Instituto Tecnológico de Costa Rica
Maestría en Electrónica, Énfasis en Sistemas Empotrados
Sistemas Empotrados de Alto Desempeño
Prof. MSc. Mauricio Caamaño
II Cuatrimestre 2019
Reporte Proyecto II
Mario Castro Morera
Óscar Dengo Alvarado
Alejandro Rodríguez Elizondo

Todos los archivos de los programas creados y ejecutados se pueden encontrar en el siguiente git: https://github.com/mcastro90/g5_proyecto_2

3.2.2. Responda las siguientes preguntas relacionadas con la instalación de Python 3 y OpenCV 4:

¿Cuáles son las ventajas de usar un ambiente virtual de Python? (2 pts)

Las aplicaciones de python utilizan paquetes y módulos que no vienen como parte de la librería standard. En algunas ocasiones las aplicaciones necesitan utilizar la versión específica de una librería ya que la aplicación requiere que un bug en particular esté arreglado o fue escrito utilizando una versión obsoleta de la librería.

Esto quiere decir que no es posible para única instalación de Python alcanzar los requerimientos de todas las aplicaciones. Si una aplicación A necesita una versión 1.0 de un módulo, pero una aplicación B necesita una versión, entonces habrá un conflicto y la instalación de la versión 1.0 o 2.0 dejará a alguna de las aplicaciones sin la posibilidad de correr.

La solución de este problema es crear un ambiente virtual, con un árbol de direcciones autocontenido que posee una instalación para una versión particular de Python y además un número de paquetes adicionales.

Diferente aplicaciones pueden usar diferentes ambientes virtuales. Para resolver el conflicto entre A y B anteriormente citado, la aplicación A puede tener su propio

ambiente virtual con la versión 1.0, mientras que la aplicación B puede tener otro ambiente con la versión B. Si B o A necesitarán hacer cambios de las versiones de librería que utilizan esto no afectará el ambiente virtual creado para la otra aplicación.

¿Qué es NumPy? (2 pts)

Numpy es la contracción de las palabras en inglés "Numeric Python". Es un paquete fundamental para computación científica en Python. Es utilizado para realizar operaciones numéricas en arreglos. Numpy es mejor que las lista de Python en términos de tamaño, velocidad y funcionalidad.

El ancestro de NumPy, Numeric, fue creado originalmente por Jim Hugunin con algunas contribuciones de otros desarrolladores. En 2005, Travis Oliphant creó NumPy incorporando características de Numarray en NumPy con algunas modificaciones. NumPy es open source

Numpy es una librería de bajo nivel escrito en C (y Fortran) para funciones matemáticas de alto nivel. NumPy supera hábilmente el problema de ejecutar algoritmos más lentos en Python utilizando matrices multidimensionales y funciones que operan en matrices. Cualquier algoritmo puede entonces ser expresado como una función con matrices, permitiendo a los algoritmos correr rápidamente.

Numpy es parte del proyecto SciPy, y fue lanzado como una librería separada por lo que las personas que necesitan únicamente los requerimientos básicos pueden usarla sin instalar el resto de SciPy.

Numpy compatible con versiones de Python 2.4 hasta 2.72 y 3.1+

Describa ejemplos de 2 aplicaciones de numpy. (2 pts)

Análisis de datos:

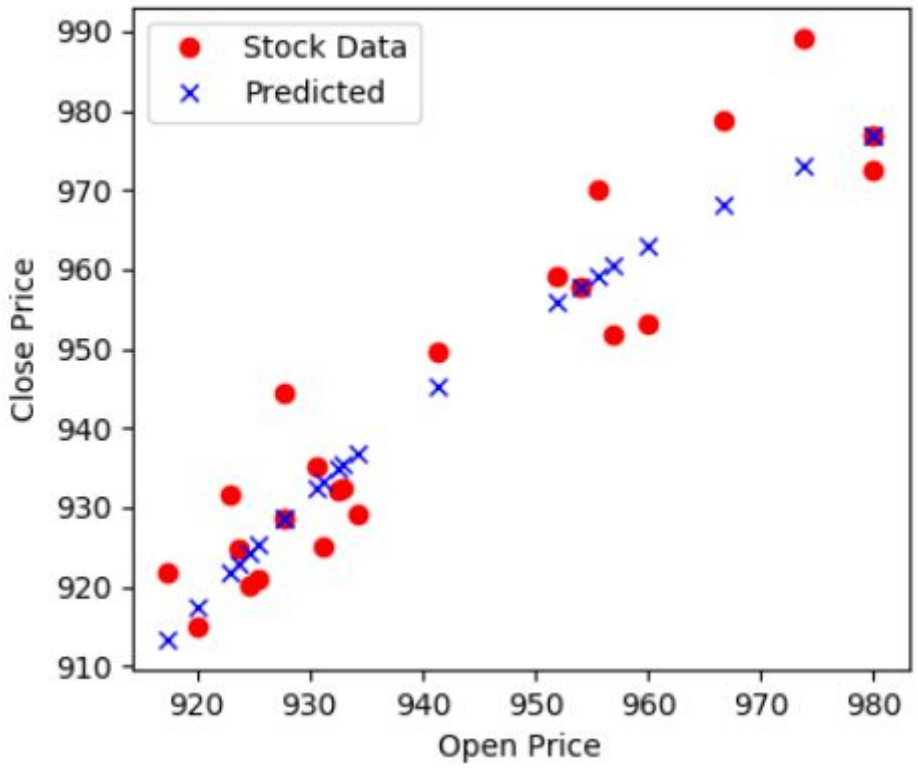
Una tarea común para los científicos e ingenieros es analizar los datos de una fuente externa que puede estar en formato de texto o de valores separados por comas (CSV). Al importar los datos a Python, se pueden realizar análisis de datos como estadísticas, tendencias o cálculos para sintetizar la información en información relevante y procesable. Numpy resulta una herramienta poderosa para tomar el extraer el arreglo de datos y realizar manipulación y exploración de la información antes de ser utilizado por las herramientas comunes de machine learning.

