



SOFE 4590U: Embedded Systems

Fall 2021

Assignment 2 : Performance evaluation of different architectures on QEMU

Owais Quadri

100697281

Due Date: Nov 11, 2021

Google Drive Video Link:

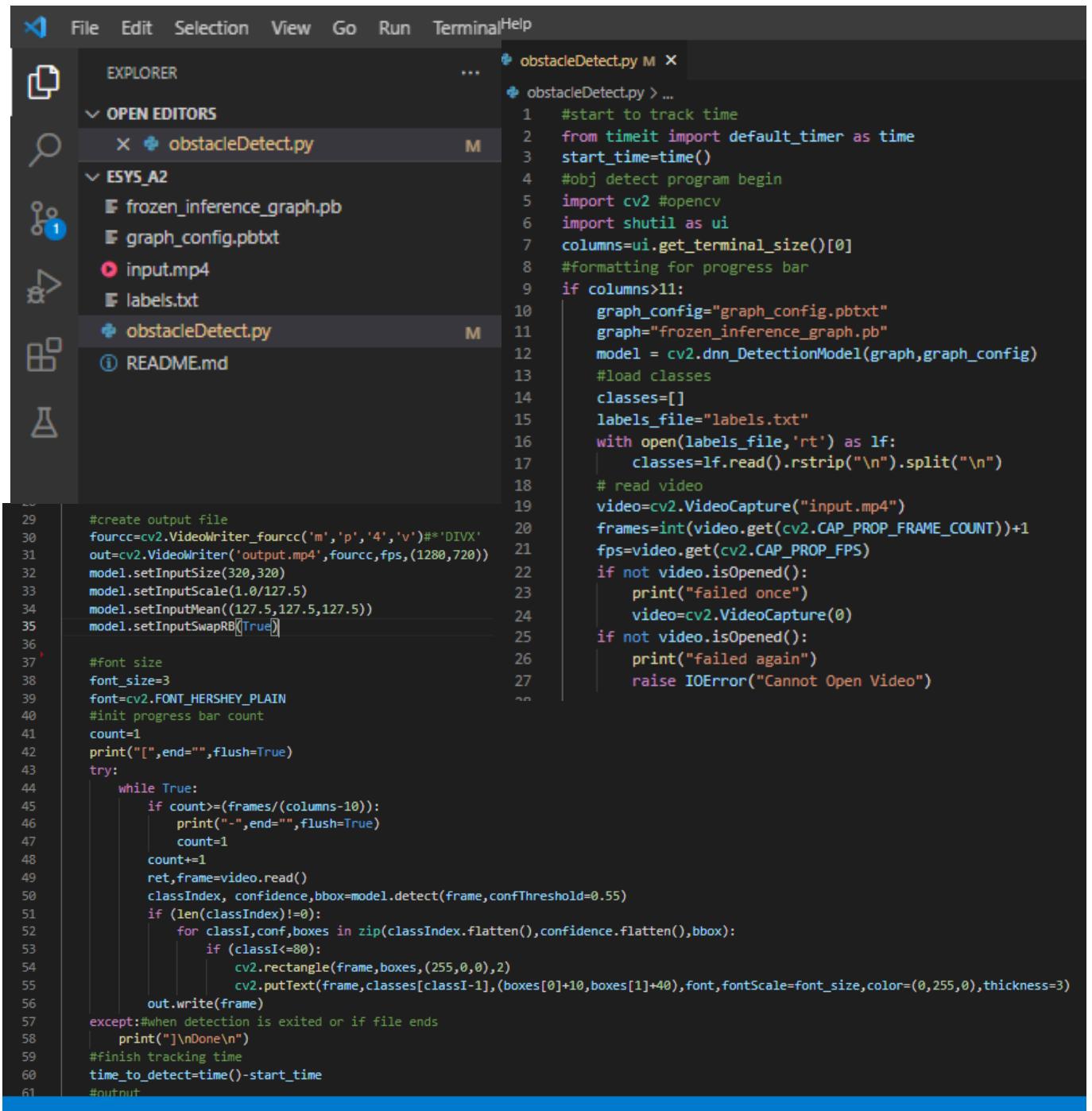
https://drive.google.com/file/d/1flxSsxy0OFSN61zIBJGbnWo7Bw_JMa3F/view?usp=sharing

Introduction

Image processing applications can often have timing constraints. For example, consider the computer vision system of an autonomous vehicle when an object is in the path of the moving vehicle, we require that objects to be detected. The goal therefore of this assignment is to develop a new or reuse an existing (please refer to the source) computer vision application using any programming language and run on QEMU for two different architectures to analyze the time difference it takes for all the objects detected between two different architectures. You can either generate videos on your own or obtain them from online sources (please refer to the source).

