Politenico di Milano

Dipartimento Elettronica, Informazione e Bioingegneria

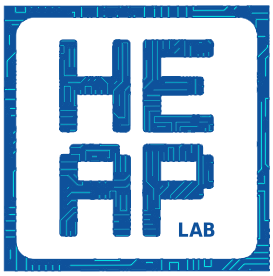
HEAPLab Project Report

OpenCV people detection algorithm for Barbeque RTRM

Author:
Giovanni Lupi

Supervisor:
Dr. Giusppe Massari

July 3, 2021



Abstract

The aim of this project is to port the sample "peopledetect" from the OpenCV library to the Barbeque Run-Time Resource Manager (BarbequeRTRM). OpenCV is a multiplaform library for real-time Computer Vision. Given a source video file, the peopledetect application provides an implementation for detecting people present in the scene. Porting this sample to BarbequeRTRM allows the application to run accordingly to the run-time managed Adaptive Execution Model. This allows in turn the application to adjust its performances according to the assigned computing resources.

1 Introduction

1.1 Goals

The goal of the present project is to port a sample application of the OpenCV library to the Barbeque Run-Time Resource Manager. In accordance with the project specifications, the "peopledetect" sample was chosen.

Peopledetect is an application that shows an implementation of an efficient method to detect humans in an image or a video. This implementation is called Histogram of Oriented Gradients (HOG) and will be described in the next few sections of the paper. Porting this sample to BarbequeRTRM allows the application to run accordingly to the run-time managed Adaptive Execution Model. This allows in turn the application to adjust its performances according to the assigned computing resources.

1.2 OpenCV

OpenCV (Open Source Computer Vision Library) is an open source library that includes hundreds of computer vision algorithms. It mainly focuses on image processing, video capture and analysis including features like face detection and object detection.

The library is also cross-platform, supporting different programming languages such as C++, Python and Java. The present project was developed in C++ using version 3.2 of the library.

1.3 People detection

Object detection is a computer vision technique that allows to identify and locate objects in an image or video. Specifically, object detection creates bounding boxes around the detected objects, allowing us to locate where they are or how they move through a given scene. Peopledetect is a sample of the OpenCV library that is specifically tailored to detect pedestrians. Therefore, the application will receive as input a video file featuring one or more pedestrians, and will generate one or more bounding boxes surronding the detected people. There are several approaches that can be used to perform object detection. Poepledetect employs a machine learning algorithm named Histogram of Oriented Gradients (HOG).

1.4 Histogram of Oriented Gradients

The Histogram of Oriented Gradients is a feature descriptor. This is an algorithm that produces a description of the visual features of the input image or video by extracting useful information and discarding the unnecessary information. A HOG descriptor converts an image into a feature vector of a certain length. In particular, the features employed are the distribution of directions of gradients.

The feature vector produced is then used as input for an image classification algorithm like Support Vector Machine (SVM). More details about the implementation of this algorithm will be provided in the implementation section.

Understanding the tunable parameters for the algorithm allows the adjustment of the application performances according to the Adaptive Execution Model.

1.5 Adaptive Execution Model

The Adaptive Execution Model drives the application through a managed execution flow, characterized by:

- Resource-awareness: the application can configure itself according to the assigned computing resources.
- Runtime performance monitoring and negotiation: the application can observe the current throughput and ask for more resources. The Peo-

pledetect application will therefore be able to adjust its performances according to the resources assigned by the BarbequeRTRM.

2 Installation and setup

The development platform chosen for this project is Ubuntu 18.04 (bionic) LTS.

BarbequeRTRM was installed according to the instuctions at https://bosp.deib.polimi.it/guide-user.html using $HOME/Work/BOSP as installation directory.

A couple of problems were found during installation, mostly due to the fact that the name of some packages had changed:

- ncurses-dev → libncurses5-dev
- premake → premake4
- git-core → git

BarbequeRTRM was then configured as explained in the guide and the system was rebooted.

As a starting point for the project, the sample https://github.com/HEAPLab/aem-template was used. The project uses CMake as buildsystem and the Qt Creator IDE was chosen as development environment since it has a good support for CMake.

To summarize:

- Operating System: Ubuntu 18.04
- IDE: QtCreator
- Compiler: gcc (Ubuntu 7.5.0-3ubuntu1 18.04) 7.5.0
- Library: OpenCV 3.2

3 Design and Implementation

The ported application provides the same functionalities as the original sample, while complying with the Adaptive Execution Model. In addition to the original functionalities, the new application is also capable of adjusting

its performances according to the available resources. This new feature was implemented by exploiting the tuning of some suitable parameters, which will be examined in greater detail later in this paragraph.

3.1 Original application

Peopledetect can use as video source a file or the computer's camera. This choice can be made by the user through the command line. Once the source has been chosen, the application runs the actual detection procedure.