En el siguiente ejemplo tomado de apmonitor.com se extraen datos finacieors de internet cuyo arreglo es manipulado por Numpy para ser utilizado por un regresor. En este caso numpy se usar para se parar del arreglo ciertas columnas y pasarlas a otras variables.

```
import pandas as pd
import matplotlib.pyplot as plt

# stock ticker symbol
url = 'https://apmonitor.com/che263/uploads/Main/goog.csv'

# import data with pandas
data = pd.read_csv(url)
print(data['Close'][0:5])
print('min: '+str(min(data['Close'][0:20])))
print('max: '+str(max(data['Close'][0:20])))

# plot data with pyplot
plt.figure()
plt.plot(data['Open'][0:20])
plt.plot(data['Close'][0:20])
plt.xlabel('days ago')
```

```
plt.ylabel('price')
plt.show()

from gekko import GEKKO
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd

# Google stock
url = 'https://apmonitor.com/che263/uploads/Main/goog.csv'

# import data with pandas
data = pd.read_csv(url)
print(data['Close'][0:5])
print('min: '+str(min(data['Close'][0:20])))
print('max: '+str(max(data['Close'][0:20])))

# GEKKO model
m = GEKKO()

# input data
x = m.Param(value=np.array(data['Open']))

# parameters to optimize
a = m.FV()
a.STATUS=1
b = m.FV()
b.STATUS=1
c = m.FV()
c.STATUS=1

# variables
y = m.CV(value=np.array(data['Close']))
y.FSTATUS=1
```

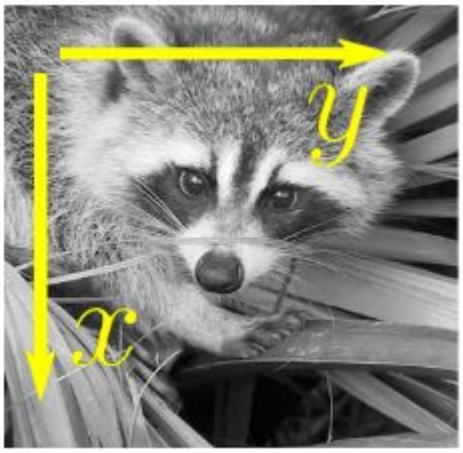
```
# regression equation
m.Equation(y==b*m.exp(a*x)+c)

# regression mode
m.options.IMODE = 2

# optimize
m.options.solver = 1
m.solve(disp=True)

# print parameters
print('Optimized, a = ' + str(a.value[0]))
print('Optimized, b = ' + str(b.value[0]))
print('Optimized, c = ' + str(c.value[0]))

# plot data
plt.figure()
plt.plot(data['Open'],data['Close'],'ro',label='Stock Data')
plt.plot(x.value,y.value,'bx',label='Predicted')
plt.xlabel('Open Price')
plt.ylabel('Close Price')
plt.legend()
plt.show()
```



Manipulación de imágenes

Numpy puede ser utilizado para la manipulación de imágenes. Las imágenes pueden ser vistas como una matriz, por lo que las operaciones de Numpy resultan de gran utilidad.

En el siguiente ejemplo tomado de scipy-lectures.org vemos algunas utilidades del uso de Numpy en la manipulación de imágenes, en este utiliziando la función ogrid que devuelve una malla abierta (es decir, no desarrollada) cuando se indexa, de modo que solo una dimensión de cada matriz devuelta es mayor que 1. La dimensión y el número de las matrices de salida son iguales al número de dimensiones de indexación.



0	1	2
3	4	5
6	7	8

```
import numpy as np
import scipy
import scipy.misc
import matplotlib.pyplot as plt

face = scipy.misc.face(gray=True)
face[10:13, 20:23]
face[100:120] = 255

lx, ly = face.shape
X, Y = np.ogrid[0:lx, 0:ly]
mask = (X - lx/2)**2 + (Y - ly/2)**2 > lx*ly/4
face[mask] = 0
face[range(400), range(400)] = 255

plt.figure(figsize=(3, 3))
plt.axes([0, 0, 1, 1])
plt.imshow(face, cmap=plt.cm.gray)
plt.axis('off')

plt.show()
```

¿Para qué sirve CMake? (2 pts)

CMake es una familia de herramientas multiplataforma de código abierto diseñada para crear, probar y empaquetar software. CMake se utiliza para controlar el proceso de compilación de software utilizando una plataforma simple y archivos de configuración independientes del compilador, y generar archivos de trabajo y espacios de trabajo nativos que se pueden utilizar en el entorno del compilador que elija. Kitware creó el conjunto de herramientas CMake en respuesta a la necesidad de un entorno de construcción potente y multiplataforma para proyectos de código abierto como ITK y VTK.

