Trabalho de Sistemas Multimídia UFF-IC-PGC 2020 – Detecção De Disparo De Armas De Fogo Em Áudio

Bruno Teixeira Gondim

Instituto de Computação – Universidade Federal Fluminense (UFF) – Niterói, RJ – Brasil

bruno.t.gondim@gmail.com

1. Objetivo

O objetivo deste trabalho foi implementar um Virtual Multimedia Sensor – VMS na arquitetura V-Prism para detecção de disparo de arma de fogo em áudio. O sensor virtual é desacoplado do dispositivo físico de captura de áudio, com comunicação via UDP. O sensor também é desacoplado do aplicativo de interface de usuário, que pode receber a informação se houve ou não disparo de arma de fogo por protocolo MQTT. Ressalta-se que apenas o VMS foi desenvolvido. O aplicativo do usuário, bem como o microfone de captura real, não fez parte do escopo do trabalho devido principalmente a limitações de disponibilidade tempo e inviabilidade de se efetuar disparos reais de arma de fogo. As operações de envio de áudio para o VMS e recepção de dados provenientes do VMS são simuladas através de comandos fora do VMS.

2. Embasamento teórico resumido

Ao puxar o gatilho de uma arma de fogo, o cartucho - composto pelo projétil (bala), capsula, pólvora (propelente) e espoleta - é acionado. A espoleta é responsável por ativar a pólvora e causar a explosão que pressiona o projétil para fora da capsula. O projétil é lançado pelo cano até a saída da arma. Quando o projétil sai do cano, toda a pressão do deslocamento de gás que impulsionou a bala é liberada para o ambiente. Esta matéria se desloca mais rápido que a velocidade do som, de aproximadamente 340 m/s em atmosfera normal. Ao se deslocar mais rápido que a velocidade do som, é gerada uma onda de choque, de frente de onda esférica, que por sua vez gera a onda sonora referente ao estouro do cano, ou *muzzle blast*. A Figura 1 exibe uma imagem de onda de choque gerada pelo *muzzle blast.*

O som do *muzzle blast* é o que se pretende identificar para detecção do disparo da arma de fogo. É uma onda sonora impulsiva, que possui como características principais uma variação abrupta em sua amplitude e alcança alto nível de amplitude. Desta forma, para se detectar o disparo, foi desenvolvido um VMS na arquitetura V-PRISM analisando estes 2 parâmetros.

Figura 1 – Imagem real da onda de choque gerada pelo estouro do cano(muzzle blast) e pelo projetil mais rápido que a velocidade do som, que é de aproximadamente 340 m/s. https://www.americanscientist.org/article/high-speed-imaging-of-shock-waves-explosions-and-gunshots

3. O VMS

Para o desenvolvimento e teste foram utilizadas as seguintes ferramentas:

- Computador com Windows 10, Intel Core i7, 16 GB de RAM
- Software VirtualBox 6.1
- Máquina Virtual Linux Ubuntu 18.04
- Visual Studio Code 1.46
- Python 3.6.9
- gst-launch-1.0 version 1.14.5
- GStreamer 1.14.5
- tkinter — Python interface to Tcl/Tk 8.6
- Outras bibliotecas Python descritas nos “import” dos arquivos “main”

Também foram utilizados como referência os VMS já desenvolvidos no ALFA V-PRISM (https://github.com/midiacom/alfa), bem como o passo a passo para implementação do ALFA.

O VMS está disponível em https://github.com/brunogondim/beta_VMS. Para realizar experimento do seu funcionamento basta clonar o repositório, instalar as bibliotecas necessárias e rodar o arquivo “main beta 2.py”. Após alguns segundos, a tela exibida na Figura 2 deve surgir:

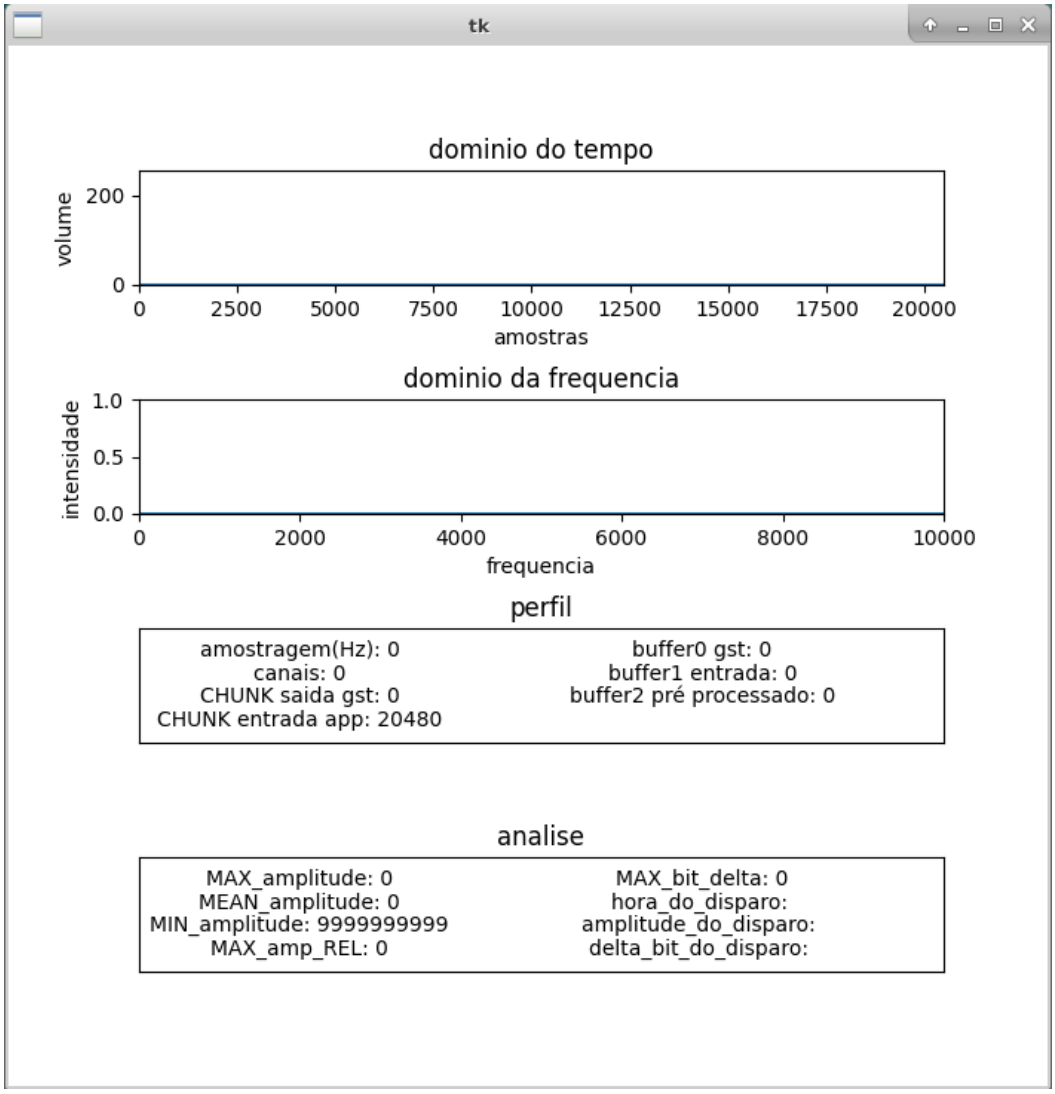


Figura 2 – Captura da tela de análise do áudio, sem áudio.

Após o surgimento desta tela, é preciso simular o recebimento do áudio do microfone. Para esta simulação, digite em um outro terminal o comando: “st-launch-1.0 filesrc location=/home/bruno/Vprism/beta_VMS/teste.wav ! decodebin ! audioconvert ! audioresample ! audio/x-raw,channels=1,depth=16,width=16,rate=44100 ! rtpL16pay ! udpsink host=localhost port=5000” (sem as aspas).

