```
/*
 * face_detection_GPU_final.cpp
 *
 *  Created on: May 23, 2017
 *  Author: Patricia Navarro Martín
 *
 By downloading, copying, installing or using the software you agree to this license.
If you do not agree to this license, do not download, install,
copy or use the software.


                          License Agreement
               For Open Source Computer Vision Library
                       (3-clause BSD License)

Copyright (C) 2000-2015, Intel Corporation, all rights reserved.
Copyright (C) 2009-2011, Willow Garage Inc., all rights reserved.
Copyright (C) 2009-2015, NVIDIA Corporation, all rights reserved.
Copyright (C) 2010-2013, Advanced Micro Devices, Inc., all rights reserved.
Copyright (C) 2015, OpenCV Foundation, all rights reserved.
Copyright (C) 2015, Itseez Inc., all rights reserved.
Third party copyrights are property of their respective owners.

Redistribution and use in source and binary forms, with or without modification,
are permitted provided that the following conditions are met:

  * Redistributions of source code must retain the above copyright notice,
    this list of conditions and the following disclaimer.

  * Redistributions in binary form must reproduce the above copyright notice,
    this list of conditions and the following disclaimer in the documentation
    and/or other materials provided with the distribution.

  * Neither the names of the copyright holders nor the names of the contributors
    may be used to endorse or promote products derived from this software
    without specific prior written permission.

This software is provided by the copyright holders and contributors "as is" and
any express or implied warranties, including, but not limited to, the implied
warranties of merchantability and fitness for a particular purpose are disclaimed.
In no event shall copyright holders or contributors be liable for any direct,
indirect, incidental, special, exemplary, or consequential damages
(including, but not limited to, procurement of substitute goods or
services;
loss of use, data, or profits; or business interruption) however caused
and on any theory of liability, whether in contract, strict liability,
or tort (including negligence or otherwise) arising in any way out of
the use of this software, even if advised of the possibility of such damage.
 */

#include <opencv2/core/core.hpp>
#include <opencv2/highgui/highgui.hpp>
#include <opencv2/imgproc/imgproc.hpp>
#include <opencv2/gpu/gpu.hpp>
#include <iostream>
#include <fstream>
#include <sys/time.h>
#include <iomanip>

using namespace std;
using namespace cv;
using namespace cv::gpu;

struct timeval crono_on, crono_off;
struct timeval empieza, acaba;

//Función para calcular los FPS
 double calc_fps(void)
 {
     gettimeofday(&crono_off,NULL);
     double us = (crono_off.tv_usec-crono_on.tv_usec);
     double s = (crono_off.tv_sec-crono_on.tv_sec);
     double total = s + (us/1000000);
     double fps = 1/total;
     cout << setw(3) << fixed << fps << " FPS "<<endl;
     gettimeofday(&crono_on,NULL);
     return fps;
 }

int main(int argc, const char** argv)
{
    //Inicio del temporizador del programa global y definición de display en la Jetson TK1
    gettimeofday(&empieza,NULL);
    setenv("DISPLAY", ":0",0);

    //Declaración de las variables iniciales
    string cascadeName;
    CascadeClassifier_GPU cascade_gpu;
    VideoCapture capture;
    if(argc != 3)
    {
        cerr<<"(!) Argumento no válido. Debe ser: video_demo/webcam <dirección_del_classificador>"<<endl;
            return -1;
```

```cpp
    }
    else
    {
            if(string(argv[1])=="webcam")
            {
                capture.open(0);
            }
            else if(string(argv[1])=="video_demo")
            {
                capture.open("/home/ubuntu/Desktop/video_demo.mp4");
            }
            else
            {
