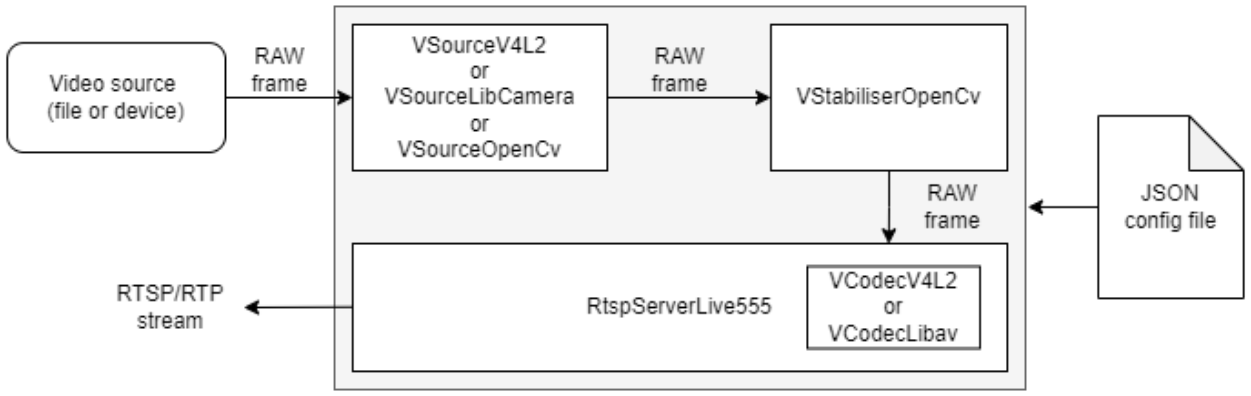
RpiStreamer application template

v3.0.0

Table of contents

Overview

RpiStreamer is a application template which users can use as basement of their applications. This application implements basic video processing pipeline: video capture > video stabilization > RTSP server for Raspberry PI4 and PI5. The application combines libraries: **VSourceV4L2** (video capture from V4L2 compatible devices), **VSourceLibCamera** (video capture from Libcamera API compatible devices), **VSourceOpenCv** (video capture based on OpenCV, supports all video sources which implemented in OpenCV), **VCodecV4L2** (H264 and JPEG video encoder for Raspberry PI4 based on V4L2 API), **VCodecLibav** (H264, H265 and JPEG encoders for RPI5 since it does not have any hardware codec), **VStabiliserOpenCv** (video stabilization library) and RtspServerLive555 (RTSP server library based on LIVE555 Streaming Media API). The applications shows how to use combination of the libraries listed above. It is built for **Raspberry PI4 (Debian Bulleye 11 x64) and PI5 (Debian Bookworm 12 x64)** but can be used on any other Linux platform by **providing VCodecXXX** library with platform compatible VCodec implementation. It supports different video sources and provides real-time video processing. Structure of video processing pipeline:



After start application reads JSON config file which includes video capture parameters, video stabilizer parameters and RTSP server parameters. If there is no config file the application will create new one with default parameters. When the config file is read the application initializes the video source. Depends on configuration params the application creates **VSourceV4L2** class object (to capture video from V4L2 compatible cameras) or **VSourceLibCamera** class object (to capture video from Libcamera API compatible cameras) or **VSourceOpenCV** (to capture video from video files or from RTSP streams). After the application creates and initializes **VStabiliserOpenCv** class object (video stabilization) by parameters from JSON config file. Then the application creates board compatible codec and then initializes **RtspServerLive555** class object (RTSP server) by parameters from JSON config file and previously created codec. When all modules are initialized the application executes the video processing pipeline: capture video frame from video source - video stabilization - send stabilized video frame to RTSP server. User can connect to RTSP server by any client (ffmpeg, gstreamer, VLC, Milestone etc.). Additionally the application creates folder "Log" to write log information.

