Application Note

Akademie věd České republiky
Ústav teorie informace a automatizace AV ČR, v.v.i.

UTIA Acoustic Phased Array Toolkit

Zdeněk Pohl, Lukáš Kohout

zdenek.pohl@utia.cas.cz, kohoutl@utia.cas.cz

Revision history

Rev.	Date	Author	Description
00	20.05.2026	Z.Pohl, L.Kohout	Document creation

Contents

Acknowledgement

Under grant 101096884, Listen2Future is co-funded by the European Union. Views and opinions expressed are however those of the author(s) only and do not necessarily reflect those of the European Union or Chips Joint Undertaking. Neither the European Union nor the granting authority can be held responsible for them. The project is supported by the CHIPS JU and its members (including top-up funding by Austria, Belgium, Czech Republic, Germany, Netherlands, Norway and Spain).

National Funding

This project has also received national funding from the Ministry of Education, Youth and Sports of the Czech Republic (MEYS) under grant agreement No 9A22004.

1 Description

The UTIA L2F Toolkit is a collection of software tools developed by the Institute of Information Theory and Automation (UTIA) within the Listen2Future (L2F) project. These tools can be divided into two groups:

1. First group contains tools designed to run on the UTIA acoustic camera embedded platform based on the Zynq UltraScale+ architecture. They support the full workflow from data acquisition to real-time AI-based detection of acoustic events.
2. Second group contains PC based applications with or without GUI for monitoring, displaying or recording data from embedded platform.

The embedded platform tools of the Toolkit deployed within the PetaLinux environment are summarized in the following table:

L2F Toolkit for Embedded Platform		
Tool	Location	Description
`capture.elf`	Petalinux /home/root	RAW data capture application for recording microphone array data and optionally also the video synchronous to audio data.
`xcalib2.elf`	Petalinux /home/root	Multipurpose tool for camera–acoustic cross-calibration, calibration verification, and beamformer configuration.
`bf_preproc2.elf`	Petalinux /home/root	Tool for preprocessing RAW array data from file into spectrogram patches as images, for AI model training. The tool uses hardware accelerated processing chain on FPGA platform.
`bf_detect2.elf`	Petalinux /home/root	Real-time application for array data capture, preprocessing, AI inference, post-processing, and cloud reporting.
`bridge.elf`	Petalinux /home/root	Connection between user application and optional LoRa module.

The PC tools are mostly provided as the python scripts executable in Linux or Windows environment:

L2F Toolkit for PC	
Tool	Description
`udp_viewer2_tk.py`	The GUI application displaying live telemetry data from embedded platform while bf_detect2.elf application is running.

`loRa_sim`	With USB-UART cable connected to LoRa UART interface on embedded platform the LoRa node communication is simulated
`udp_player2_qt.py`	Command-line application which playbacks PCM data from embedded platform while bf_detect2.elf is running. The tool also stores WAV file.

2 Prerequisites

2.1 UTIA Execution Platform

To use the tools described in this document. The dedicated embedded platform must be used. The platform consists of:

1. UTIA Ultrasound EV Board v2.x – the PCB extension board with the phased array of 32 Infineon digital microphones.
2. Trenz Electronic TE0706 carrier board – the carrier board for Zynq Based FPGA SoM providing I/O capabilities and power.
3. Trenz Electronic TE0821 SoM module with Zynq Ultrascale+ 3CG module with 4GB memory.
4. 3D printed frame.
5. TE0821 SoM Cooling radiator, optionally with fan.
6. USB Webcam.

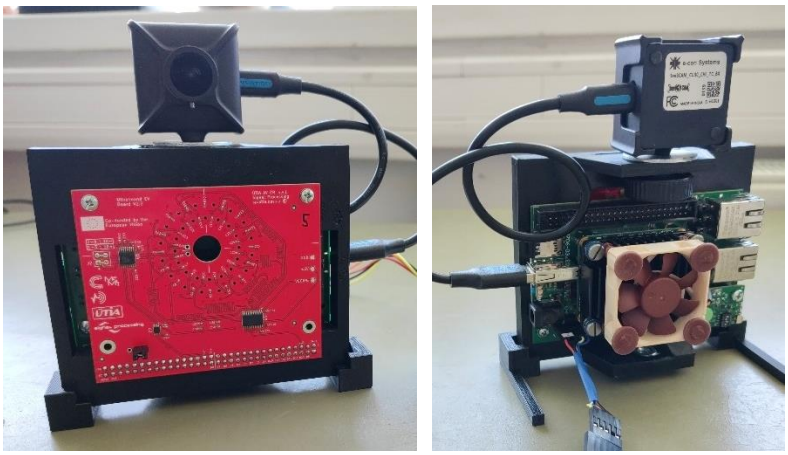

Figure 1: Embedded platform

To obtain the platform please contact authors of this document.

2.2 Firmware

To use the platform the microSD card with firmware must be used:

1. Use microSD card with capacity >= 4 GB.
2. Write the SD card image as provided with this application note in firmware folder.
3. Plug the microSD card to slot on TE0706 carrier

4. Connect the board to local network using UTP cable:
    a. Board IP address is fixed to 192.168.2.179
    b. PC must be in the same network.
5. (optional step) Connect the USB – UART cable to PC
6. Power on the board.
7. Open terminal from PC with X11 forwarding enabled.
8. (optional step) Open serial terminal connection to PC: 115200bps, none, 8bit, none
9. Use tools on embedded platform from X11 enabled terminal as shown in following chapters.

3 Embedded Platform Tools

3.1 Raw Data Capture Tool

The capture tool is used to collect raw data from the microphone array. It utilizes hardware support available in the embedded platform to ensure synchronous acquisition across all microphones.

In addition to acoustic data acquisition, the tool can also record video from a connected web camera if available. When this feature is enabled, a video stream is stored together with the recorded array data at a user-defined frame rate.

Within the Listen2Future project, this application is primarily used for collecting training datasets for neural network models as well as for array performance analysis in Matlab, capable to identify problems with individual microphones of the array.

Figure 2: Examples of capture setup. The UTIA platform with microphone array facing front is mounted on tripod, powered and connected to PC using Ethernet.

Command Line Interface

```
capture.elf [-s time_in_sec -m time_in_msec -e -c cam_index -f fps -g config_file] -o fname [-v verbosity_level -t]
```

Parameter	Description
`-v verbosity_level`	Set verbosity level: 0 – off, 1 – errors, 2 – debug
`-o filename`	Output filename without extension
`-s NUMBER`	Recording time in seconds (optional, default 1s)
`-m NUMBER`	Recording time in milliseconds (optional, default: 1000 ms). Options –s and –m options can't be used at the same time.
`-c cam_index`	Capture frames from webcam (video device index must be specified)
`-d camera_calibration`	Camera calibration XML file used for undistorting frames
`-f fps`	Video frame rate when camera capture is enabled (default: 1 FPS)
`-t`	Use external trigger (otherwise capture starts immediately)
`-g cfg_file`	Configuration file (default: `evboard_ffbf.cfg`).
`-h`	Help

Example usage

```
> capture.elf –o array_data_1s.raw –s 1 –g evboard.cfg
```

For example, when simulated partial discharge events were captured for AI training, approximately 55.8 GB of raw array recordings were acquired using this tool.

