CMPT 433

Reading and Generating QR Code from NodeJS through Webcam Stream

Last updated: Apr 11, 2023

Jason Chung, Martin Chudy, Josh Murphy, Caleb Bradley

Table of Contents

This document serves to show the user

1. Setting up webcam on the BeagleBone Green
2. Using a C program to send frame data through UDP
3. Using Javascript and libraries to analyze frame data on NodeJS server
4. Troubleshooting common problems

Guide has been tested on

- BeagleBone (Target): Debian 11.5
- PC OS (host): Debian 11.5

Hardware Used

- BeagleBone Green
- Webcam (any 720p or 1080p webcam should work). We used this one.

0. Prerequisite setup

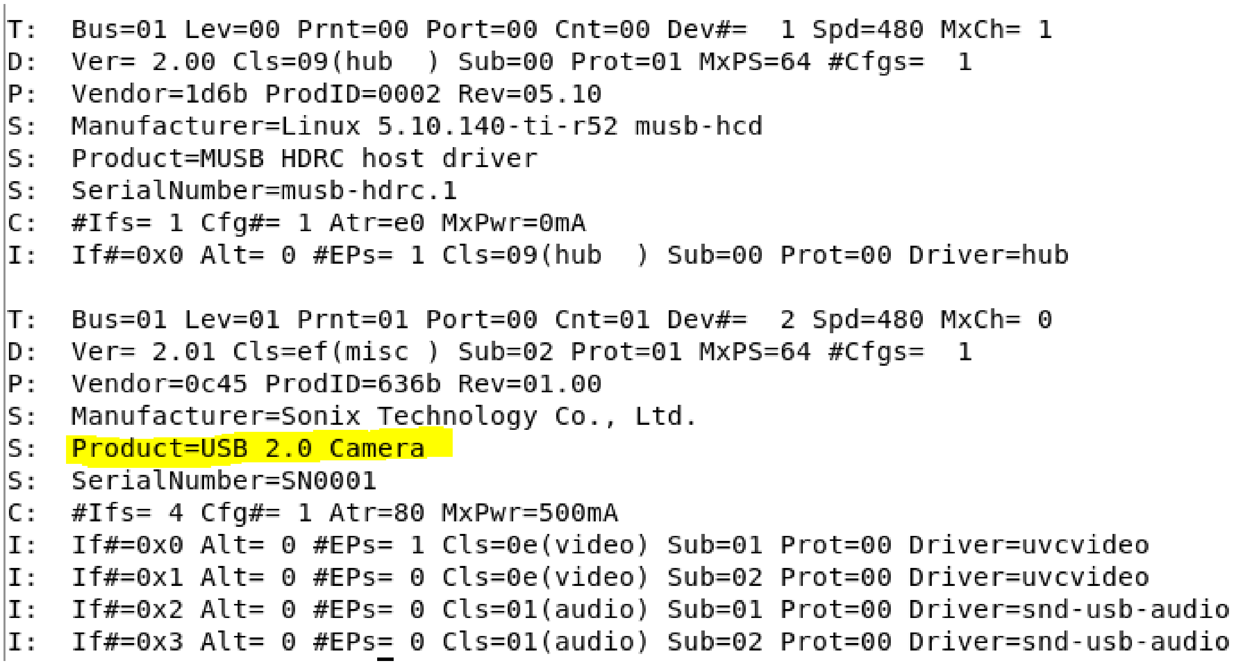
- The guide requires internet access from the BeagleBone to function properly due to the usage of UDP. We recommend setting up Ethernet over USB by following Dr.Brian's Networking Guide. Our group used the recommended ethernet over USB option (section 2).

