Medición de eficiencia de algoritmos de visión artificial implementados en raspberry pi y ordenador personal mediante python

Efficiency Measurement Machine Artificial Vision Algorithms Imple-mented in Raspberry Pi and Personal Computer Using Python*

Recibido: 11 de julio de 2016- Aceptado: 6 de septiembre de 2016

Para citar este artículo: G. Cavanzo, M. Pérez, F. Villavisan, «Medición de eficiencia de algoritmos de visión artificial implementados en raspberry pi y ordenador personal mediante Python», *Ingenium*, vol. 18. n.° 35, pp. 105-119, febrero. 2017.

Gloria Andrea Cavanzo Nisso**
Miguel Ricardo Pérez Pereira***
Fabián Villavisan Buitrago****

Resumen

Este artículo presenta la medición de tiempo de ejecución de los dos algoritmos de visión artificial más conocidos (algoritmo de seguimiento de color y algoritmo de substracción de fondo) en dos plataformas diferentes (Raspberry Pi 3 Vs PC) ambas con sistemas operativos basados en Linux, esto con el fin de determinar en cuál de las dos plataformas los algoritmos corren con mayor eficiencia, además determinar si migrar estos algoritmos a sistemas embebidos afecta su desempeño para sistemas en tiempo real. En este artículo se mostrará una explicación detallada de cada algoritmo como también explicación técnica

* Grupo de Investigación: Metis (Grupo de Investigación en Informática Organizacional), Universidad patrocinadora: Universidad Distrital Francisco José De Caldas.

** Ph. D. (c). Ingeniería de Sistemas, Universidad Nacional, Bogotá, Colombia, M. Sc. Ciencias Matemática, Universidad Nacional, Bogotá Colombia. Docente de planta Universidad Distrital Francisco José de Caldas, Bogotá, Colombia. Grupo de Investigación: Metis. E-mail: gacavanzon@udistrital.edu.co

*** Especialista en Pedagogía y Docencia Universitaria, Universidad de San Buenaventura, sede Bogotá, Colombia, Ing. Control e Instrumentación Electrónica, Universidad Distrital (FJC), Bogotá, Colombia. Docente de planta Universidad Distrital Francisco José de Caldas, Bogotá, Colombia. Grupo de Investigación: Roma. E-mail: mrperezp@udistrital.edu

**** Ing (c). Control, Universidad Distrital (FJC), Bogotá, Colombia. E-mail: jfvillavisanb@correo.udistrital.edu.co

de cada plataforma y una medición de tiempo de cada subsistema del algoritmo y al final una medición global del tiempo de ejecución de todos los algoritmos.

Palabras clave

Eficiencia, linux, openCv, python, sistema embebido, visión artificial.

Abstract

This paper evaluate the time measure between two artificial vision algorithms (color tracking algorithm and background subtraction algorithm) using two different platforms (Raspberry Pi 3 Vs PC) based on Linux. The main goal is identify which algorithm runs more efficiently, also determine if migrate these algorithms affect their performance on embedded systems for real-time processing. This article presents detailed explanation of each algorithm, platform and timing of each subsystem, and also a global execution measure time of all algorithms.

Keywords

Artificial vision, efficiency, embedded system, Linux, openCv, python.

I. Introducción

Los sistemas embebidos son ideales para implementar algoritmos que realizan tareas dedicadas y que necesiten trabajar en tiempo real, las cuales generan un costo económico elevado si se implementaran en un ordenador convencional, pero no solo el costo sería una desventaja al usar un ordenador convencional, en aplicaciones donde se requiera portabilidad y/o bajo consumo de energía esta solución presentaría una gran dificultad. Es el caso de R Neves. A Matos [1]. En donde aprovechan las características de la Raspberry Pi modelo B para implementar un sistema de visión estéreo y algoritmos paralelos de visión artificial, para detección de obstáculos sobre un vehículo marítimo, un sistema como el anterior presentaría un consumo de energía bastante alto si un computador convencional fuera utilizado, otro punto a tratar es el sistema necesario para suministrar dicha energía, ya que dicho sistema sobrepasaría la aplicación. Por otra parte encontramos el sistema de cobro de peaje mediante visión por computador desarrollado por A. Suryatali y V. B. Dharmadhikari [2] el cual fue implementando sobre plataformas Linux en Raspberry Pi, dicho sistema automáticamente detecta el tipo de vehículo que ingresa al peaje y posteriormente genera su respectivo cobro, disminuyendo el costo de ejecución del proyecto considerablemente mediante el uso de sistemas embebidos, si se tratase de una gran cantidad de cabinas de cobro de peaje, por último G. Cocorullo, P. Corsonello [3] implementan un sistema de video vigilancia de bajo costo mediante un algoritmo de sustracción de fondo, el cual presenta un desempeño aceptable.

En este artículo se implementan dos algoritmos típicos de visión artificial (rastreo de objeto mediante color y sustracción de fondo) en dos plataformas diferentes (Raspberry Pi 3 y ordenador personal) y se muestra un comparativo de tiempo de ejecución de ambos

algoritmos en ambas plataformas, con el fin de establecer cuál es el recurso en tiempo en estas plataformas y si es posible trabajar en tiempo real en plataformas embebidas.

II. II. Algoritmos

Los algoritmos de seguimiento de color y sustracción de fondo son implementados en gran parte de los sistemas de procesamiento de vídeo, entre sus usos se encuentran; la compresión de vídeo, la video vigilancia, el control de tráfico, interacción entre humano-ordenador, las consolas de videojuegos entre otras aplicaciones, debido a lo anterior estos dos algoritmos resultan ser candidatos perfectos para este estudio de velocidad de ejecución en plataformas.

2.1 Algoritmo de seguimiento de color

El primer algoritmo a implementar es el algoritmo de seguimiento de color, que consiste en que el algoritmo detecta automáticamente objetos de un color determinado en una imagen y hace seguimiento de este color en tiempo real, dicho algoritmo puede ser calibrado para adaptarse a objetos de diferentes colores con diferentes brillos y contraste, los elementos fundamentales de este algoritmo se muestran y se explican en la siguiente sección.