3.2 Camera – Acoustic Cross-Calibration Tool

The xcalib2 application provides multiple operational modes related to calibration and configuration of the acoustic camera system.

The available modes are:

- capture
- calibrate
- verify
- configure

Each mode performs a different step of the calibration workflow.

Figure 3: Signal source for camera – acoustic cross-calibration.

3.2.1 Capture Mode

The capture mode records calibration data used later for computing the spatial calibration between the camera and the acoustic array.

This mode requires:

- a connected web camera
- preferably the camera intrinsic calibration file (to correct lens distortion)
- the 40 kHz acoustic calibration source, shown in Figure 3**.**

For proper operation, the application must be executed via SSH with X11 forwarding, because it is interactive and opens a graphical window showing live:

- the camera preview
- the requested placement of the calibration source

The captured calibration data are stored in the specified output directory. The actual calibration is performed later in the calibrate step.

Command Line Interface

`xcalib2.elf capture –w NUM [–i FILE -p NUM –y –v LEVEL –h -f] –o DIR`	
Parameter	**Description**
-w NUM	Webcam device index
-i FILE	Camera intrinsics file for undistortion
-p NUM	Number of calibration points
-y	Non-interactive mode (auto-confirm prompts)
-v LEVEL	Verbosity level, 0 – none, higher value – more verbose output

-h	Help
-o DIR	Output folder for captured calibration data
-f	Flip preview

Example Usage

```
> xcalib2.elf capture –w 0 –i seecam_640_480_camera_data.xml -p 20 -o board_assembly_01 -f
```

This example initiates the acquisition of 20 calibration points using the webcam with video device index 0 and the acoustic array. The camera intrinsic parameters are used to undistort the captured images, and all collected data are stored in the directory *board_assembly_01*. Live image preview will be flipped so it behaves like mirror making manual calibration process user friendly.

The point acquisition process is interactive. The user must follow the instructions displayed in the terminal while simultaneously monitoring the preview window generated by the application. This window is forwarded to the host system via X11 forwarding, allowing the user to visually verify the placement of the calibration source during the capture process, see Figure 4. The calibration tool must be placed to position of white cross projected in the image. The tool must be powered on and setup to generate 40 kHz continuous signal. The corresponding video snapshot and acoustic data are stored when key ‘q’ is pressed.

Figure 4: Example snapshots of the capture process of calibration data for 2 of 20 points.

3.2.2 Calibration Mode

The calibrate mode computes the camera-acoustic cross-calibration using the captured data from previous step.

The computation requires:

- Acoustic data captured in ‘xcalib2 capture’ mode.
- microphone array geometry
- the projection plane used for acoustic imaging

The resulting calibration parameters are stored in the same directory as the captured calibration data.

Command Line Interface

`xcalib2.elf calibrate -i DIR -m FILE -g FILE [-h -v LEVEL -f]`	
Parameter	Description
-i DIR	Folder containing data from the capture step
-m FILE	CSV file with microphone coordinates.
-g FILE	Acoustic camera projection plane specification.
-h	Help
-v LEVEL	Verbosity level, 0 – none, higher value – more verbose output
-f	Horizontal flip preview

The ordering of microphones in the CSV file specified by the *-m* option must correspond exactly to the ordering of microphone data in the RAW data file.

The acoustic camera projection plane is defined in a configuration file by specifying the coordinates of the top-left corner point, the step sizes in the x and z directions (all in millimeters), and the number of grid rows and columns. The focal distance in millimeters (y direction) of the projection plane must also be specified in the same file.

Additionally, the configuration allows selection of the distance computation model used for acoustic projection. Depending on the application, distances can be calculated either using a point-to-plane model, suitable for far-field beamforming, or a point-to-point model, which is appropriate for near-field beamforming.

3.2.3 Verification Mode

The verify mode allows validation of the calibration outcome.

In this mode, the embedded platform operates as a live acoustic camera. The acoustic beamforming results are displayed as an overlay on the webcam image and uses semi-transparent overlay with ‘jet’ colormap. This mode also requires SSH with X11 forwarding to view the result in X11 host window.

Command Line Interface

`xcalib2.elf verify -i DIR [-h -v LEVEL -f]`	
Parameter	Description
-i DIR	Folder containing calibration results
-h	Help
-v LEVEL	Verbosity level, 0 – none, higher value – more verbose output
-f	Horizontal flip preview

Figure 5: Sample snapshot of live acoustic camera verification.

3.2.4 Configuration Mode

The configure mode is used to define measurement points in the scene.

These points have to be manually defined and placed to locations where acoustic activity should be monitored by the detector.

This step is interactive and also requires SSH with X11 forwarding. One by one, the monitored locations with circular position tolerance must be specified. At the end of the process, the tool asks for point names and their real distance from the sensor which can be manually measured by laser range finder for example.

Command Line Interface

`xcalib2.elf configure -i DIR [-y -d -h -v LEVEL]`	
Parameter	Description
-i DIR	Folder with calibration results
-y	Non interactive mode
-d	Use default point names and distances
-h	Help
-v LEVEL	Verbosity level, 0 – none, higher value – more verbose output

Figure 6: Example scene observed by the detector with two (red cross) defined locations with circular tolerance (blue).

3.3 Preprocessing Tool

The bf_preproc2 tool processes RAW array recordings and generates preprocessed datasets for AI training.

It reads microphone array data and performs preprocessing using a hardware-accelerated signal processing pipeline implemented in FPGA.

The preprocessing chain produces:

- spectrogram representations
- RMS energy estimates in selected frequency bands

The output dataset consists of spectrogram patches and corresponding RMS features, which are later used to create training, validation, and testing datasets for AI classifiers.

An example spectrogram computed by the tool together with the corresponding band-pass RMS signal is shown in Figure 7. The spectrogram contains 513 frequency bins produced by the discrete Fourier transform (FFT) and covers the frequency range 0-96 kHz.

The simulated partial discharge (PD) pulses used in the laboratory experiments are generated using an ultrasound-capable speaker from Kemo, which has a limited bandwidth of approximately 5-50 kHz. Between the pulses, interference generated by components on the embedded board can be clearly observed. For example, the horizontal line around bin 140 (approximately 23 kHz) corresponds to interference from the board's DC-DC power supply**.**

The lower part of the figure illustrates how spectrogram patches containing simulated PD pulses can be extracted using peaks in the band-pass RMS signal, which enables efficient identification and extraction of relevant training samples for the AI classifier.

