Tecnológico de Costa Rica
Escuela de Ingeniería Electrónica
Maestría en Sistemas Empotrados
MP6171 - Sistemas Empotrados de Alto Desempeño
II Cuatrimestre 2019
Estudiantes: Leonardo Araya, Gabriel Loría y Álvaro Salazar.
Reporte de Proyecto 2

Preguntas del proyecto.

1. ¿Cuáles son las ventajas de usar un ambiente virtual de Python?

Las ventajas son:

- Permiten usar diferentes versiones de Python (ejemplo: Python 2 en un ambiente y Python 3 en otro).
- Permiten usar diferentes versiones de paquetes o dependencias sin afectar otros ambientes (ejemplo: tener pypy en un ambiente y cython en otro ambiente).

2. ¿Qué es NumPy?

“NumPy es un paquete fundamental para computación científica con Python”.[1]

Entre sus funciones están el manejo de matrices multidimensionales, integración de código C/C++ y Fortran, funciones matemáticas como transformadas de Fourier y álgebra lineal. Su fuerte radica en el manejo de matrices y arreglos.

3. Describa ejemplos de dos aplicaciones de NumPy.

Un ejemplo de aplicación de NumPy es *Machine Learning*, donde basados en una gran cantidad de datos se pueden extraer patrones o tendencias a través de manipulación de dichos datos. *Machine Learning* usa varios paquetes entre los que se pueden citar sci-kit-learn y TensorFlow. Algunos de estos paquetes se basan en el paquete Pandas, el cual a su vez se basa en NumPy. Scikit-learn utiliza diferentes algoritmos basándose tanto en NumPy como Pandas para manipular datos e identificar patrones y tendencias a partir de los cuales *Machine Learning* puede predecir los siguientes resultados.

Otro ejemplo de aplicación de NumPy es en identificación de características en imágenes. Las imágenes están representadas en serie de números que se pueden acomodar en matrices como lo muestra la figura 1. Mediante manipulación de datos con NumPy, se pueden identificar grupos de valores que permiten al sistema identificar características tales como el verde del

zacate del suelo o el amarillo de una pelota. Debido a que las imágenes son una serie de valores en una matriz, algunos elementos de la matriz tienen valores muy similares a los elementos vecinos, por lo que el algoritmo los agrupa e identifica los conjuntos de valores similares. Cada grupo de valores similares representa algo importante de la imagen, llámese pelota, zacate, sol, etc.

4. ¿Para qué sirve CMake?

CMake sirve para controlar el proceso de compilado donde permite compilar, probar y crear paquetes a partir código fuente para diferentes plataformas considerando que ellas pueden usar diferentes comandos. CMake permite compilar código independientemente del sistema operativo y del compilador, además de trabajar en conjunto con el compilador nativo.

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

Entre las ventajas de CMake están:

- Es capaz de trabajar con diferentes compiladores sin necesidad de cambiar los comandos de CMake.
- Automáticamente reconoce dependencias de códigos fuentes en C y C++, tal que no es necesario definir explícitamente la importación de los *headers*.
- Para sistemas de Windows, elimina la necesidad de instalar Cygwin para usar *makes* de Unix, además de poder correr más velozmente.

Entre las desventajas están:

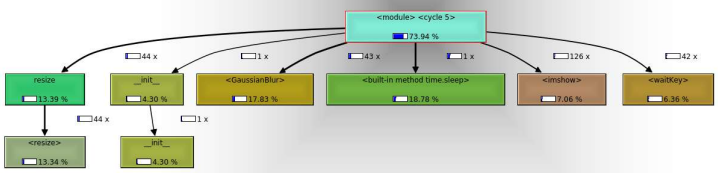
- Autotools está enfocado en Unix/Linux, por lo que sus compilaciones funcionan muy bien en Unix/Linux. CMake es multiplataforma por lo que hay que indicarle con varios parámetros que se compila para Unix/Linux.
- Autotools genera un único archivo de configuración y todos los archivos requeridos van dentro de la distribución, mientras que CMake genera varios archivos que dificultan la solución de problemas de parte del usuario.
- CMake no siempre sigue estándares de código.