2.1.1 Diagrama de flujo

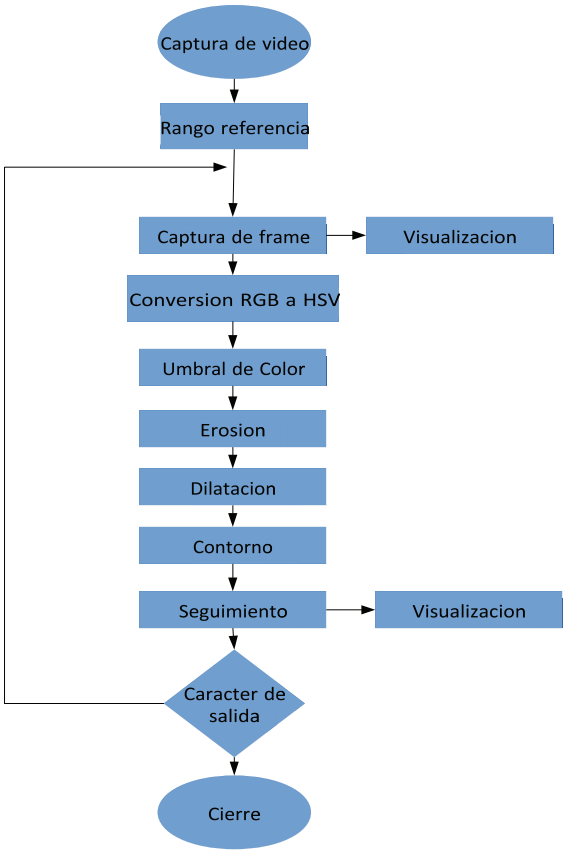


Figura 1. Diagrama de flujo algoritmo de seguimiento color. (Fuente autor).

En el diagrama de flujo de la figura 1 muestra los elementos fundamentales del algoritmo, el primer paso para la ejecución del algoritmo es capturar un frame o imagen proveniente de una cámara convencional esto de realizar la inicialización de la cámara (captura de vídeo en figura 1), después se realiza un acondicionamiento de dicha imagen, este acondicionamiento se hace con el fin de descartar la información que no es relevante para el sistema, es por esto que se aplican cambios en la imagen mediante trasformaciones espaciales (RGB a HSV), el cambio de espacio de colores primarios RGB a un espacio de progresión de color HSV es necesario debido a la facilidad de este espacio en segmentar un color por medio de un intervalo de ángulos en coordenadas H (Matiz) del espacio HSV (figura 2a) o implementar un filtro de alta frecuencia en S (Niidez) [4], actividad que presenta un poco más de complejidad matemática y ambiental si se realiza en el espacio RGB (figura 2b), ya que en este espacio una sombra y/o un brillo cambia con facilidad el tono del color a rastrear.

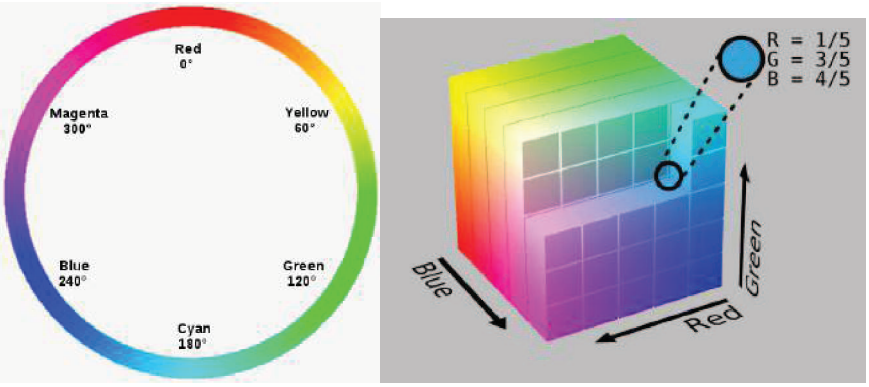


Figura 2: (a [5]) Representación de Matiz en plano HSV, (b [6]) Representación RGB

Siendo I(R, G, B) la imagen capturada se aplica la ecuación (1), para obtener f(H,S,V), donde: La transformación con $R, G, B \in [0,255]$ a $H \in [0°, 180°], S, V \in [0,1]$ y Min=min(R, G, B) y Max= max(R, G, B) [7]:

$$H = \begin{Bmatrix} indefinido, para\ Max = Min \\ 60° \frac{(G-B)}{Max-Min} + 0°, si\ Max = R\ y\ G \geq B \\ 60° \frac{(G-B)}{Max-Min} + 360°, si\ Max = R\ y\ G < B \\ 60° \frac{(B-R)}{Max-Min} + 120°, si\ Max = G \\ 60° \frac{(R-G)}{Max-Min} + 240°, si\ Max = B \end{Bmatrix} \quad (1)$$

$$S = \begin{Bmatrix} 0, si\ Max = 0 \\ 1 - \frac{Min}{Max}, en\ otro\ caso \end{Bmatrix}$$

$$V = Max$$

El siguiente paso del algoritmo es realizar una binarizacion por medio de la selección de un umbral. Usualmente, una binarizacion es obtenida desde una imagen en escala de grises

[8], pero en este caso no resulta necesario implementar esta técnica, ya que se puede ejecutar esta técnica directamente con la imagen modificada $f(H,S,V)$ por el algoritmo ejecutado en la etapa anterior, mediante la ecuación (2) se realiza dicha binarizacion:

$$b(n) = \begin{cases} 1 & \text{if } f(H,S,V) \geq T \\ 0 & \text{if } f(H,S,V) < T \end{cases} \qquad (2)$$

El resultado de la binarizacion se puede observar en la figura 3, la figura 3 (a) muestra la imagen en el formato HSV y la figura 3 (b) muestra el resultado de la binarizacion.

