CMPT433

How-To Guide:
Video Streaming from Beaglebone Green to Host with OpenCV

Qiuhao Zheng

Zhuo Ping Huang (Edmond)

Nguyen Ai Vy Tran (Ivy)

This guide shows you how to install OpenCV on both the host and target to allow video and image transfer from beaglebone green to the host computer efficiently. It introduces how to compress the image and encode the image to bytes and stored in a vector container. And send the vector through UDP to host. It is very efficient for video streaming from BBG to the host.

Required Hardware

- Linux-Based Host Computer
- Beaglebone Green Running Linux (Debian)
- Any webcam that connects via USB

Step 1: Install OpenCV on Host (Debian)

Assuming you are running Debian 11.x on Linux. Go to the terminal and run the following commands to install the required packages.

1. sudo apt-get update
2. sudo apt-get install build-essential
3. sudo apt-get install libopencv-dev
4. sudo apt-get install libgtk-3-dev
5. sudo apt-get install libavcodec-dev
6. sudo apt-get install libavformat-dev
7. sudo apt-get install libswscale-dev
8. sudo apt-get install g++ (If not already installed)

After installing the necessary tools, download the OpenCV:

1. Clone the repository to the local machine:

   Run "git clone https://github.com/opencv/opencv.git"

2. Clone the opencv_contrib modules to the local machine:

   Run " git clone https://github.com/opencv/opencv_contrib.git"

3. Create a build folder, install the OpenCV in that folder (can add more modules depends on what you need it for)

Run the following commands:

```
mkdir build
cd build
cmake -D CMAKE_BUILD_TYPE=RELEASE \
        -D CMAKE_INSTALL_PREFIX=/usr/local \
        -D INSTALL_C_EXAMPLES=ON \
        -D INSTALL_PYTHON_EXAMPLES=OFF \
        -D OPENCV_GENERATE_PKGCONFIG=ON \
        -D OPENCV_EXTRA_MODULES_PATH=../opencv_contrib/modules \
        -D BUILD_EXAMPLES=ON ..
```

4. Compile and install:

```
make -j$(nproc)
sudo make install
```

Step 2: Install OpenCV on BBG (Debian)

OpenCV for arm architecture is different as above.

Install another version OpenCV for cross compiling code with static OpenCV libraries for BBG.

(Use static OpenCV libraries, so no need to install OpenCV on BBG when running the program)

1. Clone the repository to the local machine:
    - git clone https://github.com/opencv/opencv.git
2. Clone the opencv_contrib modules to the local machine:
    - git clone https://github.com/opencv/opencv_contrib.git
3. Create a build folder, install the OpenCV in that folder. Run the following commands:

```
mkdir build
cd build
cmake -D CMAKE_BUILD_TYPE=Release \
    -D CMAKE_INSTALL_PREFIX=/usr/local/arm-linux-gnueabihf \
    -D CMAKE_TOOLCHAIN_FILE=../platforms/linux/arm-gnueabi.toolchain.cmake \
    -D BUILD_SHARED_LIBS=OFF \
    -D BUILD_TESTS=OFF \
    -D BUILD_PERF_TESTS=OFF \
    -D BUILD_EXAMPLES=OFF \
    -D OPENCV_EXTRA_MODULES_PATH=../opencv_contrib/modules ..
```

Important notes:

- Make sure the toolchain set is arm-gnueabi.toolchain.cmake. So later on, we can cross-compile the code on the host.
- Make sure SHARED_LIBS=OFF. So we will install a static OpenCV library.

4. Compile and install:

```
make -j$(nproc)
sudo make install
```

Step 3: Open Camera, and Compress and Encode the Image

In this section, all codes provided are C++ codes.

Open camera in BBG:

```
cv::VideoCapture capture(0);
```

Close camera (release camera resource) in BBG:

```
capture.release();
```

Set the resolution of the camera (640x480):

```
capture.set(cv::CAP_PROP_FRAME_WIDTH, 640);
capture.set(cv::CAP_PROP_FRAME_HEIGHT, 480);
```

Depending on what you need, you can set it to higher resolution or lower resolution.