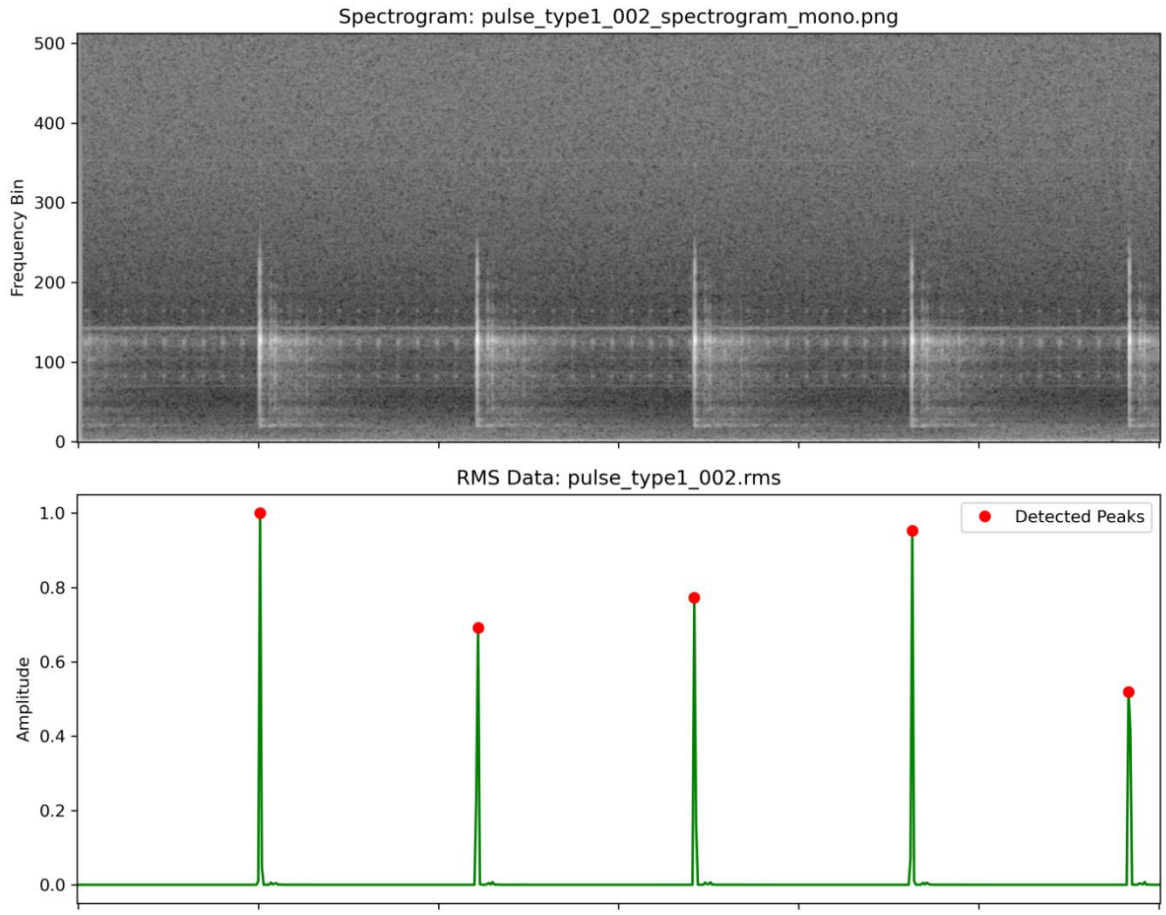

Figure 7: Sample spectrogram (top) of recorded class1 pulses. Bottom plot shows positions of training pulses found by band-pass RMS in laboratory environment.

Command Line Interface

`bf_preproc2.elf [options]`	
Parameter	**Description**
-a DIR	Folder with configuration data from the *xcalib2 configure* step
-c NUMBER	Limit number of processed channels
-d FILENAME	RAW array data file
-g FILENAME	Configuration file
-o DIR	Output directory
-p NUM	Display live preview of NUM spectrogram rows
-v LEVEL	Verbosity level, 0 – none, higher value – more verbose output

-l FREQ1	Low critical frequency of band-pass filter for RMS
-m FREQ2	High critical frequency of band-pass filter for RMS
-n FREQ	High-pass filter critical frequency for RMS
-x HOP	Enable DPU classification in profiling mode
-h	Help

As can be seen from the available options, the tool allows the computation of both a band-pass filter and a high-pass filter. The outputs of these filters can be used to compute short term RMS usable in lab environment for pulse position detection or computation of total power without DC.

Output files are stored in folder DIR:

File	Description
*.bprms	Bandpass Short-term RMS (FREQ1 to FREQ2)
*.hprms	High-pass Short-term RMS (> FREQ)
*_spectrogram_mono.png	Spectrogram – number of rows = PCM length/hopsize, number of cols = 513, nfft = 10, FFT size = 1024, Win size 800, Win type = 'Hann', zero padding, single-sided, log10 scaling
*_complex_spectrogram.bin	Complex spectrogram
metadata.txt	Parameters for dataset generation

3.4 Real-Time Detection Tool

The bf_detect2 tool implements the complete real-time inference pipeline on the embedded platform.

The application performs:

- Real-time data acquisition
- Signal preprocessing
- AI inference using the DPU
- Post-processing of classification results
- Reporting results to the cloud

The tool also transmits UDP packets containing the real-time computation state, which allows monitoring system operation and debugging potential issues.

The intended application of this tool within the Listen2Future project is the detection and classification of partial discharge events in electrical power infrastructure.

Figure 8 shows the application data flow. The processing pipeline consists of a real-time part, where the beamformer, FFT, and DNN inference blocks classify spectrogram patches of size 493 × 20 grayscale pixels at a rate exceeding 600 FPS, enabling real-time processing of the

incoming microphone array data. It also includes a software processing part, which performs more detailed analysis using band-pass IIR filter, better beamformer algorithm for acoustic camera projection, and post-processing blocks for precise evaluation and filtering of detected events.

These two parts are separated by the Detection Queue component. In the detailed analysis, the application evaluates the incoming direction of detected event, verifies the spatial isolation of the detected direction blob, and checks whether the signal power exceeds a predefined threshold.

If a detection falls within predefined regions of interest, it is included in the statistical evaluation and reported to the cloud server via a LoRa message.

The entire process can be monitored in real time using the UDP Viewer application. The tool periodically transmits UDP packets through the board's Ethernet interface while running, providing detailed insight into the internal state of the processing pipeline.