1. Setting up a webcam on the BeagleBone Green

1.1 Connect the webcam to the BeagleBone's USB port

Run the command to verify USB connection:

```
(bbg)$ usb-devices
```



```
T:  Bus=01 Lev=00 Prnt=00 Port=00 Cnt=00 Dev#=  1 Spd=480 MxCh= 1
D:  Ver= 2.00 Cls=09(hub  ) Sub=00 Prot=01 MxPS=64 #Cfgs=  1
P:  Vendor=1d6b ProdID=0002 Rev=05.10
S:  Manufacturer=Linux 5.10.140-ti-r52 musb-hcd
S:  Product=MUSB HDRC host driver
S:  SerialNumber=musb-hdrc.1
C:  #Ifs= 1 Cfg#= 1 Atr=e0 MxPwr=0mA
I:  If#=0x0 Alt= 0 #EPs= 1 Cls=09(hub  ) Sub=00 Prot=00 Driver=hub

T:  Bus=01 Lev=01 Prnt=01 Port=00 Cnt=01 Dev#=  2 Spd=480 MxCh= 0
D:  Ver= 2.01 Cls=ef(misc ) Sub=02 Prot=01 MxPS=64 #Cfgs=  1
P:  Vendor=0c45 ProdID=636b Rev=01.00
S:  Manufacturer=Sonix Technology Co., Ltd.
S:  Product=USB 2.0 Camera
S:  SerialNumber=SN0001
C:  #Ifs= 4 Cfg#= 1 Atr=80 MxPwr=500mA
I:  If#=0x0 Alt= 0 #EPs= 1 Cls=0e(video) Sub=01 Prot=00 Driver=uvcvideo
I:  If#=0x1 Alt= 0 #EPs= 0 Cls=0e(video) Sub=02 Prot=00 Driver=uvcvideo
I:  If#=0x2 Alt= 0 #EPs= 0 Cls=01(audio) Sub=01 Prot=00 Driver=snd-usb-audio
I:  If#=0x3 Alt= 0 #EPs= 0 Cls=01(audio) Sub=02 Prot=00 Driver=snd-usb-audio
```

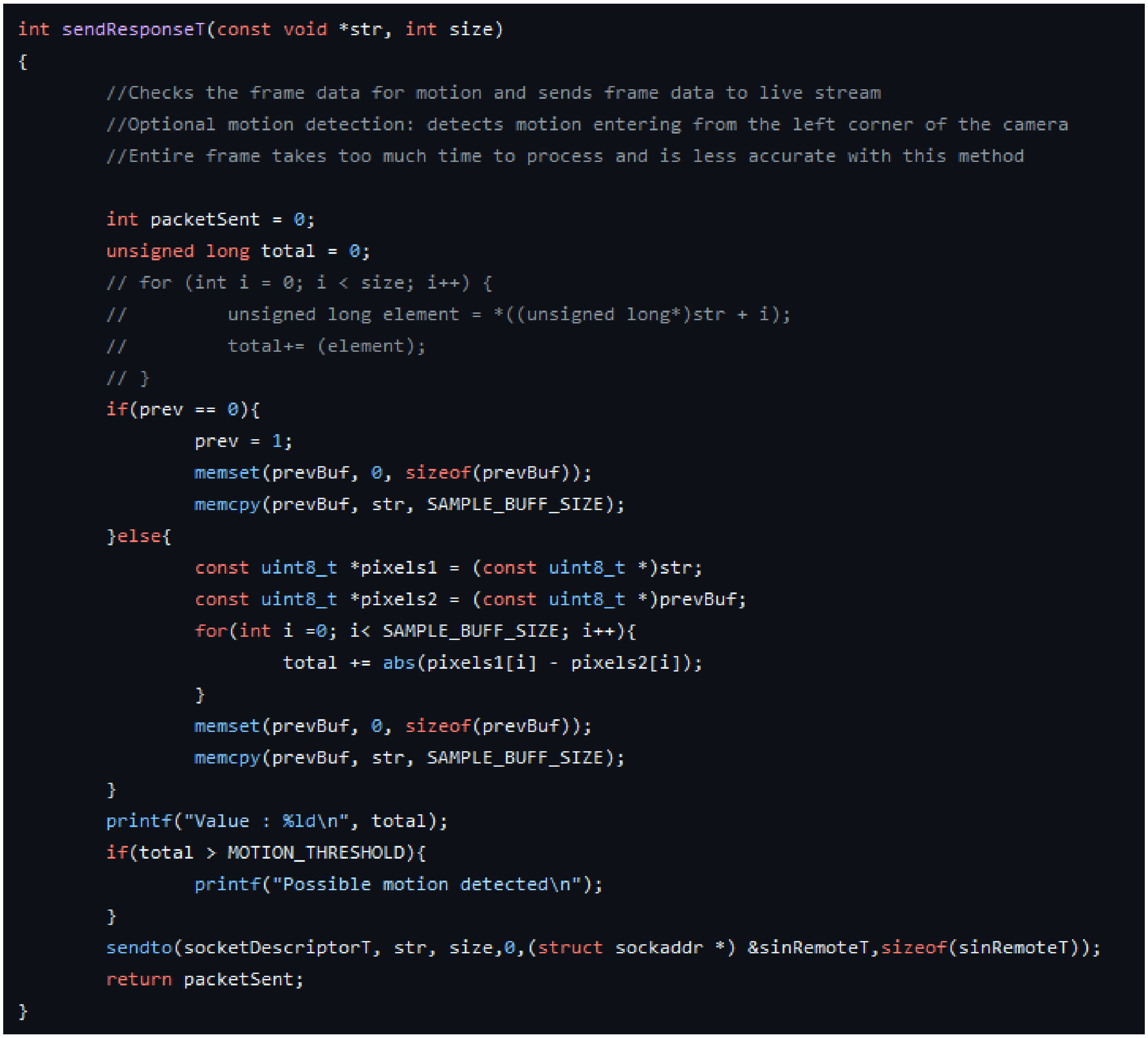
Look for an entry that resembles the camera. The maximum power should also be visible. If using an USB splitter to power multiple USB devices, keep this in mind.