            cerr<<"(!) Argumento no válido. Debe ser: video_demo/webcam <dirección_del_classificador>"<<endl;
                return -1;
            }

            cascadeName = string(argv[2]);
     }

    //DEFINICIÓN DE VARIABLES GLOBALES
    Mat frame;
    int frame_num=1;
    string log_path="./RESULTADOS/"+string(argv[1])+".csv";

    //COMPROBACIÓN DE RECURSOS
    if(!capture.isOpened())
    {
            cerr << "(!) No se pudo abrir:" << string(argv[1])<<endl;
            return -1;
    }
    if(!cascade_gpu.load(cascadeName))
    {
            cerr << "(!) No se pudo cargar el clasificador" << string(argv[2])<<endl;
            return -1;
    }

    //APERTURA DEL LOG
    ofstream log;
    log.open(log_path.c_str());

    //COMPROBACIONES PREVIAS AL VIDEOWRITER
    capture >> frame;
    if (frame.empty()) {
        cerr << "(!) Frame vacío\n";
        return -1;
    }
    bool isColor = (frame.type() == CV_8UC3);
    int height = capture.get(CV_CAP_PROP_FRAME_HEIGHT);

    //DECLARACIÓN DEL VIDEOWRITER
    VideoWriter writer;
    int codec = CV_FOURCC('M', 'J', 'P', 'G');
    double fps_write = 10.0;
    string filename = "./RESULTADOS/"+string(argv[1])+".avi";
    writer.open(filename, codec, fps_write, frame.size(), isColor);
    //Control de errores en la creación
    if (!writer.isOpened()) {
           cerr << "((!) No se pudo abrir el archivo para escribir.\n";
        return -1;
    }
    cout << "Archivo de video: " << filename << endl;

    //Iniciamos el cronómetro para el cálculo del fps
    gettimeofday(&crono_on,NULL);

    //____________INICIO DE LECTURA-ANÁLISIS-GRABACIÓN____________
   for(;;)
   {
       //Captura del frame e iniciamos el cronómetro para el cálculo del fps
      capture >> frame;

     //Comprobar que no se ha terminado el video
      if (!capture.read(frame)) {
          cout << "Video de lectura finalizado\n";
          break;
      }


      //Comienza la detección
      GpuMat faces;
      Mat frame_gray;
      cvtColor(frame, frame_gray, CV_BGR2GRAY);
      GpuMat gray_gpu(frame_gray);

      int detect_num = cascade_gpu.detectMultiScale(gray_gpu, faces,1.25,4);
      Mat obj_host;
      faces.colRange(0, detect_num).download(obj_host);
      Rect* cfaces = obj_host.ptr<Rect>();
      //Detección finalizada

      //Localización de los rostros
      for(int i=0;i<detect_num;++i)
      {
         Point pt1 = cfaces[i].tl();
         Size sz = cfaces[i].size();
         Point pt2(pt1.x+sz.width, pt1.y+sz.height);
         rectangle(frame, pt1, pt2, Scalar(255,255,0),3,8);
      }

      //Cálculo de los frames por segundo y almacenamiento del valor en el log
```

```cpp
      double fps = calc_fps();
      log<<fps<<"\n";

      //Escritura de texto informativo en frame
      ostringstream ss;
      ss<<"FPS = "<<fixed<<fps<<"  con GPU";
      putText(frame,ss.str(),Point(40,(height-25)),CV_FONT_HERSHEY_DUPLEX,0.8,Scalar(255,255,0),1,8,false);
      //Mostrar el frame resultante por pantalla y grabarlo en el archivo de video de escritura.
      imshow("Detección Facial - Versión CPU y GPU", frame);
      writer.write(frame);

      //Mantiene el frame durante 1ms y el programa se puede interrumpir si se pulsa la tecla Esc (en ASCII 27)
      int c = waitKey(1);
      if( (char)c == 27)break;

      frame_num++;
   }
   //____________FIN DE LECTURA-ANÁLISIS-GRABACIÓN____________

     //Cálculo de duración del programa total
     gettimeofday(&acaba,NULL);
     double us = (acaba.tv_usec-empieza.tv_usec);
     double s = (acaba.tv_sec-empieza.tv_sec);
     int min = (int)s/60;
     int seg = (int)s%60;
     double total = s + us/1000000;

    //Mostrar resultado por consola y registrar en log
     cout << "Tiempo de ejecución total (segundos):  "<<total<<endl;
     cout<< min <<" minutos "<<seg<<" segundos"<<endl;
     log<<total<<"segundos\n";

    //Cerrar el log
    log.close();
}
```