

# Capturing Webcam Images to Display on a Web Interface

## Introduction

The following guide presents step-by-step instructions on capturing images from a webcam with a Python script and displaying them on a web interface with a Node.js server. Our goal is to capture images and display them through a web interface in one second intervals. This guide is written for the BeagleBone Black.

## Set Up

Before starting to write any code, make sure you have all the required hardware and software components.

### Required Hardware:

- BeagleBone Black (BBB)
- Any USB webcam that is compatible with the BBB

Setting up the hardware is fairly straightforward. You need to connect the USB webcam to the BBB and ensure the BBB is connected to a network that another client has access to.

### Required Software:

- Node.js - for the server
- Sockett.IO - for connecting the server and client so they can communicate
- socketIO-client - Python Socket.IO client library for connecting the server and the Python script so they can communicate
- OpenCV - library used to capture and save images from the webcam

The software can be split into two components: image capture and server. The image capture component uses socketIO-client and OpenCV. The server component uses Node.js and Socket.IO. socketIO-client can be installed through the command line: `#pip install -U socketIO-client`. OpenCV can be installed by following steps 1 and 2 from the following guide <https://help.ubuntu.com/community/OpenCV>. Node.js should be preinstalled on the BBB. Socket.IO can be installed with the following command through the command line in the directory containing the package.json file: `#npm install`.

## Guide

We will start with a description of the folder structure, then move on the image capture component, and finally move on to the server component. We will assume you have followed the Node.js guide provided by Dr. Brian Fraser on the CMPT 433 home page. You should be able to get a basic Node.js server running and create a basic web page. For the sake of brevity, we will be expanding upon that guide. We will go through steps for each component, then provide a complete listing of the source code for each file we go through.

## Folder Structure

Create the following directory structure in your working directory along with the files (note: we will only discuss the underlined files):

server/ (directory for server code)

- lib/ (directory for server JavaScript code)
  - webcam\_server.js (JavaScript code that communicates with webcam.py)
- public/ (directory for static files served to the client)
  - images/ (directory for saving images captured by the webcam)
  - javascripts/ (directory for client JavaScript code)
    - webcam\_client.js (JavaScript code that communicates with server)
  - stylesheets/ (directory for stylesheets)
    - style.css (stylesheet)
  - index.html (main page served to the client)
- package.json (third-party packages required by the server)
- server.js (main server code)

webcam/ (directory for webcam image capture code)

- webcam.py (Python script for capturing images from the webcam)

## Image Capture - webcam.py

1. Import the following:
  - a. cv2 for image capturing
  - b. logging for debug logging (troubleshooting tip: this is required, otherwise SocketIO complains)
  - c. datetime and timedelta for timing the image capture interval
  - d. SocketIO for communicating with the server
2. Set the logging level to DEBUG to stop SocketIO from complaining.
3. Set the time of the next image capture.
4. Set up the Socket.IO client to communicate with the Node.js server. We will be running the server on localhost port 8088.
5. Set up the webcam so we can capture images from it.
6. Write a loop that captures images from the webcam.
7. Capture an image if the current time is past the image capture time set in step 3.
8. Set the time for the next image capture to be 1 second from the previous capture time.
9. Capture an image and save it to the server/public/images/ directory.
10. Alert the server that a new image is available.

Here is the final listing for webcam.py with the steps listed:

```
1 # step 1
2 import cv2
3 import logging
4 from datetime import datetime, timedelta
5 from socketIO_client import SocketIO
6 # step 2
```

```

7 logging.basicConfig(level=logging.DEBUG)
8 # step 3
9 nextCapture = datetime.now()
10 # step 4
11 socketIO = SocketIO('localhost', 8088)
12 # step 5
13 webcam = cv2.VideoCapture(0)
14 # step 6
15 while True:
16     # step 7
17     if nextCapture <= datetime.now():
18         # step 8
19         nextCapture = nextCapture + timedelta(seconds=1)
20         # step 9
21         img = webcam.read()[1]
22         cv2.imwrite('../server/public/images/img.png', img)
23         # step 10
24         socketIO.emit('new_image')

```

### Server - server/public/index.html

1. Create a `<div>` to place the image in.
2. Import the required JavaScript code.
  - a. `socket.io` for setting up a connection to the server from the client
  - b. `jQuery` for updating the image on the page
  - c. `webcam_client.js` for communicating with the server

Here is the final listing for `index.html`:

```

1 <!DOCTYPE html>
2 <html lang='en'>
3 <head>
4   <title>Webcam</title>
5   <meta charset='UTF-8'>
6   <link rel='stylesheet' href='/stylesheets/style.css'></link>
7 </head>
8
9 <body>
10  <!-- step 1 -->
11  <div id='view-box'></div>
12  <!-- step 2 -->
13  <script src='/socket.io/socket.io.js' type='text/javascript'></script>
14  <script src='https://code.jquery.com/jquery-1.11.1.min.js'
type='text/javascript'></script>
15  <script src='javascripts/webcam_client.js' type='text/javascript'></script>
16 </body>
17 </html>

```

### Server - server/public/javascripts/webcam\_client.js

1. Connect to the server's `socket.io` socket.
2. Attach a listener to the socket to act on the 'image' message.

3. Place the new image in the `view-box` div in `index.html` if there is no image.
4. Replace the image in the `view-box` div in `index.html` with the new image.

