Embedded Systems
Rishav De and Pranet Sharma

Technical Report

The Project:

The project involves creating an automatic fire detection system that can be deployed wirelessly. The concept uses the BeagleBone Black V4L2 and OpenCV libraries to take images using a Logitech Brio and send them to an ML model hosted on a Python backend to predict fires. The images are sent over the internet using curl by connecting the BeagleBone to an Ethernet cable and using a public IP tunneled to the local host using ngrok which runs the backend. The curl response is then sent back to the BeagleBone which uses a traffic light kernel module to show the risk of fire.

Aspects of the Project:

On the BeagleBone we have:

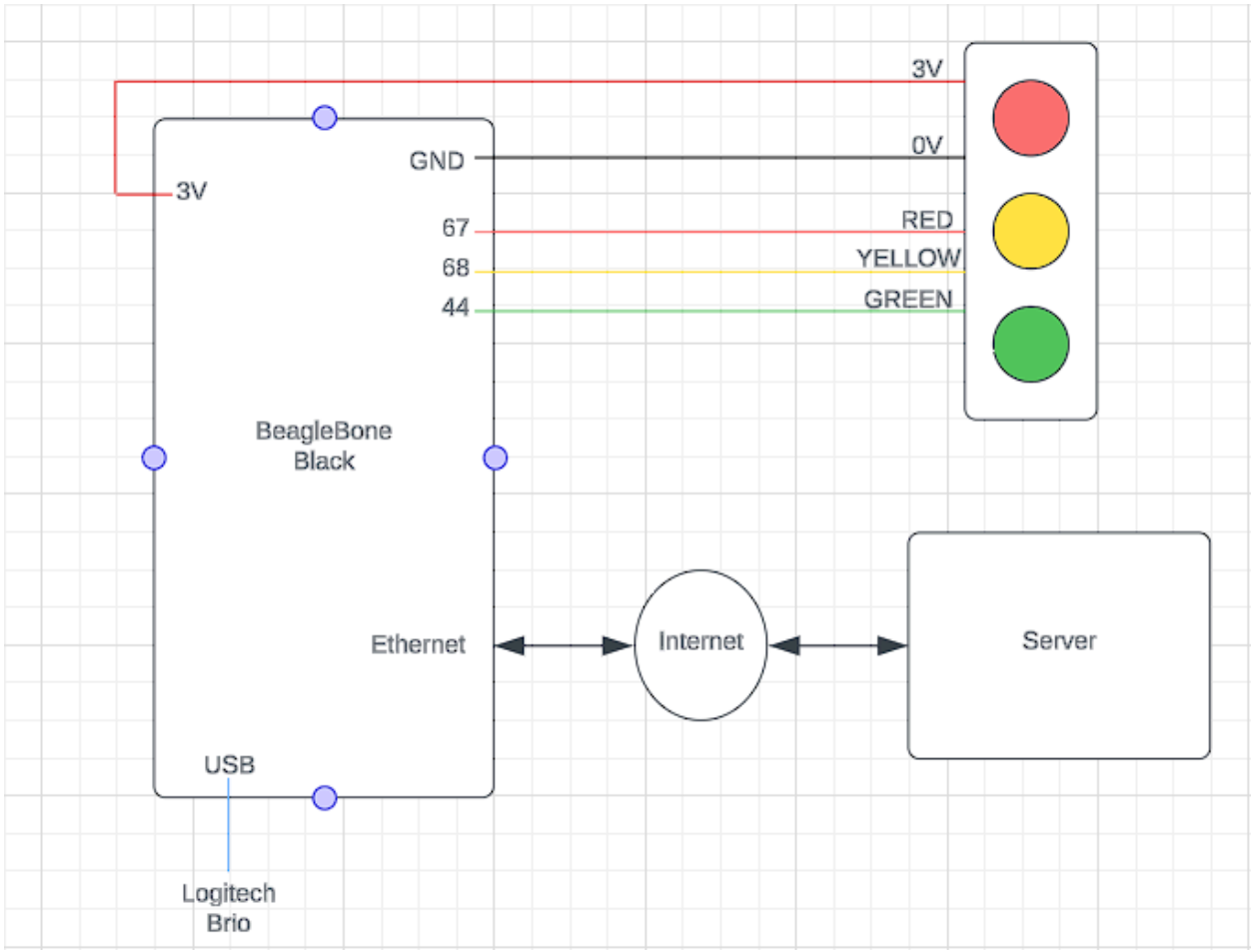
- A kernel module for the traffic light is used to show the risk of fire. This is an additional aspect that we implemented that was not mentioned in the original proposal. We did this as we felt like a visual demonstration of the end result on the BeagleBone would add complexity and be appealing.
- A .cpp userspace code file that uses OpenCV to capture an image using the camera in mpeg format.
- A shell script that runs the .cpp file and uses curl to send the image to the ngrok server every 30 seconds.
- Makefiles for the .cpp userspace file and the kernel module.