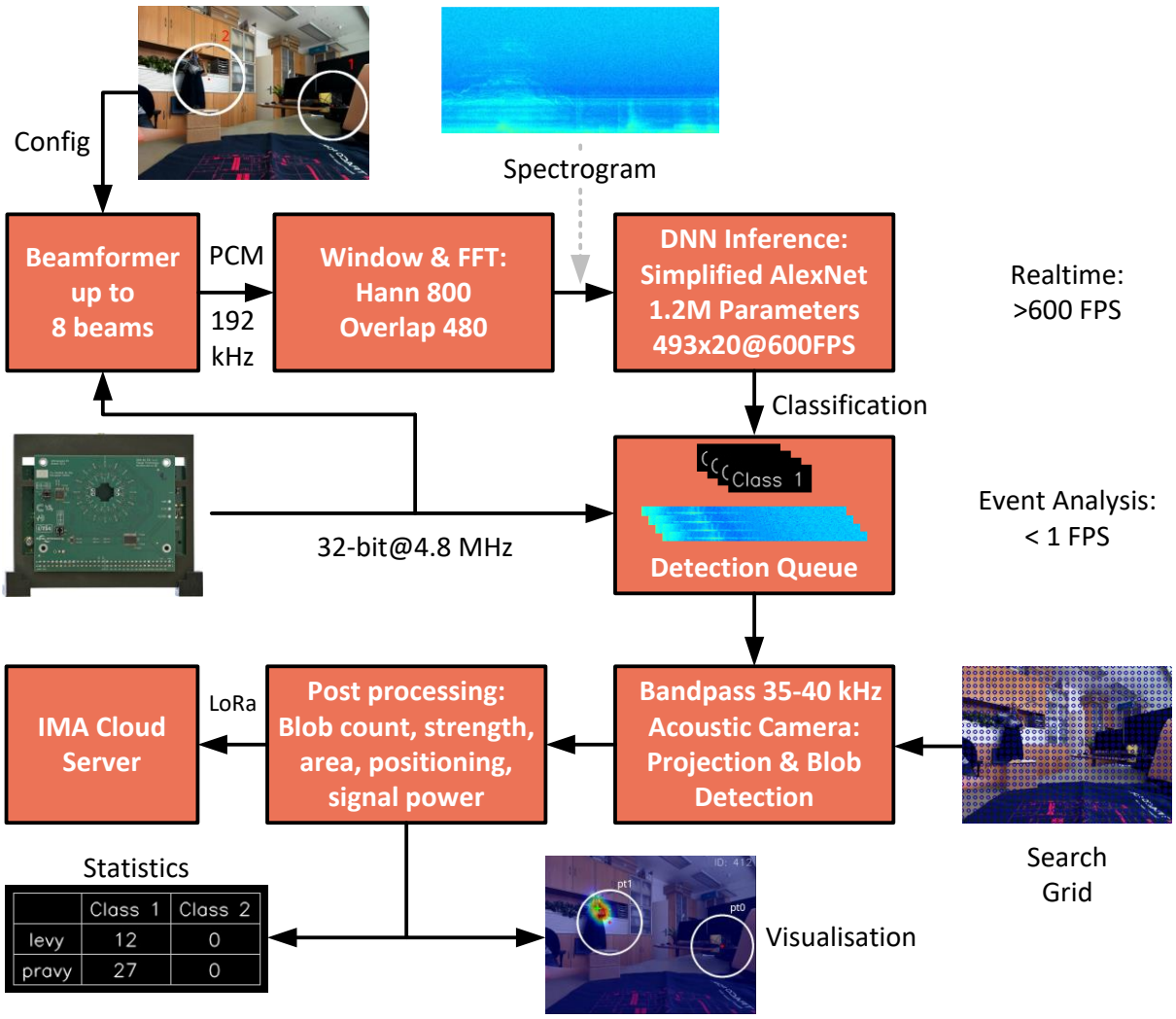

Figure 8: Computation flow of the bf_detect2 application is shown as red blocks.

Command Line Interface

`bf_detect2.elf [options]`	
Parameter	**Description**
-a DIR	Folder with configuration data from *xcalib2 configure*
-c NUMBER	Limit number of processed channels
-d FILENAME	RAW array data file
-g FILENAME	Configuration file
-p NUM	Collect profiling information for NUM iterations
-v LEVEL	Verbosity level, 0 – none, higher value – more verbose output
-m NUMBER	Special mode: store NUMBER of images classified as class 1 or 2. User must ensure that neither class 1 or class 2 events happen. This way false-positive detections of the model are collected for fine-tuning of the AI model.
-s NUMBER	Scaling factor of the Hann window
-l	AI model folder
-h	Help

The *bf_detect2* application can collect profiling information during runtime. The execution time of each operation is reported in microseconds.

The Preproc and DPU Inference stages are two parallel hardware-accelerated operations that perform AI model preprocessing and inference. The *DPU SW Preproc* stage is executed on the CPU, where input tensors are converted into a format accepted by the DPU accelerator. The final stage, *DPU SW Postproc*, processes the DPU inference results and handles output communication.

Profiling values are stored for each iteration of the detector loop. Figure 9 shows the profile of 10,000 consecutive detector-loop iterations, corresponding to 16.6 seconds of detector runtime, since one iteration processes 1.6 ms of input data.

The results show that, on average, all iterations are well below the real-time limit of 1.6 ms, with a mean execution time of approximately 1 ms. However, due to preemptive multitasking, some operations may occasionally take significantly longer than average. To handle these cases, hardware and software buffers are used to store incoming data, allowing the detector to catch up after such preemption events.