Here is the final listing for `webcam_client.js`:

```

1 // step 1
2 var socket = io.connect();
3 // step 2
4 socket.on('image', function (data) {
5   if ($('#view-box').children().length == 0) {
6     // step 3
7     $('#view-box').prepend('<img src=' + data.image_location + ' />');
8   } else {
9     // step 4
10    $('#view-box').find('img').attr('src', data.image_location);
11  }
12 });

```

### Server - `server/lib/webcam_server.js`

1. Import the `socket.io` library.
2. Export the `listen` function to make it available to `server.js`. We need to start a server to listen for incoming connections.
3. Listen for incoming connections.
4. Attach a listener to act on any new connections.
5. Attach a listener to any socket connection to act on the `'new_image'` message. The action is to broadcast the new image received in the `'new_image'` message to all clients.

Here is the complete listing for `webcam_server.js`:

```

1 // step 1
2 var socket = require('socket.io');
3 var io;
4 // step 2
5 exports.listen = function(server) {
6   // step 3
7   io = socket.listen(server);
8   // step 4
9   io.sockets.on('connection', function(socket) {
10    // step 5
11    socket.on('new_image', function(img_location) {
12      socket.broadcast.emit('image', { image_location : '/images/img.png' });
13    });
14  });
15 };

```

### Server - `server/package.json`

1. Set a dependency for `socket.io`.

Here is the complete listing for `package.json`:

```
1 {
2   "name": "webcam",
3   "version": "1.0.0",
4   "description": "Webcam",
5   "dependencies": {
6     // step 1
7     "socket.io": "~0.9.6",
8     "mime": "~1.2.7"
9   }
10 }
```

We have now covered all the important steps to get the image capture and server to work. The remaining source code files for `server/public/stylesheets/style.css` and `server/server.js` can be found in Appendix A.

In order to run the system we have created, open up to terminals and navigate to the `server/` directory in one terminal window and to the `webcam/` directory in the other. In the `server/` directory run the command: `#nodejs server.js`. In the `webcam/` directory run the command: `#python webcam.py`.

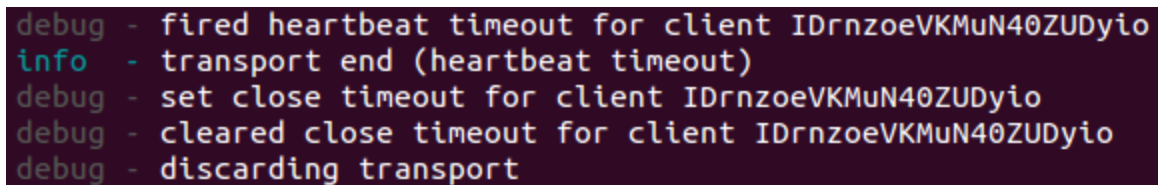
## Troubleshooting Tips

The following section goes over some problems that you may encounter during development. Quick and dirty solutions will be provided; they are not good, but will suffice for our purposes.

### Client Heartbeat Timeout:

You may notice that the client times out when it fails to send a heartbeat message to the server. The server will display message similar to the one in Figure 1 if the client times out. When this happens, the client will stop receiving any `socket.io` messages from the server. A quick fix for this is to set the heartbeat interval and heartbeat timeout to really high values in `webcam_server.js` as shown in the following code.

```
...
io = socket.listen(server);
io.set('heartbeat interval', 2000000);
io.set('heartbeat timeout', 2000000);
...
```



```
debug - fired heartbeat timeout for client IDrnzoeVKMuN40ZUDyio
info - transport end (heartbeat timeout)
debug - set close timeout for client IDrnzoeVKMuN40ZUDyio
debug - cleared close timeout for client IDrnzoeVKMuN40ZUDyio
debug - discarding transport
```

Figure 1 - Client heartbeat timeout

## New Images Fail to Load on the Web Interface

Since all images are saved to the same location, the web interface might not reload the image when a new one is available because it is using a cached copy. You can append a query string to the end of the image filename so the browser will be forced to retrieve the image instead of relying on its cache. The following change would be required in `webcam_client.js`.

```
...
} else {
  $('#view-box').find('img').attr('src', data.image_location + '?timestamp=' +
new Date().getTime());
}
...
```

