Visão computacional e Reconhecimento de gestos

Pedro Lopes de Oliveira
Instituto Nacional de Telecomunicações - Inatel
pedro.lopes@get.inatel.br

Carlos Alberto Ynoguti
Instituto Nacional de Telecomunicações - Inatel
ynoguti@inatel.br

Abstract - This document's finality is to show the implementation of a gesture recognition system, what has as principal goal ally natural human body movements to information to be managed on houses, offices or meeting rooms, as well as this main pillars, the gesture recognition algorithms, based on the Python library OpenCV with focus on the hands gestures, and the communication protocols, implemented with MQTT, widely used on IoT applications. Through these was possible to recognize the gestures and ally then to the communication, creating a bond between the human body and the automation and comfort needs, on business and residential spot.

Index Terms - Residential automation, gesture, IoT, MQTT, Python

Resumo - Este documento tem como finalidade explicitar a implementação de um sistema de reconhecimento de gestos, que tem como principal objetivo aliar movimentos naturais do corpo humano com informações a serem gerenciadas em uma casa, escritório ou sala de reunião, bem como seus principais pilares: os algoritmos de reconhecimento de gestos, desenvolvidos utilizando a biblioteca Python OpenCV com ênfase no reconhecimento das mãos, e os protocolos de comunicação na nuvem, que são implementados utilizando o protocolo MQTT, amplamente utilizado em aplicações IoT. Através destes pilares foi possível reconhecer os tais gestos e aliando-os à comunicação criar um vínculo entre movimentos do corpo humano e as diversas necessidades de automatização e conforto, tanto para ambientes empresariais, quanto para ambientes residenciais, através da ativação de serviços e/ou equipamentos com a simplicidade e comodidade de apenas gesticular.

Palavras chave - Automação residencial, gestos, IoT, MQTT, Python.

I. Introdução

O projeto desenvolvido durante esse estudo teve como base o protocolo de comunicação MQTT, algoritmos de reconhecimento de gestos e unidades microcontroladas para a obtenção, processamento e aplicação de dados, visando assistir pessoas deficientes ou idosas na realização de atividades inerentes ao dia a dia, simplesmente com a utilização de gestos, além de auxiliar no conforto e comodidade. Este pode ser resumido através do diagrama em blocos da Figura 1.

Através de uma câmera a imagem é capturada e com o auxílio de um sistema computacional e algoritmos desenvolvidos para o reconhecimento de gestos das mãos utilizando Python, são gerados dados referentes a quantos dedos estão levantados e, consequentemente quais gestos estão sendo executados. De posse dessas informações, utiliza-se o CloudMQTT, plataforma grátis, para armazenamento e distribuição desses dados para unidades controladoras conectadas tanto aos dispositivos requisitados pelo usuário, quanto ao próprio fornecedor de informações, através da internet. Portanto, estando em um cômodo é possível controlar inúmeros dispositivos, simplesmente gesticulando. Vale ressaltar, que essa aplicação também se adequa a gerar mais conforto ao usuário, podendo ser designada para servir escritórios, hospitais e indústrias, desde que seja necessário o controle remoto de dispositivos.

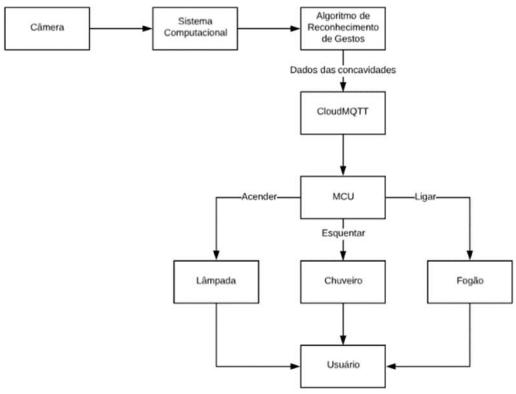


Figura 1 - Diagrama em blocos do projeto

Para a obtenção das imagens foi utilizada uma webcam de notebook Acer Aspire F15, esse também executou os algoritmos e encaminhou as informações para a aplicação na nuvem CloudMQTT. Para o controle dos dispositivos no local, foi utilizado o Node MCU, hardware pequeno, barato e que se conecta facilmente à internet, juntamente com a biblioteca <PubSubClient.h> que alia microcontroladores ao já citado protocolo e que pode ser encontrada nas referências.

