Central de Vigilância Tomba City - Relatório PBL

Víctor César, Messias Silva, Matheus Teixeira, William Soares

[1]Universidade Estadual de Feira de Santana (UEFS)
Feira de Santana – BA – Brasil

Resumo. *O documento a seguir apresenta um sistema de vigilância com 3 câmeras que, no horário estabelecido do lockdown, detectam pessoas infringindo as restrições, além de também apresentar a arquitetura do sistema com as tecnologias utilizadas, manual do sistema e manual de usuário web.*

1. Introdução

Atualmente, há uma pandemia global do *Covid-19* que já dura mais de um ano. Com o objetivo de prevenir e reduzir a proliferação do vírus, implantou-se o *lockdown*, um decreto que proíbe as pessoas de saírem de suas casas durante um determinado horário do dia. Desse modo, o prefeito de *Tomba City* solicitou o desenvolvimento de um sistema de monitoramento com uma rede de sensores capaz de detectar alguém nas ruas no horário de funcionamento do *lockdown*. Após a detecção, a imagem da câmera será salva em um *log* através de uma aplicação *Web*.

2. Arquitetura

A Central de Vigilância Tomba City (CVTC) é um sistema composto por uma aplicação *Web*, 3 câmeras e 1 *broker*. A aplicação *Web* e as câmeras se conectam ao *Broker*, que faz o repasse das mensagens que são publicadas nos tópicos. Veja a seguir na Figura 1 um diagrama de componentes do sistema:

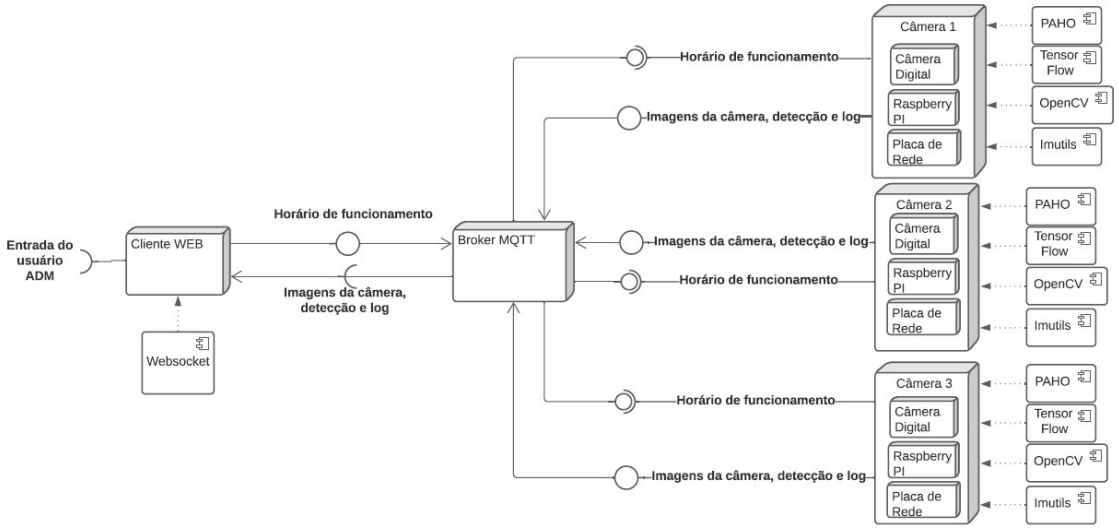


Figura 1. Diagrama de componentes

A figura 1 mostra em seu lado direito as câmeras, que não são compostas apenas com uma câmera digital, mas com uma *Raspberry PI* e uma placa de rede. Essa

composição é dada para a câmera virar um agente inteligente dentro da proposta de cidade inteligente de *Tomba City*. Este agente inteligente tem um *single board computer*, no caso a *Raspberry PI*, conectada a câmera digital e a placa de rede. Com os dados de entrada da câmera sendo processadas dentro do *Raspberry PI* e sendo enviadas para internet graças a placa de rede. Na *Raspberry PI* é feito a detecção de uma pessoa na imagem através da rede neural *Tensor Flow* com o modelo "*faster_rcnn_inception_v2_coco_2018_01_28*", *Open CV* e *Imutils* para a manipulação das imagens. Como também a utilização do *PAHO* para realizar a conexão com o *Broker MQTT*.