On the server side, we have:

- A Flask application that receives an image, passes it through the ML model and sends the prediction back to the BeagleBone as part of the response.

An ethernet cable was used to connect the BeagleBone as it did not have a WiFi module which we planned to use initially,

Schematic:



Technical Details/Setup:

On the server side, the Flask server needs to be run, and ngrok needs to be used to tunnel the localhost to a public IP address which is used later. The server gets the images sent by the BeagleBone, sends them through the model, and sends a response back of its prediction (of fire or not fire).

On the BeagleBone, the setup starts with running make on the kernel module and user space image capture program. Then, a UPLOAD_URL environment variable needs to be set that stores the URL of the ngrok server, and the kernel module needs to be instantiated using mknod and insmod. After all that is set, the shell script is ready to be run. The shell script will start sending images to the server at an interval of 30 seconds, and it does this by running the image capture program and using curl to send a POST request to the server. The script will also interface with the kernel module and control the traffic light. The Logitech Brio should be connected via USB and the BeagleBone should be connected to the internet via Ethernet.

Technical Setbacks:

We had numerous setbacks during the project which ate into our time. Firstly, the original camera we had tried did not interact with the installed version of OpenCV on the default Debian image. This was because the camera did not use a commonly used video format, and the pre-installed OpenCV version was old. Thus, we tried using the microSD card image that we had made for a lab. However, that did not work since v4l2-utils, a library required for camera setup, and OpenCV was not installed by default. Thus, we tried building the OpenCV libraries from source code as recommended by various forums. However, we ran into memory issues due to the size of the library and the partitions that the SD card had from the lab. So, we spent time trying to properly partition the microSD card and were eventually successful in doing so. However, there were many hiccups including us accidentally corrupting the SD card and having to figure out a way to reformat it.

Eventually, we were able to make the microSD card partitions big enough such that we could make the OpenCV library. However, the make was taking very long and had not finished even after 12 hours of letting it run. After spending a few more days trying to debug these issues regarding memory and OpenCV we gave up on that front. We eventually bought a camera that was able to interface with the OpenCV version that existed in the default Debian image.

After that, we had issues with making the local server using Flask visible to the internet, and after some time we found out about ngrok. Issues also popped up regarding sending the images to the server. We could not use the curl libraries in C++ due to some issue with installations and we spent some time trying to figure out the root cause. Eventually, we decided to ditch the idea of a program to send the images using curl and used a shell script and the curl command instead.

Lastly, we had memory issues again when we tried cloning our repository onto the default Debian distribution but ran out of memory as we mistakenly cloned the image data we had used for training our model. However, even after removing the local repository we still had memory issues (presumably due to the git history and commit logs) and we spent some time trying to use the microSD card (which did not work) and eventually spent time removing unwanted libraries and freeing enough space for the code we needed on the BeagleBone. During this time, our Serial connector also got damaged and we had to spend time fixing it so we could connect to the BeagleBone.

Technical Skills Learned:

Through the course of the project, the following technical skills were applied and learned:

1. Kernel Module Development: Developing a kernel module for controlling the traffic light on the BeagleBone involved understanding kernel programming concepts, device drivers, and low-level system interactions.

2. Image Processing with OpenCV: Troubleshooting issues with camera compatibility and OpenCV versions helped develop skills in image processing and computer vision. Building OpenCV from source enhanced our understanding of library compilation and dependencies.
3. Web Development with Flask: Building a Flask application for receiving and processing images involved setting up routes, handling API requests, and integrating with external components.
4. Networking and Internet Connectivity: Configuring network connections, dealing with Ethernet setups, and utilizing tools like ngrok for creating secure tunnels exposed us to networking concepts and embedded communications.
5. Shell Scripting: Creating a shell script to automate the image capture and upload process helped us develop proficiency in shell scripting and allowed us to automate many tasks.
6. Debugging and Troubleshooting: Dealing with various setbacks and technical challenges required debugging and troubleshooting skills. Identifying the root causes of issues and finding effective solutions were skills that we improved through this project.