Compress and encode the image:

```
vector<uchar> buf;
cv::imencode(".jpg", frame,buf,vector<int>{cv::IMWRITE_JPEG_QUALITY, 25});
```

In the code above, the image is compressed into 25% of the original image quality. It can be 50%, or 80% depending on what you need. The image is encoded to bytes and stored in a vector. We then use UDP to send the vector to the host. This increases the efficiency of video streaming from beaglebone green to the host. The lower "IMWRITE_JPEG_QUALITY", the more efficient.

Step 4: Decode the Image and Display it on Host

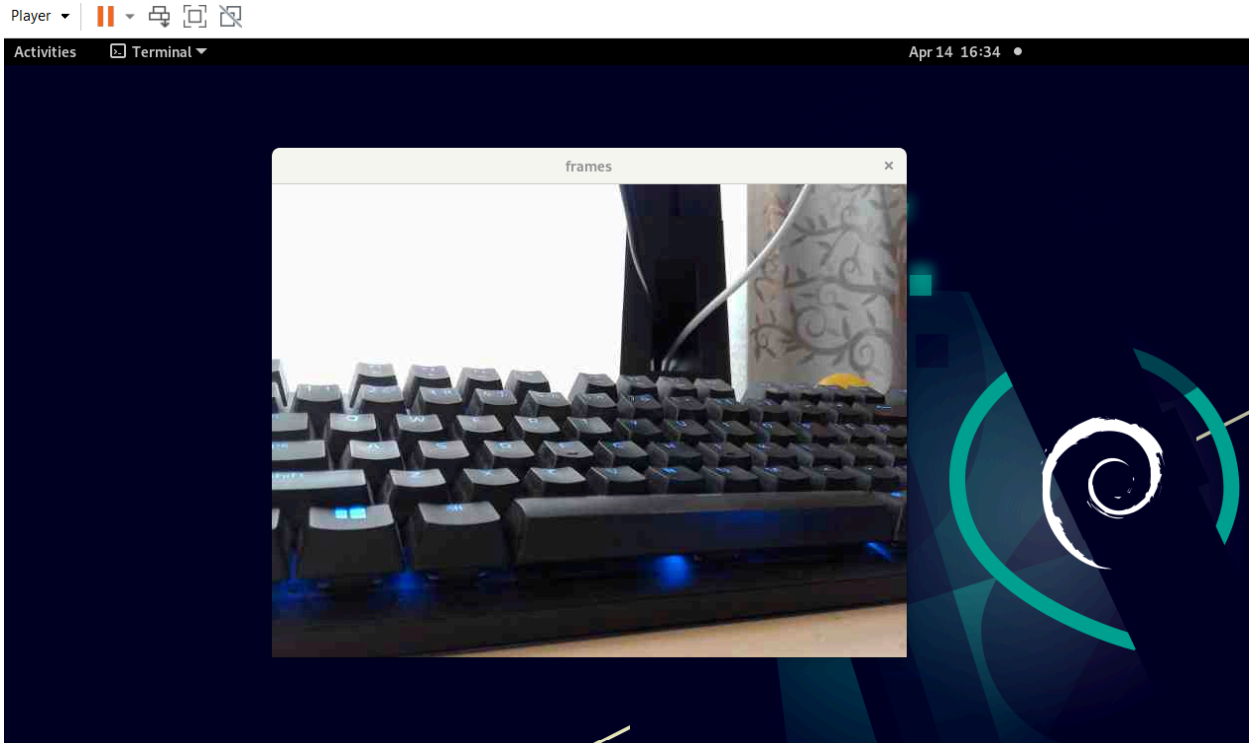
1. Receive the bytes (stored in vector) from BBG. Decode the bytes to image:

   cv::Mat frame = cv::imdecode(cv::Mat(buffer), cv::IMREAD_COLOR);

2. Create a window to display the image:

   cv::imshow("frames", frame);

As shown in the figure below, a window named “frame” will be created to display the image (or video streaming) on the host.



