How to Perform Object Detection using OpenCV on the BYAI

Group Name: Purr-hibition
Group Members: Jennifer Kim, Raymond Kong, Mathew Tse, Alvin Tsang
Last Updated: April 7, 2025

Guide has be tested on:

BeagleY-AI (target):	Debian 12.8
PC OS (host):	Debian 12.8

This document guides the user through

1. Setting up the camera on the BYAI
2. Object detection on the BYAI

Table of Contents

Formatting

1. Commands for the host Linux's console are shown as:
   `(host)$ echo "Hello PC world!"`
2. Commands for the target (BeagleY-AI) Linux's console are shown as:
   `(byai)$ echo "Hello embedded world!"`
3. All commands are case-sensitive.

Hardware Setup

1. **USB camera**
   - Before plugging the USB camera in, you should expect
     ```
     (byai)$ ls /dev/video*
     ls: cannot access '/dev/video*': No such file or directory
     ```
   - Plug in the USB camera into the BYAI USB port
2. **Camera information**
   - After plugging the USB camera into the BYAI, you might expect something like this
     ```
     (byai)$ ls /dev/video*
     /dev/video0
     /dev/video1
     /dev/video2
     /dev/video3
     ```
3. **Troubleshooting**
   - If you don't see a device when running `ls /dev/video*`, there may be an issue with the camera or the USB port itself. Try a different USB port to see if it works. You may also find.
   - If the camera is still not found, try
     ```
     (byai)$ dmesg
     ```
     or
     ```
     (byai)$ dmesg | grep usb
     ```
     If you don't see your camera, then the BYAI is not detecting your camera. Consider a different camera of a different port.

Software Setup

1. **Host setup**
   - On the host, install the following:
     ```
     (host)$ sudo apt update
     (host)$ sudo apt install -y ffmpeg
     ```
2. **Target Setup**
   - On the target, install the following:
     ```
     (byai)$ sudo apt update
     (byai)$ sudo apt install -y ffmpeg \
                                 v4l-utils \
                                 xvfb \
                                 python3 \
                                 python3-pip
     ```
3. **Optional step: Python virtual environment**
   - Install a virtual environment on the target. We will use Conda, but any Python virtual environment should do. It is recommended to use a virtual environment so that package installations and development are nicely separated from the global Python configuration.
     ```
     (byai)$ cd ~
     ```

```
(byai)$ sudo apt update
(byai)$ sudo apt install wget
(byai)$ wget
https://github.com/conda-forge/miniforge/releases/download/2
4.3.0-0/Mambaforge-24.3.0-0-Linux-aarch64.sh
(byai)$ bash Mambaforge-24.3.0-0-Linux-aarch64.sh -b
(byai)$ source ~/mambaforge/etc/profile.d/conda.sh
(byai)$ conda init bash
(byai)$ rm Mambaforge-24.3.0-0-Linux-aarch64.sh
```

4. **Install the Python packages**
   - If you are using a Python virtual environment, make sure to create and activate it
     ```
     (byai)$ conda create --name <CONDA_ENV_NAME> python=3.11 -y
     (byai)$ conda activate <CONDA_ENV_NAME>
     ```
   - Install Python dependencies
     ```
     (byai)$ pip install numpy==1.26.4 \
                         opencv-python-headless
     ```

USB Camera with OpenCV

1. **Plug the camera into a USB port**
2. **Identify the camera**
   ```
   (byai)$ ls /dev/video*
   video0
   video1
   video2
   video0
   ```
3. **Open the camera with OpenCV**
   - The following code should be run on the target
   - Import Numpy and OpenCV
     ```
     import numpy as np
     import cv2
     ```
   - Identify the device path and start capturing the frames
     ```
     cap = cv2.VideoCapture("/dev/video3")

     while True:
          ret, frame = cap.read()
          if not ret:
               print("Failed to capture a frame!")
               continue
     ```

4. **Troubleshooting**
    - If you don't see a device when running `ls /dev/video*`, there may be an issue with the camera or the USB port itself. Try a different one to see if it works
    - When identifying the device path, you might have to try different `video*` options

Object Detection With USB Camera

This section is heavily based on a guide produced by Tauseef Ahmad.