The Detector class contains all the information required for the detection. The detect function performs the detection: it takes as input a frame from the source video and outputs a vector of rectangles that circumscribe the individuals. Detect is as a wrapper for another function, called detectMultiScale which can run in two different modes:

- Daimler mode → More accurate detection but computationally expensive
- Default mode → Less CPU intensive but less detection power

By default, the original Peopledetect application only uses the default mode algorithm, and the Daimler mode can only be selected manually. The execution flow is contained in the main function and can be summed up in this way:

```
initialize detection
while (there are frames)
{
    read a frame from the video
    detect people in the frame
    draw the frame with detections highlighted with bounding rectangles
}
```

3.2 Ported application

The original application included all the control flow in the main function; all the steps of the detection belonged to a "for" loop.

The main loop was therefore split in separate functions (callbacks for BarbequeRTRM) in a separate class called PeopleDetect (a subclass of bbque::rtlib::BbqueEXC):

- onSetup → set up video source and define Barbeque CPS goal
- onConfigure → get and display Barbeque configuration parameters
- onRun → process a single frame (detect and display)
- onMonitor → check the performance and adjust the detector parameters accordingly
- onSuspend → not used

The main() function now does the following:

```
parse command line arguments
set up Barbeque RTRM
instantiate the PeopleDetect class in Barbeque
start
```

3.3 Performance tuning

After finishing the porting, some additional effort was put into understanding how to tune the performance of the application. The high level idea was to set a target performance and, whenever below this threshold, allow the application to give up some of its detection accuracy to improve its speed.

This tradeoff between speed and accuracy is especially important when running the detector in real-time on resource constrained devices. This goal was achieved by analyzing and tweaking the parameters of the detectMultiScale function.

detectMultiScale has two key parameters that, if changed, can dramatically impact both the speed and the accuracy of the application. These parameters are:

- winStride → The winStride parameter is a 2-tuple that dictates the "step size" in both the x and y location of the sliding window. A sliding window is a rectangular region of fixed width and height that "slides" across an image. At each stop of the sliding window we (1) extract HOG features and (2) pass these features on to our Linear SVM for classification. The process of feature extraction and classifier decision is very computationally expensive. Clearly, the smaller the window size, the more windows will need to be evaluated, lowering dramatically the

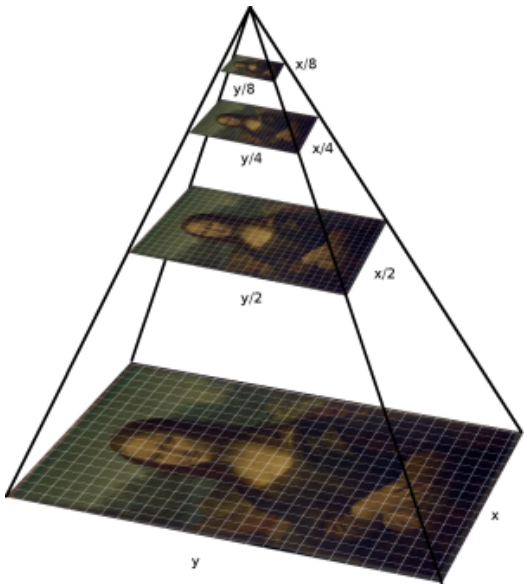


Figure 1: Pyramid example

speed of the algorithm. On the other hand, increasing the value of winStride allows to reduce the computational burden.

- scale → An image pyramid is a multi-scale representation of an image, as in Figure **??**.

  At each layer of the image pyramid the image is downsized and (optionally) smoothed via a Gaussian filter.

  The scale parameter controls the factor in which the image is resized at each layer of the image pyramid, ultimately influencing the number of levels in the image pyramid.

  A smaller scale will increase the number of layers in the image pyramid and increase the amount of time it takes to process the image. A larger scale will decrease the number of layers in the pyramid as well as decrease the amount of time it takes to detect objects in an image.

Therefore, performance optimization was implemented as follows: in the onSetup function a target CPS goal was set. In onMonitor, the resulting CPS was examined and:

```
every 16 calls to onMonitor:
```

```
if the CPS is substantially lower than the target CPS:
    switch from Daimler algorithm to default algorithm
    return
if the CPS is lower than the target CPS:
    increase the scale by 0.02
    return
OPTIONAL: increase the stride (not performed)
```

The numeric values were determined experimentally.

4 Experimental Results

To test the performance of the ported code, the video file `sample2.mp4` was used; the video shows people walking in a station. The video file was scaled from the original 1280x720 to a smaller 640x360, because the original resolution was too challenging for the available CPU resources.

A desired target CPS of 6 was determined empirically with some tests. When the application starts, the Daimler HOG is used and the initial CPS is

```
PeopleDetect::onMonitor(): CPS=1.13868
```

After some cycles, the detection switches to the Default HOG and the performance starts improving, at first without a noticeable loss of accuracy.

```
PeopleDetect::onMonitor(): CPS=2.21363
```

At this point, the detection starts increasing the scale by 0.02 each time until a maximum of 1.4 is reached. Beyond this point the Detector cannot detect people any more. A final

```
PeopleDetect::onMonitor(): CPS=4.65304
```

is reached, with a final performance which is 4 times the initial performance, at the cost of dropping from 14 detected people to only 3 or 4.