However, this would create a problem because a file with the appended query string does not actually exist on the server. The problem would be as seen in figure 2. The quick fix would be to ignore the query string from all client requests. The following change be required in `server.js`.

```
...
var url = require('url');
...
filePath = 'public' + request.url; // original
...
filePath = 'public' + url.parse(request.url).pathname; // fixed
...
```

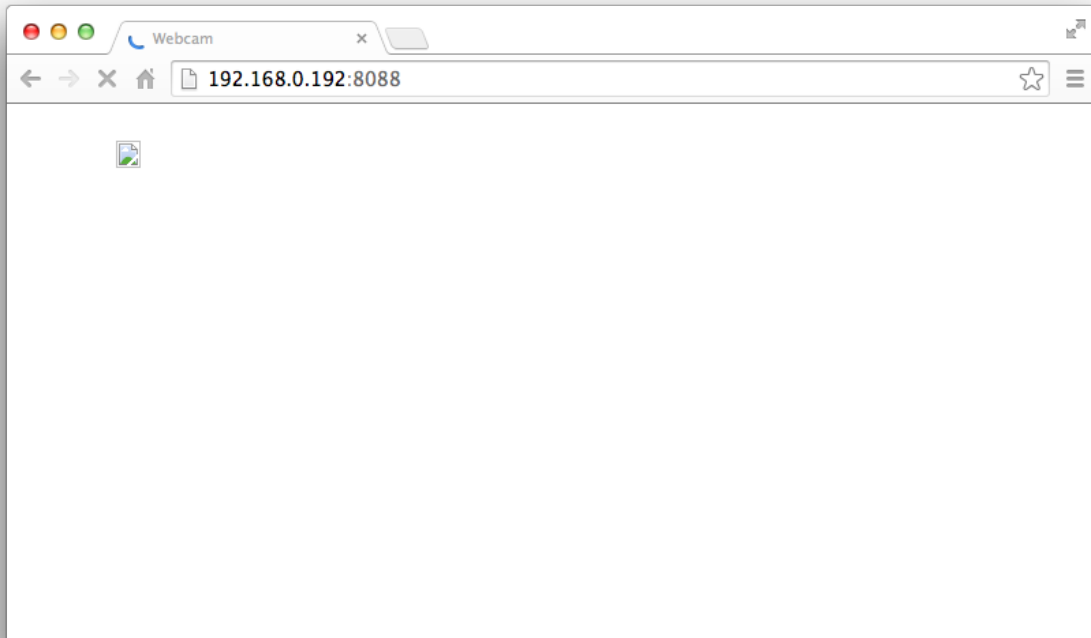


Figure 2 - Browser unable to load new image

## Appendix A - Complete Source Code Listing

style.css

```
1 body, html {
2     width: 800px;
3     margin-top: 10pt;
4     margin-left: auto;
5     margin-right: auto;
6     padding: 0;
7 }
8
9 img {
10    max-width: 100%;
11    max-height: 100%;
12 }
13
14 #view-box {
15    width: 640px;
16    height: 360px;
17    margin-left: auto;
18    margin-right: auto;
19 }
```

```
server.js
1 var PORT_NUMBER = 8088;
2 var http = require('http');
3 var fs = require('fs');
4 var path = require('path');
5 var mime = require('mime');
6 var url = require('url');
7
8 var server = http.createServer(function(request, response) {
9   var filePath = false;
10  if (request.url == '/') {
11    filePath = 'public/index.html';
12  } else {
13    filePath = 'public' + url.parse(request.url).pathname;
14  }
15  var absPath = './' + filePath;
16  serveStatic(response, absPath);
17 });
18
19 server.listen(PORT_NUMBER, function() {
20   console.log("Server listening on port " + PORT_NUMBER);
21 });
22
23 function serveStatic(response, absPath) {
24   fs.exists(absPath, function(exists) {
25     if (exists) {
26       fs.readFile(absPath, function(err, data) {
27         if (err) {
28           send404(response);
29         } else {
30           sendFile(response, absPath, data);
31         }
32       });
33     } else {
34       send404(response);
35     }
36   });
37 }
38
39 function send404(response) {
40   response.writeHead(404, {'Content-Type': 'text/plain'});
41   response.write('Error 404: resource not found. ');
42   response.end();
43 }
44
45 function sendFile(response, filePath, fileContents) {
```



```
46 response.writeHead(  
47     200,  
48     {"content-type": mime.lookup(path.basename(filePath))}  
49 );  
50 response.end(fileContents);  
51 }  
52  
53 var webcamServer = require('./lib/webcam_server');  
54 webcamServer.listen(server);
```