Figura 3 a) HSV b) Binarizada. (Fuente autor)

La conversión a de la imagen de una matriz HSV a una matriz binaria resulta importante en cuanto a las operaciones matemáticas, ya que estas se reducen a ecuaciones booleanas, por ende el procesamiento es más eficiente debido a la disminución considerable de información.

Obtenida ya b(n) como la imagen binarizada, es importante determinar el contorno del elemento que se desea rastrear, para este paso es útil la implementación de técnicas de filtrado basadas en la morfología de los elementos u objetos que se encuentran en la imagen, destacando únicamente las estructuras de los objetos y/o su esqueleto, la dilatación y la erosión* son operaciones básicas de procesamiento morfológico [9], en este caso tan solo con erosionar la imagen es suficiente:

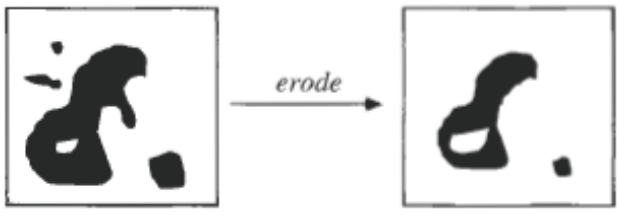


Figura 4. a) Imagen (Binarizada) b(n), b) imagen erosionada [9]

* Erosión: Este filtro es un proceso de suavizado de la imagen o de los objetos contenidos en la imagen, pero también es un filtro de tamaño, es decir este filtro también reduce la cantidad de objetos de tamaño pequeño (tamaño determinado por la matriz estructurante B) contenidos en la imagen, en general los objetos aislados son eliminados, la figura 4 muestra los resultados de la erosión aplicada a una imagen.

La imagen $E(\mathrm{n})$ es resultante de la erosión a la imagen $b(\mathrm{n})$ mediante la matriz estructurante B_{nxm}, donde dicha matriz puede ser cuadrada, diagonal o un vector, al superponer los centros de $b(\mathrm{n})$ y B_{nxm} y aplicando la ecuación 3, y realizando una traslación de por $B_{nxm}\ por\ b(\mathrm{n})$:

$$E(n) = B_{nxm} \wedge b(n) \tag{3}$$

Donde los pixeles de $b(\mathrm{n})$ totalmente contenidos en B_{nxm} se retendrán, mientras que los que no, se asignaran como 0 lógico.

Por último se superpone la imagen resultante (pintada de algún color) sobre la imagen original con el fin de visualizar en tiempo real el seguimiento del objeto, ver resultados del algoritmo en la figura 5.

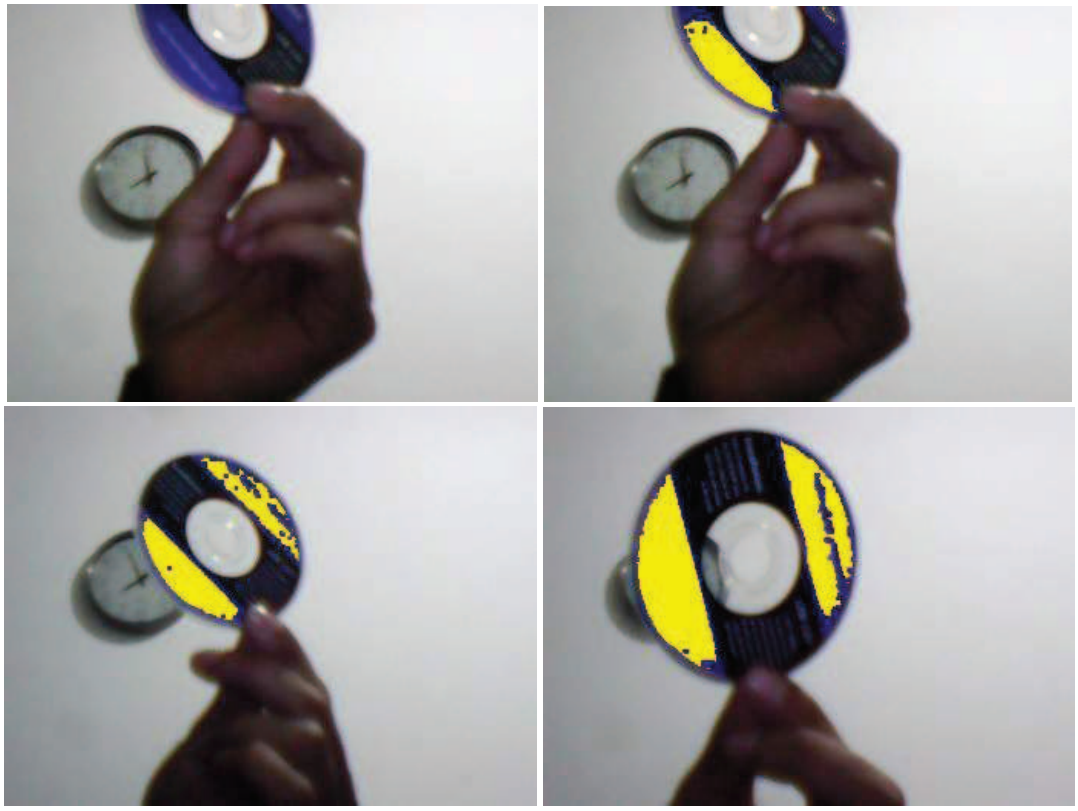


Figura 5 resultado final del algoritmo de seguimiento de color (Fuente autor)

2.2 Algoritmo de sustracción de fondo

El algoritmo de sustracción de fondo se utiliza para encontrar las diferencias entre dos imágenes, este algoritmo resta de una imagen prototipo la imagen actual tomada por la cámara, y así determina que cambia entre escena y escena, este algoritmo es la base fundamental de algoritmos más complejos implementados en sistemas de detección de movimiento, video vigilancia automática y rastreo de objetos en tiempo real.