Este comando simula o envio de áudio, via UDP, do microfone para o sensor, em um outro ponto qualquer da rede. O valor do parâmetro “filesrc location” deve ser alterado de acordo com o caminho do diretório onde o VMS foi clonado. A taxa de atualização

da tela de visualização dos dados é baixa, entre 1 a 5 vezes por segundo, dependendo do hardware utilizado. Embora a taxa de visualização dos dados seja baixa, a taxa de análise dos dados propriamente dita é em tempo real. A Figura 3 exibe um exemplo de análise do áudio.

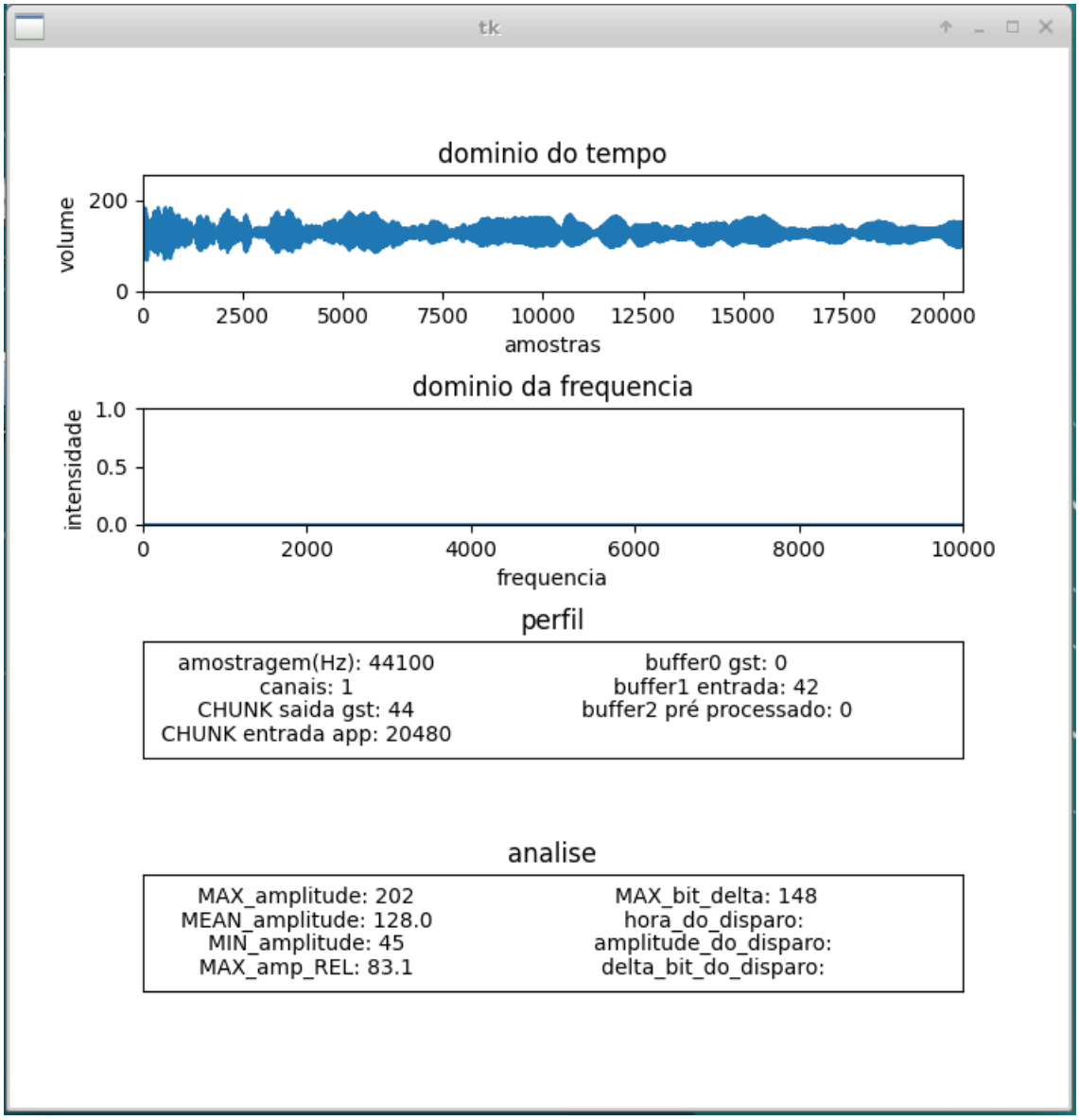


Figura 3 – Captura da tela de análise do áudio.

A visualização dos dados para análise e desenvolvimento dos critérios de detecção é feita com recursos visuais do Matplotlib.

Domínio do tempo

Representa graficamente os bits obtidos do pipeline gstreamer, variando de 0 a 255 (quantização de 8 bits), sendo 128 o valor de referência de meio de eixo. Quanto mais

distante de 128, maior a amplitude. No eixo X, estão as amostras. Como a taxa de amostragem é de 44100, 20480 representa um pouco menos que meio segundo de áudio.

Domínio da frequência

Esta representação gráfica foi desativada pois não foi possível realizar a transformada para o domínio da frequência sem que houvesse atraso no processamento. A taxa de bits da entrada do buffer estava maior que a vazão, ou seja, entrava mais que saia, gerando um atraso crescente e insustentável. É possível paralelizar e coordenar esta análise em trabalho futuro. Apenas quando ocorre a detecção do disparo a transformada é realizada, mas de qualquer forma não está sendo utilizada na análise do disparo.

Perfil (ou "profile")

São os parâmetros internos do processamento:

- Amostragem (Hz): quantos bits são transmitidos por segundo. Esta propriedade é definida no pipeline do gstreamer.
- Canais: Quantidade de canais de áudio. Mesmo que se receba por UDP 2 ou mais canais de áudio, como o estéreo, o pipeline do gstreamer converte para um canal apenas, para eficiência da análise.
- CHUNK saída gst: quantidade de bits que saem do pipeline gstreamer por ciclo. Esta taxa é variável e precisa ser convertida para a taxa de análise.
- CHUNK entrada app: quantidade de bits definida para análise. Este parâmetro pode ser alterado no programa e corresponde a quantos bits serão analisados por ciclo. Aproximadamente é analisado meio segundo de áudio por ciclo.
- Buffer0 gst: quantidade de conjunto de bits (CHUNK) do final do pipeline gstreamer aguardando serem recuperados pelo programa.
- Buffer1 entrada: quantidade de conjunto de bits (CHUNK) que foram recuperados pelo programa e estão aguardando operação de concatenação para formar o chunk de tamanho adequado ao processamento.
- Buffer2 pré-processado: quantidade de conjunto de bits (CHUNK) que foram concatenados e estão aguardando serem processados no próximo ciclo de análise.

Análise

São os parâmetros da informação do áudio propriamente dito que são usados como critérios para detecção do disparo:

- MAX_amplitude: maior valor absoluto recuperado do pipeline do gstreamer
- MEAN_amplitude: valor médio recuperado do pipeline do gstreamer
- MIN_amplitude: menor valor absoluto recuperado do pipeline do gstreamer
- MAX_amp_REL: maior valor distante do valor médio. É a amplitude relativa ao ponto médio. Esta é a amplitude utilizada como critério de detecção. Na visualização é mostrado o maior valor já detectado até o momento.

- MAX_bit_delta: Maior diferença entre 2 bits consecutivos. Este critério é utilizado na detecção do disparo. Quanto maior este valor, mais impulsiva é a onda. Na visualização é mostrado o maior valor já detectado até o momento.
- Hora_do_disparo: É o “time stamp” da detecção do tiro. Quando ocorre o disparo, esta informação é congelada na tela por 5 segundos, assim como os outros 2 critérios seguintes.
- Amplitude_do_disparo: É a amplitude relativa do ponto médio no momento do disparo. O limiar é 83. Passando deste valor é um indicativo que ocorreu o disparo. Deve ocorrer junto com o próximo critério. Entre parênteses é a amostra que acusou o rompimento do limiar. Caso haja mais de uma ocorrência, as quantidades adicionais são mostradas como +x, onde x é o número de incidências adicionais no trecho sob análise.
- Delta_bit_do_disparo: É a variação de bits no momento de disparo. Deve ser maior que 20. Com uma variação maior que 20, alcançando o limiar de 83 é caracterizado o disparo de arma de fogo. Incidências adicionais de ultrapassagem do valor de referência também são representadas por +x.