Deliverables:

1. Code Overview



The screenshot shows a code editor interface with the following details:

- File Menu:** File, Edit, Selection, View, Go, Run, Terminal, Help.
- Explorer:** Shows the project structure:
 - OPEN EDITORS: obstacleDetect.py (marked with a 'M' icon)
 - ESYS_A2 folder:
 - frozen_inference_graph.pb
 - graph_config.pbtxt
 - input.mp4
 - labels.txt
 - obstacleDetect.py (marked with a 'M' icon)
 - README.md
- Code Editor:** Displays the content of `obstacleDetect.py`. The code is a Python script for video processing using OpenCV and a pre-trained model. It includes imports for `timeit`, `cv2`, and `shutil`, defines a progress bar, and processes frames from an input video to detect objects using a frozen inference graph and a specific configuration file.

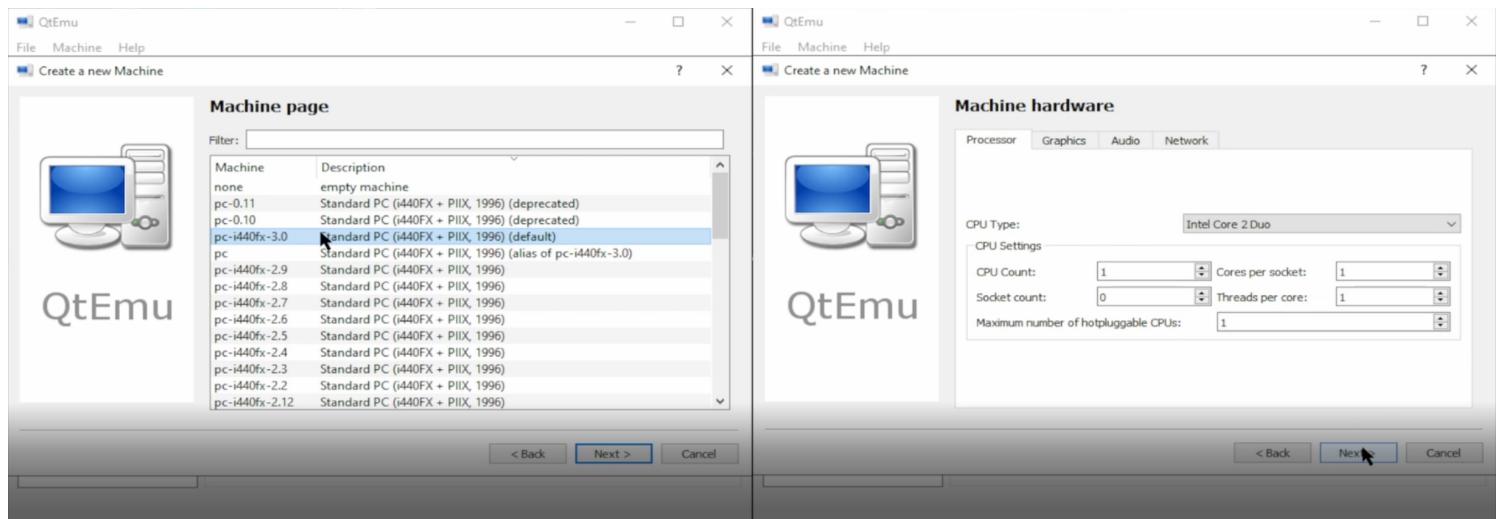
```
#start to track time
from timeit import default_timer as time
start_time=time()
#obj detect program begin
import cv2 #opencv
import shutil as ui
columns=ui.get_terminal_size()[0]
#formatting for progress bar
if columns>11:
    graph_config="graph_config.pbtxt"
    graph="frozen_inference_graph.pb"
    model = cv2.dnn_DetectionModel(graph,graph_config)
    #load classes
    classes=[]
    labels_file="labels.txt"
    with open(labels_file,'rt') as lf:
        classes=lf.read().rstrip("\n").split("\n")
    # read video
    video=cv2.VideoCapture("input.mp4")
    frames=int(video.get(cv2.CAP_PROP_FRAME_COUNT))+1
    fps=video.get(cv2.CAP_PROP_FPS)
    if not video.isOpened():
        print("failed once")
        video=cv2.VideoCapture(0)
    if not video.isOpened():
        print("failed again")
        raise IOError("Cannot Open Video")
#create output file
fourcc=cv2.VideoWriter_fourcc('m','p','4','v')##'DIVX'
out=cv2.VideoWriter('output.mp4',fourcc,fps,(1280,720))
model.setInputSize(320,320)
model.setInputScale(1.0/127.5)
model.setInputMean((127.5,127.5,127.5))
model.setInputSwapRB[True]
#font size
font_size=3
font=cv2.FONT_HERSHEY_PLAIN
#init progress bar count
count=1
print("[",end="",flush=True)
try:
    while True:
        if count>=(frames/(columns-10)):
            print("-",end="",flush=True)
            count=1
        count+=1
        ret,frame=video.read()
        classIndex, confidence,bbox=model.detect(frame,confThreshold=0.55)
        if (len(classIndex)!=0):
            for classI,conf,boxes in zip(classIndex.flatten(),confidence.flatten(),bbox):
                if (classI<=80):
                    cv2.rectangle(frame,boxes,(255,0,0),2)
                    cv2.putText(frame,classes[classI-1],(boxes[0]+10,boxes[1]+40),font,fontScale=font_size,color=(0,255,0),thickness=3)
        out.write(frame)
except:#when detection is exited or if file ends
    print("]\nDone\n")
#finish tracking time
time_to_detect=time()-start_time
#output
```

