How To Stream Video Through Udp Using A Webcam On BeagleBone Green With Audio

This is a guide that will help you get started with streaming video and audio separately in the case that your webcam doesn't support video formats that have an audio component to it like H264 or YUYV. This guide is made for a webcam that can record a video in format like MJPEG that records individual frames and then stitch them together later without the sounds component but come with a microphone to stream the audio separately.

Table of Contents

Requirements

Required Hardware:

- BeagleBone Green
- Logitech c270 Webcam (There are better camera that can stream video(MP4) right away, but this one will stream mpeg)

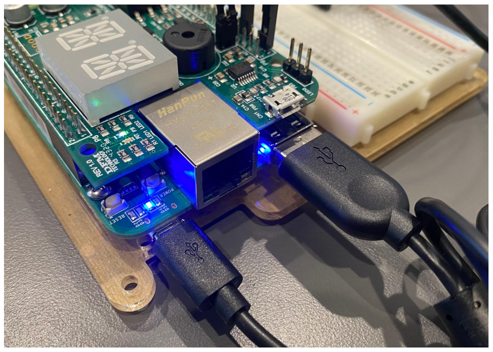
Required library or software:

- Node
- Npm
- Socket.io
- Alsa-utils
- v4l2
- FFmpeg
- jQuery?

Video Streaming

1. Connect webcam to BeagleBone

First step is to connect the webcam usb-A cable to the Beaglebone's USB-A port next to the ethernet port on the BeagleBone.



The user can easily check if the webcam connects to the BeagleBone successfully by using command `lsusb` on the terminal of the BeagleBone.

```
(target)$ lsusb
Bus 001 Device 002: ID 046d:0825 Logitech, Inc. Webcam C270
```

It's important that you check if `/dev/video0` exists on the Beaglebone board.

```
(target)$ ls /dev | grep video0
video0
```

Troubleshooting:

- If you can't find the webcam when its plugged in or it disconnects randomly
  - The webcam might require more power. Try plugging in the webcam to a powered usb hub and then into the Beaglebone board.

2. Install necessary software on host and BeagleBone

```
(host & target)$ sudo apt update
(host & target)$ sudo apt install ffmpeg
(target)$ sudo apt-get install v4l-utils libv41-dev
```

When cross compiling we will need Beaglebone's copy of `libv4l2.so`. Attach your NFS and create a new directory and copy the file over:

```
(target)$ mkdir /mnt/remote/v4l2_lib_BB
(target)$ cp /usr/lib/arm-linux-gnueabihf/libv4l2.so /mnt/remote/v4l2_lib_BB/libv4l2.so
```

We will have to link this in our Makefile like so:

```
LFLAGS = -L$(HOME)/cmpt433/public/v4l2_lib_BB

app:
	$(CC_C) $(CFLAGS) $(SOURCES) -o $(OUTDIR)/$(TARGET) $(LFLAGS) -lv4l2
```

This is an example snippet of how you want to add it to your Makefile.

3. Method 1 - Streaming with FFMpeg to FFMpeg:

We are able to stream our camera feed via UDP out of the box with only a minor modification using Derek Malloy's `capture.c` found here:
https://github.com/derekmolloy/boneCV.git.

Download `capture.c` and create a rudimentary `Makefile` like this:

```
TARGET= bbg_exec
SOURCES= camera.c

PUBDIR = $(HOME)/cmpt433/public/myApps
OUTDIR = $(PUBDIR)
CROSS_TOOL = arm-linux-gnueabihf-
CC_CPP = $(CROSS_TOOL)g++
CC_C = $(CROSS_TOOL)gcc

CFLAGS = -Wall -g -std=c99 -D _POSIX_C_SOURCE=200809L -Werror -Wshadow

LFLAGS = -L$(HOME)/cmpt433/public/v4l2_lib_BB

app:
	$(CC_C) $(CFLAGS) $(SOURCES) -o $(OUTDIR)/$(TARGET) $(LFLAGS) -lv4l2

clean:
	rm -f $(OUTDIR)/$(TARGET)
```

Modify this specific section of `capture.c` from:

```
if (force_format) {
	  if (force_format==2){
		fmt.fmt.pix.width       = 1920;
		fmt.fmt.pix.height      = 1080;
		fmt.fmt.pix.pixelformat = V4L2_PIX_FMT_H264;
		fmt.fmt.pix.field       = V4L2_FIELD_INTERLACED;
	  }
```