As Figuras 4, 5, 6 e 7 exibem telas obtidas durante operação normal do sensor. A figura 4 é obtida durante um trecho de áudio sem disparo. As figuras 5, 6 e 7 exibem momentos em que o disparo foi detectado, em 3 testes diferentes.

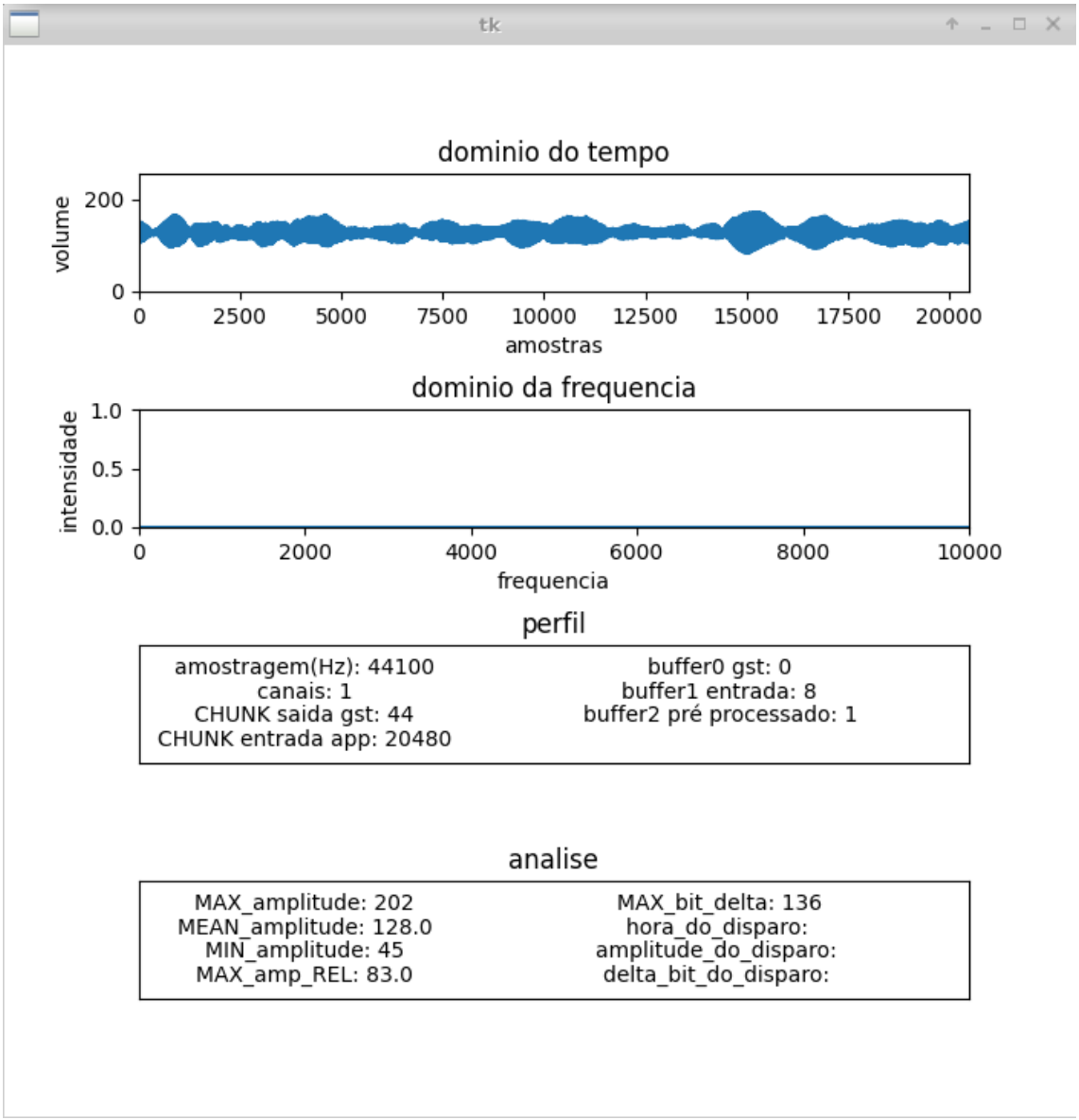


Figura 4 – Captura da tela de análise do áudio.

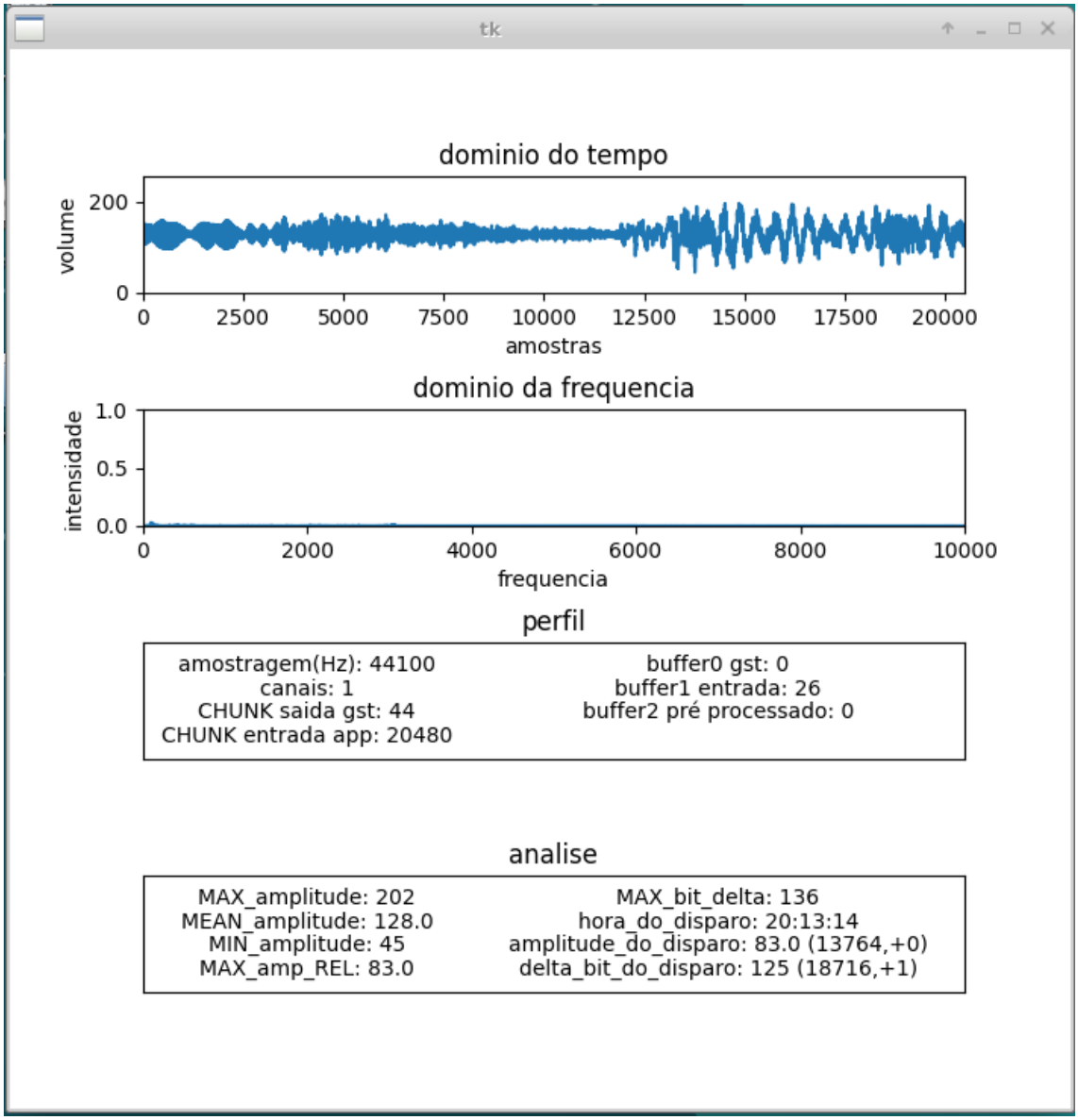


Figura 5 – Captura da tela de análise do áudio no momento em que ocorre a detecção do disparo (teste nº1).