The application chooses the right VCodec implementation on its own, based on the board it's compiled on. If it's on a Raspberry Pi 4, it picks **VCodecV4L2** because this board supports hardware codecs. However, for Raspberry Pi 5, it goes for **VCodecLibav** since this board doesn't have hardware codecs and needs a software-based solution.

Versions

Table 1 - Application versions.

Version	Release date	What's new
1.0.0	11.09.2023	First version.
1.1.0	19.09.2023	- VSourceLibCamera and VSourceOpenCv libraries added to support different video sources.
2.0.0	22.11.2023	- RTSP server implementation is changed. - External RTSP server app is excluded. RTSP server built in application.
2.1.0	15.01.2024	- RTSP server updated. - Code optimized.

Version	Release date	What's new
3.0.0	24.01.2024	- Raspberry PI5 support added. - RtspServerLive555 updated. - VSourceLibCamera updated.

Source code files

The application is supplied by source code only. The user is given a set of files in the form of a CMake project (repository). The repository structure is shown below:

```
CMakeLists.txt ---------------- Main CMake file.
3rdparty ---------------------- Folder with third-party libraries.
    CMakeLists.txt ------------ CMake file to include third-party libraries.
    RtspServerLive555 --------- Source code of RtspServerLive555 library.
    VCodecV4L2 ---------------- Source code of VCodecV4L2 library.
    VCodecLibav --------------- Source code of VCodecLibav library.
    VSourceLibCamera ---------- Source code of VSourceLibCamera library.
    VSourceOpenCv ------------- Source code of VSourceOpenCv library.
    VSourceV4l2 --------------- Source code of VSourceV4l2 library.
    VStabiliserOpenCv --------- Source code of VStabiliserOpenCv library.
src --------------------------- Folder with application source code.
    CMakeLists.txt ------------ CMake file.
    RpiStreamerVersion.h ------ Header file with application version.
    RpiStreamerVersion.h.in --- File for CMake to generate version header.
    main.cpp ------------------ Application source code file.
```

Config file

RpiStreamer application reads config file **RpiStreamer.json** in the same folded with application executable file. Config file content:

```
{
    "Params":
    {
        "videoSource":
        {
            "initString": "0;1280;720;30;YUYV"
        },
        "videoStabiliser":
        {
            "aFilterCoeff": 0.8999999761581421,
            "aOffsetLimit": 10.0,
            "constAOffset": 0.0,
            "constXOffset": 0,
            "constYOffset": 0,
            "cutFrequencyHz": 2.0,
            "enable": true,
```

```
            "fps": 30.0,
            "logMod": 0,
            "scaleFactor": 1,
            "transparentBorder": true,
            "type": 2,
            "xFilterCoeff": 0.8999999761581421,
            "xOffsetLimit": 150,
            "yFilterCoeff": 0.8999999761581421,
            "yOffsetLimit": 150
        },
        "videoStream":
        {
            "bitrateKbps": 3000,
            "bitrateMode": 0,
            "codec": "H264",
            "enable": true,
            "fitMode": 0,
            "fps": 30.0,
            "gop": 30,
            "h264Profile": 0,
            "height": 720,
            "ip": "192.168.0.2",
            "jpegQuality": 80,
            "maxBitrateKbps": 5000,
            "minBitrateKbps": 1000,
            "overlayMode": true,
            "password": "",
            "port": 8554,
            "suffix": "unicast",
            "user": "",
            "width": 1280
        }
    }
}
```


Table 2 - Config file parameters description.

Parameter	type	Description
Video source parameters:		