To:

```
if (force_format) {
	  if (force_format==2){
		fmt.fmt.pix.width       = 640;
		fmt.fmt.pix.height      = 360;
```

```
                fmt.fmt.pix.pixelformat = V4L2_PIX_FMT_MJPEG;
                fmt.fmt.pix.field       = V4L2_FIELD_NONE;
           }
```

Make the `capture.c` and execute this command to start streaming via UDP using FFMpeg. We are letting FFMpeg do all the heavy lifting.

```
(target)$ cd /mnt/remote/myApps && ./bbg_exec -F -o -c0 | ffmpeg -i pipe:0
-vcodec copy -f mjpeg udp://192.168.7.1:1234
```

The executable `bbg_exec` is dumping raw camera data into `stdout`, which ffmpeg takes in as input and directly streams the output to the UDP address `192.168.7.1:1234.` We will need ffmpeg to capture this UDP stream of raw video data on the other end (in step 5).

You should see the webcam turned on if everything goes right.

Troubleshooting:

- If you see error messages in the console when running the executable with ffmpeg. You can ignore most of those. Our experiments show that it still works.
    - If it doesn't work. You may need to refer to ffmpeg documentation.

4. Method 2 - Streaming with C UDP to FFMpeg:

We can also stream our camera feed using C code. First we would need Derek Malloy's `capture.c` and modify it to stream the camera feed instead of dumping it into stdout. You can find the `capture.c` here:

`https://github.com/derekmolloy/boneCV.git`.

From here, we will be following this guide in particular: CMPT433 How-to-Guide (sfu.ca). As it already shows the bare bone modifications required. Here is a TLDR:

Add to `capture.c`:

```
#include <sys/socket.h>
#include <netinet/in.h>
#include <stdbool.h>
#include <arpa/inet.h>
#include <netdb.h>

#define PORT_T 3000
#define RPORT_T 1234 //Port for NodeJS
static struct sockaddr_in sinT;
static struct sockaddr_in sinRemoteT;
static int socketDescriptorT;


void openConnectionT()
{
        memset(&sinT, 0, sizeof(sinT));
        sinT.sin_family = AF_INET;
        sinT.sin_addr.s_addr = htonl(INADDR_ANY);
        sinT.sin_port = htons(PORT_T);
        socketDescriptorT = socket(PF_INET, SOCK_DGRAM, 0);
        bind(socketDescriptorT, (struct sockaddr*) &sinT, sizeof(sinT));
        sinRemoteT.sin_family = AF_INET;
        sinRemoteT.sin_port = htons(RPORT_T);
        sinRemoteT.sin_addr.s_addr = inet_addr("192.168.7.1");
}

int sendResponseT(const void *str, int size)
{
        int packetSent = 0;
        sendto(socketDescriptorT, str,size, 0,
        (struct sockaddr *) &sinRemoteT,
        sizeof(sinRemoteT) );

        return packetSent;
}

void closeConnectionT()
{
      close(socketDescriptorT);
}
```

Modify `process_image` to send the output to UDP instead of `stdout`:

```
static void process_image(const void *p, int size)
{
        if (out_buf) {
                // printf("Sending response via udp\n");
                sendResponseT(p, size);
                // fwrite(p, size, 1, stdout);
        }
```

Modify `main` function to start the UDP connection and remove cli flags to streamline the executable:

```
force_format=2;
out_buf++;
frame_count = 0;

openConnectionT();
open_device();
init_device();
start_capturing();
mainloop();
stop_capturing();
closeConnectionT();
uninit_device();
close_device();
```

Just like in method 1 modify this section:
From:

```
if (force_format) {
           if (force_format==2){
                fmt.fmt.pix.width       = 1920;
                fmt.fmt.pix.height      = 1080;
                fmt.fmt.pix.pixelformat = V4L2_PIX_FMT_H264;
                fmt.fmt.pix.field       = V4L2_FIELD_INTERLACED;
           }
```

To:

```
if (force_format) {
           if (force_format==2){
                fmt.fmt.pix.width       = 640;
                fmt.fmt.pix.height      = 360;
                fmt.fmt.pix.pixelformat = V4L2_PIX_FMT_MJPEG;
                fmt.fmt.pix.field       = V4L2_FIELD_NONE;
           }
```

Refer to method 1 to compile `capture.c`, but run the executable like so:

```
(target)$ ./bbg_exec
```

Troubleshooting:

- If you are having problems compiling the capture.c please refer to this guide CMPT433 How-to-Guide (sfu.ca). They go into more detail in each step.