2.2.1 Diagrama de flujo

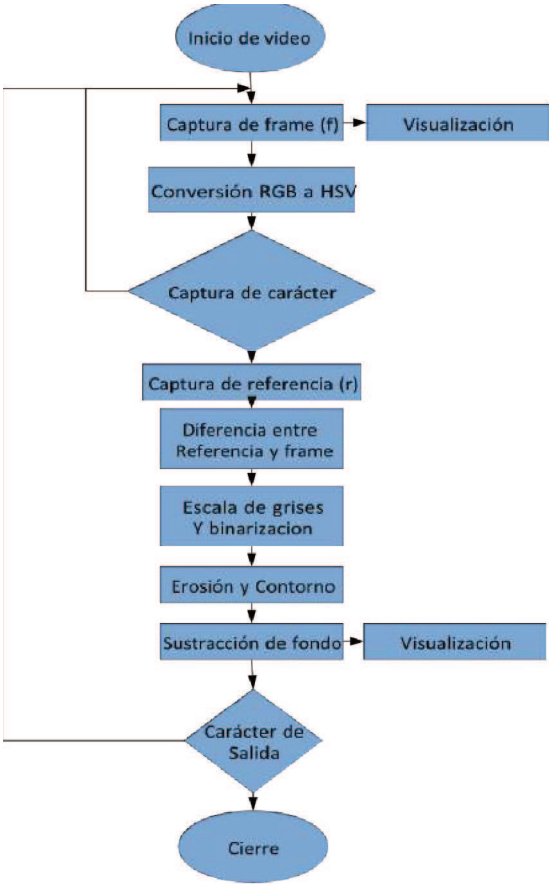


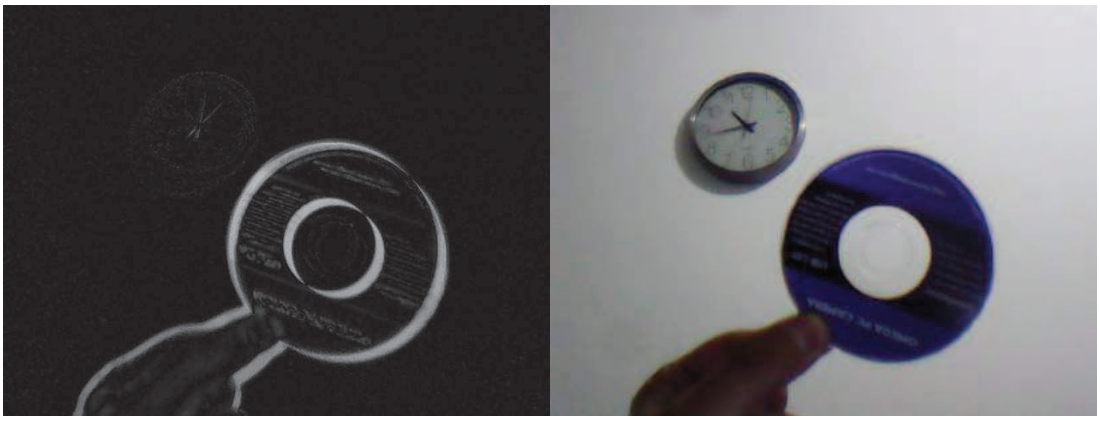
Figura 6 Diagrama de flujo algoritmo sustracción de fondo (fuente autor)

El primer paso del algoritmo de sustracción de fondo es tomar la imagen referencia $r\,(\mathrm{R,G,B})$, dicha imagen es la encargada de decirle al algoritmo los parámetros nominales de la escena. Dicha imagen es restada pixel a pixel con el frame capturado en tiempo real $f\,(\mathrm{R,G,B})$, esta resta es una resta matricial dada por:

$$I_{ij} = r_{ij} - f_{ij} \qquad (4)$$

La imagen resultante $I\,(\mathrm{R,G,B})$ es de tamaño mxn al igual que $r\,(\mathrm{R,G,B})$ y $f\,(\mathrm{R,G,B})$

Como en el algoritmo de seguimiento de color es necesario realizar una eliminación de ruido, objetos e información innecesaria, para dicho proceso la translación de espacios es bastante importante, pero en este caso es una conversión del espacio de color al espacio de intensidad de grises, esta transformación disminuye considerablemente la cantidad de información existente en la imagen, ya que se pasa de una imagen $I\,(\mathrm{R,G,B})$ con 24 bits a una imagen de tan solo 8 bits, ver resultado en la figura 7, la cual es una imagen con menos datos y que brinda la misma información, esto con el fin de volver más eficiente el procesamiento, y así facilitar la eliminación de la información innecesaria.



(a) (b)

Figura 7 a) Imagen a color b) Imagen a escala de grises, (Fuente Autor)

El proceso anterior se puede hacer de múltiples maneras; promedio, luminancia, saturación, descomposición, entre otras [10]. Ya que la clave para procesar cualquier imagen es obtener o identificar las regiones. [11] Para este caso se mostrarán dos (promedio y luminancia), las cuales son las más básicas y son las implementadas por python con la librería OpenCv, el promedio y la luminancia están determinados respectivamente por las siguientes ecuaciones mostradas en 5:

$$g(n) = \frac{R + G + B}{3} \quad (5)$$

a) Conversión escala de grises promedio,

$$\left(g(n) = R(0.299) + G(.0587) + B(0.114)\right) \quad (6)$$

b) Conversión escala de grises por luminancia

Posterior a este procedimiento se implementa el mismo filtro (erosión de imagen)), que fue implementado en el algoritmo anterior, y resulta la matriz , imagen sin ningun ruido, la cual posee la deteccion de fondo. Por ultimo se superponen a la imagen rectángulos con las coordenadas de cada uno de los elementos detectados en para visualizar en tiempo real los elementos diferentes al fondo referencia , la figura 8 muestra una sucesión de imágenes donde se muestra el resultado de este algoritmo.