Parameter	type	Description
initString	string	Video source initialization string. Based on initialization string the application creates **VSourceV4L2** class object (to capture video from V4L2 compatible cameras) or **VSourceLibCamera** class object (to capture video from Libcamera API compatible cameras) or **VSourceOpenCV** (to capture video from video files or from RTSP streams). So, user can defilene backend to capture video: Initialization string for **V4l2 or Libcamera compatible video devices** (cameras). Valid formats: **[full device name];[width];[height];[fps];[fourcc]** or **[full device name];[width];[height];[fps]** or **[full device name];[width];[height]** or **[full device name]** Initialization string parameters: **[full device name]** - Full name of video device file (for example **"/dev/video0"**) for V4L2 or device number for Libcamera (for example **"0"**). **[width]** - Video frame width. Video frame height must be set as well. The library will try set this value in video capture hardware parameters. If set to **0** the library will detect frame width automatically according to existing video device parameters. **[height]** - Video frame height. Video frame width must be set as well. The library will try set this value in video capture hardware parameters. If set to **0** the library will detect frame width automatically according to existing video device parameters. **[fps]** - FPS. The library will try set this value in video capture hardware parameters. If set to **0** the library will detect frame width automatically according to existing video device parameters. **[fourcc]** - Pixel format. Valid values: BGR24, RGB24, GRAY, YUV24, YUYV, UYVY, NV12, NV21, YV12, YU12. If fourcc not set the library will chose appropriate format according to existing video device parameters. Initialization string for **OpenCV video sources** can be video file name (for example **"test.mp4"**) or RTSP stream (for example **"rtsp://192.168.0.1:7100/live"**).
Video stabilizer parameters:		
scaleFactor	int	Scale factor. Value: If 1 the library will process original frame size, if 2 the library will scale original frame size by 2, if 3 - by 3. VStabiliserOpenCv class supports 2D FFT algorithm which scales input image to wise 512x512 or 256x256 depends on scale factor. To chose particular size (512x512 or 256x256) the library devide input frame height by scale factor. If result >= 512 the library will use 512x512 size for internal algorithms. Otherwise the library will use 256x256 size.

Parameter	type	Description
xOffsetLimit	int	Maximum horizontal image shift in pixels per video frame. If image shift bigger than this limit the library will compensate only xOffsetLimit shift.
yOffsetLimit	int	Maximum vertical image shift in pixels per video frame. If image shift bigger than this limit the library will compensate only yOffsetLimit shift.
aOffsetLimit	float	Maximum rotational image angle in radians per video frame. If image absolute rotational angle bigger than this limit the library will compensate only aOffsetLimit angle.
xFilterCoeff	float	Horizontal smoothing coefficient of constant camera movement. Values [0-1]: 0 - the library will not compensate for constant camera motion, video will not be stabilized, 1 - no smoothing of constant camera motion (the library will compensate for the current picture drift completely without considering constant motion). **If set 0 the library will detect necessary coefficients automatically.**
yFilterCoeff	float	Vertical smoothing coefficient of constant camera movement. Values [0-1]: 0 - the library will not compensate for constant camera motion, video will not be stabilized, 1 - no smoothing of constant camera motion (the library will compensate for the current picture drift completely without considering constant motion). **If set 0 the library will detect necessary coefficients automatically.**
aFilterCoeff	float	Rotational smoothing coefficient of constant camera movement. Values [0-1]: 0 - the library will not compensate for constant camera motion, video will not be stabilized, 1 - no smoothing of constant camera motion (the library will compensate for the current picture drift completely without considering constant motion). **If set 0 the library will detect necessary coefficients automatically.**
enable	bool	Enable/disable stabilization.
transparentBorder	bool	Enable/disable transparent borders.
constXOffset	int	Constant horizontal image offset in pixels. The library will add this offset to each processed video frame. This values used for boresight correction.
constYOffset	int	Constant vertical image offset in pixels. The library will add this offset to each processed video frame. This values used for boresight correction.
constAOffset	float	Constant rotational angle in radians. The library will add this offset to each processed video frame. This values used for boresight correction.