Es comparable al GNU build system de Unix en que el proceso es controlado por ficheros de configuración, en el caso de CMake llamados CMakeLists.txt. Al contrario que el GNU build system, que está restringido a plataformas Unix, CMake soporta la generación de ficheros para varios sistemas operativos, lo que facilita el mantenimiento y elimina la necesidad de tener varios conjuntos de ficheros para cada plataforma.

CMake puede manejar compilaciones in situ y fuera de lugar, permitiendo varias compilaciones desde el mismo árbol de origen y compilación cruzada. La capacidad de crear un árbol de directorios fuera del árbol de origen es una característica clave, que garantiza que si se elimina un directorio de compilación, los archivos de origen no se verán afectados.

CMake puede localizar ejecutables, archivos y bibliotecas. Estas ubicaciones se almacenan en un caché, que luego se puede adaptar antes de generar los archivos de compilación de destino. El caché se puede editar con un editor gráfico, que se incluye en el proyecto.

CMake admite bien las jerarquías de directorios complicadas y las aplicaciones que dependen de varias bibliotecas. Por ejemplo, CMake puede acomodar un proyecto que tiene múltiples kits de herramientas o bibliotecas que tienen múltiples directorios. Además, CMake puede trabajar con proyectos que requieren la creación de ejecutables antes de generar el código que se compilará para la aplicación final. Su diseño de código abierto y extensible permite que CMake se adapte según sea necesario para un proyecto específico

¿Qué ventajas/desventajas tiene CMake ante otras herramientas de configuración de Makefiles (Autotools, qmake, Scons, etc) en sistemas embebidos? (2 pts)

La principal ventaja de CMake es que es multiplataforma (tanto en el sentido de los sistemas operativos como de los sistemas de construcción en general). Un solo archivo de configuración CMake puede generar un sistema de compilación nativo para make (Makefiles), Visual Studio (Visual Studio soultions & projects), ninja, NMake, Code :: Blocks, Eclipse, KDEdevelop, ... Y genera un sistema de compilación nativo para eso herramienta, para que los desarrolladores que usan el sistema de compilación trabajen de la manera en que están acostumbrados.

Incluso para proyectos de plataforma única, CMake ofrece mecanismos de nivel superior como la detección y configuración de la biblioteca externa (es decir, la configuración automática de las definiciones, incluir directorios y archivos de enlace

para trabajar con una biblioteca de terceros), soporte para generar la instalación (y el empaquetado), integración con la utilidad de prueba CTest y así sucesivamente.

CMake también resume algunos detalles específicos del compilador como "qué indicadores pasar para crear una biblioteca compartida" o "cómo habilitar PIC u optimización interprocedural". Por lo tanto, puede generar un archivo MAKE para gcc, Clang o el compilador Sun CC sin preocuparse necesariamente por los detalles de su interfaz.

Una de las de las desventajas es el lenguaje CMake no es algo que pueda comparar con lo que probablemente haya usado antes. No hay clases, mapas, funciones virtuales o lambdas. Incluso tareas como "analizar los argumentos de entrada de una función" y "devolver el resultado de una función" son bastante difíciles para los principiantes.

3.3 Perfilado de aplicaciones en Raspberry Pi.

Con la ayuda de la cámara del Raspberry Pi, la librería de opencv y Python, se realizaron varias pruebas para poder estudiar el comportamiento en términos del tiempo de ejecucion del codigo motion_detection.py y utilizando herramientas con kcachegrind y pstats, se logran visualizar, como se presenta a continuación:

```
Ordered by: internal time
List reduced from 1289 to 10 due to restriction <10>

ncalls  tottime  percall  cumtime  percall filename:lineno(function)
   153    3.460    0.023    3.460    0.023 {GaussianBlur}
   153    3.098    0.020    3.098    0.020 {resize}
     1    2.002    2.002    2.002    2.002 {built-in method time.sleep}
   152    1.526    0.010    1.526    0.010 {waitKey}
    13    0.696    0.054    0.710    0.055 {built-in method _imp.create_dynamic}
   456    0.599    0.001    0.599    0.001 {imshow}
     1    0.418    0.418    0.418    0.418 {method 'read' of 'cv2.VideoCapture' objects}
   152    0.374    0.002    0.374    0.002 {findContours}
   304    0.219    0.001    0.219    0.001 {putText}
   152    0.162    0.001    0.162    0.001 {dilate}
```