II. O Protocolo MQTT

O MQTT (*Message Queue Telemetry Transport*) foi desenvolvido e inventado pela IBM em meados dos anos 90, segundo registros da própria empresa. Este tinha como objetivo original a vinculação de sensores em linhas de tubos de petróleo a satélites. Porém, após 2014 esse protocolo se tornou um padrão aberto, o que desencadeou sua "tradução" para diversas linguagens de programação populares e de fácil manuseio, por consequência, este se tornou o protocolo padrão quando se trata de aplicações IoT, ou seja, as que necessitam de conexão constante com a Internet. Esse protocolo é um protocolo de mensagem M2M (*Machine To Machine*),

funciona através da troca de mensagens entre máquinas, que publicam ou se inscrevem em tópicos de informações.
Para entender melhor seu funcionamento, se faz necessária a definição de alguns conceitos:

Broker: o *broker* é o intermediário entre as comunicações das máquinas. Ele tem como principal objetivo guardar todas as informações publicadas em tópicos e fornecê-las a quem se inscrever nestes.

Tópico: um tópico é uma parte do *broker*, cada tópico possui um nome e pode ou não possuir sub-tópicos. É nele que serão publicadas as informações para que, posteriormente, alguém ou algo as utilize.

Exemplo de estrutura de um tópico com sub-tópicos: /casa/quarto/abajur/.
Temos que o tópico é casa e os seus sub-tópicos são quarto e abajur

Publicação/Publish: uma publicação é uma informação guardada em um tópico, dentro do broker.

Inscrição/Subscribe: o ato de inscrição é a obtenção das informações guardadas em um determinado tópico ou subtópico.

Entendendo esses conceitos, se torna fácil o entendimento do funcionamento do protocolo:
De um lado do *broker*, temos os publicadores, sistemas que coletam dados de algum meio e que precisam enviar estes para um ou mais dispositivos, do outro lado temos os receptores, comumente sistemas que coletam os dados dos publicadores para mostrar ao usuário. Portanto, o publicador publicará seus dados em um tópico do *broker*, que as deixará armazenadas, e se um dos dispositivos necessitar dessa informação ele simplesmente fará sua inscrição no tópico que a contém, tendo o dado requisitado.

Vale ressaltar que todos os dispositivos envolvidos podem publicar e se inscrever em tópicos, portanto não existe sistema que só tenha permissão para publicar ou apenas para se inscrever, como mostrado na Figura 2, ambas as atividades são inerentes a todos os dispositivos envolvidos.

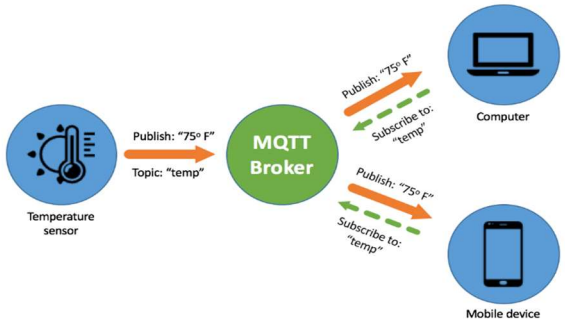


Figura 2 - Funcionamento do protocolo MQTT

III. O Algoritmo de reconhecimento

Este algoritmo foi construído a partir de uma biblioteca Python denominada OpenCV, desenvolvida originalmente pela Intel em 2000 e essa biblioteca tem como principal propósito o estudo da visão computacional, possuindo para isso, módulos de processamento de imagem, entradas e saídas de vídeo, álgebra linear e interfaces gráficas, além de mais de 300 algoritmos de visão computacional que incluem:

- Filtros de imagem;
- Calibração de câmera;
- Reconhecimento de objetos;
- Análise estrutural.

Porém, não se encontra reconhecimento de gestos já fornecidos pela própria plataforma, o que levou ao surgimento dessa aplicação. Vale também ressaltar que essa biblioteca abrange inúmeras linguagens mas tem como as principais Python, Java e Visual Basic. Nessa aplicação foi utilizado exclusivamente Python para a construção do algoritmo de reconhecimento de gestos.

Para que haja um bom entendimento do funcionamento desse algoritmo é necessário dividi-lo em partes, para expor cada uma das etapas até o resultado final, portanto este pode ser divido em:

- Captura e remoção do plano de fundo ou background;
- Aplicação de filtros;
- Binarização;
- Contornos;
- Fecho convexo e falhas de convexidade.