Parameter	type	Description
type	int	Three types of stabilisation algorithm are implemented in the library: **0 - 2D based on FFT**. Fastest algorithm but only for 2D stabilisation. Works stable for low light conditions and for low contrast images. **1 - 2D based on optical flow**. Gives good accuracy but lower speed as 2D FFT and requires contrast objects on video. **2 - 3D based on optical flow**. Gives best accuracy but lower speed as 2D FFT and requires contrast objects on video.
cutFrequencyHz	float	**Not supported** by VStabiliserOpenCv.
fps	float	**Not supported** by VStabiliserOpenCv.
RTSP server parameters:		
enable	string	RTSP server mode: false - Off, true - On.
width	int	Video stream width from 8 to 8192. If the resolution of the source video frame is different from that specified in the parameters for the RTSP server, the source video will be scaled.
height	int	Video stream height from 8 to 8192. If the resolution of the source video frame is different from that specified in the parameters for the RTSP server, the source video will be scaled.
ip	string	RTSP server IP. It is recommended to use **"0.0.0.0"** independent from board IP.
port	int	RTSP server port. RTSP initialization string to receive video will be: **"rtsp//IP:port/suffix"**.
user	string	RTSP server user: "" - no user.
password	string	RTSP server password: "" - no password.
suffix	string	RTSP stream suffix (stream name). For example **"live"** or **"unicast"**. RTSP initialization string to receive video will be: **"rtsp//IP:port/suffix"**.
minBitrateKbps	int	**Not supported** by RtspServerLive555.
maxBitrateKbps	int	**Not supported** by RtspServerLive555.
bitrateKbps	int	Dedicated bitrate, kbps. Will be set to **VCodecV4L2** codec or **VCodecLibav** codec.
bitrateMode	int	**Not supported** by RtspServerLive555.
fps	int	Video stream FPS. Regardless of the FPS of the camera, the video stream will be in accordance with the specified FPS.
gop	int	GOP size for H264 codecs. Will be set to **VCodecV4L2** codec or **VCodecLibav** codec.

Parameter	type	Description
h264Profile	int	H264 profile: 0 - baseline, 1 - main, 2 - high. Will be set to **VCodecV4L2** codec or **VCodecLibav** codec.
jpegQuality	int	JPEG quality from 1 to 100% for JPEG codec. Will be set to **VCodecV4L2** codec.
codec	string	Encoding format. PI4 supports : JPEG and H264. PI5 supports : H264 and HEVC.
fitMode	int	Scaling mode: 0 - fit, 1 - cut. Defines how to scale video to RTSP stream resolution.
overlayMode	bool	Overlay enable/disable flag. true - enable, false - disable.

Run application

Copy application (RpiStreamer executable and RpiStreamer.json) to any folder and run:

```
./RpiStreamer -vv
```

When the application is started for the first time, it creates a configuration file named **RpiStreamer.json** if file does not exist (refer to the Config file section). RTSP initialization string to receive video will be: **"rtsp//IP:port/suffix"**. If the application is run as a superuser using sudo, the file will be owned by the root user. Therefore, to modify the configuration file, superuser privileges will be necessary. You can run application manually or you can add application to auto start as **systemd** servive. To add application as **systemd** service create service file:

```
sudo nano /etc/systemd/system/rpistreamer.service
```

and add content:

```
[Unit]
Description=RpiStreamer

Wants=network.target
After=syslog.target network-online.target

[Service]
Type=simple
ExecStart=/home/pi/App/RpiStreamer
Restart=on-failure
RestartSec=10
KillMode=control-group
KillSignal=SIGTERM
[Install]
WantedBy=multi-user.target
```

3.6. Save **ctrl + s** and close **ctrl + x**.

3.7. Reload **systemd** services:

```
sudo systemctl daemon-reload
```

3.8. Command to control service:

Start service:

```
sudo systemctl start rpistreamer
```

Enable for auto start:

```
sudo systemctl enable rpistreamer
```

Stop service:

```
sudo systemctl stop rpistreamer
```

Check status:

```
sudo systemctl status rpistreamer
```

Build application

Before build user should configure raspberry (Raspberry PI configuration). Typical build commands:

```
cd RpiStreamer
mkdir build
cd build
cmake -DCMAKE_BUILD_TYPE=Release ..
make
```

Raspberry PI configuration

OS install

1. Download **Raspberry PI imager**.
2. Choose Operating System: Raspberry Pi OS (other) -> Raspberry Pi OS Lite (64-bit) Debian Bulleye 11 for Raspberry PI4 or Raspberry Pi OS Lite (64-bit) Debian Bookworm 12 for Raspberry PI5.
3. Choose storage (MicroSD) card. If it possible use 128 GB MicroSD.