2. Using a C program to analyze/send frame data through UDP

2.1 Download the capture.c code from the BoneCV library

- Modify the capture.c code following the steps from this guide (section 1.3).
- Additional modifications can be made based on your needs.
- Resolution and display size can be changed by adjusting the fmt.fmt.pix.width and fmt.fmt.pix.height in force_format.
- OpenCV processing can also be done directly without the use of a NodeJS server if desired. The frame data is sent to the Node server

through UDP with the function *sendResponseT()*. Below is a sample lightweight code for detecting motion on the BBG using sum of absolute difference before sending off the packet:



```
int sendResponseT(const void *str, int size)
{
        //Checks the frame data for motion and sends frame data to live stream
        //Optional motion detection: detects motion entering from the left corner of the camera
        //Entire frame takes too much time to process and is less accurate with this method

        int packetSent = 0;
        unsigned long total = 0;
        // for (int i = 0; i < size; i++) {
        //         unsigned long element = *((unsigned long*)str + i);
        //         total+= (element);
        // }
        if(prev == 0){
                prev = 1;
                memset(prevBuf, 0, sizeof(prevBuf));
                memcpy(prevBuf, str, SAMPLE_BUFF_SIZE);
        }else{
                const uint8_t *pixels1 = (const uint8_t *)str;
                const uint8_t *pixels2 = (const uint8_t *)prevBuf;
                for(int i =0; i< SAMPLE_BUFF_SIZE; i++){
                        total += abs(pixels1[i] - pixels2[i]);
                }
                memset(prevBuf, 0, sizeof(prevBuf));
                memcpy(prevBuf, str, SAMPLE_BUFF_SIZE);
        }
        printf("Value : %ld\n", total);
        if(total > MOTION_THRESHOLD){
                printf("Possible motion detected\n");
        }
        sendto(socketDescriptorT, str, size,0,(struct sockaddr *) &sinRemoteT,sizeof(sinRemoteT));
        return packetSent;
}
```

3. Using Javascript and libs to analyze frame data on NodeJS server

3.1 Setting up NodeJS server

- Follow section 2 of this guide (skip the 2.9 portion)
- Run the commands:

```
(bbg)$ ./capture.c
(host)$ node index.js
```

Make sure the stream is visible on the browser.