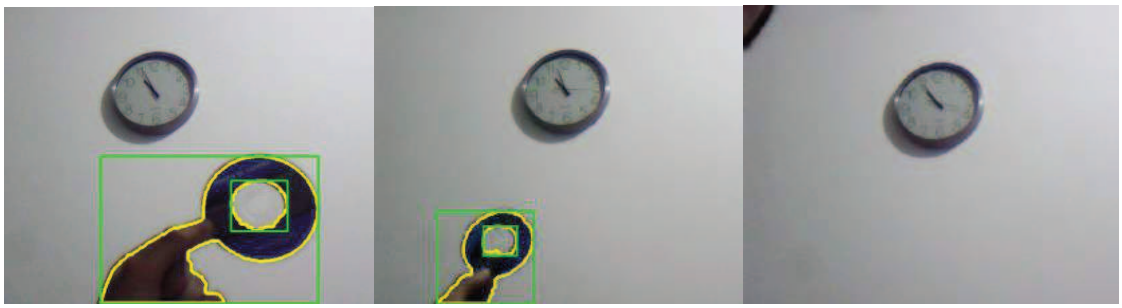


Figura 8 Resultado del seguimiento de objeto. (Fuente autor)

III. Plataformas empleadas

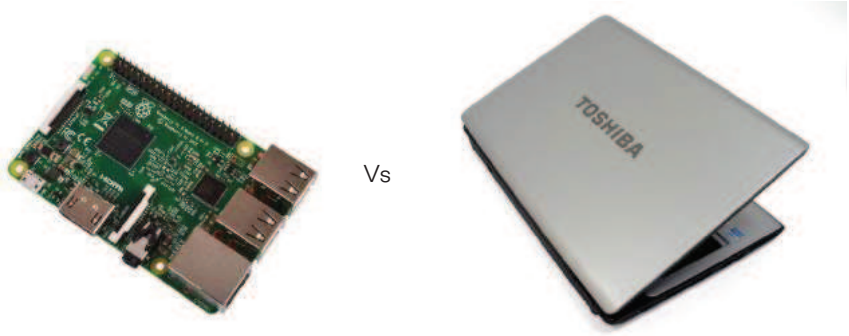


Figura 9: a).Ordenador personal [12], b) Raspberry Pi 3 [13]

Para realizar la comparación se implementaron los dos algoritmos en dos plataformas diferentes, las plataformas empleadas son un sistema embebido, Raspberry Pi modelo 3, y un ordenador convencional ver figura 9, ambos basados en Linux, por una parte se encuentra la Raspberry la cual implementa Debian en su versión para Raspberry, y por otra se encuentra un computador de recursos intermedios (más adelante se habla sobre esto) con Linux en su versión de Ubuntu, el software implementado para el desarrollo de los algoritmos anteriormente mencionados es python y su plataforma de desarrollo Stani's Python Editor (SPE), al igual que las librerías para procesamiento de vídeo «OpenCv», la librería para operaciones matemáticas «matplotlib» y por último la librería encargada de medir los tiempos de ejecución de todas y cada una de las instrucciones «time».

3.1 Ordenador personal empleado

El ordenador utilizado para la elaboración de este documento es un ordenador de características bajas, es un ordenador de fácil acceso, como el que se usa en colegios, universidades o centros académicos, las características de dicho equipo se denotan a continuación.

3.1.1 Características

- Procesador Intel core i5-2450 @ 2.50 Ghz
- 6092200kB de memoria RAM
- Sistema operativo Linux Ubuntu v16.04
- Software: SPE v0.8.4 librería OpenCv '2.4.9.1'

3.2 Raspberry Pi empleado

La Raspberry empleada para la elaboración de este documento es la Raspberry Pi 3, ya que posee características que se aproximan a las de cualquier ordenador básico de costo medio, dichas características son las presentadas en el ítem siguiente.

3.2.1 Características de la Raspberry Pi 3 modelo B

- A 1.2GHz 64-bit quad-core ARMv8 CPU
- 1000000 kB de memoria RAM
- Sistema operativo Linux Raspbian jessie; Kernel Version 4.4
- Cámara de 800x600 pixeles*

IV. Evaluación de rendimiento

Existen factores que influyen a la hora de determinar la eficiencia de un algoritmo, dichos factores son de complejidad espacial (recursos físicos) y de complejidad temporal, en este artículo se mostrará el análisis del factor temporal; la eficiencia temporal de un algoritmo, medida por tiempos de ejecución es una técnica clásica para comparar las prestaciones de diferentes ordenadores. Existen unos conjuntos de programas destinados a facilitar esta comparativa entre ordenadores, llamados Benchmark [14]. En python existen dos librerías capaces de entregar medidas de tiempos de ejecución de un algoritmo o instrucción, medidas útiles para su respectiva comparación, dichas librerías son: la librería «time» y la librería clock, en este documento los datos se basan en los resultados arrojados por la librería «time».

4.1 Medición de velocidad del algoritmo de seguimiento de color en las dos plataformas

Las siguientes gráficas muestran los resultados del tiempo de ejecución de cada una de las instrucciones implementadas en la Raspberry Pi (color azul) vs. Ordenador (color rojo), dichas gráficas son el resultado de varias mediciones para lograr un tiempo promedio que pueda ser válido y estable para el estudio, también se muestra una gráfica del tiempo de ejecución total de cada algoritmo.

La primera etapa según 2.1.1 es la captura de una imagen, la figura 10 muestra el tiempo de ejecución de dicha instrucción en las dos plataformas, en donde el tiempo promedio de ejecución para pc es de 70ms (gráfica azul) y el tiempo promedio para la Raspberry Pi es de 20 ms (gráfica roja).