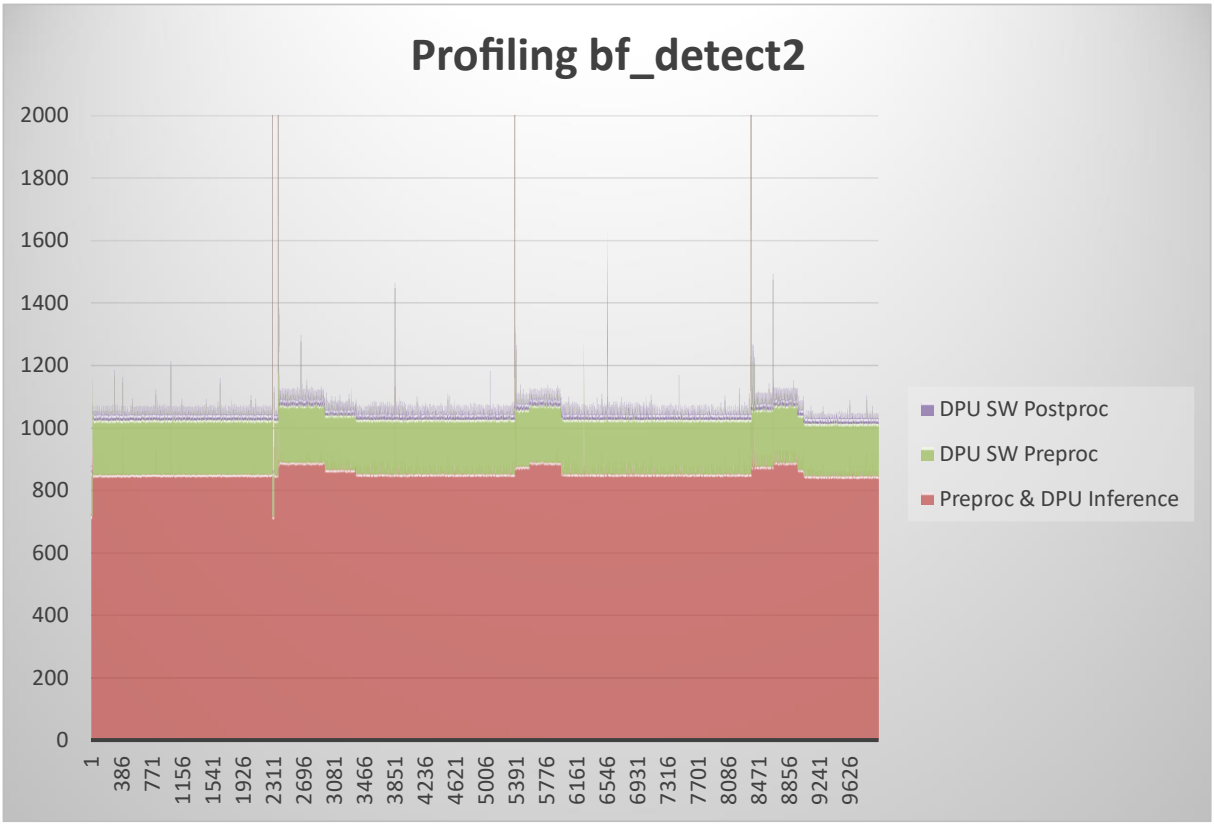

Figure 9: Execution profile of the bf_detect2 application.

3.5 LoRa Bridge

To communicate with the LoRa module over a serial interface using the defined protocol, a dedicated application, *bridge.elf*, was developed. Its structure and communication protocol are illustrated in Figure 10.

The application accepts messages either from within the operating system via UNIX socket, or from external sources via UART. It then queues and packs the messages before forwarding them over another UART interface to the LoRa module.

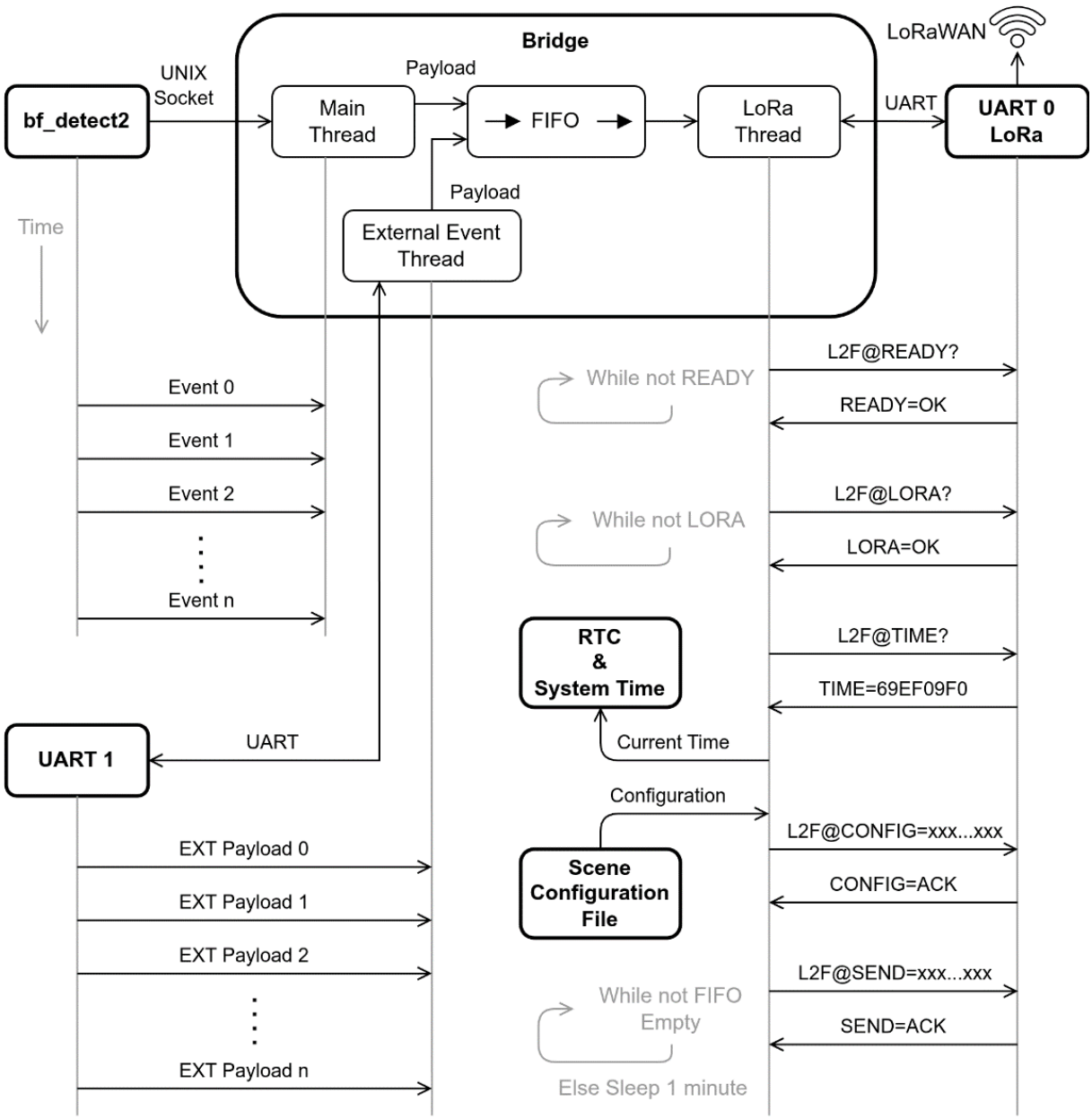

Figure 10: Bridge Application Communication Protocol

The application behavior can be modified in configuration file:

```
# socket
socket=/tmp/lora_sock

# enable/disable RTC config - true/false
use_rtc=true

#define timezone, if empty
time_zone="UTC-2"

# enable/disable uart_1 to uart_0 forwarding - true/false
uart_1_forward=false
```

```
# Number of attempts of LORA commands
rdy_cnt=5
lora_cnt=1000
send_cnt=10

# LoRa thread sleep interval in seconds
lora_sleep=60

# Camera Config
cam_cfg=assembly_xcalib2_LKO/mu_config.yml

# uart timeout in milliseconds
uart_timeout=60000

# enable/disable all debug prints
debug_en=true
```

Command Line Interface

`./bridge.elf [-c CONFIG_FILE | -h]`	
Parameter	**Description**
-c CONFIG_FILE	Override default configuration file, optional
-h	Show help

3.6 PC Applications

3.6.1 UDP Viewer application

This application runss on PC connected to the UTIA embedded platform and is intended to visualize internal operation of the bf_detect2 PD detection application.