No lado esquerdo e no centro da Figura 1 temos respectivamente a aplicação *Web* e o *Broker MQTT*. A aplicação *Web* simples para a exibição das imagens das câmeras, como também ela terá um registro "*log*"com as imagens das quebras de *lockdown*, o horário e a câmera que capturou a imagem. A comunicação entre as câmeras e a *web* é feita através do protocolo *Message Queuing Telemetry Transport* (MQTT), onde o *Broker* é o computador responsável em fazer essa comunicação, basicamente o servidor sendo o *Broker* e os demais sendo clientes. Entretanto, a comunicação é feita através do fornecimento do endereço e da porta, mas ao conectar é necessário a autenticação e também a inscrição em tópicos. A seguir há uma lista com todos os tópicos:

- cam/#
  - cam/camera1
  - cam/camera2
  - cam/camera3
- log/#
  - log/imagem
  - log/texto
- horario/#
  - horario/inicial
  - horario/final

Os tópicos são utilizados da seguinte maneira. O cliente *Web* se inscreve no tópico *cam/#* e *log/#* onde pega todos os subtópicos dentro deles e publica no *horario/#* em ambos subtópicos. Já as câmeras se inscrevem no tópico *horário/#* em ambos subtópicos e publica em ambos subtópicos do *log/#*, como também publica em um único subtópico da *cam/#*, por exemplo, se for a câmera 1 irá publicar as imagens no subtópico *cam/camera1*. Sendo assim, o tópico *cam/#* são para as imagens em tempo real, o *log/#* para o registro de quebras de *lockdown* e o *horário/#* para configurar o horário que o *lockdown* está ocorrendo.

A CVTC tem a capacidade de transmitir uma imagem por segundo em tempo real, como também a capacidade de detectar as quebras de *lockdown* nas imagens e assim gerando um registro contendo a data, hora, imagem e em qual câmera foi flagrado a pessoa violando a restrição, além de todos os dados serem comunicados entre as aplicações através do protocolo MQTT. A seguir é listado os requisitos funcionais mencionados neste parágrafo:

- Transmitir de imagens em tempo real;
- Detectar de quebras de *lockdown*;
- Comunicar entre aplicações com MQTT;
- Gerar *log* de quebras de *lockdown*.

Na CVTC possui o uso de redes neurais para a detecção de pessoas nas imagens, como também há a necessidade de autenticar os dispositivos, onde fornecem o nome do usuário e senha para conectar corretamente ao *Broker*, além disso, o *Broker* possuí a capacidade de efetuar dois tipos diferentes de conexão, a conexão *Transmission Control Protocol* (TCP) e a conexão *Websocket*, por fim, o código possui um baixo acoplamento e alta coesão. A seguir é listado os requisitos não funcionais mencionados neste parágrafo:

- Autenticação de dispositivo;
- Uso de redes neurais;
- Múltiplos tipos de conexão (TCP e Websocket);
- Baixo acoplamento e alta coesão.

Contudo, a CVTC é apenas um protótipo, todos os dados são simulados a partir de dados reais, como também o protótipo *web* está apenas rodando em servidor local, mas o centro funcional do sistema está devidamente preparado para o funcionamento real, ou seja, o *Broker* está configurado de acordo com a aplicação real, podendo se conectar online. Entretanto, o serviço DDNS para hospedagem tem um tempo limite, sendo assim, para finalizar o projeto é necessário ter a assinatura desse serviço.

3. Manual de usuário da página web

Como o MQTT não possui suporte nativo para *web*, foi necessário implementar este módulo através do *Websocket*, um protocolo de comunicação que utiliza conexão TCP para permitir a comunicação bidirecional de baixa latência (com alto volume de dados e baixo *delay*) [Pedamkar 2020]. Dessa forma, o módulo *web* estabelece a conexão com o *Broker* através do HTTP e então ”migra”sua conexão para o *websocket*, que, em conjunto com o MQTT, permite que todas as funcionalidades deste último estejam disponíveis [Cope 2021b]. Essa comunicação foi implementada utilizando a biblioteca *PAHO* para *JavaScript*.

Figura 2. Interface da página Web

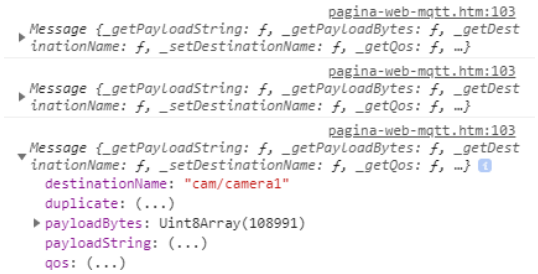
Através do módulo, deve-se inserir o endereço e a porta do *Broker* com o qual se deseja conectar. Feito isso, pode-se definir os horários inicial e final nos quais a quebra

de *lockdown* será detectada. Caso nenhum horário seja definido pela página, a detecção acontecerá em todos os horários até primeiro de janeiro de 2022. A Figura 2 mostra como é a interface da página ao executá-la.