Tabla 1. paso 3.3.2. Datos obtenidos del archivo prof_rpi_cam.out

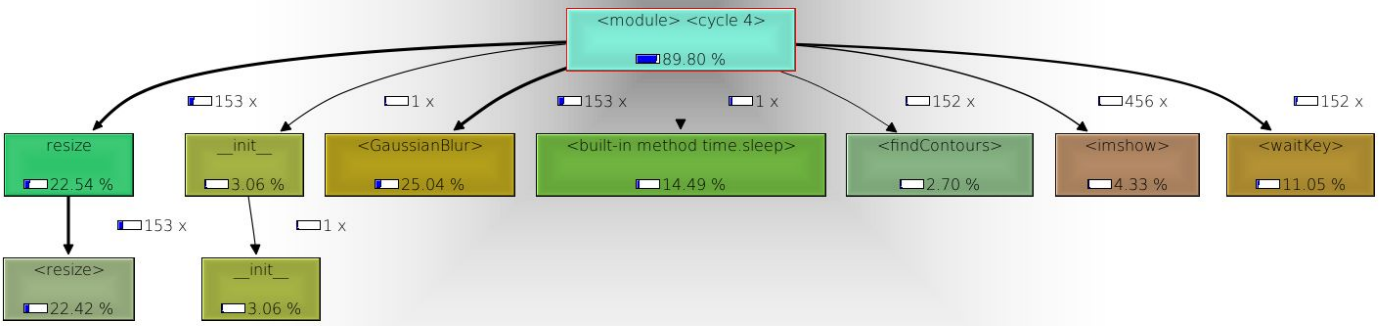


Fig 1. Call Graph obtenido utilizando Kcachegrind.

```
         82874 function calls (80097 primitive calls) in 21.647 seconds

Ordered by: internal time
List reduced from 1285 to 10 due to restriction <10>

ncalls  tottime  percall  cumtime  percall filename:lineno(function)
   137    8.401    0.061    8.401    0.061 {resize}
   138    6.491    0.047    6.491    0.047 {method 'read' of 'cv2.VideoCapture' objects}
   137    2.592    0.019    2.592    0.019 {GaussianBlur}
   137    0.964    0.007    0.964    0.007 {waitKey}
    13    0.698    0.054    0.712    0.055 {built-in method _imp.create_dynamic}
   411    0.544    0.001    0.544    0.001 {imshow}
     1    0.351    0.351   21.648   21.648 motion_detector.py:6(<module>)
   137    0.275    0.002    0.275    0.002 {findContours}
   274    0.194    0.001    0.194    0.001 {putText}
   166    0.111    0.001    0.111    0.001 {built-in method marshal.loads}
```

Tabla 2. paso 3.3.7. Datos obtenidos del archivo prof_rpi_vid1.out

```
      82673 function calls (79896 primitive calls) in 18.688 seconds

Ordered by: internal time
List reduced from 1285 to 10 due to restriction <10>

ncalls  tottime  percall  cumtime  percall filename:lineno(function)
   137    9.515    0.069    9.515    0.069 {resize}
   138    6.231    0.045    6.231    0.045 {method 'read' of 'cv2.VideoCapture' objects}
    13    0.713    0.055    0.727    0.056 {built-in method _imp.create_dynamic}
   411    0.341    0.001    0.341    0.001 {imshow}
     1    0.324    0.324   18.689   18.689 motion_detector.py:6(<module>)
   137    0.322    0.002    0.322    0.002 {waitKey}
   137    0.233    0.002    0.233    0.002 {GaussianBlur}
   274    0.121    0.000    0.121    0.000 {putText}
   166    0.111    0.001    0.111    0.001 {built-in method marshal.loads}
480/477    0.050    0.000    0.112    0.000 {built-in method builtins.__build_class__}
```

Tabla 3. paso 3.3.8. frame = imutils.resize(frame, width=100). Datos obtenidos del archivo prof_rpi_vid2.out

Compare con los resultados obtenidos en el paso 3.3.7. Justifique las diferencias de las funciones resize y GaussianBlur entre ambos resultados.

Ambas respuestas del sistema corren la función resize la misma cantidad de veces, sin embargo toma menor tiempo por llamado cuando se utiliza el código con width=100.

El proceso GaussianBlur muestra una gran diferencia entre ambas corridas, ya que la corrida con width=100 presenta un consumo en tiempo menor que con width original. Sin embargo el número de llamados a esta función se mantiene igual.

Este resultado se debe a que al reducir la resolución de la imagen de video, los procesos se ejecutan con mayor rapidez ya que el blur se aplica a una imagen de menor tamaño y por lo tanto debe procesar menos datos.

3.3.9. (Opcional)

Mejora de la función resize.

El código original posee una función de la librería imutils:

frame = imutils.resize(frame, width=500)

Esta permite hacer manipulacion de imagenes y video de forma sencilla, esta recibe dos parámetros, el primero este caso llamado frame contiene la lectura del video, el otro parámetro es width o ancho de la imagen . Esto hace que sea más fácil el

ajuste del tamaño de la imagen de video ya que esta ajusta por sí sola la relación de aspecto, requiriendo solo del ancho de la imagen como parámetro. Sin embargo esta facilidad de uso impone un "trade-off" en cuanto al rendimiento, ya que si bien es cierto para muchas otras aplicaciones no es tan importante el tiempo que tarda esta función en completar, para nuestro caso en particular se busca una alternativa que permita reducir el tiempo que esta tarda..

Por lo anterior se estudió, la posibilidad de utilizar la función de ajuste de tamaño de la cv2, la cual es un poco más compleja de utilizar, ya que el usuario debe de proveer más detalles acerca de la resolución de la imagen para mantener la relación de aspecto, y algunos otros parámetros opcionales.

Proceso.

La función a utilizar tiene la siguiente forma:

frame = cv2.resize(frame, (500, 281))

Se observa que a diferencia de la función anterior, esta es parte de la librería cv2, y además posee el valor del heigth, que sería la altura de la imagen o eje y. Para este caso en particular se mantiene el valor de width=500, con la diferencia de que solo se expresa el valor de este, además se agrega el valor del height que sería 281, para mantener la relación de aspecto que posee la función imutils.size. Para esto se corre un código simple en python sobre el mismo video que se utilizó para las pruebas anteriores, para extraer dicha relación.