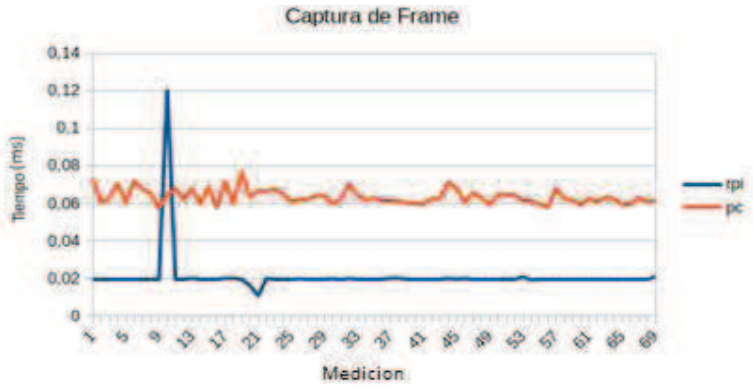


Figura 10 Tiempo de captura de Frame. (Fuente autor)

* La cámara implementada en ambos algoritmos fue la misma, esto con el fin de procesar la misma cantidad de información, en cada una de las plataformas.

La etapa que continúa en el algoritmo de seguimiento de color es la transformación espacial, mencionada en la sección 2.1, la figura 11 presenta la comparativa de los tiempos que le toma a los dispositivos ejecutar la ecuación 1. Con un tiempo promedio de 30 ms para la Raspberry Pi 3 y 5 ms para el ordenador personal.

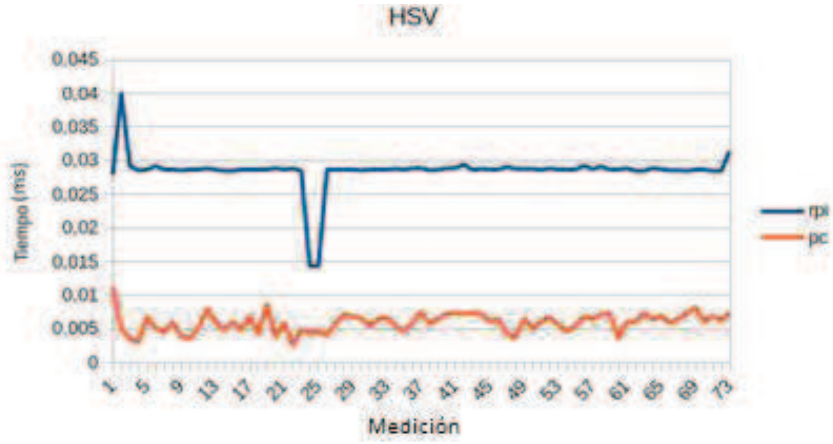


Figura 11 Tiempo de conversión de espacio RGB a HSV. (Fuente autor)

El umbral para la binarizacion es ejecutado según la ecuación 2, es una operación matemática más sencilla para el proceso, lo cual se ve reflejado directamente en las gráficas (figura 12) de los tiempos que conlleva a cada dispositivo ejecutar dicha instrucción.

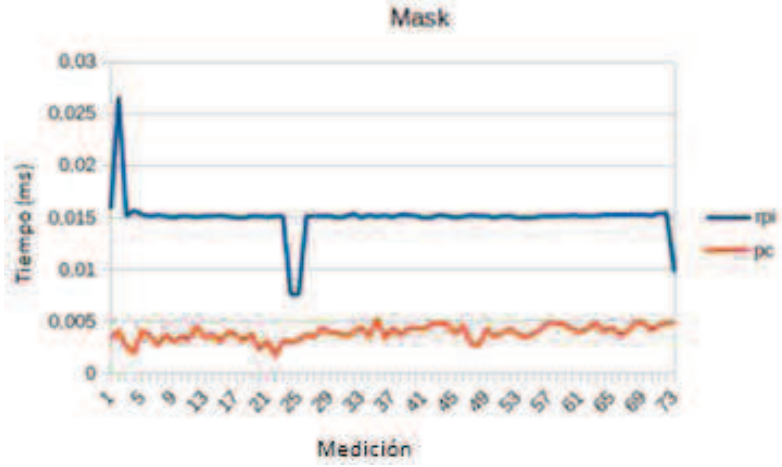


Figura 12 Tiempo de ejecución instrucción «Mask» (Fuente autor)

Por último queda la gráfica de erosión de la imagen, en donde los tiempos presentan una gran diferencia y aunque es una operación booleana, los tiempos de ejecución de la Raspberry Pi (25 ms) son aproximadamente 25 veces más lentos que los tiempos que le toma al ordenador (1m) completar esta instrucción. Esto debido a que es una función que se repite cantidad de veces (dependiendo del tamaño de la matriz estructurante al igual que el tamaño de la imagen), en la misma instrucción, estas repeticiones son acumulativas (en cuanto a tiempo), y directamente relevantes en la comparativa que trata de mostrar este documento. Véase la figura 13.

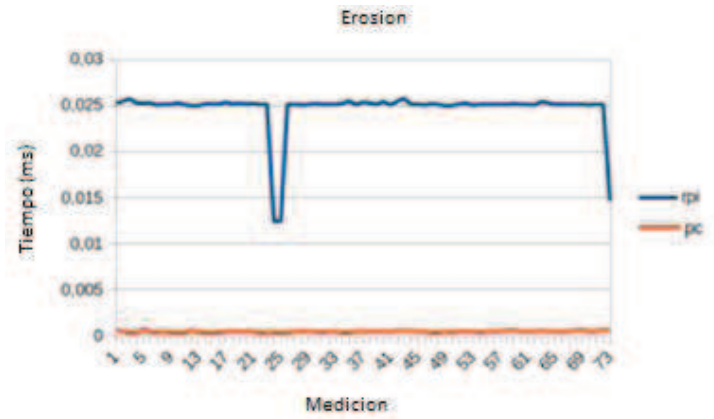


Figura 13 Tiempo de erosión. (Fuente autor)

La publicación de los resultados visualmente, no le toma tiempos considerables a los dispositivos implementados (Raspberry Pi y pc), los cuales son presentados en la figura 14.

