Streaming Webcam Video to a Browser

Team: Beagle Rangers

This guide will walk you through the necessary steps to display onto a browser the video from a webcam. Using C and Node.js, the overall process is made significantly easier due to a JavaScript library called *JSMpeg*. This guide assumes that you already have a website and a Node.js server, all set and ready to go.

1. Follow a guide to set up the webcam for your BeagleBone

There are some good guides that you can follow to get the webcam working for your BeagleBone. Here are some notable ones that worked for us:

- Recording Webcam Videos with the BeagleBone Black
- Capturing and Streaming Webcam Video with the BeagleBone Green

Before going to the next step, ensure that your webcam works with the BeagleBone and is able to send its video feed over a UDP stream. You should have a working *capture.c* file (from Derek Molloy's *boneCV* repository) working with *FFmpeg*.

2. Set up a website

In your *index.html* file, include a *canvas* element with a set *id*. Additionally, include the *JSMpeg* package by downloading the bundled *.min.js* file and referencing it from a *script* tag.

```
<body>
      <canvas id="self-view"></canvas>

      <script src="js/jsmpeg.min.js" type="text/javascript"></script>
      <script src="js/webcam.js" type="text/javascript"></script>
</body>
```

Add a *JSMpeg* Player to the client by typing the following to *webcam.js*:

```
// Code adapted from the sample code from jsmpeg's GitHub:
// https://github.com/phoboslab/jsmpeg/blob/master/view-stream.html
const WS_ADDR = "192.168.7.2";
const WS_PORT = 8088;
const options = {
      canvas: $("#self-view")
};

new JSMpeg.Player(`ws://${WS_ADDR}:${WS_PORT}`, options);
```

3. Set up a Node.js server

On the Node.js side of things, create two files, *server.js* and *stream.js*. Write the normal code in your *server.js* file to serve your *index.html* to the user. *stream.js* is there to help with the modularity. In *server.js*, make sure to add the following to activate the *stream.js* code.

```
// server.js
import stream from "./lib/stream.js";
stream.listen();
```

In *stream.js*, add the following:

```
// Code adapted from the sample code from jsmpeg's GitHub:
// https://github.com/phoboslab/jsmpeg/blob/master/websocket-relay.js

import WebSocket, {WebSocketServer} from "ws";
import dgram from "dgram";

const UDP_STREAM_ADDR = "192.168.7.2";
const UDP_STREAM_PORT = 8080;
const WS_PORT = 8088;
const wSocketServer = new WebSocketServer({
      perMessageDeflate: false,
      port: WS_PORT
});
```

```
wSocketServer.broadcast = (data) => {
      wSocketServer.clients.forEach((client) => {
            if (client.readyState === WebSocket.OPEN) {
                  client.send(data);
            }
      });
};
wSocketServer.connectionCount = 0;
wSocketServer.on("connection", (socket, upgradeReq) => {
      wSocketServer.connectionCount++;
      wSocketServer.on("close", (code, message) => {
            wSocketServer.connectionCount--;
      });
});

export default {
      listen: () => {
            const udpSocket = dgram.createSocket("udp4");

            udpSocket.bind(UDP_STREAM_PORT, UDP_STREAM_ADDR);
            udpSocket.on("listening", () => {});
            udpSocket.on("message", (chunk, rinfo) => {
                  wSocketServer.broadcast(chunk);
            });
      }
};
```

From the code in *webcam.js*, we created a listening UDP socket that catches the UDP stream of the webcam video from *capture.c*. We also need to create a WebSocket server that passes the encoded webcam video from the UDP stream to be received and decoded by a *JSMpeg* Player in the client (hence, the code written in *webcam.js*).

4. Run the binary from *capture.c*

Once you compile your *capture.c* program, you will be able to run the resulting binary to drive the webcam and pipe the raw output to *FFmpeg*. Although the above guides from the first step do show you the exact command for it, we will need to alter the command slightly. We want to work with the *mpegts* format instead of the *mpeg* format because *JSMpeg* only works with the *mpegts* format. This allows it to automatically decode the encoded stream and display it quickly. The new terminal command is as follows:

```
./capture -F -o -c0 | ffmpeg -i pipe:0 \
        -f mpegts -codec:v mpeg1video \
        -s 640x480 -b:v 1024k -bf 0 udp://192.168.7.2:8080
```

5. Run the Node.js server

Hopefully, all should be working at this point. You can now run your Node server, serving your users with a website that receives and displays a webcam stream.

6. Stuck? Some troubleshooting tips below

Make sure to go through a webcam guide before proceeding with this guide. Some useful guides are listed in the first section (Step 1). Specifically, verify that *FFmpeg* is able to stream successfully to your *VLC* media player.

Verify that you have your IP addresses and port numbers set correctly. Specifically, there are three different port numbers to keep in mind. One port is used for the HTTP server to serve the website. Another is used for the UDP stream between *FFmpeg* command and the listening UDP socket on the Node server, and the third one is used between the WebSocket server (on the Node server) and the client's *JSMpeg* Player.

For more information about *JSMpeg*, take a look at its detailed documentation.