The other tunable parameter, stride, was not modified by onMonitor, because modifying both scale and stride at the same time led to a very fast loss of detection.

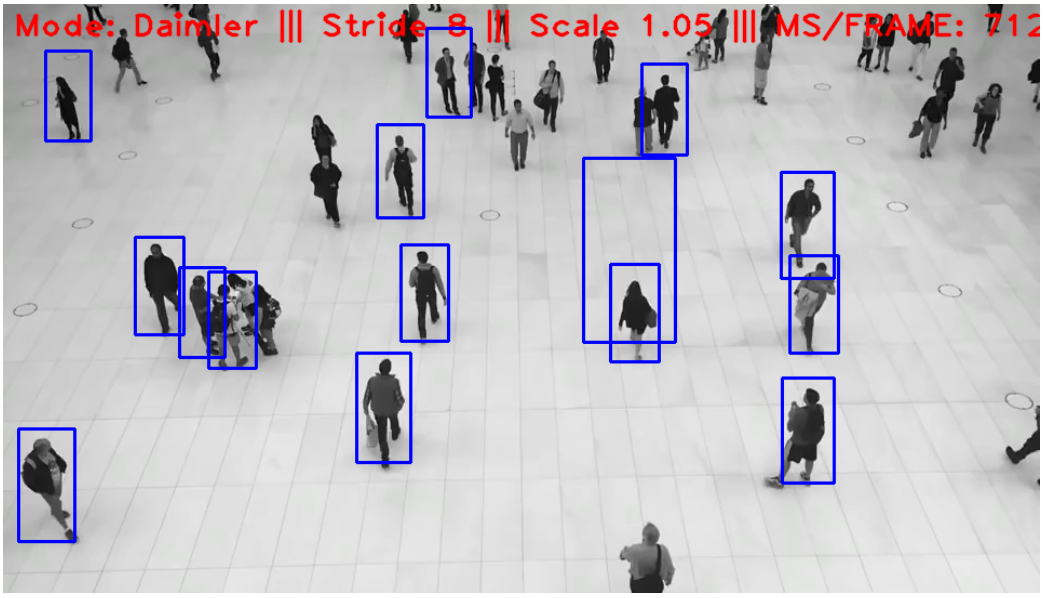


Figure 2: Daimler HOG with initial scale

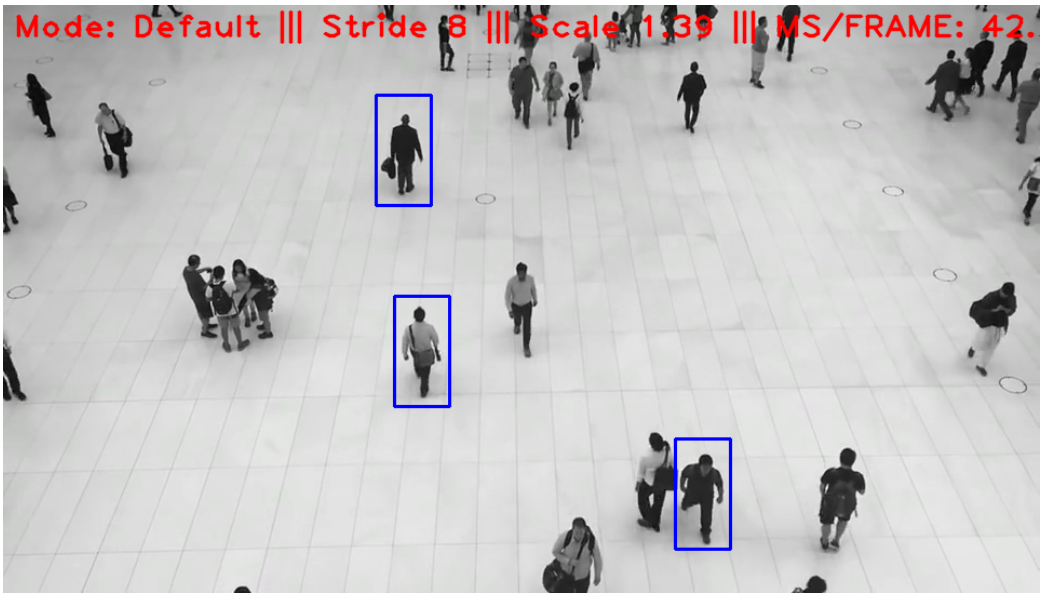


Figure 3: Default HOG with final scale

5 Conclusions

The original "peopledetect" sample consisted of a main loop which analyzed a video frame at a time in order to perform people detection. This code organization matched the application structure required by the BarbequeRTRM; the analysis of the video frame was moved to the onRun() function of the environment, while the preparatory steps became part of onSetup().

Optimizing the performance was more complex, mostly because the lack of documentation of HOG detector, and the effect of changes to the parameters had to be determined empirically.

In the end, a reasonable performance improvement through runtime adaptation of the parameters was reached at the cost of a loss of precision in the detection.