El codigo a continuacion:

```python
from imutils.video import VideoStream
import argparse
import datetime
import imutils
import time
import cv2

ap = argparse.ArgumentParser()
ap.add_argument("-v", "--video", help="path to the video file")
ap.add_argument("-a", "--min-area", type=int, default=500, help="minimum area size")
args = vars(ap.parse_args())
vs = cv2.VideoCapture(args["video"])

_, frame = vs.read()

(w,h,c) = frame.shape

img = imutils.resize(frame,width=500)

print("original width and height")
print(w,h)
print("resized values h, w" )
print(img.shape)

```

Fig 2. Código para extraer la relación de aspecto de la función imutils.

Una vez realizado dicho cambio, se procede a realizar las pruebas con el video, con la función original de imutils y luego con la cv2, manteniendo como se menciona anteriormente el width en 500, y luego se visualizan los resultados utilizando pstats. La primera imagen corresponde a la prueba realizada con la función original imutils.resize.

```
     82874 function calls (80097 primitive calls) in 21.516 seconds

Ordered by: internal time
List reduced from 1285 to 10 due to restriction <10>

ncalls  tottime  percall  cumtime  percall filename:lineno(function)
   137    8.331    0.061    8.331    0.061 {resize}
   138    6.347    0.046    6.347    0.046 {method 'read' of 'cv2.VideoCapture' objects}
   137    2.733    0.020    2.733    0.020 {GaussianBlur}
   137    0.919    0.007    0.919    0.007 {waitKey}
    13    0.708    0.054    0.721    0.055 {built-in method _imp.create_dynamic}
   411    0.539    0.001    0.539    0.001 {imshow}
     1    0.344    0.344   21.517   21.517 motion_detector.py:6(<module>)
   137    0.278    0.002    0.278    0.002 {findContours}
   274    0.189    0.001    0.189    0.001 {putText}
   166    0.110    0.001    0.110    0.001 {built-in method marshal.loads}
```

Tabla 4. Prueba con función imutils.resize

```
      82737 function calls (79960 primitive calls) in 15.700 seconds

Ordered by: internal time
List reduced from 1284 to 10 due to restriction <10>

ncalls  tottime  percall  cumtime  percall filename:lineno(function)
   138    6.565    0.048    6.565    0.048 {method 'read' of 'cv2.VideoCapture' objects}
   137    3.687    0.027    3.687    0.027 {GaussianBlur}
   137    1.203    0.009    1.203    0.009 {resize}
   137    0.971    0.007    0.971    0.007 {waitKey}
    13    0.714    0.055    0.728    0.056 {built-in method _imp.create_dynamic}
   411    0.550    0.001    0.550    0.001 {imshow}
     1    0.355    0.355   15.701   15.701 motion_detector.py:6(<module>)
   137    0.278    0.002    0.278    0.002 {findContours}
   274    0.193    0.001    0.193    0.001 {putText}
   166    0.114    0.001    0.114    0.001 {built-in method marshal.loads}
```

Tabla 5. Prueba con función cv2.resize

Los resultados anteriores muestran que para esta aplicación en específico, donde se desea mantener la relación de aspecto limitada por el ancho de la imagen en 500, la función cv2.resize es mas rapida, y esto puede deberse al hecho de que el programador provee los detalles de la relación de aspecto, por lo que la función no realiza dicho cálculo, resultando en un proceso de menor peso computacional, y por lo tanto más rápido, reduciendo el tiempo total de 21.5 segundos a 15.7 segundos, con una diferencia de 5.8 segundos, lo que equivale a un 27% de mejora con respecto a la función original.