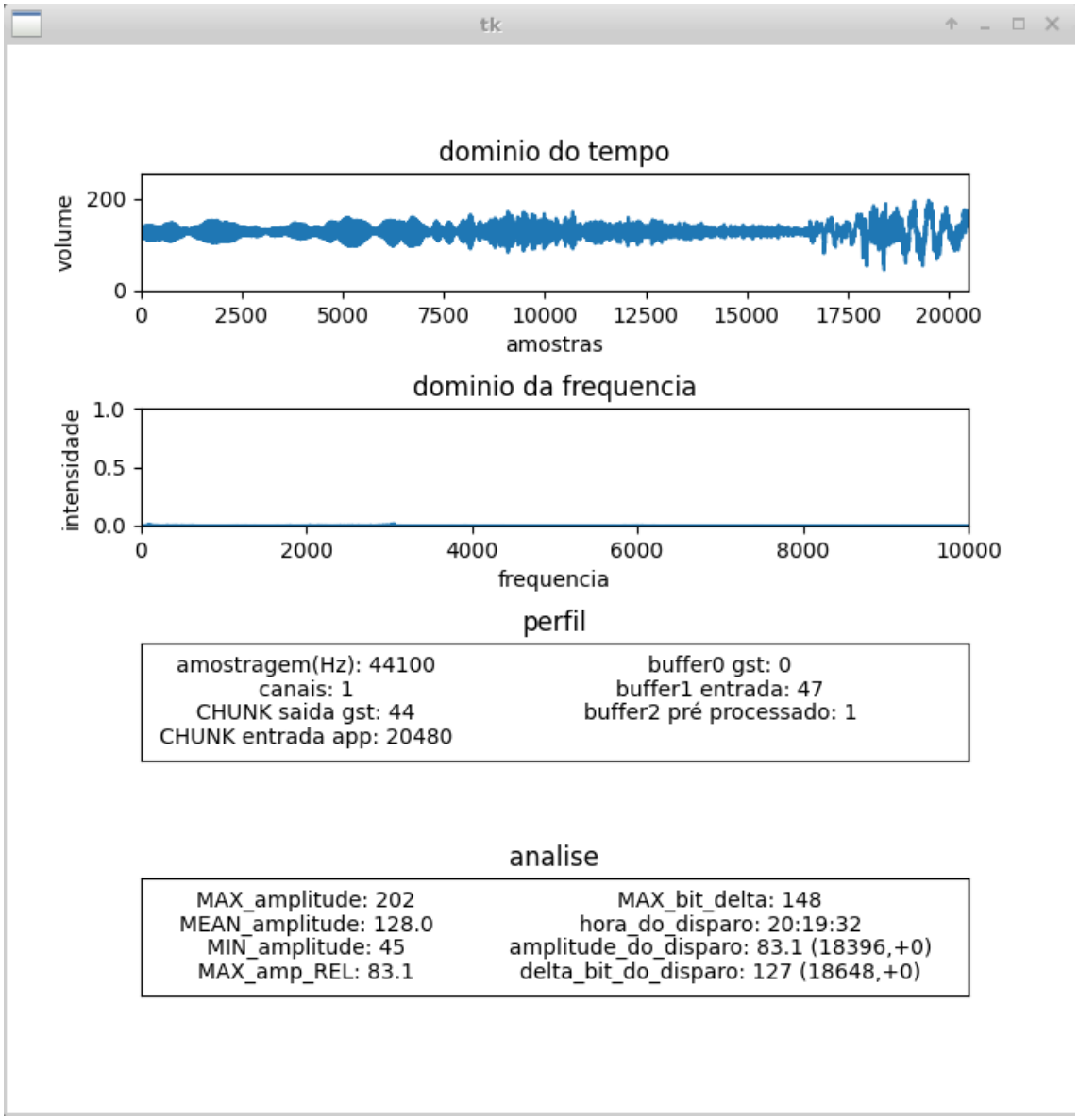


Figura 6 – Captura da tela de análise do áudio no momento em que ocorre a detecção do disparo (teste nº2).

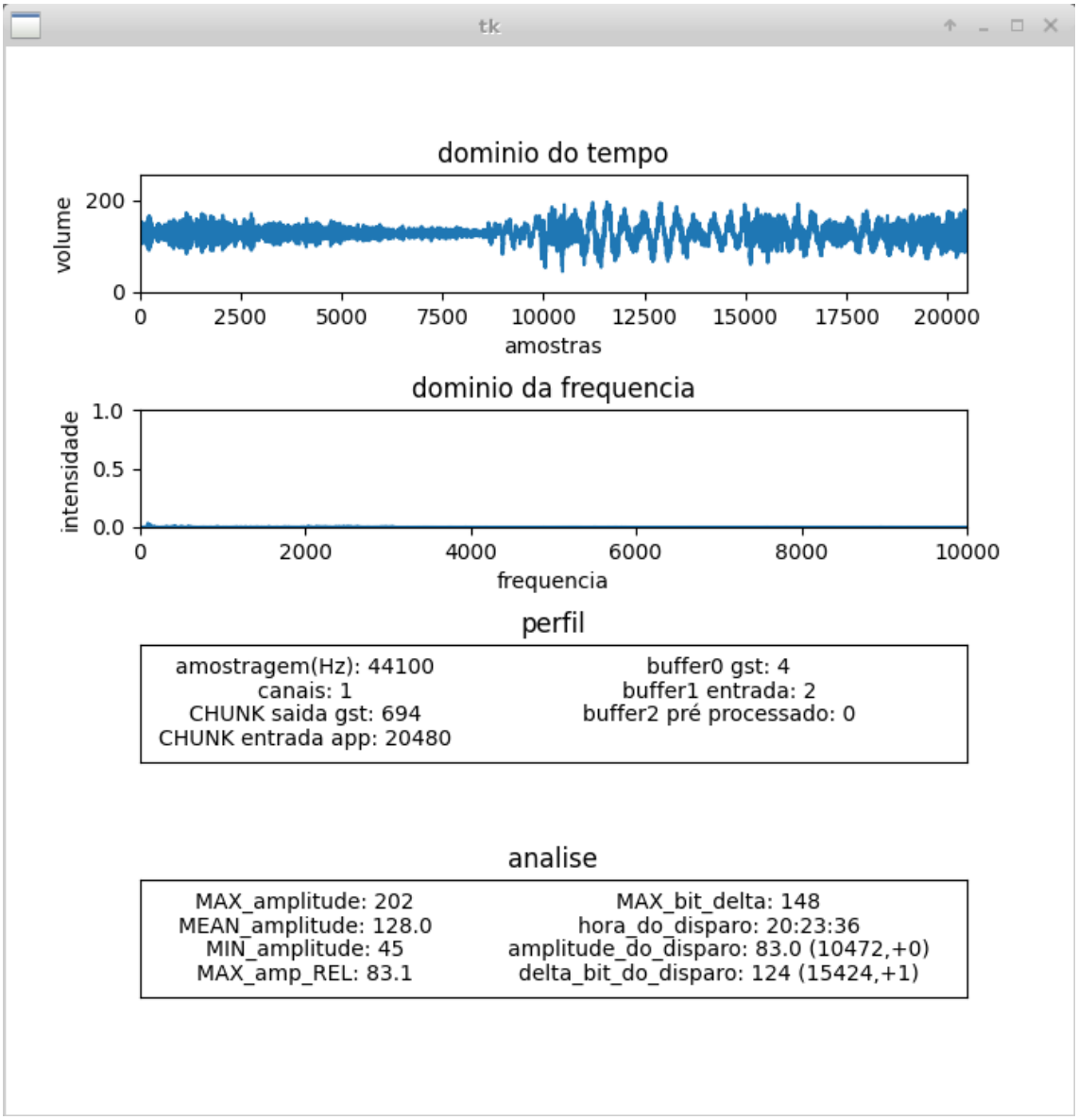


Figura 7 – Captura da tela de análise do áudio no momento em que ocorre a detecção do disparo (teste nº3).

