CMPT 433
Taking a picture, sending it over TCP and using AWS Rekognition to a detect human
By Adam Atbi, Daven Chohan, Hong Quang Cung

1. USB Camera

This provides basic instructions on how to use a USB camera with your Beaglebone in C.

Step 1: Install these libraries on both the host and target

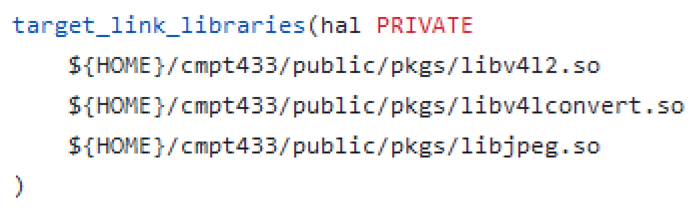
- They are required on the host for the IDE to not complain which can make coding easier
- They are required on the target to be utilized by the Beaglebone
    - sudo apt-get install libv4l-dev
    - sudo apt-get install v4l-utils

Step 2: Move library files to a shared location

- Have a shared folder between the target and the host (via NFS). Here is the NFS guide: https://opencoursehub.cs.sfu.ca/bfraser/grav-cms/cmpt433/guides/files/NFSGuide.pdf
- Copy the following files to the shared folder (located in **/usr/lib/arm-linux-gnueabihf**)
    - libv4lconvert.so
    - libjpeg.so
    - libv4l2.so

Step 3: Copy picture taking code

- Clone the grabber.c file (credit to Derek Molloy)
    - https://github.com/derekmolloy/boneCV/blob/master/grabber.c
- Add the code to an existing file or make grabber.c an executable using cmake
- In CMakeLists.txt file and make sure you link the libraries you copied in Step 2.



```
target_link_libraries(hal PRIVATE
    ${HOME}/cmpt433/public/pkgs/libv4l2.so
    ${HOME}/cmpt433/public/pkgs/libv4lconvert.so
    ${HOME}/cmpt433/public/pkgs/libjpeg.so
)
```

Step 4: Connect USB Camera

- Plug in your USB camera to the Beaglebone
- Ensure the Beaglebone recognizes the camera by running **lsusb**

Step 5: Run your executable

- Your Beaglebone needs write permissions to create the image file and save it
- The image is saved wherever your executable was run from

- The image is saved as a .ppm file

Troubleshooting

- *v4l2 library not being found:*
    - Make sure you provide the correct file path to the .so files in Step 3

- *Resolution is unexpected:*
    - Edit the resolution within the code to your expected resolution
        - fmt.fmt.pix.width = WIDTH_VALUE;
        - fmt.fmt.pix.height = HEIGHT_VALUE;

2. AI human detection

There are many ways to set up for AWS Rekognition service, this instruction provides the most convenient approach which allows you directly work with an image in local storage (in Host).

Step 1: Setup AWS Rekognition

- Register an AWS account (select region: ‘us-east-1’, or choose your region)
- Setup IAM Policy:
    - Open AWS Console
    - In “IAM”, select “Users” (left side panel)
    - Click on “Create User” to create new user
    - Provide name for “User”
    - You might need to grant “programmatic access” - follow the link: https://docs.aws.amazon.com/rekognition/latest/dg/sdk-programmatic-access.html#programmatic-access-general
    - In the “Permission Options”, for simplicity, select “Attach policies directly”
    - In “Attach policies directly” select “AmazonRekognitionFullAccess”, then create
- Setup Access key:
    - Open the “User” is created in the above section
    - Click on “Create access key”
    - Under “Access key best practices & alternatives”, select “Application running outside AWS”
    - Click “Create access key” to complete the process
    - Save the “Access key” and “Secret access key” - later use to access AWS Rekognition

Step 2: Install Dependencies on Host (Linux - debian 11)

- Install python and pip on host machine using below command

```
(host) ~$ sudo apt update
(host) ~$ sudo apt install python3
(host) ~$ sudo apt install python3-pip
```

- Install boto3 package

```
(host) ~$ sudo pip3 install boto3
```

Step 3: Setup Python Server - TCP Protocol

- Libraries requires: “boto3”, “os”
- Use “os” library to get environment variable at run-time
- Provide “Access key” and “Secret Key” at run-time (AWS & Git does not allow you to expose these information publicly)

Example:

```
AWS_Access_key:fake_access_key AWS_Secret_Key:fake_secret_key python3
humandetection.py
```

- In server code, include this snippet to run AWS Rekognition:

```
AWS_ID = os.environ.get('AWS_ID')
AWS_ACCESS = os.environ.get('AWS_ACCESS')
client = boto3.client('rekognition',
    aws_access_key_id=AWS_ACCESS_KEY,
    aws_secret_access_key=AWS_SECRET_ACCESS,
    region_name='us-east-1')
```

Step 4: Analyze image with AWS Rekognition

- The response from AWS Rekognition is in JSON format - access via field such as “Labels”, “Names”, or “Person”
- Understand bounding box: https://docs.aws.amazon.com/rekognition/latest/dg/images-displaying-bounding-boxes.html

Step 5: Recommend workflow for transmitting an image via TCP Protocol between Client (in C) and Server (in Python)

- Server (Python)
  - Establish a server (in Python) to listen for incoming connections - set limit number of connection as you need
  - Create a separate thread to handle the request concurrently
  - Initially, receive the metadata of the image/video, including the file type and size
  - Confirm with the client before proceeding with data processing
  - Receive all data pertaining to the file from the client
  - Reconstruct the file, store in local storage then send confirmation
  - Call APIs to detect human then send the results
  - Join the thread after connection closure
- Client (C):
  - Send the metadata, two most important things: file type (for reconstruct the file) and file size (expect how many bytes to receive)
  - Wait for server to response before transmitting actual data
  - Segment the image/video data into chunks of data, typically 1024 bytes each
  - Wait for confirmation of successful receiving data

    - Listening to server’s response for the result of human detection
    - Close connection

Troubleshooting

- *Unable to call AWS Rekognition APIs:*
    - Ensure that the "Access Key" and "Secret Key" are provided at runtime.
    - Check the region

- *The file corrupted or server cannot reconstruct the file:*
    - Forcing Clients to Wait for Confirmation: Slows down the process of sending and receiving images.
    - Verifying the Buffer to Prevent Overwrite: Ensures that the client doesn't send data too quickly, potentially overwriting the server’s buffer and causing data loss.
    - Sending Metadata Before Data Transmission: Ensures that the metadata is accurate and transferred before sending the actual data.
    - Debugging by Printing Content: This is an efficient debugging method.
    - Comparing Image Sizes on Server and Client Sides: This is a good approach to detecting errors.

References

- Previous CMPT 433 Webcam guide
  https://opencoursehub.cs.sfu.ca/bfraser/grav-cms/cmpt433/links/files/2016-student-howtos/WebcamJPEGtoWeb.pdf
- USB camera code:
  https://github.com/derekmolloy/boneCV
- Developer Guide:
  https://docs.aws.amazon.com/pdfs/rekognition/latest/dg/rekognition-dg.pdf#what-is
- Analyze images in Amazon S3 bucket:
  https://docs.aws.amazon.com/rekognition/latest/dg/images-s3.html