3.4.2 Guarde el resultado con nombre: prof_jetson_vid1.out (5 pts)

Archivo adjunto

3.4.3 Escriba el resultado obtenido con “stats 10” en el reporte final. (5 pts)

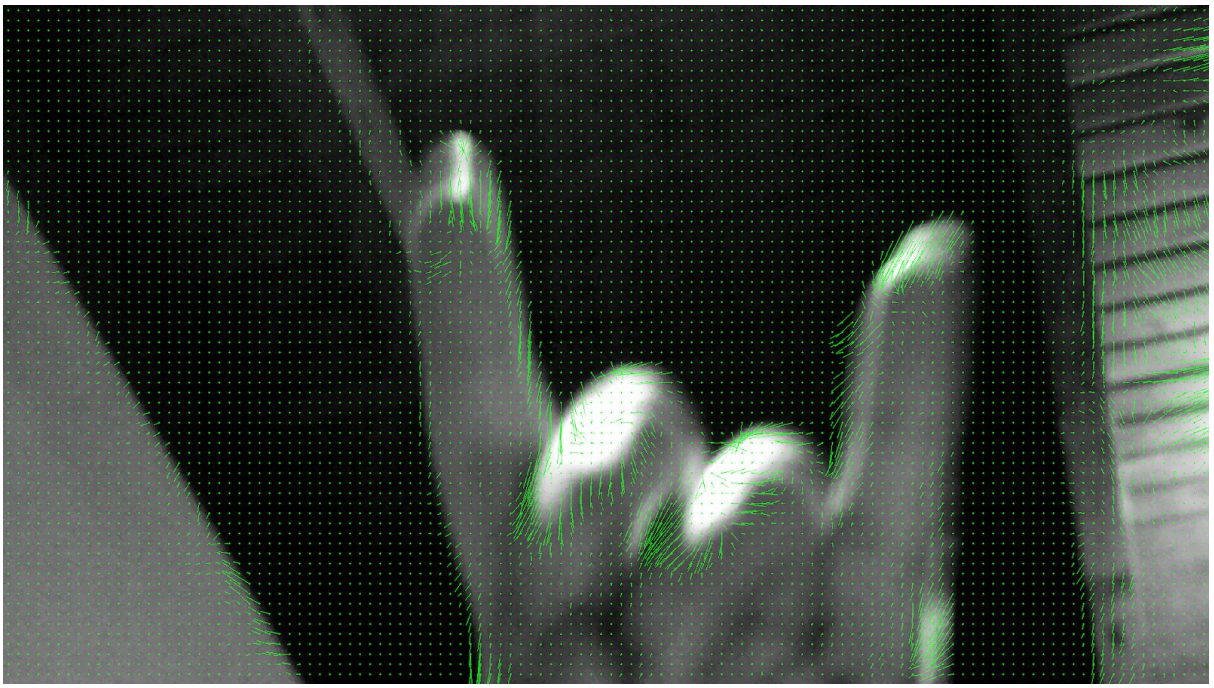
```
Welcome to the profile statistics browser.
prof_jetson_vid1.out% strip
prof_jetson_vid1.out% sort time
prof_jetson_vid1.out% stats 10
Fri Aug  2 21:04:28 2019    prof_jetson_vid1.out

         127855 function calls (124173 primitive calls) in 8.114 seconds

   Ordered by: internal time
   List reduced from 1352 to 10 due to restriction <10>
```

```
  ncalls  tottime  percall  cumtime  percall filename:lineno(function)
     137    3.875    0.028    3.875    0.028 {resize}
     138    1.855    0.013    1.855    0.013 {method 'read' of 'cv2.VideoCapture' objects}
     136    0.440    0.003    0.440    0.003 {waitKey}