4. Set additional options: do not set hostname “raspberrypi.local”, enable SSH (Use password authentication), set username **"pi"** and password **"pi"**, configure wireless LAN according your settings. You will need wi-fi for software installation, set appropriate time zone and wireless zone.
5. Save changes and push “Write” button. After push “Yes” to rewrite data on MicroSD. At the end remove MicroSD.
6. Insert MicroSD in Raspberry and power up Raspberry.

LAN configuration

1. Configure LAN IP address on your PC. For Windows 11 got to Settings -> Network & Internet -> Advanced Network Settings -> More network adapter options. Click right button on LAN connection used to connect to Raspberry PI and chose “Properties”. Double click on “Internet Protocol Version 4 (TCP/IPv4)”. Set static IP address 192.168.0.1 and mask 255.255.255.0.
2. Connect raspberry via LAN cable. After power up you don’t know IP address of Raspberry board but you can connect to raspberry via SSH using “raspberrypi” name. In Windows 11 open windows terminal or power shell terminal and type command **ssh pi@raspberrypi**. After connection type yes to establish authenticity. **WARNING:** If you work with Windows we recommend delete information about previous connections. Go to folder **C:/Users/[your user name]/.ssh**. Open file **known_hosts** is exists in text editor. Delete all lines **“raspberrypi”**.
3. You have to set static IP address in Raspberry PI but not **NOW**. After static IP set you will lose wi-fi connection which you need for software installation.

Install dependencies

1. Connect to raspberry via SSH: **ssh pi@raspberrypi**. **WARNING:** If you work with Windows we recommend delete information about previous connections. Go to folder **C:/Users/[your user name]/.ssh**. Open file **known_hosts** is exists in text editor. Delete all lines **“raspberrypi”**.
2. Install libraries:

```
sudo apt-get update
sudo apt install cmake build-essential libopencv-dev
```

3. If Libcamera API not installed you have to install it by commands from source code :

```
sudo apt-get install meson
git clone https://git.libcamera.org/libcamera/libcamera.git
cd libcamera
meson setup build
ninja -C build install
```

   or by commands from packet mamager:

```
sudo apt-get install libcamera-dev
```

4. Install OpenSSL by command:

```
sudo apt-get install libssl-dev
```

5. Install live555

- Run command :

```
sudo apt-get install liblivemedia-dev
```

If you get error from package manager, go to next step and install live555 from source code.

Important note : There will be a compilation error if your system has live555 from source code installation and package manager installation at the same time. In order to solve problem you should remove one of them. For example:

```
sudo apt-get remove liblivemedia-dev
```

5. Install required libraries for libav codec. If you are using RPI4 this step can be skipped.

- Run command :

```
sudo apt-get install -y ffmpeg libavcodec-dev libavutil-dev libavformat-dev
libavdevice-dev libavfilter-dev libcurl4
```

6. Reboot the system.

```
sudo reboot
```

Setup hight performance mode (overclock) if necessary

1. Open config file:

```
sudo nano /boot/config.txt
```

2. Add line at the end of file:
    - For RPI4 :

```
force_turbo=1
arm_freq=2000
arm_freq_min=2000
over_voltage=6
```

    - For RPI5 :

```
force_turbo=1
over_voltage_delta=500000
arm_freq=3000
gpu_freq=800
```

3. Save changes **ctrl+s** and close **ctrl+x**. Reboot raspberry **sudo reboot**.

V4L2 camera stack configuration

1. To enable legacy (V4L2) camera support in Raspberry PI you have to change system settings. Run configuration manager:

   ```
   sudo raspi-config
   ```

2. Select “Interface Options” -> “Legacy camera”. After select **yes**.
3. Go to **Finish** menu and close configuration manager.
4. Reboot raspberry **sudo reboot**.

Static IP configuration on Raspberry.