5. Capturing a Video Stream with FFMpeg

In short this is the command for ffmpeg to capture a UDP video stream. This command works for both methods above.

```
(host)$ ffmpeg -re -y -i udp://192.168.7.1:1234 -preset ultrafast -f mjpeg pipe:1
```

This isn't really that useful alone. What this is doing it's telling ffmpeg to receive input from `192.168.7.1:1234` and output it into `stdout`(pipe:1).

There are many other guides that show how to capture the video stream into something more useful. To keep this guide short please refer to other guides that go into more detail. Such as displaying the video stream on a website (refer to this guide if you are interested CMPT433 How-to-Guide)

Audio streaming:

Audio streaming similarly uses UDP to stream packets over the network. To stream UDP, make use of the ffmpeg command line interface to convert raw WAV data to mp3 before streaming packets over the network. Afterwards you can do whatever with the streamed data.

1. Install Necessary Software And Finding Audio Device

We need alsa-utils to find an audio device to sample audio from.

```
(target)$ sudo apt-get install alsa-utils
(target)$ arecord -L

default:CARD=webcam
    webcam, USB Audio
    Default Audio Device
```

You should see this audio device listed, remember this audio device name. We will need it as our input device for our ffmpeg command.

Troubleshooting:

- If you cannot find similar audio devices listed. Make sure the webcam is plugged in.
  - If you still cannot find it, it may be named differently. This will require you to test each audio device until you find the right one.

2. Streaming Audio Data to Another Device

We will be using ffmpeg again to stream the output of our webcam microphone.

```
(target)$ ffmpeg -hide_banner -loglevel error -ar 44100 -f alsa -i
default:CARD=WEBCAM -acodec mp3 -f mp3 udp://192.168.7.1:10001
```

- `-hide_banner`: This option hides the FFmpeg banner information from the output.
- `-loglevel error`: This sets the logging level to "error", which suppresses all but the most important error messages.
- `-ar 44100`: This sets the audio sample rate to 44100 Hz.
- `-f alsa`: This specifies the input format as ALSA.
- `-i default:CARD=WEBCAM`: This specifies the audio input device.
- `-acodec mp3`: This specifies the audio codec as MP3.
- `-f mp3`: This specifies the output format as MP3.
- `-udp://192.168.7.1:10001`: This specifies the output destination as a UDP stream

3. Receiving and Saving Audio Stream From Another Device

Using ffmpeg, we can receive and save the mp3 audio stream into an mp3 file:

```
(host)$ ffmpeg -f mp3 -i udp://192.168.7.1:10001 -f mp3 - > test.mp3
```

Stop writing into the file with a simple `CTRL + C` and playback the mp3 file to hear the audio captured from the hardware on the target device.

```
(host)$ ^C
```

This may be useless to most, but the concept of utilizing the video stream applies here. For example we can use NodeJS to spawn an ffmpeg command and have it read the output in `stdout` and utilize it however we wish.

```
const express = require("express");
const app = express();
const child = require('child_process');

const port = 3000;

let ffm = child.spawn(
   "ffmpeg",
   "-f mp3 -i udp://192.168.7.1:10001 -f mp3 -".split(" ")
);

app.get("/audio", (req, res) => {
   console.log("Audio request");
   res.writeHead(200, { "Content-Type": "audio/mp3" });

   ffm.stdout.on("data", (data) => {
      res.write(data);
   });

});

var listener = server.listen(process.env.PORT || port, () =>
console.log(`Listening on ${listener.address().port}`))
```

Have fun and get creative!

References:

https://github.com/derekmolloy/boneCV

https://opencoursehub.cs.sfu.ca/bfraser/grav-cms/cmpt433/links/files/2017-student-howtos/CapturingAndStreamingWebcamVideoOnBBG.pdf

https://opencoursehub.cs.sfu.ca/bfraser/grav-cms/cmpt433/links/files/2022-student-howtos/StreamingWebcamFromBeagleBoneToNodeJSServer.pdf

https://opencoursehub.cs.sfu.ca/bfraser/grav-cms/cmpt433/links/files/2015-student-howtos/RecordingWebcamVideos.pdf