3.2 Installing required libraries for QR

- Install JIMP: this will be used to read .jpg files and the received frame data
  ```
  (host)$ npm install qrcode-reader jimp
  ```
- Install the jsQR library: this is used for scanning QR codes. The other QR library does not work for scanning.
  ```
  (host)$ npm install jsqr --save
  ```
- Install QRcode
  ```
  (host)$ npm install qrcode
  ```

3.3 Import the libraries to our javascript file

Inside index.js, add the following lines to the start of the file:

```
const jsQR = require("jsqr");
const qrCodeReader = require('qrcode-reader');
const qrGenerator = require('qrcode');
const Jimp = require("jimp");
const fs = require("fs");
```

3.4 QR Code Scanning

The function *ffmpeg.stdout.on('data', function (data)* is in charge of dealing with the frame data received from the C application running on the BBG. You may not want to call the QR scan function too often, otherwise the stream can get choppy and you may run into a malformed image data error. To add a delay, wrap the function with the following:

```
if( (Date.now()/1000) - last_time > 1 ){
          last_time = (Date.now())/1000;
          //rest of code
}
```

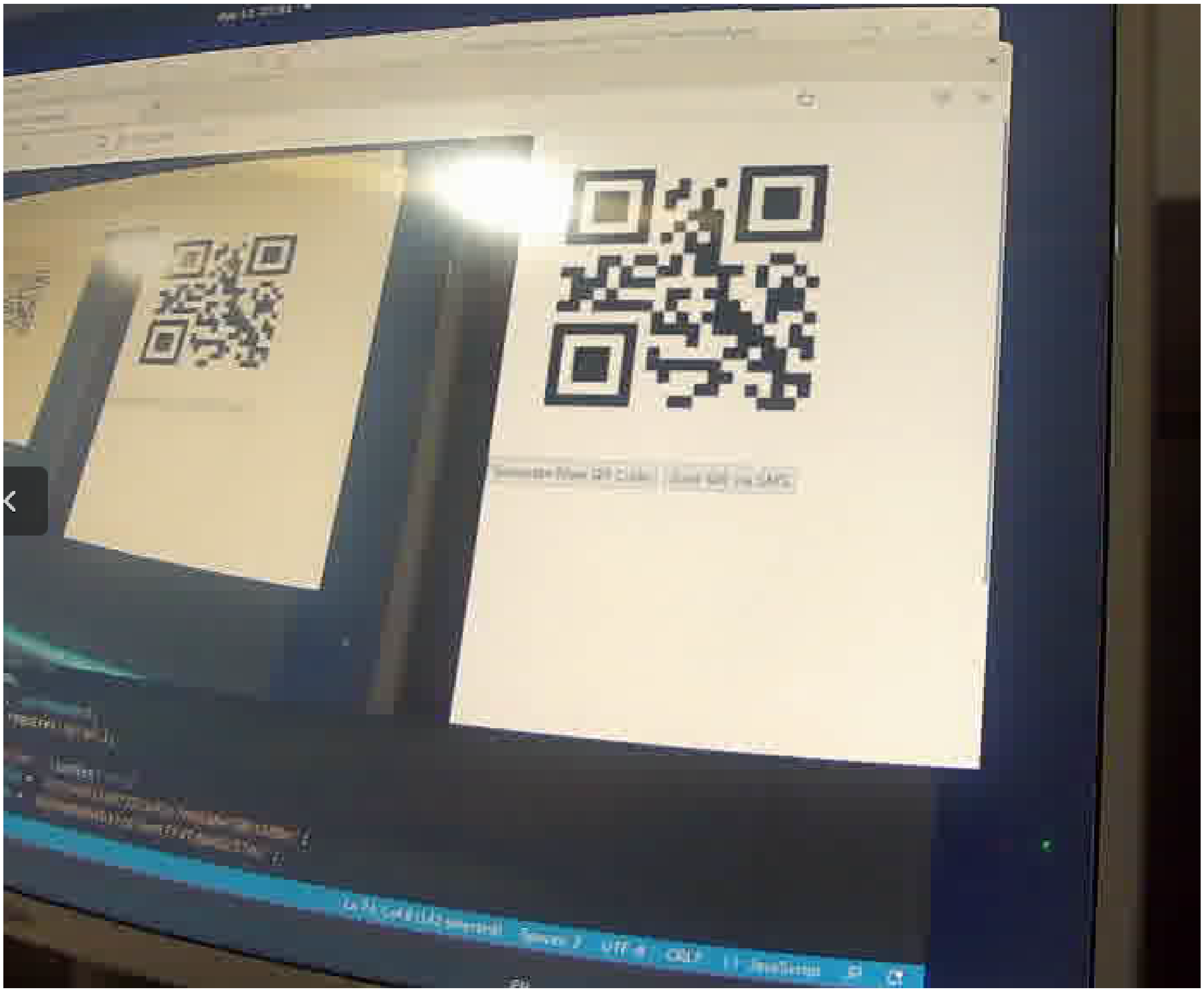
3.4.1 Convert frame data to image

- We first save the frame data as a .jpg image using fs.
  ```
  fs.writeFileSync("jee.jpg", data);
  ```
- Convert it to .png format so that it can be processed later on.

  ```
  Jimp.read("jee.jpg", function (err, image) {
            //will print the error
            if (err) {
            console.log(err)
            }
            //Convert the image into PNG format and save
  ```

```
     else {
     image.write("jee.png")
     }
})
// Read the image file
fs.readFile('jee.png', (err, data) => {
if (err){
console.log(err);
return;
}
```

[Sample Image saved to jee.png]



3.4.2 Scan the image for possible QR code

- Read in the image using JIMP

```
// Create a Jimp image object from the data
```

```
Jimp.read(data, (err, image) => {
if (err){
     console.log(err);
     return;
}
// Get the image as a Uint8ClampedArray
const imageData = image.bitmap.data;
```

- Check for QR code:

  ```
  // Do something with the image data(scan)