     137    0.429    0.003    0.429    0.003 {GaussianBlur}
      22    0.235    0.011    0.246    0.011 {built-in method _imp.create_dynamic}
     408    0.181    0.000    0.181    0.000 {imshow}
     136    0.118    0.001    0.118    0.001 {findContours}
     137    0.084    0.001    0.084    0.001 {cvtColor}
       1    0.074    0.074    8.117    8.117 motion_detector.py:6(<module>)
     172    0.062    0.000    0.062    0.000 {built-in method marshal.loads}


prof_jetson_vid1.out%
```

3.4.4 Incluya en el reporte pdf, dos fotos obtenidas de correr el código, las cuales se guardan en el directorio outputs/cpu y outputs/gpu (5 pts)

Imagen obtenida en outputs/cpu:

cpu_frame_36.png



cpu_frame_128.png:

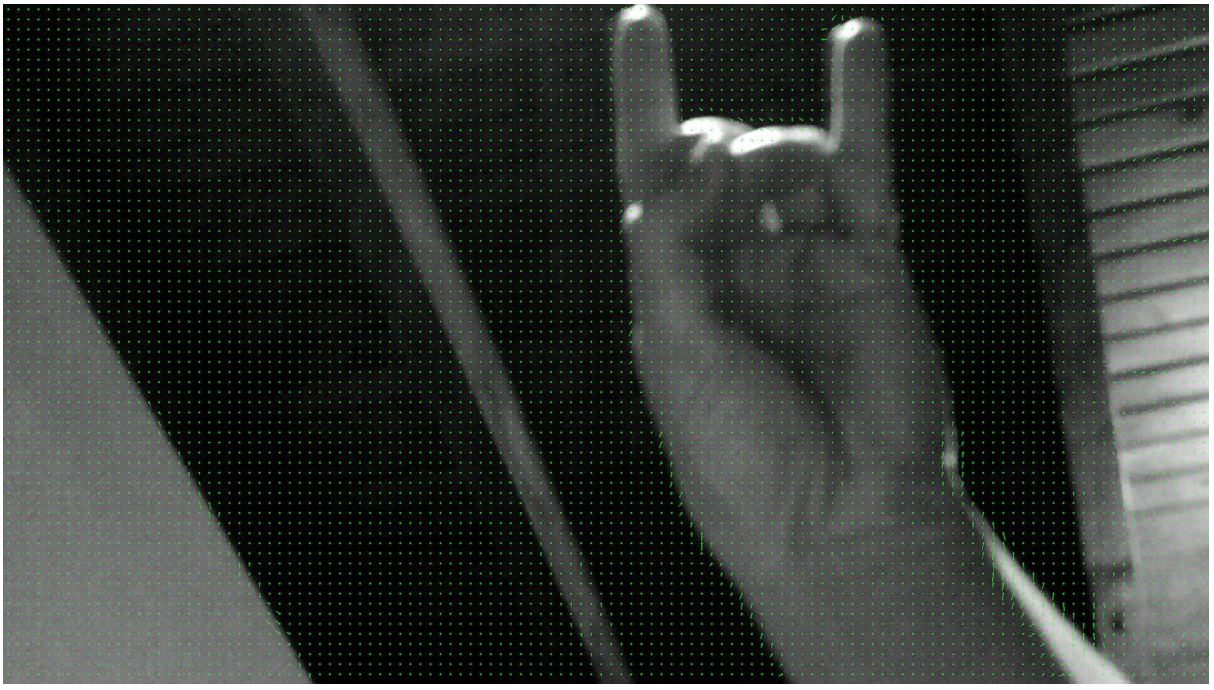
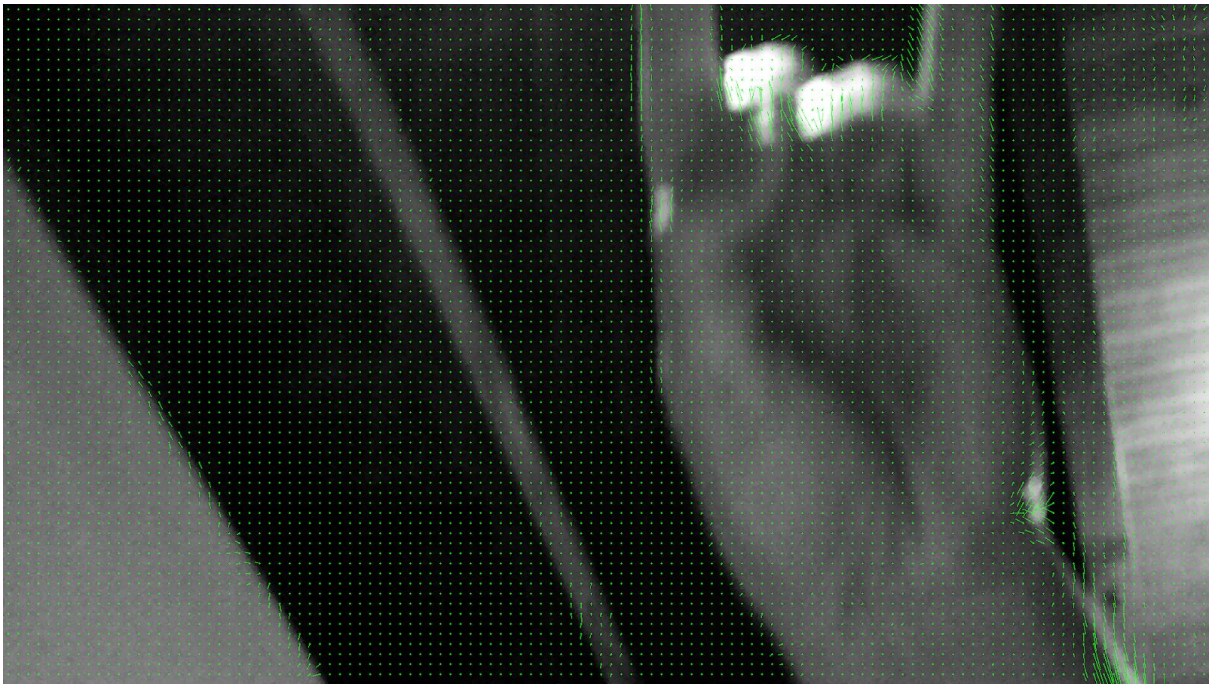