A. Captura e remoção do background

A remoção do plano de fundo ou background é o pilar inicial do algoritmo, sem ela, a detecção de objetos se torna algo inviável, uma vez que inúmeros artefatos estarão interferindo no funcionamento do algoritmo. Essa remoção é feita tomando como base o fundo inicial, sem os objetos a serem detectados. Sendo assim, essa imagem é removida e tudo o que aparecer, posteriormente à remoção e que for diferente dos estados anteriores, será considerado um novo objeto e será mostrado.

Vale ressaltar, que a distância influenciará na qualidade da captura, sendo assim, quanto maior a distância da câmera em relação aos objetos envolvidos no plano de fundo, melhor será a captura. Porém, quando se trata dos objetos a serem detectados, a situação é inversa, pois a distância também influenciará, mas de maneira oposta, portanto, a detecção dos objetos será melhor quanto mais perto estiverem da câmera.

Por fim, se faz importante a definição de algumas situações que interferem no funcionamento do algoritmo. A luz é o que nos permite enxergar objetos e o mundo a nossa volta, porém, para o algoritmo, seu excesso é um vilão, portanto, para amenizar seus efeitos, a câmera nunca deve estar voltada diretamente para uma fonte de luz forte. Outros vilões são os objetos em movimento durante a captura do background, estes podem causar *bugs* no momento da remoção e devem ser evitados. Portanto, chegou-se à conclusão que o melhor plano de fundo é uma parede lisa ou um plano de fundo amplo, com objetos, que não devem ser detectados, longe da câmera.

Para tornar a implementação possível, a biblioteca OpenCV disponibiliza algumas funções prontas, que são de extrema valia para simplificar o funcionamento do código. São elas:

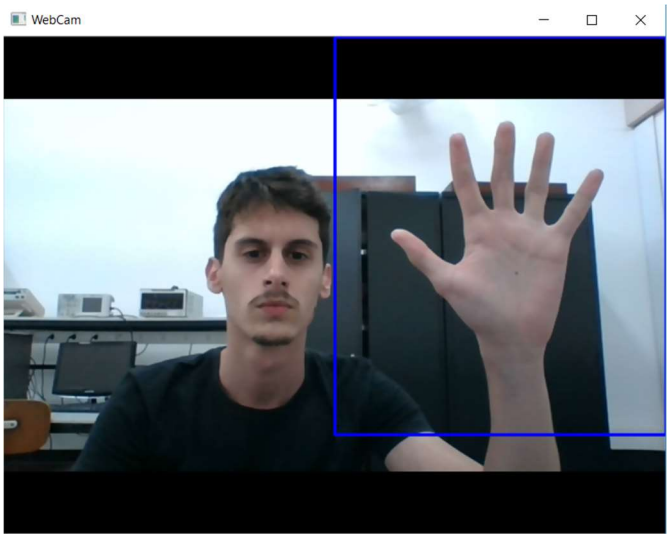
Bgmodel = cv2.BackgroundSubtractorMOG2(0, subTr)

É uma função que cria um modelo de subtração de background a partir de alguns parâmetros e que deverá ser utilizado posteriormente.
No caso utilizou-se subTr igual a 50.

fgmask = bgModel.apply(frame)

Aplica o modelo criado previamente à imagem de saída da câmera, fazendo a subtração.

Resultados obtidos:



*F*igura 3 - Imagem original obtida pela WebCam

Figura 4 - Background removido

B. Filtros de imagem

A utilização de filtros de imagem é imprescindível para obter melhores resultados quando se trata de uma aplicação que utiliza de detalhes capturados por câmera. O principal problema das imagens capturadas em câmeras de baixa ou média qualidade é o ruído, portanto, para que esse ruído seja consideravelmente diminuído, são utilizadas algumas técnicas, dentre elas encontra-se o desfoque, comumente conhecido como "blur" ou "smoothing", que como o nome sugere desfoca a imagem, o que permite a obtenção de contornos mais precisos, uma vez que a barreira entre o que deve ser contornado e o que não deve ser, é suavizada, facilitando o processo e impedindo eventuais erros.

Existem inúmeras formas e técnicas de desfocar uma imagem, dentre elas tem-se:

- Desfoque Gaussiano ou Gaussian Blur;
- Filtro por média ou Median Filter;
- Filtros bilaterais ou Bilateral Filters.