  const code = jsQR(imageData, 800, 600);
  ```

  If `code` returns true, that means a QR code has been successfully scanned. Print it out using `console.log("Found QR code", code);` to view the QR code info. code.data contains the text value stored within a QR code. To validate using QR code, do not compare the scanned QR code image to an existing one. Save the text value of your existing QR in a text file and compare it to code.data.

3.4.3 Generating a new QR code

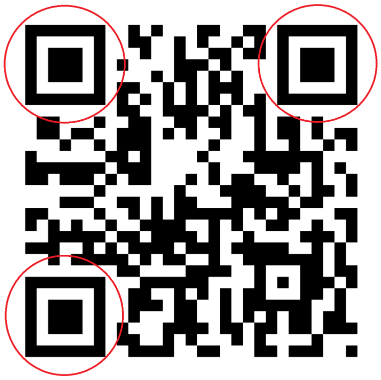
- Save the text value you want to encode
  `let newcode = “new text value”`
- Use qrcode lib to generate a QR code and save it as png

  ```
  qrGenerator.toFile('qrcode/qr.png', newcode, {
       errorCorrectionLevel: 'H'
       }, function(err) {
       if (err) throw err;
       });
  ```

- Optional: save the new text data to a text file

4. Troubleshooting

4.1 Receiving the error while scanning QR code: couldn't find enough finder patterns: 2 patterns found



The finder patterns are the black squares within a QR code. The qrcode library runs into this issue frequently. The fixes listed here may or may not fix the problem. We recommend using the jsQR library instead as mentioned in the guide.

4.2 Error: Malformed data passed to binarizer.

The output images may be malformed if for every frame received, the image gets overwritten. This is why we wrapped the entire function to only run if at least 1 second has passed since the last execution. If you are still encountering this error frequently, increase the timer. Additionally, you may wish to use async-mutex but we will not get into that.

4.3 Choppy frame rate

Reduce the size of each frame and refresh the browser.

5. References

[1] Webcam stream from BeagleBone to Node
https://opencoursehub.cs.sfu.ca/bfraser/grav-cms/cmpt433/links/files/2022-student-howtos/StreamingWebcamFromBeagleBoneToNodeJSServer.pdf

[2] Setting up internet on the BeagleBone
https://opencoursehub.cs.sfu.ca/bfraser/grav-cms/cmpt433/guides/files/Networking.pdf

[3] QR code generation
https://blog.logrocket.com/create-read-qr-codes-node-js/

[4] jsQR library
https://github.com/cozmo/jsQR

[5] BoneCV library
https://github.com/derekmolloy/boneCV