1. Connect to raspberry via SSH **ssh pi@raspberrypi** after reboot. **WARNING:** If you work with Windows we recommend delete information about previous connections. Go to folder **C:/Users/[your user name]/.ssh**. Open file **known_hosts** is exists in text editor. Delete all lines **“raspberrypi”**.
2. Open config file:

   ```
   sudo nano /etc/dhcpcd.conf
   ```

3. Go dawn to file and delete the comments in the following lines and set IP 192.168.0.2:

   ```
   interface eth0
   static ip_address=192.168.0.2/24
   static ip6_address=fd51:42f8:caae:d92e::ff/64
   static routers=192.168.0.2
   static domain_name_servers=192.168.0.2 8.8.8.8 fd51:42f8:caae:d92e::1
   ```

4. Save changes **ctrl + s** and exit **ctrl + x**. Reboot after **sudo reboot**.
5. After configuring static IP you can connect to to raspberry by IP.

   ```
   ssh pi@192.168.0.2
   ```

Source code

Bellow source code of main.cpp file:

```
#include <iostream>
#include <string>
#include <cstring>
#include <cstdint>
#include <cctype>
#include <string.h>
#include "RpiStreamerVersion.h"
#include <VSourceV4L2.h>
#include <VSourceLibCamera.h>
#include <VSourceOpenCv.h>
```

```
#include <VStabiliserOpenCv.h>
#include <RtspServerLive555.h>

#ifdef isRPI5
    #include <VCodecLibav.h>
#else
    #include <VCodecV4L2.h>
#endif

// Link namespaces.
using namespace std;
using namespace cr::utils;
using namespace cr::video;
using namespace cr::vstab;
using namespace cr::rtsp;

/// Application parameters.
class Params
{
public:

    /// Stream params.
    class VideoSourceParams
    {
    public:
        /// Video source init string
        string initString {"/dev/video0;640;480;30;YUYV"};

        JSON_READABLE(VideoSourceParams, initString)
    };

    /// Video stream params.
    RtspServerParams videoStream;
    /// Video source params.
    VideoSourceParams videoSource;
    /// Video stabiliser params.
    VStabiliserParams videoStabiliser;

    JSON_READABLE(Params, videoStream, videoSource, videoStabiliser)
};

/// Application params.
Params g_params;
/// Logger.
Logger g_log;
/// Log flag.
PrintFlag g_logFlag{PrintFlag::FILE};

/**
 * @brief Load configuration params from JSON file.
 * @return TRUE if parameters loaded or FALSE if not.
 */
bool loadConfig()
{
    // Init variables.
    ConfigReader config;
```

```
    // Open config json file (if not exist - create new and exit).
    if(config.readFromFile("RpiStreamer.json"))
    {
        // Read values and set to params
        if(!config.get(g_params, "Params"))
        {
            g_log.print(PrintColor::RED, g_logFlag) <<
            "[" << __LOGFILENAME__ << "][" << __LINE__ << "] ERROR: " <<
            "Params were not read" << endl;
            return false;
        }
    }
    else
    {
        // Set default params.
        config.set(g_params, "Params");

        // Save config file.
        if (!config.writeToFile("RpiStreamer.json"))
        {
            g_log.print(PrintColor::CYAN, g_logFlag) <<
            "[" << __LOGFILENAME__ << "][" << __LINE__ << "]: " <<
            "Config file created" << endl;
            return false;
        }
    }

    return true;
}

// Entry point.
int main(int argc, char **argv)
{
    // Configure logger.
    Logger::setSaveLogParams("Log", "log", 20, 1);

    // Welcome message.
    g_log.print(PrintColor::BLUE, g_logFlag) <<
    "[" << __LOGFILENAME__ << "][" << __LINE__ << "] " <<
    "RpiStreamer application v" << RPI_STREAMER_VERSION << endl;

    // Check verbose mode option.
    if (argc > 1)
    {
        string str = string(argv[1]);