Antes de decidir qual filtro utilizar é necessário entender um parâmetro importante e inerente a todos os filtros de imagem, bem como seu princípio de funcionamento básico. Todo filtro de imagens pode ser determinado como a soma ponderada de pixels, ou seja, multiplicadas por um parâmetro, que resulta em um outro pixel com características alteradas. Além disso, todos estes possuem um kernel, o kernel é o conjunto de coeficientes de um filtro, o que ajuda a visualizar o filtro como uma soma ponderada de pixels, uma vez que o ponderamento é feito através do kernel. O funcionamento básico de um filtro pode ser detalhado através da equação de convolução (1):

$$g(i,j) = \sum_{k,l} f(i+k, j+l) \times h(k,l) \qquad (1)$$

Onde:

$g(i,j)$ é o píxel que sofrerá desfoque e suas respectivas coordenas i e j;
f(i + k, j + l) é a função que faz as operações necessárias para obter o resultado desejado.
$h(k,l)$ é o kernel, que multiplicará o valor do pixel por um valor paramétrico necessário.

Porém, para essa aplicação foi utilizado especificamente o filtro denominado Desfoque Gaussiano ou *Gaussian Blur*, que tem seu princípio de funcionamento baseado na distribuição probabilística de Gauss, a escolha desse tipo de filtro foi baseada no grande número de publicações que demonstram sua superioridade em relação a outras técnicas, como o artigo citado nas referências, além dos testes realizados. Basicamente, seu funcionamento é determinado pela substituição de um pixel de entrada, por um outro pixel alterado através da curva de distribuição gaussiana. O pixel localizado entre dois outros será o de maior peso, enquanto que seus vizinhos terão os pesos decrescidos, quanto mais distante do pixel de origem, menor será o peso do pixel em questão, criando desfoque.

A curva utilizada para os cálculos é, como já dito, a curva de Gauss, representada por:

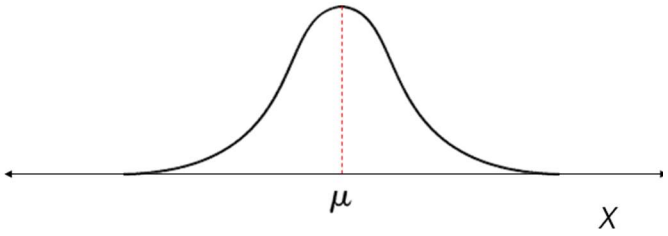


Figura 5 - Curva de distribuição normal ou Gaussiana

Por fim, a obtenção do novo valor do pixel, pode ser obtida através da equação (2):

$$G_o(x,y) = e^{-\frac{(x-\mu_x)^2}{2\sigma_x^2}-\frac{(y-\mu_y)^2}{2\sigma_y^2}} \quad (2)$$

Onde:

$G_o(x,y)$ é o novo valor de pixel obtido;
μ é o valor do pico da função, possuindo valores para x e y;
σ é a variância, possuindo valores para x e y.

Por se tratar de uma implementação muito complexa, foi utilizada uma função pronta disponibilizada pela biblioteca e que possui a seguinte sintaxe:

blur = cv2.GaussianBlur(img, (sx, sy), b)

Onde:

blur é a imagem de saída;
img é a imagem de entrada;
(sx, sp) é o valor das variâncias em x e y, respectivamente. No caso utilizou-se (41,41);
b é o tipo de borda. No caso utilizou-se 0, para não aplicar nenhuma.

Porém, antes de utilizá-la é necessária a conversão da imagem colorida para preto e branco, uma vez que essa função só funciona para imagens desse tipo. Portanto, utiliza-se a função de conversão, que possui a seguinte sintaxe:

gray = cv2.cvtColor(img, cor)

Onde:

gray é a imagem de saída;
img é a imagem de entrada;
cor é o parâmetro de conversão de cores. No caso utilizou-se cv2.COLOR_BGR2GRAY que converte de RGB para preto e branco.