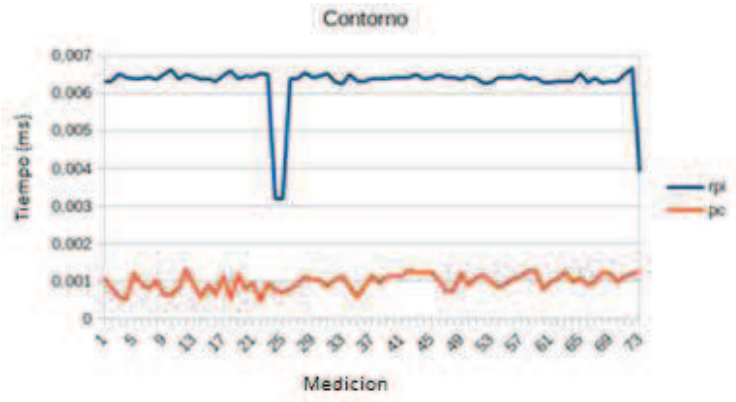


Figura 14 Tiempo de delimitación del contorno. (Fuente autor)

Por último la figura 15 muestra el tiempo de ejecución total del algoritmo en cada plataforma, con un resultado promedio de 50 ms para el procesamiento de la imagen por medio del ordenador personal y 200 ms promedio para el procesamiento de la imagen para el sistema embebido Raspberry Pi. Desde la captura de la imagen hasta la visualización por parte del usuario.

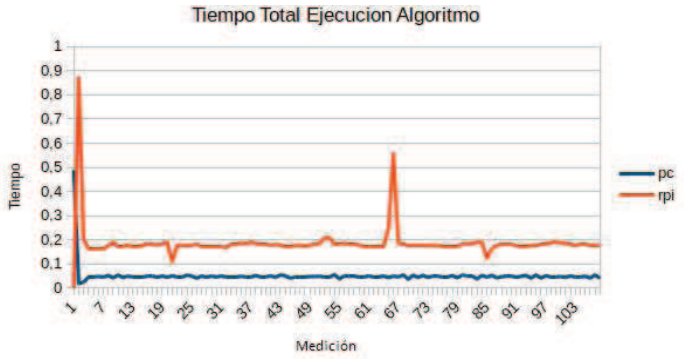


Figura 15 Tiempo total de ejecución de algoritmo de seguimiento de color. (Fuente autor)

4.2 Medición de velocidad del algoritmo de sustracción de fondo en las dos plataformas

Igual que el apartado 4.1, este apartado muestra en gráficas comparativas los resultados de tiempo de ejecución de las instrucciones del algoritmo de sustracción de fondo. Omitiendo al igual que en ítem 2.2 las instrucciones que se repiten en ambos algoritmos, se presenta a continuación en la figura 16 el tiempo que implica realizar la diferencia entre imágenes por cada uno de los dispositivos.

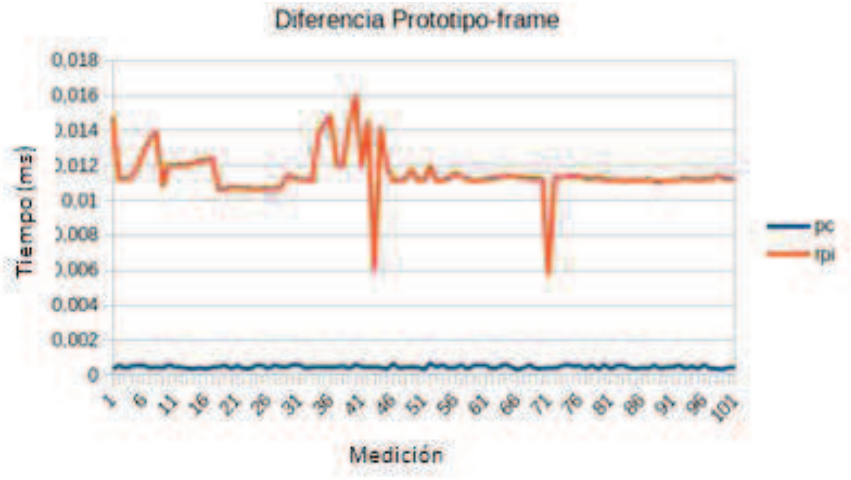


Figura 16 Diferencia entre imagen prototipo y actual (Fuente autor)

La gráfica de comparación del tiempo de conversión de RGB a escala de grises según la ecuación 5 del ítem 2.2, es la mostrada en la figura 17 con un tiempo promedio de 8 ms para el sistema embebido Raspberry Pi y 1 ms para el ordenador personal.

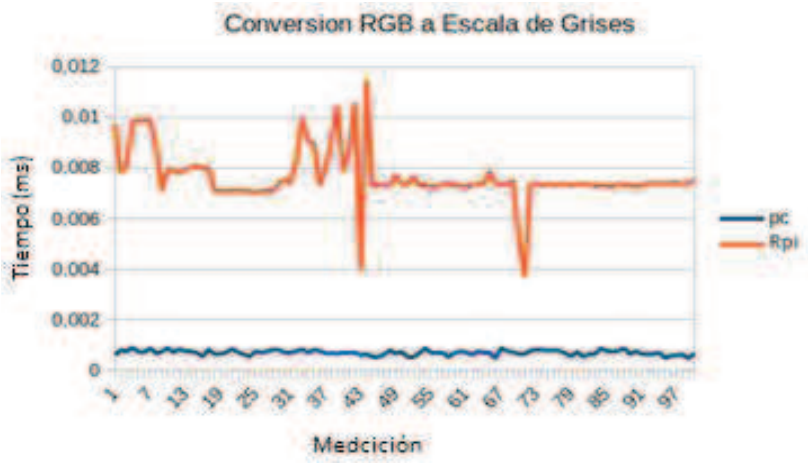


Figura 17 Tiempo de conversión RGB a Escala de gris. (Fuente autor)

Por último en cuanto a comparación de tiempos por instrucción, se obtuvo la figura 18 en donde se observa el tiempo que toma a cada uno de los sistemas implementar el umbral, para la binarizacion. Con tiempos de 3 ms aproximadamente para la Raspberry Pi y menos de 1 m para el ordenador personal que se implementó para la elaboración de este documento.