        if (str == "-v" || str == "-vv")
        {
            // Set printing in console flag.
            g_logFlag = PrintFlag::CONSOLE_AND_FILE;
        }
    }

    // Load config file.
    if (!loadConfig())
    {
```

```
        g_log.print(PrintColor::RED, g_logFlag) <<
        "[" << __LOGFILENAME__ << "][" << __LINE__ << "] ERROR: " <<
        "Can't load config file" << endl;
        return -1;
    }
    g_log.print(PrintColor::CYAN, g_logFlag) <<
    "[" << __LOGFILENAME__ << "][" << __LINE__ << "] " <<
    "Config file loaded" << endl;

    // Create video source object.
    VSource *videoSource;
    if (g_params.videoSource.initString.find("/dev/") != string::npos)
    {
        // Create video source based on V4L2 API.
        videoSource = new VSourceV4L2();
    }
    else if (isdigit(g_params.videoSource.initString[0]))
    {
        // Create video source based on libcamera API.
        videoSource = new VSourceLibCamera();
    }
    else
    {
        // Create video source based on OpenCV API.
        videoSource = new VSourceOpenCv();
    }

    // Init video source.
    if (!videoSource->openVSource(g_params.videoSource.initString))
    {
        g_log.print(PrintColor::RED, g_logFlag) <<
        "[" << __LOGFILENAME__ << "][" << __LINE__ << "] ERROR: " <<
        "Can't initialise video source" << endl;
        return -1;
    }
    g_log.print(PrintColor::CYAN, g_logFlag) <<
    "[" << __LOGFILENAME__ << "][" << __LINE__ << "] INFO: " <<
    "Video source init" << endl;

    // Init video stabilizer.
    VStabiliser* videoStabiliser = new VStabiliserOpenCv();
    if (!videoStabiliser->initVStabiliser(g_params.videoStabiliser))
    {
        g_log.print(PrintColor::RED, g_logFlag) <<
        "[" << __LOGFILENAME__ << "][" << __LINE__ << "] ERROR: " <<
        "Can't initialise video stabilizer" << endl;
        return -1;
    }
    g_log.print(PrintColor::CYAN, g_logFlag) <<
    "[" << __LOGFILENAME__ << "][" << __LINE__ << "] INFO: " <<
    "Video stabilizer init" << endl;

    // Codec for server
    cr::video::VCodec *codec;

    #ifdef isRPI5
```

```
    codec = new VCodecLibav();
    codec->setParam(VCodecParam::TYPE, 1); // Raspberry Pi 5 doesn't support hardware
codec.
    #else
    codec = new VCodecV4L2();
    #endif

    // Init RTSP server.
    cr::rtsp::RtspServerLive555 server;
    if(!server.init(g_params.videoStream, codec))
    {
        g_log.print(PrintColor::RED, g_logFlag) <<
        "[" << __LOGFILENAME__ << "][" << __LINE__ << "] ERROR: " <<
        "Rtsp server can't init" << endl;
        return -1;
    }

    // Init frames.
    Frame frame;
    Frame stabilizedFrame;

    // Main loop.
    while (true)
    {
        // Wait new video frame 1000 msec.
        if (!videoSource->getFrame(frame, 1000))
        {
            g_log.print(PrintColor::RED, g_logFlag) <<
            "[" << __LOGFILENAME__ << "][" << __LINE__ << "] ERROR: " <<
            "No input video frame" << endl;
            continue;
        }

        // Stabilize frame.
        if (!videoStabiliser->stabilise(frame, stabilizedFrame))
        {
            g_log.print(PrintColor::RED, g_logFlag) <<
            "[" << __LOGFILENAME__ << "][" << __LINE__ << "] ERROR: " <<
            "Can't stabilize frame" << endl;
        }

        // Send frame to RTSP server.
        if(!server.sendFrame(stabilizedFrame))
        {
            g_log.print(PrintColor::RED, g_logFlag) <<
            "[" << __LOGFILENAME__ << "][" << __LINE__ << "] ERROR: " <<
            "Can't send frame to rtsp" << endl;
        }
    }
    return 1;
}
```