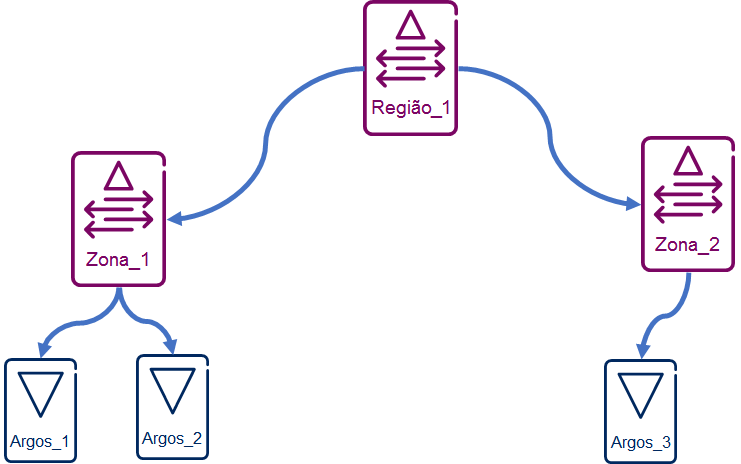


Fig. 4. Esquema de tópicos, para divisão por regiões

Uma interface GUI rudimentar foi criada, apenas para possibilitar a visualização de fotos recebidas de maneira rápida.

1. Desenvolvimentos futuros

Enumeramos à seguir sugestões para futura continuação deste projeto, algumas mais trabalhosas, porém algumas pequenas e pontuais o bastante para serem implementadas em curto espaço de tempo.

* 1. *Melhorias na GUI*

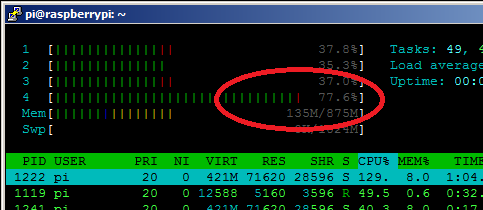
Um dos principais avanços para o atual projeto será a criação de uma interface gráfica de usuário mais amigável e intuitiva, capaz de ser utilizada por pessoas comuns. Hoje, mesmo um programador experiente levará algum tempo até entender a interface e poder operá-la de maneira ágil. Para um projeto real, 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).

* 1. *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. Vislumbramos 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. Convém notar que a biblioteca OpenCV já permite a utilização de GPUs para aumentar o poder de processamento, bastando que isto seja implementado no sistema. Inclusive os ambiente como Amazon AWS já permitem o uso de GPU *farm*.

* 1. *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. Embora hoje 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. 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 provê diversas rotinas auxiliares para processamento paralelo.



*Fig. 5. Uso intensivo de apenas uma única CPU*

* 1. *Empilhamento de classes*

Embora a captura das imagens pela câmera seja uma tarefa não paralelizável, nada impede que as rotinas da biblioteca OpenCV sejam utilizadas para processar frames em paralelo, simultaneamente, indo de encontro com o tópico anterior. 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 possibilitaria ao sistema executar reconhecimentos como ‘busca criança perdida’, ‘busca idoso de óculos’, ‘contagem de pessoas com sacola de comprar’, todos em tempo real. Os modelos de reconhecimento seriam empilhados, e 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.

* 1. *Limitações do MQtt*

O protocolo MQtt, sendo originalmente voltado para pequenos processadores embarcados, tem algumas limitações, sendo que a que mais nos afeta é o tamanho máximo do campo payload. Para *brokers* como o *Mosquitto*, este tamanho é bastante confortável, de aproximadamente 255 MB [3]. Já para o *broker* da *Amazon AWS* existe um limite bastante pequeno de apenas 128 kB [4]. 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. Talvez mesmo as classes Java ou modelos .xml necessitem deste tratamento. A integração com protocolos de dados como *ftp* também pode ser usa saída elegante.

1. Referências
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. Eclipse Mosquitto™ manual [Online]. Disponível: <https://mosquitto.org/man/mosquitto-conf-5.html>
5. Amazon Web Services, General Reference (Version 1.0). Disponível: <https://docs.aws.amazon.com/general/latest/gr/aws_service_limits.html#limits_iot>

¹ **Kleber de Mattos Dobrowolski** nasceu em Taubaté, SP, em 03 de novembro de 1975. Possui os títulos: Bacharel em Computação Científica (UNITAU, 2000) e Mestrado em Computação Aplicada (INPE, 2004).

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