The code for obstacle detection was coded by me, after watching a guide on youtube for [opencv](#)[1]. I created the code before testing the code on the different architectures because I wanted to make sure that the code would run properly. I used the `timeit.default_timer` to measure the time that it took (performance) to process the video. I also included feedback to show how long that the processing should take (the progress line proceeds until near the end of the terminal window horizontally). I used `shutil` to find the terminal width and proceeded with that. Then, I loaded the [tensorflow model](#)[2] using the graph and [config files](#)[3] using opencv (cv2). After that, I loaded the [labels](#)[4] for the model, I loaded the `input.mp4`[5] [video](#) and while the video was being iterated through, each frame was getting its objects detected and rectangles drawn around each detected item with its corresponding labels. Then, the output video writer was being written to and when the loop is broken (video ends or is escaped by using ‘`Ctrl+C`’), the time for processing is calculated, the result is printed and the processed video is written to a file called ‘`output.mp4`’.

2. Snapshots of obstacle detection running on different architectures:

First Architecture: Machine = pc-i440fx-3.0 : PC(i440FX + PIIX, 1996) ,

CPU Type = Intel Core 2 Duo



Program running on first architecture:

- Input video screenshot:



- Terminal :

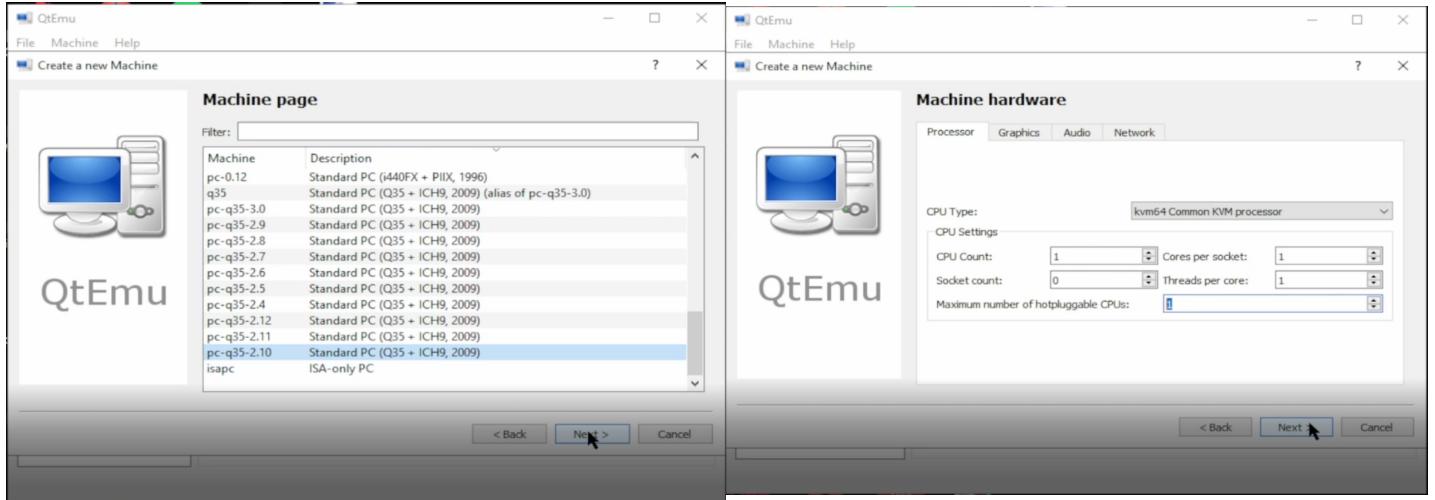
A screenshot of a terminal window on an Ubuntu system. The terminal shows the command `sudo apt-get install python3-pip` followed by `pip install opencv-python`. It then runs the script `python3 obstacleDetect.py` on an input video file. The terminal output includes package download progress bars and a message indicating the script was successfully executed. The final output shows the processed video saved as `output.mp4`.

- Output video screenshot:



Second Architecture: Machine = pc-q35-2.10 : PC(Q35 + ICH9, 2009) ,

CPU Type = kvm64 Common KVM processor



Program running on second architecture:

- Input video screenshot:



- Terminal :

```
ubuntu@ubuntu: ~/Documents/ESYS_A2$ pip install opencv-python
Collecting opencv-python
  Downloading opencv_python-4.5.4.58-cp38-cp38-manylinux2014_x86_64.whl (60.3
B)
Collecting numpy<1.17.3
  Downloading numpy-1.21.4-cp38-cp38-manylinux_2_12_x86_64.manylinux2010_x86_
64.whl (15.7 MB)
Installing collected packages: numpy, opencv-python
  WARNING: The scripts f2py, f2py3 and f2py3.8 are installed in '/home/ubuntu/
local/bin' which is not on PATH.
  Consider adding this directory to PATH or, if you prefer to suppress this w
arning, use --no-warn-script-location.
Successfully installed numpy-1.21.4 opencv-python-4.5.4.58
ubuntu@ubuntu:~/Documents/ESYS_A2$ python3 obstacleDetect.py
[-----]
Done

Time taken to process input.mp4: 120.2862414000002 s
Processed video saved in: output.mp4
ubuntu@ubuntu:~/Documents/ESYS_A2$
```

- Output video screenshot:



Explanation for differences:

In some cases, the Q35 is better than the i440FX , but for machine learning applications, this source indicates that the i440FX should perform [29.7%](#) better than the Q35 [6]. Meanwhile, when tested, the i440FX completed the processing for a 60 second video in 118.1880 seconds where the Q35 completed the processing on the same input video in 120.2286 seconds. This means that the i440FX ran only 1.73% faster than the Q35 machine.

Machine Learning	1115	860	129.7%
QEMU Standard PC (i440FX + PIIX, 1996)			
QEMU Standard PC (Q35 + ICH9, 2009)			

3. Link to Video:

https://drive.google.com/file/d/1flxSsxy0OFSN61zIBJGbnWo7Bw_JMa3F/view?usp=sharing

References

- [1] OpenCV Guide <https://www.youtube.com/watch?v=RFqvTmEFtOE>
- [2] Tensorflow model used:
<https://github.com/opencv/opencv/wiki/TensorFlow-Object-Detection-API>
- [3] Config file: <https://gist.github.com/dkurt/54a8e8b51beb3bd3f770b79e56927bd7>
- [4] Classes label list data: <https://github.com/pjreddie/darknet/blob/master/data/coco.names>
- [5] Video used: <https://www.youtube.com/watch?v=aMJNBUDk69Q> (2:23-3:23)
- [6] i440FX vs Q35 <https://browser.geekbench.com/v5/cpu/compare/4589842?baseline=4589992>