Resultados obtidos:

Figura 6 - Conversão para preto e branco

Figura 7 - Desfoque utilizando GaussianBlur

C. Binarização

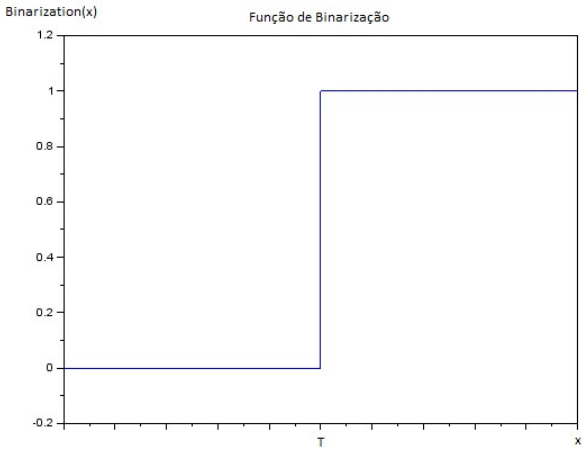
A conversão de uma imagem em tons de cinza para uma imagem com representação binária é de grande valia para inúmeros objetivos, dentre eles pode-se listar a identificação de objetos, separação de níveis e análise de formas. Todos esses conceitos são base para o funcionamento desse algoritmo que tem por finalidade o isolamento da mão, que nada mais é, que um objeto.

É possível binarizar uma imagem a partir do momento em que são conhecidas as intensidades luminosas dos pixels que a compõem. De posse dessa informação, a implementação é simples, uma vez que por se tratar de uma transformação para binário, somente dois valores são esperados: '1' ou '0', níveis lógicos altos ou níveis lógicos baixos.

Para isso, pode-se determinar uma função, que para cada valor de intensidade luminosa ou nível de cinza, retorne um valor corresponde a '1' ou '0'. Portanto, é necessário definir um valor padrão que limite o que será branco e, consequentemente, o que será preto, esse valor é denominado *treshold* (T). Sendo assim é possível determinar o comportamento do sinal de saída através da função composta:

$$\boldsymbol{Binarization}(x) = \begin{cases} \mathbf{0}, & x < T \\ \mathbf{1}, & x > T \end{cases}$$

Este pode ser facilmente desenhado com o auxílio de um plotador de gráficos como o MatLAB, resultando em:



Vale ressaltar que o sinal obtido é um sinal do tipo degrau unitário deslocado de T, que é o *treshold*.

Mesmo se tratando de uma implementação simples, a biblioteca OpenCV, utilizada para toda a construção do algoritmo, disponibiliza uma função pronta que executa o processo, listada a seguir:

ret, thresh = cv2.threshold(blur, threshold, 255, transf)

Onde:

tresh é a imagem de saída;
blur é a imagem de entrada. No caso foi utilizada a imagem já desfocada com o método GaussianBlur;
treshold é o parâmetro que define a margem do que é branco e do que é preto. No caso utilizou-se uma trackbar para ajustar esse valor de acordo com a necessidade;
transf é o tipo de transformação a ser aplicado na imagem. No caso utilizou-se cv2.THRESH_BINARY, que binariza a imagem.

Por fim, é evidente que a binarização é inerente ao funcionamento do algoritmo, uma vez que aliada à subtração de background, essa se torna muito fácil de implementar, além de facilitar muito os processos posteriores, que envolverão o contorno das mãos e a detecção de falhas de convexidade, que seriam extremamente difíceis de aplicar, se não se tratasse de uma imagem binarizada e tratada, causando várias confusões e estados ambíguos para o algoritmo.

Resultados obtidos:

Figura 8 - Pós Binarização

D. Fecho convexo e contornos

O fecho convexo tem como principal objetivo delimitar uma região definida por um conjunto de pontos. Porém antes de entender seu funcionamento, faz-se necessária a definição de um conceito:

Envoltória convexa: a envoltória convexa de um conjunto Q de pontos é o menor polígono convexo, ou seja, de menor área, em que estão contidos todos os pontos do conjunto, sendo eles extremidades ou não.

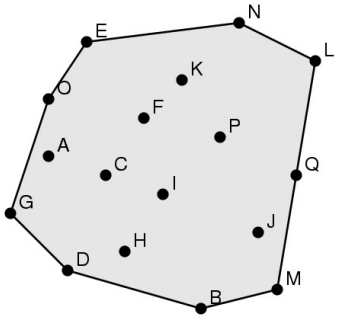


Figura 9 - Envoltória convexa

Portanto, o objetivo dos algoritmos de fecho convexo é obter a envoltória convexa. Para tanto, a implementação desta técnica pode ser feita através do algoritmo de Graham, pioneiro no campo dos algoritmos geométricos.