4. O VMS no V-PRISM

Para implementar o VMS no V-PRISM é necessário que este esteja dentro de um container Docker. E por isso também foi necessário desabilitar a visualização dos dados, pois não há interface gráfica neste container. A visualização dos dados serviu para desenvolver a lógica e parâmetros de detecção do disparo. O disparo continua ser detectado, mas sem interface gráfica.

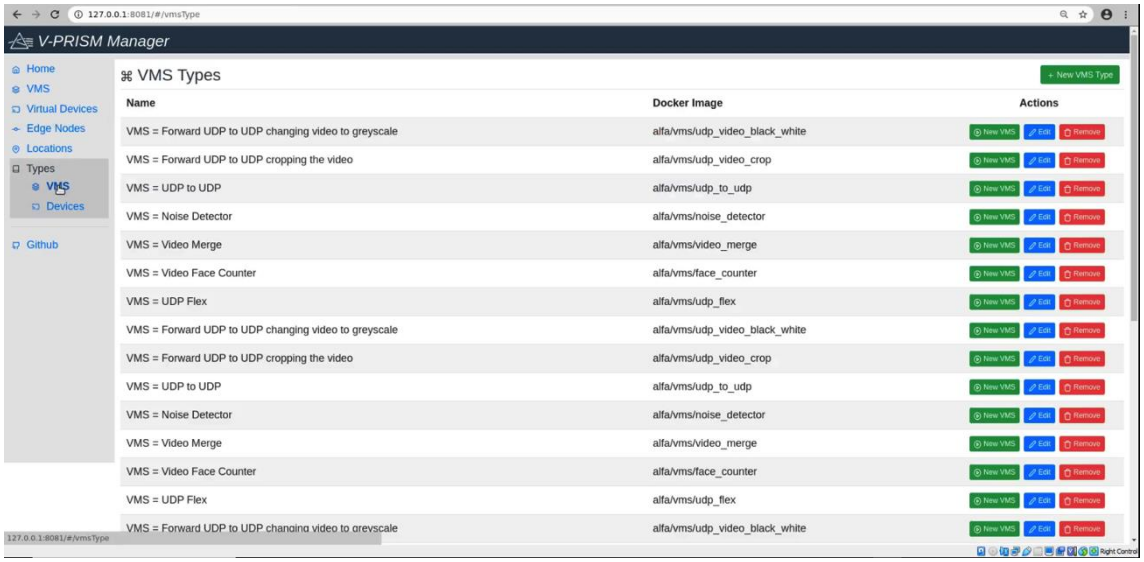
Para testar a saída do VMS que implementado no V-PRISM, execute o teste realizado no item anterior, mas desta vez rodando o “main final.py” em vez de “main beta 2.py”. A saída do sensor sem a interface gráfica será no terminal, simulando o envio das informações ao servidor MQTT a cada 3 segundos:

Para iniciar a implementação do VMS no ALFA V-PRISM, deve-se seguir os passos abaixo:

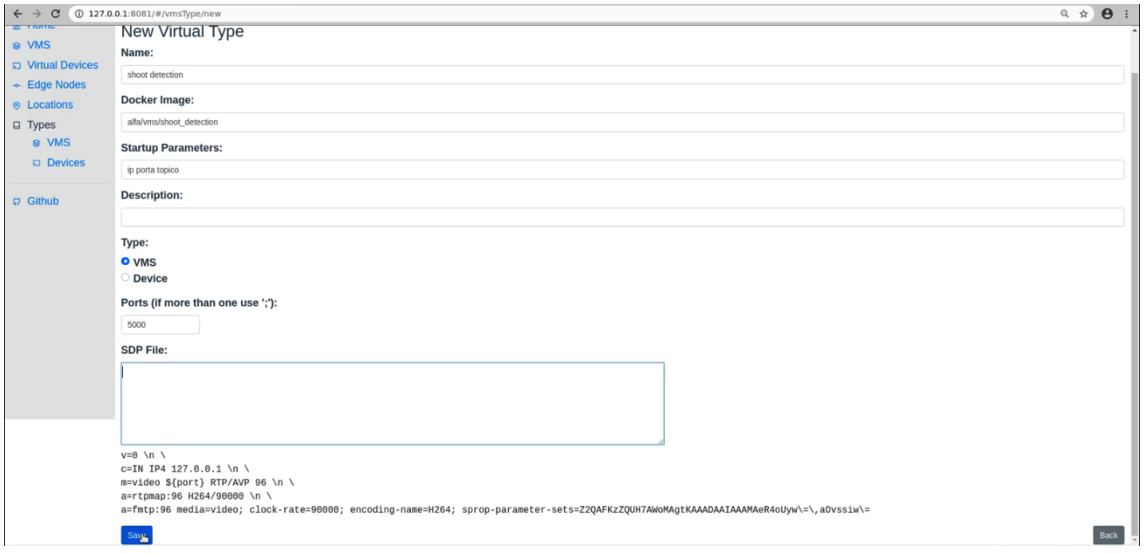
- **$ cd beta_VMS** (no terminal do Linux, vá para a pasta que foi clonada)
- **$ docker build . -t alfa/vms/shoot_detection** (construa a imagem docker)

```
bruno@unbruno:~/Vprism/beta_VMS$ docker build . -t alfa/vms/shoot_detection
Sending build context to Docker daemon  559.6kB
Step 1/15 : FROM python:3.6-slim-stretch
 ---> bc653a6a7211
Step 2/15 : RUN apt-get -y update
 ---> Using cache
 ---> ce3a6c68e729
Step 3/15 : RUN apt-get install -y --fix-missing     build-essential     cmake     gfortran     git     w
get     curl     graphicsmagick     libgraphicsmagick1-dev     libatlas-dev     libavcodec-dev     libavf
ormat-dev     libgtk2.0-dev     libjpeg-dev     liblapack-dev     libswscale-dev     pkg-config     pytho
n3-dev     python3-numpy     software-properties-common     zip     bash     python-gi     python-gi-cair
o     python-dbus     gir1.2-gtk-3.0     gir1.2-gstreamer-1.0     gir1.2-gst-plugins-base-1.0     gstream
er1.0-plugins-base     gstreamer1.0-plugins-good     gstreamer1.0-plugins-ugly     gstreamer1.0-pulseaudi
o     gstreamer1.0-plugins-bad     && apt-get clean && rm -rf /tmp/* /var/tmp/*
 ---> Using cache
 ---> b5af1aecb300
Step 4/15 : RUN apt-get install -y --fix-missing     python3-gi     python3-gst-1.0
 ---> Using cache
 ---> b9be1fe27d91
Step 5/15 : RUN python -m pip install --upgrade pip
 ---> Using cache
 ---> 660a86bc3eb3
Step 6/15 : RUN pip3 install paho-mqtt
 ---> Using cache
 ---> 4eaac1accd40
Step 7/15 : RUN pip3 install opencv-python
 ---> Using cache
 ---> e531eb6befd6
Step 8/15 : RUN pip3 install numpy==1.15.4
 ---> Using cache
 ---> cb8f3de2fa43
Step 9/15 : RUN apt-get install -y --fix-missing libgirepository1.0-dev
 ---> Using cache
 ---> 00cd110f39bd
Step 10/15 : RUN pip install --no-binary gobject PyGObject
 ---> Using cache
 ---> 5e6cc0ef634c
Step 11/15 : RUN apt-get install -y --fix-missing gstreamer1.0-libav
 ---> Using cache
 ---> 5635b91374d2
Step 12/15 : RUN apt-get install -y --fix-missing     tcpdump     net-tools
 ---> Using cache
 ---> 89f07b8109e6
Step 13/15 : RUN pip3 install scipy
 ---> Running in ddac5760ef30
Collecting scipy
  Downloading scipy-1.5.0-cp36-cp36m-manylinux1_x86_64.whl (25.9 MB)
Requirement already satisfied: numpy>=1.14.5 in /usr/local/lib/python3.6/site-packages (from scipy) (1.15
.4)
Installing collected packages: scipy
Successfully installed scipy-1.5.0
Removing intermediate container ddac5760ef30
 ---> 76e1e135bb32
Step 14/15 : COPY . /root/shoot_detection
 ---> 8adb766210c7
Step 15/15 : ENTRYPOINT ["./root/shoot_detection/start.sh"]
 ---> Running in c7c4cbbbf812
Removing intermediate container c7c4cbbbf812
 ---> ca322c25c7b9
Successfully built ca322c25c7b9
Successfully tagged alfa/vms/shoot_detection:latest
```