3. If you want to save the image, run:

   cv::imwrite(filename, frame);

Note. “filename” includes the path where you want to save the pic.

Troubleshooting

- When you run the program, if it shows the error “cannot find opencv.hpp or something related with opencv library”, it probably means you cross compile the code with dynamic OpenCV library. On BBG, there is an OpenCV library installed. Check if the SHARED_LIBS is OFF when you install OpenCV. Reinstall the OpenCV.
- If dependencies are not installed, make sure *apt-get update* is run first.
- If the webcam is not recognized, check that /dev/video1 or video0 is present. If still not working, then try changing:
  cv::VideoCapture capture(0) to cv::VideoCapture capture(1) in camera.cpp

Receive.cpp : Basic implementation on Host

```
void receiveFunction(const char* host, int port) {
    // Create a UDP socket
    int sockfd = socket(AF_INET, SOCK_DGRAM, 0);
    if (sockfd < 0) {
        std::cerr << "Failed to create socket!" << std::endl;
        return;
    }

    // Set up the server address structure
    struct sockaddr_in server_addr;
    memset(&server_addr, 0, sizeof(server_addr));
    server_addr.sin_family = AF_INET;
    server_addr.sin_port = htons(port);
    server_addr.sin_addr.s_addr = inet_addr(host);

    // Bind the socket to the specified IP address and port
    if (bind(sockfd, (struct sockaddr*)&server_addr,
sizeof(server_addr)) < 0) {
        std::cerr << "Bind failed!" << std::endl;
        return;
    }

    // Buffer to hold received data
    std::vector<uchar> buffer(65536);
```

```
    while (true) {
        // Receive data
        struct sockaddr_in client_addr;
        socklen_t addr_len = sizeof(client_addr);
        int recv_len = recvfrom(sockfd, buffer.data(), buffer.size(),
0, (struct sockaddr*)&client_addr, &addr_len);

        // If data was received
        if (recv_len > 0) {
            // Decode the received data as an image
            cv::Mat frame = cv::imdecode(cv::Mat(buffer),
cv::IMREAD_COLOR);

            // If the image is not empty, display it
            if (!frame.empty()) {
                cv::imshow("frames", frame);
            }
        }

    }

    // Close the socket
    close(sockfd);
}
```

Camera.cpp : Send video feed Implementation on Target

```
void sendVideoFeed(const char* host, int port) {
    // Create a UDP socket
    int sockfd = socket(AF_INET, SOCK_DGRAM, 0);
    if (sockfd < 0) {
        std::cerr << "Failed to create socket!" << std::endl;
        return;
    }

    // Set up the server address structure
    struct sockaddr_in server_addr;
    memset(&server_addr, 0, sizeof(server_addr));
```

```
    server_addr.sin_family = AF_INET;
    server_addr.sin_port = htons(port);
    server_addr.sin_addr.s_addr = inet_addr(host);

    cv::VideoCapture capture(0);
    if (!capture.isOpened()) {
        std::cerr << "Failed to open camera" << std::endl;
        return;
    }

    while (true) {
        // Capture a frame
        cv::Mat frame;
        if (capture.read(frame)) {
            // Encode the frame as a JPEG image
            std::vector<uchar> buf;
            cv::imencode(".jpg", frame, buf);

            // Send the encoded image
            sendto(sockfd, buf.data(), buf.size(), 0, (struct
sockaddr*)&server_addr, sizeof(server_addr));
        }


    }

    // Release the camera and close the socket
    capture.release();
    close(sockfd);
}
```

Source for Dependencies:
https://vegastack.com/tutorials/how-to-install-opencv-on-debian-11/