Ao se conectar com o *Broker*, o módulo automaticamente se inscreve nos tópicos *cam/#* e *log/#*. Dessa forma, garante-se que ele receba todas as informações enviadas através desses tópicos e de seus respectivos subtópicos. Para enviar os horários inicial e final, o módulo publica as mensagens nos tópicos *horario/inicial* e *horario/final*, respectivamente. A partir daí, basta apenas esperar que as imagens sejam enviadas pelos sensores.

Na Figura 3 é possível ver a imagem que chega da câmera 1, enquanto o console do navegador mostra as mensagens recebidas no tópico *cam/camera1*.

(a) Imagem da câmera 1

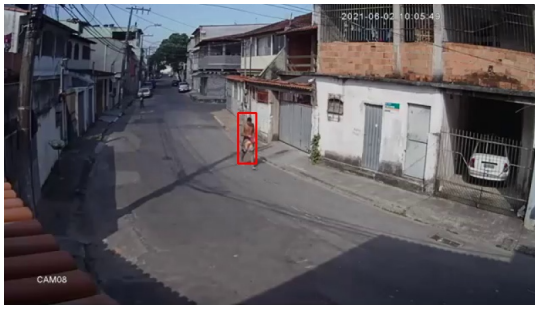


(b) Console do navegador

Figura 3. Câmera 1

O módulo também é responsável por exibir um *log* com as informações de cada uma das vezes em que a quebra de *lockdown* foi detectada. A imagem com a detecção é gravada e exibida junto com as informações de câmera, data e hora nas quais a quebra foi registrada. É possível abrir a imagem em nova guia para visualizá-la melhor (Figura 4).

(a) Log das detecções



(b) Detecção de quebra de lockdown

Figura 4. Log

O código pode ser encontrado em "Câmera Inteligente" junto também terá o código das câmeras, como também o modelo e os vídeos utilizados [Teixeira et al. 2021].

4. Manual de usuário das câmeras do sistema

Para a simulação das câmeras será necessário fazer a instalação das bibliotecas que foram utilizadas no projeto, o *Tensor Flow*, *Opencv*, *Imutils*, além da *PAHO* que foi usada na comunicação, que será descrita posteriormente.

Para a instalação das bibliotecas basta digitar os seguintes comandos no *Prompt de Comando* (CMD). O CMD deve ser executado na pasta onde contém o *pip* do *Python*:

- *pip install tensorflow*, para a instalação do *Tensor Flow*;
- *pip install imutils*, para a instalação do *Imutils*;
- *pip install opencv-python*, para a instalação do *OpenCV*;
- *pip install paho-mqtt*, para a instalação do *PAHO*;

O *Tensor Flow* é usado no processamento das imagens, onde foi utilizada uma *Application Programming Interface* (API) de detecção para processar os *frames* do vídeo, esta API utiliza um modelo pré-treinado para realizar o processamento, o modelo escolhido para este detector foi o "*faster_rcnn_inception_v2_coco_2018_01_28*", que pode ser baixado pasta do repositório de modelos do *Tensor Flow*.

Para usar o modelo é necessário fazer o download acessando o link: Modelo [pkulzc and vivek rathod 2018]. Em seguida escolha o modelo "*faster_rcnn_inception_v2_coco*". Com o arquivo baixado, agora é necessário extrair a pasta no mesmo diretório que o código da câmera está.

Ao processar a imagem, a API de detecção retorna a caixa delimitadora (coordenadas para criação do retângulo em volta das detecções), a pontuação, que é usada para determinar o grau de confiança que utilizada na filtragem das detecções, que no caso deve ser maior que 0.9, e a categoria que deve ser 1 para detecção humana [Vidanapathirana 2018].

O *OpenCV* é usado para capturar os *frames* do vídeo, onde é necessário informar o diretório do vídeo a ser analisado, seu nome e extensão, como no exemplo: "*cap = cv2.VideoCapture('video.mp4')*". Já o *Imutils* é usado apenas no redimensionamento da imagem após a captura e não precisa de nenhuma informação ou ação específica.

Com as bibliotecas instaladas, a pasta do modelo pré-treinada no mesmo diretório do projeto, com o vídeo devidamente especificado, e com o *Broker* iniciado, o projeto pode ser executado. Há algumas limitações que devido ao *hardware* inferior, o carregamento do modelo do *Tensor Flow* e o salvamento da imagem antes de enviar para o *Broker* que geram uma lentidão na execução, e será exibido alguns alertas devido a ausência da biblioteca CUDA, porém a CUDA é utilizada para melhorar o desempenho das redes neurais em placas de vídeo, sendo assim, não é necessário pois afinal será rodado em uma SBC.