Figure 11 shows a snapshot of the UDP Viewer GUI. On the left top side of the screen, a rolling spectrogram covering the frequency range from 0 Hz to 96 kHz is displayed. Below the spectrogram, the short-term RMS of the full spectrum is shown in red, while the RMS computed in the 35-40 kHz band is displayed in green.

Between the RMS and spectrogram plots, detections produced by the AI classifier are visualized. The cumulative counts of detected classes are displayed below as horizontal green bars**.**

The spectrogram, RMS plots, and intermediate classification results are updated in real time and can sometimes be difficult to follow. For this reason, the following data are displayed only when a detection occurs and remain visible until the next positive detection.

In the bottom-left corner, the spectrogram patch of the most recently detected event is shown. For better visibility, the patch is transposed and magnified by a factor of two, allowing easier inspection of the pulse spectrogram.

The right side of the screen displays a static image of the monitored scene (which remains unchanged during operation) with an acoustic camera overlay derived from the detailed analysis output inside the bf_detect2 application. An additional overlay indicates predefined

monitoring directions, visualized as red crosses with blue circles representing the tolerance radius. The name of each monitored location is displayed next to the corresponding circle.

A white cross indicates the location of the maximum of the detected acoustic blob.

Below this visualization, a statistical table summarizing the events that falls into pre-defined areas and are reported to the cloud via LoRa communication.

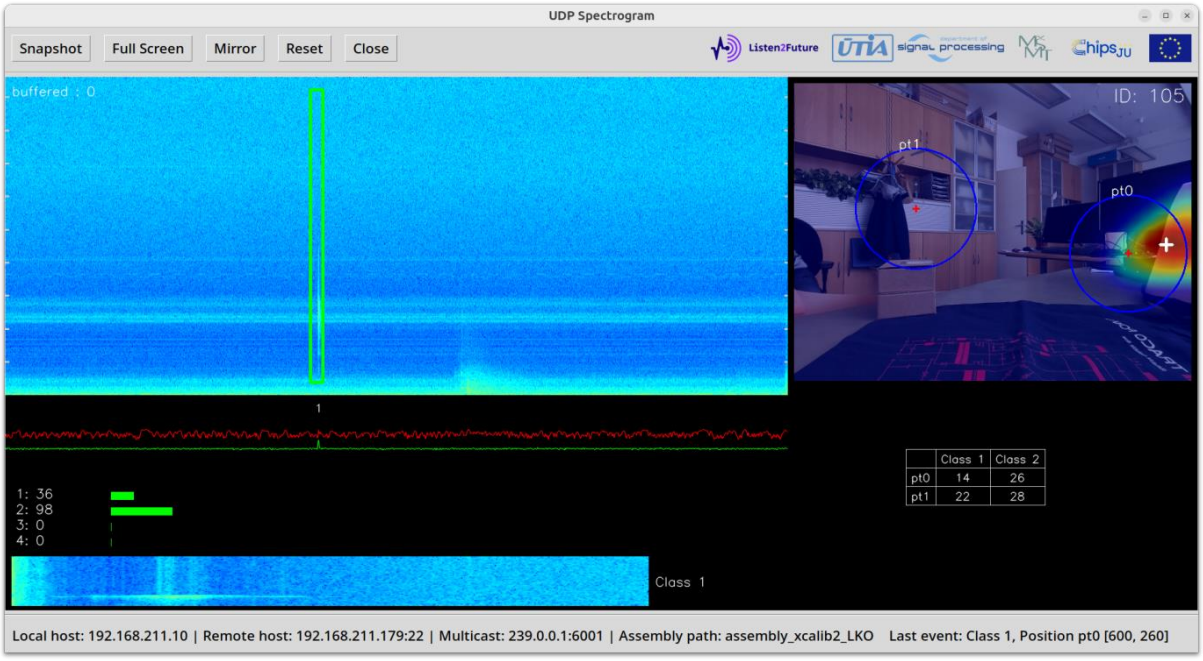

Figure 11: Example UDP Viewer snapshot showing actual detection of class 1 simulated discharge pulse positively identified as coming from nearby monitored point 'pt0' which is back annotated in the spectrogram as green rectangle.

Before running the application, udp_viewer2_tk.cfg must be properly set:

```
# IP of the computer where the VIEWER is running
LOCAL_HOST_IP = 192.168.211.235

# IP of the Zynq device
REMOTE_HOST_IP = 192.168.211.179

# Multicast configuration
MCAST_GRP = 239.0.0.1
MCAST_PORT = 6001

# Name of the remote folder containing MIC array and camera calibration
# setup
ASSEMBLY = assembly_xcalib2_LKO

# mirror camera wiew
CAM_MIRROR = True

# Disable all runtime prints to console except initialization
# It has strong impact on performance
NO_PRINT = False
```

Run the tool by the command:

```
python3 udp_viewer2_tk.py
```

3.6.2 UDP Player and Recorder

In addition to the UDP viewer, a tool for array data playback and recording has been implemented. The tool receives PCM data broadcast by the board over UDP immediately after the beamforming and decimation stages. The data are sampled at 192 kHz and converted into a suitable format for playback on a PC. The tool can also record the received data to a WAV file.

The bf_detect2 application is expected to be running on the UTIA execution platform with the beamformer directions properly configured.