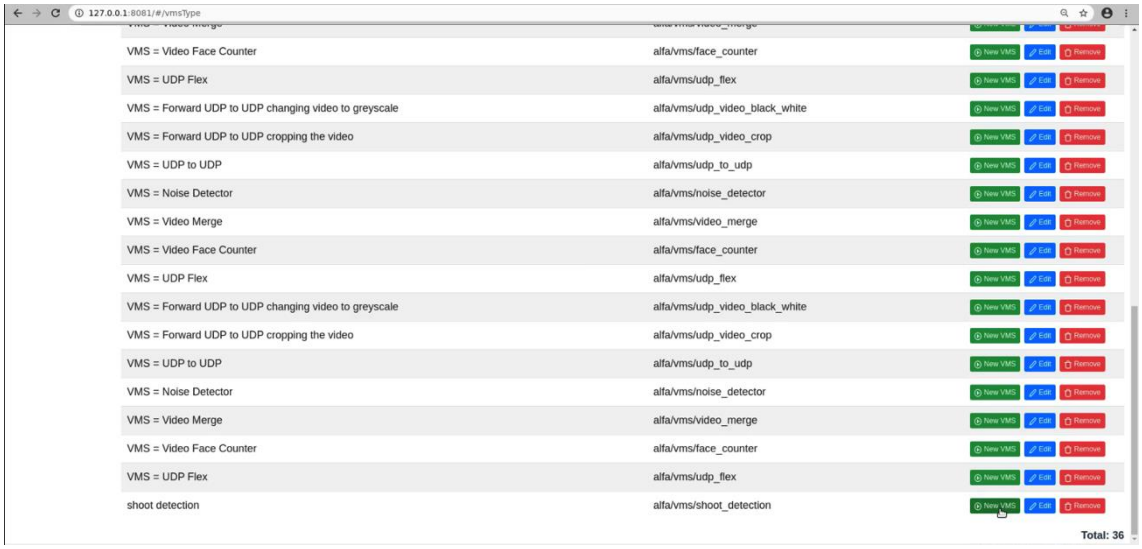
- **Acessar a interface WEB do ALFA e clicar em VMS dentro do menu Types** (Antes deste passo é necessário baixar, implementar e rodar o ALFA conforme instruções no https://github.com/midiacom/alfa)
- **Clicar em "New VMS Type"**



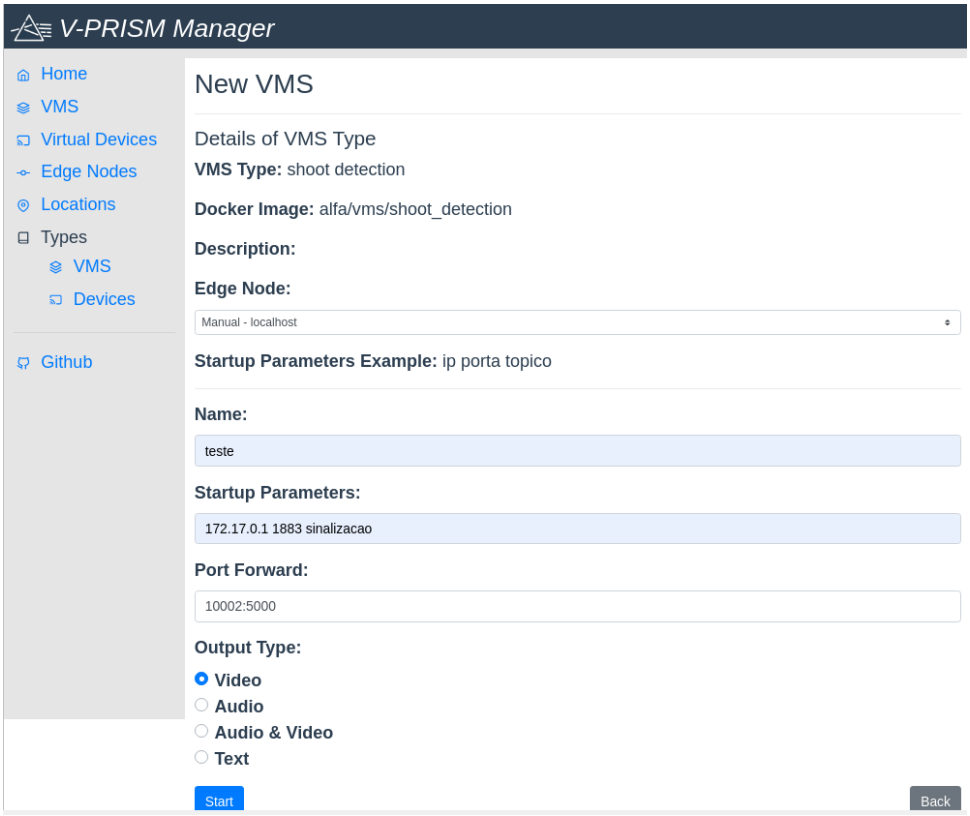
- **Preencher os campos conforme exemplo abaixo e clicar em "Save"** (Esta etapa cria um tipo de VMS no ALFA)



- **Ir novamente na tela que lista os tipos de VMS clicar em "New VMS" na linha correspondente ao tipo de VMS que acabou de ser criado** (Esta etapa cria o VMS propriamente dito dentro do ALFA)



- **Ir novamente na tela que lista os tipos de VMS clicar em "New VMS" na linha correspondente ao tipo de VMS que acabou de ser criado**
- **Preencher os campos conforme exemplo abaixo** (Esta etapa cria o VMS propriamente dito dentro do ALFA, com startup parameters: 172.17.0.1 1883 sinalizacao. Estes são o endereço IP e a porta do servidor MQTT para onde o VMS irá enviar a sinalizacao do disparo da arma de fogo. "sinalizacao" é um nome dado ao tópico do MQTT)