Os códigos das câmeras, da web, como também o modelo e os vídeos utilizados podem ser encontrados através do *link*: Câmera Inteligente [Teixeira et al. 2021].

5. Manual de usuário para configuração do *Broker Mosquitto*

O *Broker mosquitto* foi configurado para rodar no Sistema Operacional (SO) *Windows*, em sua versão 10. Ao instalar o *mosquitto* no sistema, deve-se abrir o arquivo *mosquito.conf* afim de editar parâmetros para o melhor funcionamento do *broker*, permitindo se comunicar com a aplicação *Web* e os módulos das câmeras.

Dentro do arquivo *mosquitto.conf* alguns parâmetros a serem editados são:

- *allow_anonymous*: Deve-se colocar *false* nesse parâmetro, quando é definido para *false* os clientes que não possuem um identificador autenticado, não podem se conectar ao *Broker* [Cope 2021a];
- *per_listener_settings*: Deve-se colocar *true*. Esse parâmetro define que as configurações iram ser aplicadas a todos os ouvintes conectados no *broker* [Cope 2021a];
- *Password_file*: O parâmetro deve conter o caminho do arquivo criptografado, gerado pela ferramenta de criação de senha e usuário fornecida pelo *mosquitto*, afim de autenticar todos os dispositivos que tentarem se conectar ao *Broker* [Locatelli 2020];

Outras configurações do *Broker*, passam pelos parâmetros de portas e protocolos, deve-se definir quais portas serão usadas e quais protocolos serão usados. Deste modo dentro do arquivo de configuração os parâmetros a serem editados são:

- *listener*: Indica qual porta o *Broker* irá escutar. No caso da aplicação as portas são 1883 e 9001, sendo a 9001 para *Websockets*. Deve-se indicar para cada porta o arquivo de autentificação, caso esteja usando o método de usuário e senha;
- *protocol*: Indica qual protocolo de comunicação será usado para cada porta. Na aplicação a porta 9001 esta usando o protocolo *websockets* e na 1883 o padrão do *mosquitto broker*;

6. Manual para configuração no SO e roteador

Para que o *mosquitto* funcione satisfatoriamente no *windows*, deve-se liberar as portas usadas pelo *Broker* no *firewall*, podendo criar as regras de entrada para cada porta.

No roteador, deve-se definir um IP estático para a maquina que esta rodando o *Broker*, além de habilitar o redirecionamento de pacotes chegam com endereço do *Broker* para a maquina que esta rodando.

Para possibilitar que o *Broker* seja acessado remotamente, seria necessário um IP publico como endereço da maquina, porém os ips disponíveis são privados, sendo necessário configurar um *ddns* no roteador.Para isso, utilizou-se o *no-ip* que fornece esse serviço de *ddns*. Desde modo, o *Broker* pode ser acessado fora da rede privada.

7. Conclusão

Após tudo o que foi apresentado, é nítido que a câmera é um agente inteligente no meio de *Tomba City*, podendo ser utilizada não apenas nesse sistema, mas em outros também. Além disso, o protocolo MQTT é essencial para a comunicação entre dispositivos de maneira leve. Entretanto, é possível deixar mais fluído e rápido se substituir a rede neural *Tensor Flow* por outra mais leve, como também colocar uma aplicação intermediária entre a aplicação *Web* e o *Broker* MQTT. Afinal, o meio de comunicação é essencial para uma *smart city*, saber como usar um protocolo leve de troca de mensagens como o MQTT é de suma importância para os engenheiros da computação.

Referências

Cope, S. (2021a). Quick guide to the mosquitto.conf file with examples. *http://www.steves-internet-guide.com/mossquitto-conf-file/*.

Cope, S. (2021b). Using mqtt over websockets with mosquitto. *http://www.steves-internet-guide.com/mqtt-websockets/*.

Locatelli, C. (2020). Segurança no mqtt. *https://www.curtocircuito.com.br/blog/seguranca-no-mqtt*.

Pedamkar, P. (2020). Mqtt vs websocket. *https://www.educba.com/mqtt-vs-websocket/*.

pkulzc and vivek rathod (2018). Faster rcnn inception v2 coco model. `https://github.com/tensorflow/models/blob/master/research/object_detection/g3doc/tf1_detection_zoo.md`.

Teixeira, M., Júnior, M., César, V., and Soares, W. (2021). Câmera inteligente. *https://github.com/WilliamOSoares/CameraInteligente*.

Vidanapathirana, M. (2018). Real-time human detection in computer vision — part 2. *https://medium.com/@madhawavidanapathirana/real-time-human-detection-in-computer-vision-part-2-c7eda27115c6*.