Run the tool by the command:

```
python3 udp_player2_qt.py
```

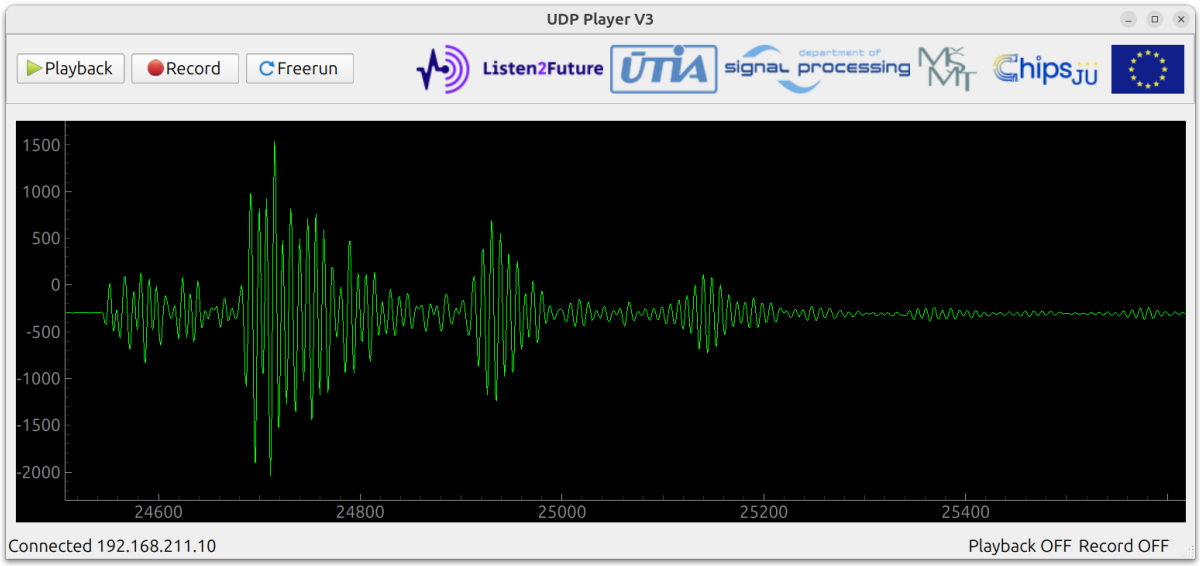

Figure 12: UDP Player and Recorder

3.6.3 LoRa Simulator

LoRa simulator serves as a replacement for a physical LoRa Node device. This allows the programmer to develop an application that communicates with simulated LoRa Node without performing real LoRa network data exchange. The LoRa Node device is provided by IMA under the Listen2Future project. Like a real LoRa Node, the LoRa simulator uses an UART interface on which it implements the communication protocol identical to LoRa Node. The LoRa Node device is connected to the Zynq UltraScale+ device via UART interface on UTIA execution platform. The LoRa simulator communicates with the same UART interface on UTIA execution platform as LoRa Node does by USB-to-UART cable. In other words, the user develops the application using the LoRa simulator and then simply replaces it with a physical LoRa Node device without any modifications of code.

Command Line Interface

`./lora_sim -d <UART_DEVICE> [-t] [-c] [-i <IMG_FILE>] [-m] [-h]`	
Parameter	**Description**
-d <UART_DEVICE>	/dev/ttyUL1, /dev/ttyUL2, /dev/ttyUSB0, ...
-t	TEST_MODE_DETAILED, default=TEST_MODE_NORMAL
-c	Save events to CSV file events.csv. If it exists, backup is created
-I <IMG_FILE>	Show IMG_FILE (JPG) - draw event locations
-m	Mirror IMG_FILE (JPG)
-h	Print this help

Output

```
./lora_sim -d /dev/ttyUSB2 -i mu_view.jpg -m
LoRaSIM: Test mode: NORMAL
LoRaSIM: Open mu_view.jpg
LoRaSIM: OK
LoRaSIM: Connected to -d 115200 8N1
LoRaSIM: Receive messages up to length of 384
LoRaSIM: A message ends with <CR><LF>
LoRaSIM: IN: L2F@READY?
LoRaSIM: OUT: READY=OK
LoRaSIM: IN: L2F@LORA?
LoRaSIM: OUT: LORA=OK
LoRaSIM: IN: L2F@TIME?
LoRaSIM: OUT: TIME=69EF09F0
LoRaSIM: IN: L2F@CONFIG=0200220022010022005d00200020020096004400210021
LoRaSIM: CONFIG:
LoRaSIM:   N_LOC               = 2
LoRaSIM:   SCALE_X_10UM_PER_PX = 34
LoRaSIM:   SCALE_Y_10UM_PER_PX = 34
LoRaSIM:   N_LOC[2] =
LoRaSIM:     #1: { [34,93], [32,32] }
LoRaSIM:     #2: { [150,68], [33,33] }
LoRaSIM: OUT: CONFIG=ACK
LoRaSIM: IN:
L2F@SEND=69ef0a0102e1008e00360000fb23312169ef0a050141008e005e00001823311169ef0a0903
520006006c00000323311169ef0a18034e008e005100001f23312169ef0a1d034f008e004a000021233
11169ef0a22035f008e004a00001f23312169ef0a270350008e004a000021233111
LoRaSIM: EVENT LIST: 7 events
LoRaSIM:   #0
LoRaSIM:     UTC:          2026-04-27 07:02:25.737
LoRaSIM:     LOC:          [142, 54, 0]
LoRaSIM:     PEAK:         251 dB
LoRaSIM:     F_BAND:       35 kHz
LoRaSIM:     BURST:        3 - Moderate (10 - 100 us)
LoRaSIM:     SYSTEM SRC:   1 - Central unit
LoRaSIM:     PD_TYPE:      2 - Surface or tracking PD
LoRaSIM:     PD_SEVERITY:  1 - Low
LoRaSIM:   #1
```

```
LoRaSIM:      UTC:          2026-04-27 07:02:29.321
LoRaSIM:      LOC:          [142, 94, 0]
LoRaSIM:      PEAK:         24 dB
LoRaSIM:      F_BAND:       35 kHz
LoRaSIM:      BURST:        3 - Moderate (10 - 100 us)
LoRaSIM:      SYSTEM SRC:   1 - Central unit
LoRaSIM:      PD_TYPE:      1 - Internal PD
LoRaSIM:      PD_SEVERITY:  1 - Low
LoRaSIM:    #2
LoRaSIM:      UTC:          2026-04-27 07:02:33.850
LoRaSIM:      LOC:          [6, 108, 0]
LoRaSIM:      PEAK:         3 dB
LoRaSIM:      F_BAND:       35 kHz
LoRaSIM:      BURST:        3 - Moderate (10 - 100 us)
LoRaSIM:      SYSTEM SRC:   1 - Central unit
LoRaSIM:      PD_TYPE:      1 - Internal PD
LoRaSIM:      PD_SEVERITY:  1 - Low
LoRaSIM:    #3
LoRaSIM:      UTC:          2026-04-27 07:02:48.846
LoRaSIM:      LOC:          [142, 81, 0]
LoRaSIM:      PEAK:         31 dB
LoRaSIM:      F_BAND:       35 kHz
LoRaSIM:      BURST:        3 - Moderate (10 - 100 us)
LoRaSIM:      SYSTEM SRC:   1 - Central unit
LoRaSIM:      PD_TYPE:      2 - Surface or tracking PD
LoRaSIM:      PD_SEVERITY:  1 - Low
LoRaSIM:    #4
LoRaSIM:      UTC:          2026-04-27 07:02:53.847
LoRaSIM:      LOC:          [142, 74, 0]
LoRaSIM:      PEAK:         33 dB
LoRaSIM:      F_BAND:       35 kHz
LoRaSIM:      BURST:        3 - Moderate (10 - 100 us)
LoRaSIM:      SYSTEM SRC:   1 - Central unit
LoRaSIM:      PD_TYPE:      1 - Internal PD
LoRaSIM:      PD_SEVERITY:  1 - Low
LoRaSIM:    #5
LoRaSIM:      UTC:          2026-04-27 07:02:58.863
LoRaSIM:      LOC:          [142, 74, 0]
LoRaSIM:      PEAK:         31 dB
LoRaSIM:      F_BAND:       35 kHz
LoRaSIM:      BURST:        3 - Moderate (10 - 100 us)
LoRaSIM:      SYSTEM SRC:   1 - Central unit
LoRaSIM:      PD_TYPE:      2 - Surface or tracking PD
LoRaSIM:      PD_SEVERITY:  1 - Low
LoRaSIM:    #6
LoRaSIM:      UTC:          2026-04-27 07:03:03.848
LoRaSIM:      LOC:          [142, 74, 0]
LoRaSIM:      PEAK:         33 dB
LoRaSIM:      F_BAND:       35 kHz
LoRaSIM:      BURST:        3 - Moderate (10 - 100 us)
LoRaSIM:      SYSTEM SRC:   1 - Central unit
LoRaSIM:      PD_TYPE:      1 - Internal PD
LoRaSIM:      PD_SEVERITY:  1 - Low
LoRaSIM: OUT: SEND=ACK
```

Once LoRa Simulator is running on PC, it takes background image of current detector configuration and shows detection areas and individual detections within defined detection tolerance, see Figure 13.