Resultados de Raspberry Pi 2

Ejecución de Stats10 del motion_detector.py (sección 3.3.3):

```
out.prof% stats 10
Tue Aug  6 22:06:39 2019    out.prof

         115268 function calls (111527 primitive calls) in 10.660 seconds

   Ordered by: internal time
   List reduced from 1305 to 10 due to restriction <10>

   ncalls  tottime  percall  cumtime  percall filename:lineno(function)
        1    2.002    2.002    2.002    2.002 {built-in method time.sleep}
       43    1.901    0.044    1.901    0.044 {GaussianBlur}
       44    1.422    0.032    1.422    0.032 {resize}
       23    1.257    0.055    1.274    0.055 {built-in method _imp.create_dynamic}
      126    0.753    0.006    0.753    0.006 {imshow}
       42    0.679    0.016    0.679    0.016 {waitKey}
        1    0.451    0.451    0.451    0.451 {method 'read' of 'cv2.VideoCapture' objects}
      317    0.166    0.001    0.166    0.001 {built-in method builtins.compile}
      173    0.163    0.001    0.163    0.001 {built-in method marshal.loads}
       84    0.141    0.002    0.141    0.002 {putText}
```

Imagen del Call Graph (sección 3.3.4):



Ejecución de Stats10 del video grabado con motion_detector.py (sección 3.3.7):

```
prof_rpi_vid1.out% stats 10
Wed Aug  7 00:08:32 2019    prof_rpi_vid1.out

         120161 function calls (116413 primitive calls) in 38.265 seconds

   Ordered by: internal time
   List reduced from 1304 to 10 due to restriction <10>

   ncalls  tottime  percall  cumtime  percall filename:lineno(function)
      143   13.646    0.095   13.646    0.095 {resize}
      144   11.003    0.076   11.003    0.076 {method 'read' of 'cv2.VideoCapture' objects}
      143    5.424    0.038    5.424    0.038 {GaussianBlur}
      142    1.987    0.014    1.987    0.014 {waitKey}
       23    1.290    0.056    1.307    0.057 {built-in method _imp.create_dynamic}
      426    0.927    0.002    0.927    0.002 {imshow}
        1    0.635    0.635   38.273   38.273 motion_detector.py:6(<module>)
      142    0.438    0.003    0.438    0.003 {findContours}
      284    0.348    0.001    0.348    0.001 {putText}
      142    0.252    0.002    0.252    0.002 {dilate}
```

Ejecución de Stats10 del video grabado con *resize* de 100 (sección 3.3.8):

```
prof_rpi_vid2.out% stats 10
Wed Aug  7 00:23:49 2019    prof_rpi_vid2.out

         118628 function calls (114880 primitive calls) in 29.522 seconds

   Ordered by: internal time
   List reduced from 1304 to 10 due to restriction <10>

   ncalls  tottime  percall  cumtime  percall filename:lineno(function)
      143   14.858    0.104   14.858    0.104 {resize}
      144    9.410    0.065    9.410    0.065 {method 'read' of 'cv2.VideoCapture' objects}
       23    1.227    0.053    1.243    0.054 {built-in method _imp.create_dynamic}
      426    0.681    0.002    0.681    0.002 {imshow}
        1    0.547    0.547   29.529   29.529 motion_detector.py:6(<module>)
      142    0.479    0.003    0.479    0.003 {waitKey}
      143    0.375    0.003    0.375    0.003 {GaussianBlur}
      284    0.207    0.001    0.207    0.001 {putText}
      317    0.162    0.001    0.162    0.001 {built-in method builtins.compile}
      173    0.159    0.001    0.159    0.001 {built-in method marshal.loads}
```

Los primeros resultados son del perfilado del python sin ningún video, por lo que la función del motion_detector.py no sale en estos resultados. La tabla 1 muestra los resultados de las funciones de interés, las cuales son resize, GaussianBlur y motion_detector.py. Se puede observar que las llamadas de esas funciones aumentan al utilizar el video grabado con la cámara del Raspberry Pi 2. Adicionalmente, los tiempos de las tres funciones se incrementan al utilizar el mismo video.

No obstante, llama la atención el caso de la función GaussianBlur cuyo tiempo con video y resize es menor que el tiempo sin video. Posiblemente, la causa sea que se produzcan varios errores internos debido a la ausencia de un archivo para su procesamiento, sea éste una imagen o un video.

La tabla 2 muestra las diferencias de los tiempos entre el perfilado con video y el perfilado con video y resize (Tiempo de video y resize menos tiempo de video). Para el caso de la función resize, se da un incremento de tiempo esperado que ronda el 9%, debido a que es requerida para ajustar el tamaño del video. En el caso de GaussianBlur y motion_detector.py, se da una reducción de los tiempos como se espera, debido a la reducción de la cantidad de datos a utilizar por ambas funciones.

Tabla 1. Resultados de las funciones resize, GaussianBlur y motion_detector.py.