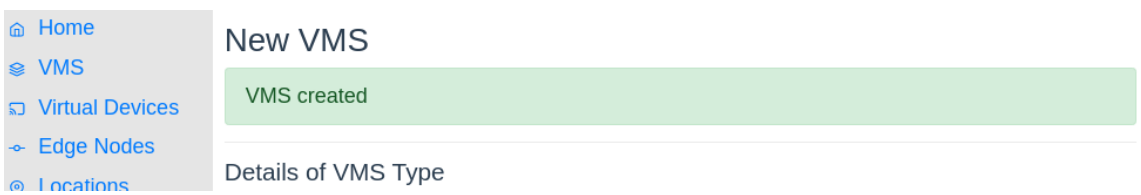
- **Antes de clicar em "Start" verifique se o endereço e a porta do servidor MQTT**
- **$ ifconfig** (verificar o ip docker0)

```
bruno@unbruno:~$ ifconfig
docker0: flags=4099<UP,BROADCAST,MULTICAST>  mtu 1500
        inet 172.17.0.1  netmask 255.255.0.0  broadcast 172.17.255.255
        inet6 fe80::42:feff:fe70:4952  prefixlen 64  scopeid 0x20<link>
        ether 02:42:fe:70:49:52  txqueuelen 0  (Ethernet)
        RX packets 29492  bytes 1774981 (1.7 MB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 33792  bytes 106276223 (106.2 MB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

docker_gwbridge: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 172.18.0.1  netmask 255.255.0.0  broadcast 172.18.255.255
        inet6 fe80::42:b4ff:fe8e:ace8  prefixlen 64  scopeid 0x20<link>
        ether 02:42:b4:8e:ac:e8  txqueuelen 0  (Ethernet)
        RX packets 1679  bytes 11632436 (11.6 MB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 1968  bytes 455107 (455.1 KB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

enp0s3: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 10.0.2.15  netmask 255.255.255.0  broadcast 10.0.2.255
        inet6 fe80::a00:27ff:fe89:4ff4  prefixlen 64  scopeid 0x20<link>
        ether 08:00:27:89:4f:f4  txqueuelen 1000  (Ethernet)
        RX packets 353309  bytes 366921007 (366.9 MB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 102062  bytes 7376809 (7.3 MB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

lo: flags=73<UP,LOOPBACK,RUNNING>  mtu 65536
        inet 127.0.0.1  netmask 255.0.0.0
        inet6 ::1  prefixlen 128  scopeid 0x10<host>
        loop  txqueuelen 1000  (Local Loopback)
        RX packets 28565  bytes 15576208 (15.5 MB)
        RX errors 0  dropped 0  overruns 0  frame 0
```

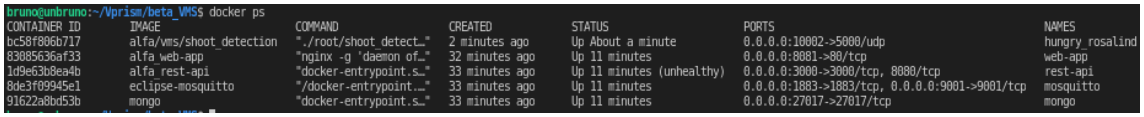
- **$ ./list_docker_fw.sh** (verificar a porta do servidor MQTT)

```
bruno@unbruno:~/Vprism/alfa/tools$ ./list_docker_fw.sh
CONTAINER ID        NAMES               PORTS
b70cbc9781c8        web-app             0.0.0.0:8081->80/tcp
05b6cef0f3f2        rest-api            0.0.0.0:3000->3000/tcp, 8080/tcp
53a45ece5f4a        mosquitto           0.0.0.0:1883->1883/tcp, 0.0.0.0:9001->90
01/tcp
83268c1c5a71        mongo               0.0.0.0:27017->27017/tcp
```

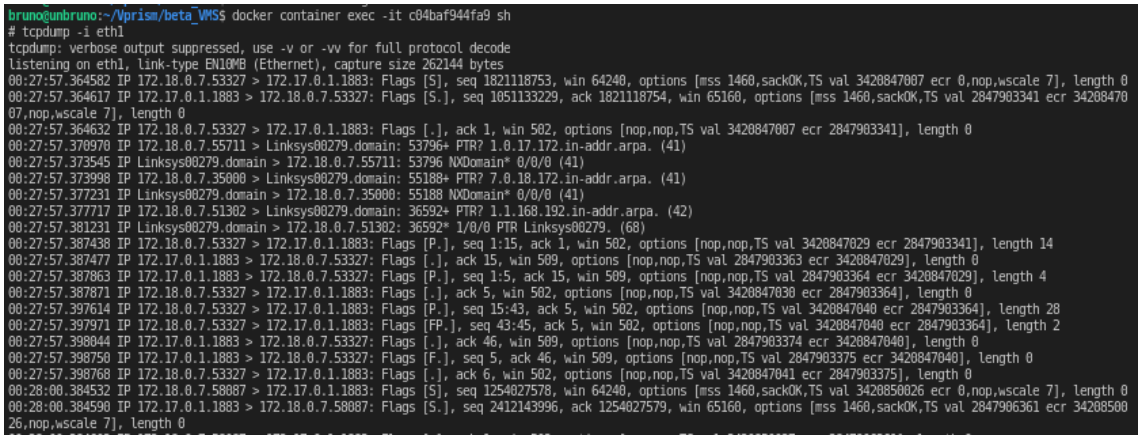
- **Após iniciar o VMS, a tela abaixo será mostrada**



- **$ Docker ps** (para verificar se o container esta rodando com os demais containers do ALFA)



- **$ gst-launch-1.0 filesrc location=/home/bruno/Vprism/beta_VMS/teste.wav ! decodebin ! audioconvert ! audioresample ! audio/x-raw,channels=1,depth=16,width=16,rate=44100 ! rtpL16pay ! udpsink host=172.17.0.1 port=10002** (comando para testar a captura de audio dentro do container – pode ser digitado em um terminal fora do container, ou seja, no mesmo terminal que estava sendo usado. Este comando vai enviar um áudio teste da máquina local para o VMS, que teoricamente poderia estar em qualquer lugar da internet)
- **$ docker container exec -it xxxxxx sh** (entre no container, substitua xxxxxx por CONTAINER ID do VMS observável pelo comando **Docker ps**)
- **$ tcpdump -i eth1** (visualize o fluxo enviado ao servidor MQTT, sinalizando a detecção do áudio e eventual identificação do disparo, e o fluxo UDP do áudio recebido, respectivamente, nos prints abaixo)



```
00:21:54.175612 IP 172.18.0.1.40973 > 172.18.0.7.5000: UDP, length 1400
00:21:54.190776 IP 172.18.0.1.40973 > 172.18.0.7.5000: UDP, length 1400
00:21:54.206883 IP 172.18.0.1.40973 > 172.18.0.7.5000: UDP, length 1400
00:21:54.222244 IP 172.18.0.1.40973 > 172.18.0.7.5000: UDP, length 1400
00:21:54.238097 IP 172.18.0.1.40973 > 172.18.0.7.5000: UDP, length 106
00:21:54.239057 IP 172.18.0.1.40973 > 172.18.0.7.5000: UDP, length 1400
00:21:54.255185 IP 172.18.0.1.40973 > 172.18.0.7.5000: UDP, length 1400
00:21:54.270808 IP 172.18.0.1.40973 > 172.18.0.7.5000: UDP, length 1400
00:21:54.286431 IP 172.18.0.1.40973 > 172.18.0.7.5000: UDP, length 1400
00:21:54.302125 IP 172.18.0.1.40973 > 172.18.0.7.5000: UDP, length 104
00:21:54.303371 IP 172.18.0.1.40973 > 172.18.0.7.5000: UDP, length 1400
00:21:54.319105 IP 172.18.0.1.40973 > 172.18.0.7.5000: UDP, length 1400
00:21:54.334702 IP 172.18.0.1.40973 > 172.18.0.7.5000: UDP, length 1400
00:21:54.350444 IP 172.18.0.1.40973 > 172.18.0.7.5000: UDP, length 1400
00:21:54.366030 IP 172.18.0.1.40973 > 172.18.0.7.5000: UDP, length 106
00:21:54.367178 IP 172.18.0.1.40973 > 172.18.0.7.5000: UDP, length 1400
00:21:54.383219 IP 172.18.0.1.40973 > 172.18.0.7.5000: UDP, length 1400
00:21:54.398819 IP 172.18.0.1.40973 > 172.18.0.7.5000: UDP, length 1400
00:21:54.414430 IP 172.18.0.1.40973 > 172.18.0.7.5000: UDP, length 1400
00:21:54.430193 IP 172.18.0.1.40973 > 172.18.0.7.5000: UDP, length 104
00:21:54.431463 IP 172.18.0.1.40973 > 172.18.0.7.5000: UDP, length 1400
00:21:54.447638 IP 172.18.0.1.40973 > 172.18.0.7.5000: UDP, length 1400
00:21:54.462656 IP 172.18.0.1.40973 > 172.18.0.7.5000: UDP, length 1400
00:21:54.478351 IP 172.18.0.1.40973 > 172.18.0.7.5000: UDP, length 1400
00:21:54.494015 IP 172.18.0.1.40973 > 172.18.0.7.5000: UDP, length 104
00:21:54.495259 IP 172.18.0.1.40973 > 172.18.0.7.5000: UDP, length 1400
00:21:54.510896 IP 172.18.0.1.40973 > 172.18.0.7.5000: UDP, length 34
00:21:54.511695 IP 172.18.0.1.40973 > 172.18.0.7.5000: UDP, length 100
```