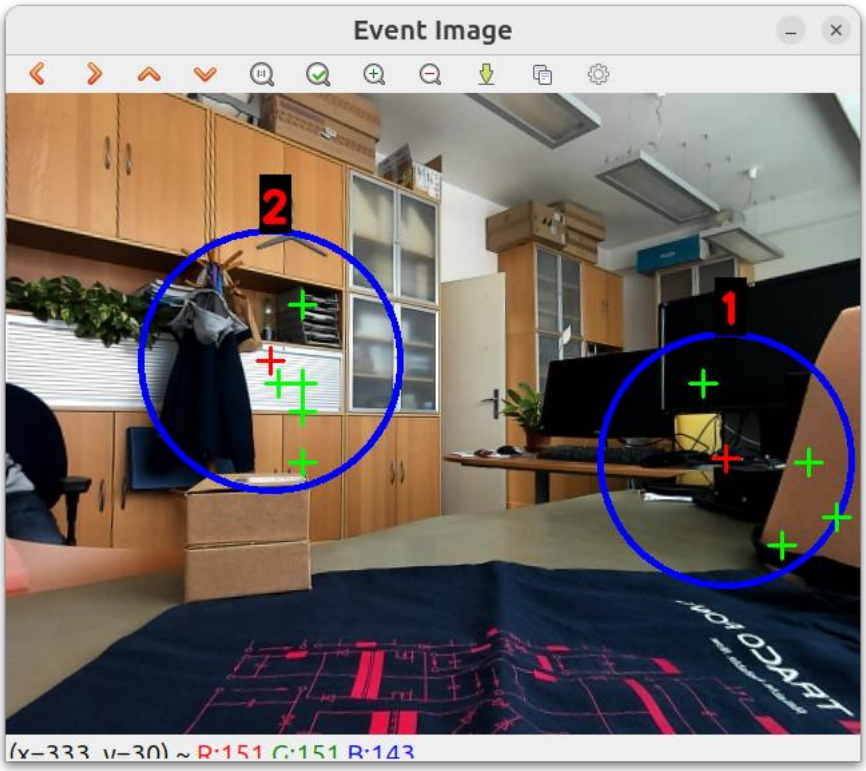

Figure 13: LoRa Simulator Output

4 How to Setup the Board

Before running the tools on the board, make sure the board is booted and that all auxiliary files are in the correct locations. To prepare a bootable microSD card, follow these steps:

1. Use a microSD card larger than 4 GB.
2. Locate the SD card image file at data/firmware/sd_card.img
3. Use an SD card image writer to write the .img file to the microSD card.
4. Insert the microSD card into the board's card slot.
5. Power on the board.
6. Log in as user: root password: root.
7. After the board has booted, run the required filesystem setup commands:

```
# resize ext4 partition to stretch all remaining SD card capacity
cd ~

# resize partition
resize-part /dev/mmcblk1p2
```

```
# go to fat32 partition with files
cd /run/media/mmcblk1p1

# copy files to root folder
cp * /home/root

# unzip folder contents
unzip folder.zip

# go to home
cd ~
```

5 License

The package is provided by UTIA AV CR as is, free of charge.

6 Content of the package

data	
├── firmware	SD card image and models folder
│ ├── models	Models folder is not integrated into image, must be copied later to /home/root on running system
│ │ ├── model_AlexNetBN_PD_BUT_UTIAnoise_ft	
│ │ │ ├── md5sum.txt	
│ │ │ ├── meta.json	
│ │ │ ├── pd_det.xmodel	
│ │ │ └── preproc_params.txt	
│ │ ├── model_AlexNetBN_PD_UTIA_ft	
│ │ │ ├── md5sum.txt	
│ │ │ ├── meta.json	
│ │ │ ├── pd_det.xmodel	
│ │ │ └── preproc_params.txt	
│ │ └── readme.txt	
│ └── sd_card.img	SD card image to be written into SD card
├── LoRa_SIM	Lora simulator tool for Ubuntu PC
│ ├── lora_sim	
│ └── mu_view.jpg	
├── UDP_player2	UDP Player2 tool for PC with or without qt GUI (python application)
│ ├── logo-chipsju-full.png	
│ ├── logo-eu.png	
│ ├── logo-l2f.png	
│ ├── logo-msmt.png	
│ ├── logo-utia.png	
│ ├── udp_player2.py	
│ ├── udp_player2_qt.py	
│ └── udp_player2_qt.ui	
└── UDP_viewer2	UDP Viewer2 tool for PC with tkinter GUI (python application)
├── logo-chipsju-full.png	
├── logo-eu.png	
├── logo-l2f.png	
├── logo-msmt.png	
├── logo-utia.png	
├── shortcuts.csv	
├── shortcuts.csv.win	
├── shortcuts.py	
├── snapshots	Folder for snapshots
├── udp_viewer2_tk.cfg	
├── udp_viewer2_tk.py	
├── udp_viewer_icon_64.ico	
└── udp_viewer_icon_64.png	

7 References

[1] Listen2Future project web pages: https://www.listen2future.eu/home

8 Annex A

Example command sequence required to setup the detection on the embedded platform after new firmware is installed without any pre-defined configuration. Boot the board, copy models to /home/root and execute on embedded platform connected by ssh terminal from Linux PC with X11 forwarding enabled:

```
#change to /home/root

cd ~

#test capture (1 second to test_data.raw)
./capture.elf -o test_data -s 1

# capture calibration data after webcam is attached
./xcalib2.elf capture -w 0 -i seecam_640_480_camera_data.xml -p 12 -o
my_config

# compute calibration
./xcalib2.elf calibrate -i my_config -m utia_ev_board_v2_x.csv -g grid.cfg

# verify calibration by showing live acoustic camera overlay
./xcalib2.elf verify -i my_config

# configure directions to be monitored/tracked
./xcalib2.elf configure -i my_config

# if needed, raw data can be pre-processed to form AI training dataset
./bf_preproc2.elf -a my_config -d test_data -o preproc_data -g preproc.cfg
-l 17000 -m 27000

# optionally run LoRa bridge
./bridge.elf &

# run detector, to run detector a trained AI model must be provided in
'model' folder or
./bf_detect2.elf -a my_config -g default.cfg -s 1.0
```