Función		GaussianBlur	resize	motion_detector.py
Llamadas	Python	43	44	0
	Video	143	143	1
	Video con resize	143	143	1
Tiempo acumulado	Python	1,901	1,422	0
	Video	5,424	13,646	38,273
	Video con resize	0,375	14,858	29,529
Tiempo acumulado por llamada	Python	0,044	0,032	0,000
	Video	0,038	0,095	38,273
	Video con resize	0,003	0,104	29,529

Tabla 2. Diferencias entre tiempos con video y tiempos con video y resize.

Función		GaussianBlur	resize	motion_detector.py
Diferencias de tiempos	Acumulado	-5,049	1,212	-8,744
	Por llamada	-0,035	0,009	-8,744
	Porcentaje	-92,1%	9,0%	-22,8%

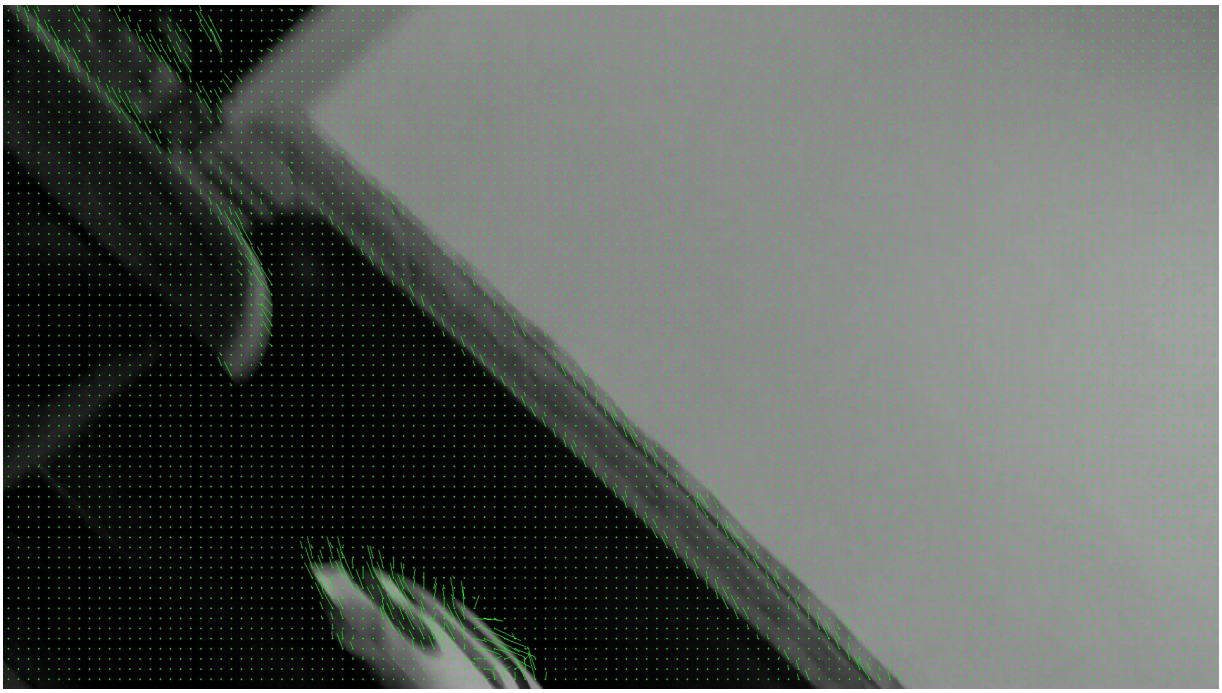
Resultados de Jetson Nano

Ejecución de Stats10 del video con motion_detector.py (sección 3.4.3):