É utilizada a varredura rotacional, que processa os vértices na ordem dos ângulos polares que eles formam com o vértice de referência, com isso, é decidido quais pontos serão vértices do envoltório e quais estarão apenas contidos neste. Caso o ângulo formado por três pontos seja menor do que 90 graus, não há convexidade, mas sim concavidade e, sendo assim, esse conjunto é excluído.

Portanto, na aplicação, o fecho convexo tem como prioridade a detecção de falhas de convexidade, que em nossa aplicação são dadas pelos vãos entre os dedos, uma vez que é formada uma concavidade. Com essa informação é possível plotar pontos que se encontram nessas falhas e conta-los, dando origem a

informação de um gesto. Cada gesto é formado pelo número de falhas de convexidade, ou vãos entre os dedos mostrados, dando origem a cinco possibilidades, representadas nos resultados obtidos desta seção.

Como sempre, a biblioteca OpenCV disponibiliza várias funções prontas para as implementações. Para a obtenção do contorno da mão foi utilizada:

contours, hierarchy = cv2.findContours(img, p1, p2)

Onde:

contours é a imagem de saída;
img é a imagem de entrada;
p1 é um parâmetro. No caso utilizou-se cv2.RETR_TREE;
p2 é um parâmetro. No caso utilizou-se cv2.CHAIN_APPROX_SIMPLE.

Já para a obtenção do fecho convexo e das falhas de convexidade, respectivamente, foram utilizadas as funções:

hull = cv2.convexHull(img)

Onde:

hull é a variável que armazenará as coordenadas do fecho convexo;
img é a imagem de entrada.

defects = cv2.convexityDefects(img, hull)

Onde:

defects é a imagem de saída;
hull é a variável que armazena as coordenadas do fecho convexo;
img é a imagem de entrada.

Resultados obtidos:

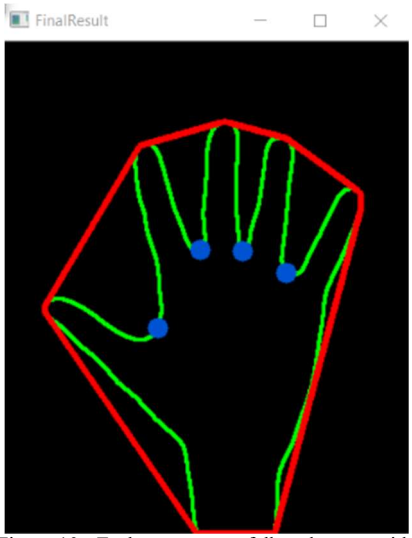


Figura 10 - Fecho convexo e falhas de convexidade.

Gestos reconhecidos:

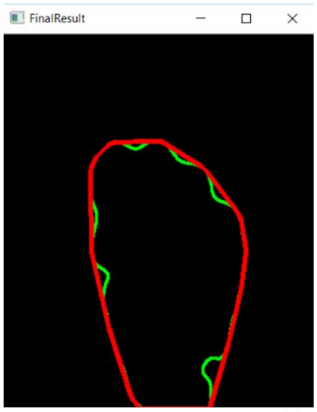


Figura 11 – Gesto com zero concavidades

Figura 12 - Gestos com uma e duas concavidades

Figura 13 - Gestos com três e quatro concavidades

IV. Referências

[1] Bradskym G. R.; Pisarevsky, V.; Bouguet, J., “Learning OpenCV: Computer Vision with OpenCV Library. Springer, 2006.
[2] Teresa Cristina T. V. Carneiro, “Segmentação”.
[3] OpenCV development team, “OpenCV API Reference”, 2011, docs.opencv.org.
[4] Rosanna M. R. Silveira, “Algoritmos geométricos”.
[5] Izane, “Fingers detection using OpenCV and Python”, 2016, www.izane.com.

[6] Leonardo H. Monteiro, “Binarização por Otsu e outras técnicas utilizadas na detecção de placas”, www.ic.uff.br.
[7] Douglas Zuqueto, “Do Hello world à integração com MQTT”, 2017, www.douglaszuqueto.com.
[8] Pedro Bertoleti, “Controle e monitoramento IoT com NodeMCU e MQTT”, 2016, www.filipeflop.com.
[9] Ivan Grokhotkov, “ESP8266 Arduino core’s documentation”, 2017, ESP8266 Arduino Core.
[10] Edison O. Jesus, Roberto Costa Jr., “A utilização de filtros Gaussianos na análise de imagens digitais”.