- **$ ps -elf** (verifique se os processos estão rodando normalmente. Caso não consiga visualizar o fluxo de saída pelo tcpdump, execute manualmente o programa dentro do container: # python /root/shoot_detection/main_final.py 172.17.0.1 1883 sinalizacao)

```
# ps -elf
F S UID        PID  PPID  C PRI  NI ADDR SZ WCHAN  STIME TTY          TIME CMD
4 S root         1     0  1  80   0 -  4928 wait   01:11 ?        00:00:00 /bin/bash ./root/shoot_detection/start.sh 172.17.0.1 1883 sinalizacao
4 S root         7     1 66  80   0 - 231389 futex_ 01:11 ?        00:00:36 python /root/shoot_detection/main_final.py 172.17.0.1 1883 sinalizacao
4 S root        22     0  2  80   0 -  1070 wait   01:12 pts/0    00:00:00 sh
0 R root        29    22  0  80   0 -  9596 -      01:12 pts/0    00:00:00 ps -elf
#
```

- **$ A sinalização é enviada a cada 3 segundos** (não há visualização gráfica)

```
detectando...
detectando...
detectando...
detectando...
detectando...
detectando...
detectando...
detectando...
detectando...
detectando...
detectando...
detectando...
detectando...
detectando...
disparo ocorrido às00:15:27
disparo ocorrido às00:15:27
disparo ocorrido às00:15:27
detectando...
```

5. Limitações e trabalhos futuros

O algoritmo de detecção de disparos pode ser considerado um algoritmo “dummy”, com único propósito de servir de início de desenvolvimento de um detector de disparos com taxa de detecção e precisão razoáveis. Detectar com qualidade o som do *muzzle blast* ("estouro do cano"), através da onda impulsiva (variação abrupta da amplitude), com energia concentrada em faixas de frequência mais baixas não é trivial e se encaixa em um objetivo de melhoria continua.

A escolha do limiar de amplitude e variação da amplitude dependem da regulação do microfone de captura, ou seja, precisam ser regulados considerando um ambiente real, onde cada microfone vai enviar o seu áudio com “volume” diferente do outro.

A eventual perda de pacotes por UDP geralmente faz com que haja ocorrência de falsa variação abrupta de amplitude, já que o último bit de uma amostra não bate com o primeiro bit da amostra seguinte. A variação de amplitude entre um bit e outro (variável MAX_bit_delta) entre duas amostras que não são naturalmente sequenciais pode ser um número fora da realidade esperada, e geralmente é.

A Transformada Rápida de Fourier inicialmente utilizada para análise precisou ser desativada por não ser tão rápida quanto a chegada de novas amostras no buffer para processamento. Também é preciso verificar se as frequências detectadas “casam” com as frequências indicadas no gráfico.

Análises mais complexas, como a de Transformada Rápida de Fourier, podem ser adicionadas ao algoritmo em tempo real, mais precisam ser estruturadas de forma paralela e coordenada para que não haja entrada no buffer maior que a saída.

A análise da amplitude é feita por “chunks”, ou partes isoladas. Caso o início do som do disparo esteja no final de um “chunk”, e o restante do som esteja no “chunk” seguinte, este pode passar sem ser detectado.

A análise da amplitude e da variação da amplitude também dependem do “momento” da taxa de amostragem. Como um mesmo som pode ter um “momento” de quantização diferente dependendo de que instante ele começou a ser digitalizado, o mesmo som de disparo transmitido pode ora ser detectado ora não. Caso os limiares de detecção estejam muito elevados, para reduzir incidência de falsos positivos por exemplo. O pico alcançado na amostra pode não acionar a detecção do disparo. Nos testes realizados, o limiar de amplitude foi 80.0. Em 3 testes, no mesmo trecho foram detectados os valores máximos: 83.0; 83.1; 83.0, em cada teste. Caso o limiar fosse de 83.1, o mesmo disparo teria 33% de chance de ser detectado a cada vez que ocorresse.

Para aprimorar o algoritmo de detecção é primordial que se aprimore a visualização dos dados. A visualização dos dados não está com taxa de atualização razoável e isto dificulta o desenvolvimento dos critérios de detecção.

Possíveis próximos passos:

- Aprimorar a visualização dos dados
- Aprimorar o critério de limiar de amplitude e variação de amplitude
- Adicionar análise de frequência (analisar qual a frequência predominante no som do disparo e utilizar este critério como detecção)

- Adicionar análise por redes neurais (Em sistemas proprietários, a detecção do disparo pode ser dividida em 2 etapas. 1 - A detecção da onda impulsiva e 2 – A identificação de qual tipo de arma que efetuou o disparo. A primeira etapa ocorre “no próprio sensor”. Ao ser detectada, o trecho de áudio correspondente à onda impulsiva é gravado e enviado para a nuvem, para uma análise mais robusta, utilizando redes neurais)

6. Conclusão

É possível implementar, com precisão razoável, a detecção de disparos de arma de fogo em um VMS na arquitetura V-PRISM. A qualidade da detecção depende principalmente da qualidade do algoritmo de detecção e da taxa de amostragem. Aprimoramentos podem ser feitos no presente algoritmo para que a sua taxa de detecção e precisão sejam aumentados. Por se tratar de análise em tempo real, a qualidade de conexão de rede entre a captura do áudio e o VMS também é essencial.

Principais Referências Utilizadas

High-speed Imaging of Shock Waves, Explosions and Gunshots BY GARY S. SETTLES https://www.americanscientist.org/article/high-speed-imaging-of-shock-waves-explosions-and-gunshots

Métodos para Implementação de Sistemas de Detecção de Disparos de Arma de Fogo de Baixo Custo - Igor Dantas dos Santos Miranda - Universidade Federal da Bahia – UFBA - Programa de Pós-Graduação em Engenharia Industrial - Dezembro de 2017 https://repositorio.ufba.br/ri/bitstream/ri/25024/1/Tese%20-%20Igor%20Miranda%20-%20Vers%c3%a3o%20Final%20com%20Capa.pdf

Acoustic and psychoacoustic analysis of the noise produced by the police force firearms - Análise acústica e psicoacústica do ruído de armas utilizadas pela Polícia Militar - Heraldo Lorena Guida1, Thiago Hernandes Diniz2, Sérgio Koodi Kinoshita3 http://oldfiles.bjorl.org/conteudo/acervo/acervo.asp?id=4122

Gunshot detection in noisy environments - Izabela L. Freire and Jos´e A. Apolin´ario Jr. Program of Defense Engineering Military Institute of Engineering (IME) Rio de Janeiro, Brazil https://www.researchgate.net/publication/228741330_Gunshot_detection_in_noisy_environments