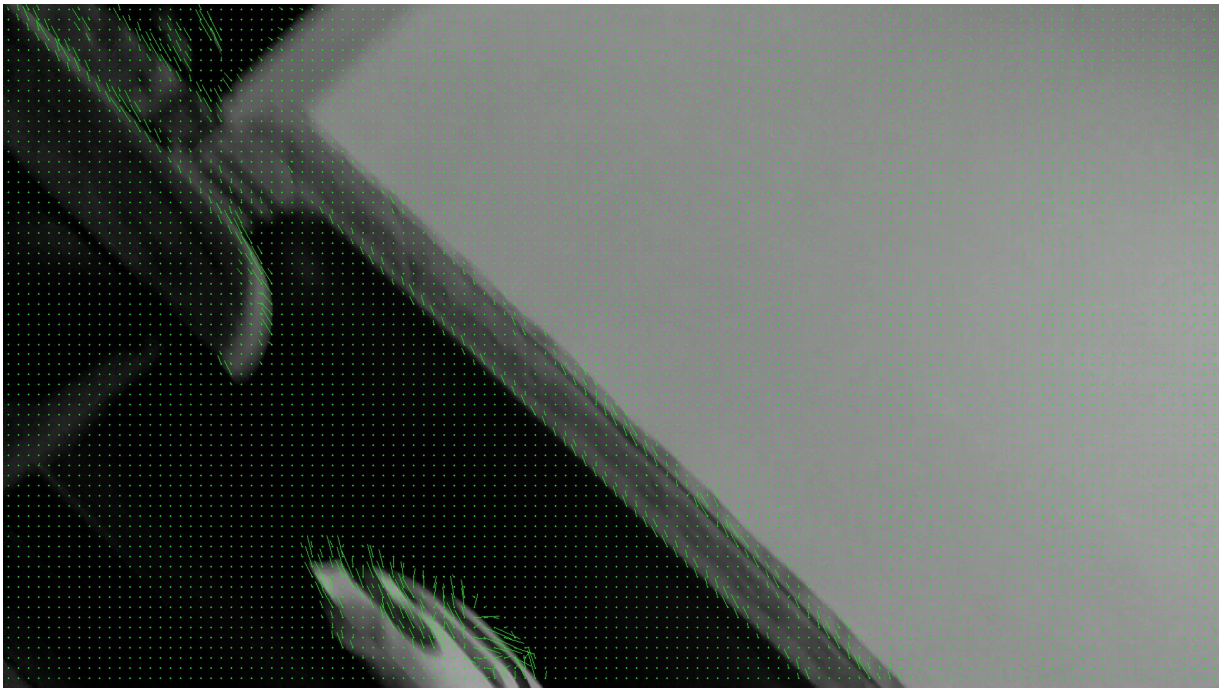
```
Tue Aug  6 22:34:18 2019    prof_jetson_vid1.out

         127985 function calls (124303 primitive calls) in 8.729 seconds

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

   ncalls  tottime  percall  cumtime  percall filename:lineno(function)
      143    4.095    0.029    4.095    0.029 {resize}
      144    1.963    0.014    1.963    0.014 {method 'read' of 'cv2.VideoCapture' objects}
      142    0.476    0.003    0.476    0.003 {waitKey}
      143    0.475    0.003    0.475    0.003 {GaussianBlur}
       22    0.307    0.014    0.318    0.014 {built-in method _imp.create_dynamic}
      426    0.203    0.000    0.203    0.000 {imshow}
      142    0.133    0.001    0.133    0.001 {findContours}
      143    0.103    0.001    0.103    0.001 {cvtColor}
        1    0.075    0.075    8.732    8.732 motion_detector.py:6(<module>)
      172    0.065    0.000    0.065    0.000 {built-in method marshal.loads}
```

Ejecución de la aplicación de Jetson Nano (sección 3.4.4):

CPU frame



GPU frame



Respuesta de la pregunta de la sección 3.4.5:

Las 3 funciones que más tiempo requieren para la GPU son boxFilter5, gaussianBlur y updateMatrices, todas del paquete optflowfarneback. (sección 3.4.5). De parte de API, las 3 funciones que más tiempo requieren son cudaFree, cudaMallocPitch y cudaStreamSynchronize

Respuesta de la pregunta de la sección 3.4.6:

El comando utilizado es sudo /usr/local/cuda/bin/nvprof --log-file nvprof_jetson_gputrace_5sec.log --print-gpu-trace --timeout 10 python3 gpu-opt_flow.py

Respuesta de la pregunta de la sección 3.4.7:

Resultados de métricas del nvprof con la aplicación cpu-out_flow.py:

No es posible obtener resultados debido a que el CPU no usa CUDA.

Referencias:

[1] NumPy.org https://numpy.org/

[2] Where X=cmake. Learn X in Y minutes. https://learnxinyminutes.com/docs/cmake/

[3] Mukherjee, A. Is it worth replacing autotools with cmake? Quora. 21 de enero de 2019. https://www.quora.com/Is-it-worth-replacing-autotools-with-cmake

[4] What are the differences between Autotools, Cmake and Scons? stackoverflow. https://stackoverflow.com/questions/4071880/what-are-the-differences-between-autotools-cmake-and-scons

[5] Johnson, J. Python Numpy Tutorial. http://cs231n.github.io/python-numpy-tutorial/

[6] Malik, F. Why Should We Use NumPy? Medium. 31 de Marzo de 2019. https://medium.com/fintechexplained/why-should-we-use-numpy-c14a4fb03ee9