1. **Download the pre-trained model**
    - For a lightweight, pre-trained object detection model, we will use Mobilenet SSD, a pre-trained convolutional architecture for fast feature embedding (CAFFE) deep learning framework designed for mobile and embedded devices.
    - In your project, create a directory called `MobileNetSSD/` and copy the following file contents into the directory:
        - MobileNetSSD_deploy.prototxt: Model definition
        - MobileNetSSD_deploy.caffemodel: Pre-trained model weights

2. **Initialize Model**
    - At the top of the object detection Python file, import NumPy and OpenCV

```
import numpy as np
import cv2
```

    - Read the pre-trained network model

```
prototxt = "MobileNetSSD/MobileNetSSD_deploy.prototxt"
model = "MobileNetSSD/MobileNetSSD_deploy.caffemodel"
net = cv2.dnn.readNetFromCaffe(prototxt, model)
```

    - Out of the box, MobileNetSSD can detect 21 different objects. We save this in a map

```
class_names = {
  0: "background",
  1: "aeroplane",
  2: "bicycle",
  3: "bird",
  4: "boat",
  5: "bottle",
  6: "bus",
  7: "car",
  8: "cat",
  9: "chair",
  10: "cow",
  11: "diningtable",
  12: "dog",
```

```
  13: "horse",
  14: "motorbike",
  15: "person",
  16: "pottedplant",
  17: "sheep",
  18: "sofa",
  19: "train",
  20: "tvmonitor"
}
```

3. **Add the camera**
    - Using the USB camera, we can use the captured frame and attempt to detect any of the 21 objects

```
cap = cv2.VideoCapture("/dev/video3")

while True:
      ret, frame = cap.read()
      if not ret:
            print(“Failed to capture a frame!”)
            continue

      # construct a blob from the image
      blob = cv2.dnn.blobFromImage(
            frame,
            scalefactor = 1/127.5,
            size = (300, 300),
            mean = (127.5, 127.5, 127.5),
            swapRB=True,
            crop=False
      )

      # blob object is passed as input to the object
      net.setInput(blob)

      # network prediction
      detections = net.forward()
```

4. **Object detection**
    - The forward pass through the MobileNetSSD network returns a collection of detections. Each detection has the following format:
        - `image_id`: ID of the image batch
        - `label`: Predicted class ID. Use `class_names` to find its string representation
        - `conf`: Confidence score of the predicted class
        - `x_min`: x-coordinate of the top left bounding box corner

- `y_min`: y-coordinate of the top left bounding box corner
- `x_max`: x-coordinate of the bottom right bounding box corner
- `y_max`: y-coordinate of the bottom right bounding box corner

```
    for i in range(detections.shape[2]):
         conf = detections[0, 0, i, 2]

         # set confidence level threshold
         if conf > 0.5:
              class_id = int(detections[0, 0, i, 1])

          if class_id in classNames:
              print(f"Object {classNames[class_id]}
              detected with {conf:.2f}% confidence")
```

5. Troubleshooting

- The overall performance with operating the camera, object detection and live streaming was considerably poor. We took the following measures to improve the overall performance
  - Asynchrony/Concurrency: We used the `asyncio` library to operate the camera, object detection and live streaming service as independent tasks.
  - Batch processing: Instead of performing object detection one frame at a time, we are processing a batch of frames captured by the camera asynchronously. So, for a camera capturing 30 frames per second, we are invoking the model once per second rather than 30 times per second
  - Frame skipping: By skipping every nth frame captured by the camera, we are decreasing the number of frames that need to be processed by the object detector. However, this may impact the overall video quality. We found a good balance with skipping every 4th frame.
- Since we are running the model directly on the board, there is no way to visually see through our camera. Consider streaming the captured frame back to the host machine. See this for reference