Imagen obtenida en outputs/gpu:

gpu_frame_49.png

gpu_frame_86.png



3.4.5 Guarde el resultado como nvprof_jetson_gpu.log (10 pts)

```
==10128== NVPROF is profiling process 10128, command: /root/.virtualenvs/cv/bin/python
gpu-opt_flow.py
==10128== Warning: Unified Memory Profiling is not supported on the underlying platform. System
requirements for unified memory can be found at:
http://docs.nvidia.com/cuda/cuda-c-programming-guide/index.html#um-requirements
==10128== Profiling application: /root/.virtualenvs/cv/bin/python gpu-opt_flow.py
==10128== Profiling result:
            Type  Time(%)      Time     Calls       Avg       Min       Max  Name
 GPU activities:   48.17%  31.2854s      1632  19.170ms  729.69us  62.174ms
cv::cuda::device::optflow_farneback::boxFilter5(int, int, cv::cuda::PtrStep<float>, int, float,
cv::cuda::PtrStep<float>)
                  20.35%  13.2197s      1088  12.150ms  6.6678ms  260.57ms  void
cv::cuda::device::optflow_farneback::gaussianBlur<cv::cuda::device::BrdReflect101<float>>(int, int,
cv::cuda::PtrStep<float>, int, float, cv::cuda::PtrStep)
                  13.67%  8.87797s      1632  5.4399ms  217.87us  19.805ms
cv::cuda::device::optflow_farneback::updateMatrices(int, int, cv::cuda::PtrStep<float>,
cv::cuda::PtrStep<float>, cv::cuda::PtrStep<float>, cv::cuda::PtrStep<float>, cv::cuda::PtrStep<float>)
                   5.91%  3.84147s       136  28.246ms  8.4208ms  488.50ms  [CUDA memcpy DtoH]
                   5.58%  3.62670s      1088  3.3334ms  160.42us  14.437ms  void
cv::cuda::device::optflow_farneback::polynomialExpansion<int=5>(int, int, cv::cuda::PtrStep<float>,
cv::cuda::PtrStep)
                   2.79%  1.81241s      1632  1.1105ms  56.302us  3.5102ms
cv::cuda::device::optflow_farneback::updateFlow(int, int, cv::cuda::PtrStep<float>,
cv::cuda::PtrStep<float>, cv::cuda::PtrStep<float>)
                   1.60%  1.04076s      1632  637.72us  117.29us  2.5975ms  void
cv::cuda::device::resize_linear<float>(cv::cuda::PtrStepSz<float>, float, float, float)
                   0.49%  318.49ms       272  1.1709ms  1.1549ms  1.4724ms  [CUDA memcpy DtoD]
                   0.48%  310.57ms       816  380.60us  52.553us  925.01us  void
cv::cudev::grid_transform_detail::transformSmart<int=4, float, float,
_GLOBAL__N__53_tmpxft_00003629_00000000_9_gpu_mat_compute_75_cpp1_ii_71482d89::Converto
r<float, float, float>, cv::cudev::WithOutMask>(cv::cudev::GlobPtr<float>,
cv::cudev::grid_transform_detail::transformSmart<int=4, float, float,
_GLOBAL__N__53_tmpxft_00003629_00000000_9_gpu_mat_compute_75_cpp1_ii_71482d89::Converto
r<float, float, float, float>, cv::cudev::WithOutMask>, float, float, int, int)
                   0.40%  258.31ms       136  1.8993ms  1.8869ms  2.0342ms  void
cv::cudev::grid_split_merge_detail::mergeC2<cv::cudev::GlobPtr<int>, cv::cudev::GlobPtr<int>, int2,
cv::cudev::WithOutMask>(int, cv::cudev::GlobPtr<int>, cv::cudev::GlobPtr<cv::cudev::GlobPtr<int>>,
int2, int, int)
                   0.28%  184.83ms      1904  97.077us     208ns  1.9567ms  [CUDA memcpy HtoD]
                   0.26%  171.50ms       272  630.50us  584.33us  4.2017ms  void
cv::cudev::grid_transform_detail::transformSmart<int=4, unsigned char, float,
cv::cudev::saturate_cast_func<unsigned char, float>,
cv::cudev::WithOutMask>(cv::cudev::GlobPtr<unsigned char>,
cv::cudev::grid_transform_detail::transformSmart<int=4, unsigned char, float,
cv::cudev::saturate_cast_func<unsigned char, float, float>, cv::cudev::WithOutMask>, unsigned char,
float, int, int)
                   0.00%  130.21us       272     478ns     208ns  3.6980us  [CUDA memset]
      API calls:   42.74%  43.1745s      5304  8.1400ms  26.824us  249.45ms  cudaFree
                  33.78%  34.1210s      5304  6.4331ms  20.781us  4.43402s  cudaMallocPitch
                  17.65%  17.8289s      7752  2.2999ms  3.8020us  353.80ms  cudaStreamSynchronize
                   4.24%  4.28510s       408  10.503ms  961.94us  490.83ms  cudaMemcpy2D
                   0.96%  971.09ms      9928  97.813us  32.552us  1.9850ms  cudaLaunchKernel
                   0.28%  287.90ms       136  2.1169ms  1.9345ms  3.3794ms  cudaDeviceSynchronize
                   0.19%  194.11ms      1632  118.94us  28.489us  4.2877ms  cudaMemcpyToSymbol
                   0.07%  70.649ms       680  103.90us  17.656us  1.7979ms  cudaStreamDestroy
                   0.02%  25.048ms       272  92.087us  71.668us  180.26us  cudaMemcpy2DAsync
                   0.02%  22.323ms       272  82.068us  51.511us  558.23us  cudaMemset2DAsync
                   0.02%  16.066ms      9928  1.6180us     572ns  201.67us  cudaGetLastError
                   0.01%  14.097ms      1768  7.9730us  2.7080us  130.16us  cudaGetDevice
                   0.01%  8.6782ms       680  12.762us  6.1980us  346.25us  cudaStreamCreate
                   0.00%  214.59us        96  2.2350us  1.3540us  32.083us  cuDeviceGetAttribute
                   0.00%  184.69us         1  184.69us  184.69us  184.69us  cudaGetDeviceProperties
                   0.00%  9.3230us         3  3.1070us  1.8750us  4.3750us  cuDeviceGetCount
```

```
0.00%  9.0620us         1  9.0620us  9.0620us  9.0620us  cuDeviceTotalMem
0.00%  5.2080us         3  1.7360us     833ns  2.6040us  cudaGetDeviceCount
0.00%  4.0630us         2  2.0310us  1.7710us  2.2920us  cuDeviceGet
0.00%  2.1870us         1  2.1870us  2.1870us  2.1870us  cuDeviceGetName
0.00%  1.7710us         1  1.7710us  1.7710us  1.7710us  cuDeviceGetUuid
```

Responda: Cuáles son las 3 funciones que más tiempo requieren, según el resultado? (3 pts)

- boxFilter5 (31 segundos)
- gaussianBlur (13 segundos)
- updateMatrices (8 segundos)

Api calls:

- cudaFree (43 segundos)
- cudaMallocPitch (34 segundos)
- cudaStreamSynchronize (17 segundos)

3.4.6 Escriba en el reporte final el comando utilizado (5 pts)

El comando de python no funcionaba con la instalación de OpenCV al estar contenido en virtualenv, por lo que la llamada se hizo al ejecutable de Python dentro del respectivo virtualenv.

sudo /usr/local/cuda/bin/nvprof --print-gpu-trace --timeout 10 --log-file nvprof_jetson_gputrace_5sec.log ~/.virtualenvs/cv/bin/python gpu-opt_flow.py

Guarde el resultado como nvprof_jetson_gputrace_5sec.log (5 pts)

Archivo adjunto, contenido muy extenso como para adjuntarlo al documento.

3.4.7 Responda: Porque no es posible obtener resultados? (2 pts)

Contenido de nvprof_jetson_cpu.log

```
======== Warning: No CUDA application was profiled, exiting
```

Es una aplicación que va a utilizar el CPU para el procesamiento y no utiliza código especializado para CUDA, por lo que no se pueden obtener datos esperado o útiles.