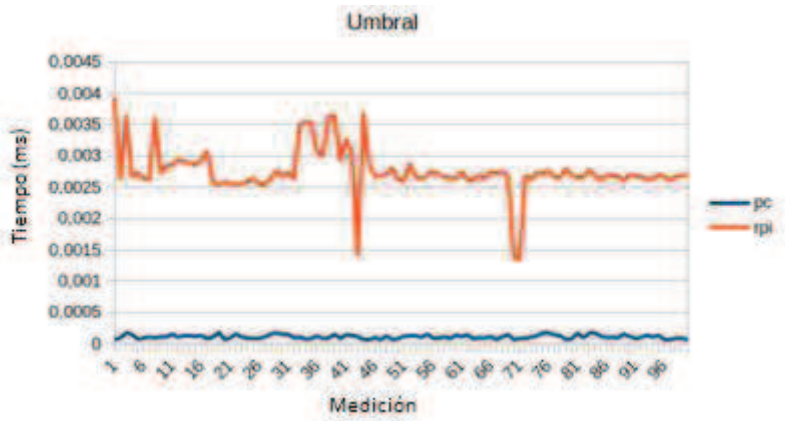


Figura 18 Tiempo ejecución binarizacion mediante Umbral. (Fuente autor)

La figura 19 muestra el tiempo total de ejecución del algoritmo de sustracción de fondo, con todas y cada una de las instrucciones ejecutadas consecutivamente por los dos dispositivos implementados.

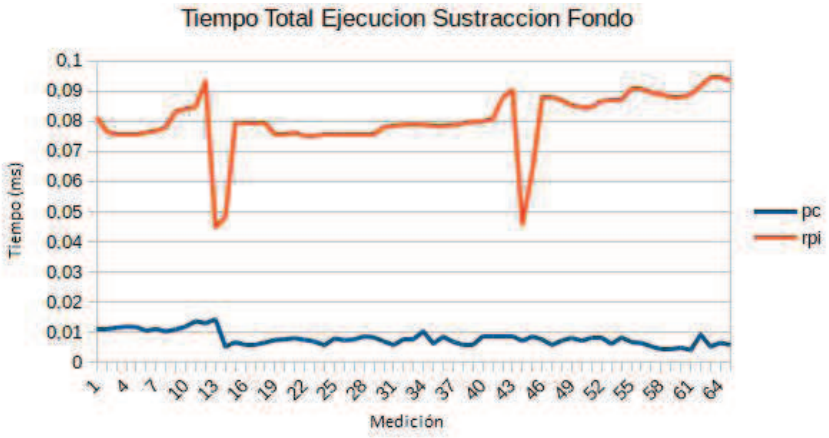


Figura 19 Tiempo ejecución total de algoritmo sustracción de fondo. (Fuente autor)

V. Resultados

Las gráficas mostradas en la sección IV evaluación de rendimiento de este documento muestran una diferencia poco amplia, en cuanto a tiempos de ejecución de todas y cada una de las instrucciones de procesamiento de imágenes que se implementaron, aunque si reflejan comportamientos diferentes, en cuanto a la estabilidad de dichos tiempos, se observa una fluctuación en los tiempos que le toma a la Raspberry Pi realizar el algoritmo de sustracción de fondo. Por otra parte en la gráfica mostrada en la figura 10 se observa la diferencia a favor que posee la Raspberry Pi para inicializar y tomar información de dispositivos periféricos.

La Raspberry Pi con las técnicas de filtrado morfológico presentan una gran desventaja en tiempos de ejecución comparando dichos tiempos en la gráfica 17. Aunque el tiempo de ejecución del algoritmo de sustracción de fondo en la Raspberry pi, se encuentra dentro de la tasa de refrescamiento normal (más de 12 fps) lo cual puede ser calificado como

procesamiento de vídeo (Imágenes en movimiento en una tasa normal para la percepción humana) en tiempo real.

VI. Conclusiones

Después de analizar los tiempos de procesamiento de la Raspberry PI 3 para ambos algoritmos (apartado 4 en el artículo), es concluyente que este dispositivo puede ser utilizado para procesar imágenes en tiempo real, ya que los tiempos medidos para este dispositivo son concordantes con los establecidos por la teoría para este tipo de procesamiento.

Se puede observar en las gráficas de medición, variaciones repentinas en el tiempo de ejecución, las cuales no son concordantes con mediciones previas, esto puede deberse a un error de la librería TIME la cual no resetea los contadores internos todas las veces.

Los algoritmos de filtrado basados en morfología, como lo es la erosión, no son adecuados para ejecutarse consecutivamente en dispositivos embebidos como la Raspberry Pi, ya que estos consumen mucho tiempo y retrasa el tiempo total de la aplicación .

Referencias

[1] R. Neves, A. Matos. (-). Raspberry PI Based Stereo Vision For Small Size ASVs. pp: 1-6.

[2] A. Suryatali, V. B. Dharmadhikari. (2015). Computer Vision Based Vehicle Detection for Toll Collection System Using Embedded Linux. International Conference on Circuit, Power and Computing Technologies (ICCPCT), -, Pg: 1-7.

[3] G. Cococrullo, P. Cornosello, F. Frustaci, L. Guacho, S.Perri. (-). Embedded Survillance System Using Background Subtraction and Raspberry PI. Department Of Electronics, Computer Sciences and Systems DIMES - University of Calabria, Pg: 1-5.

[4] K., Milos; P., Jiri; Sánchez Reyes, Jazmín. Evaluation of Quality in Imaging Systems Based on Psychovisual Attributes.INGENIUM, [S.l.], v. 13, n. 26, p. 73-77, Jun. 2013. ISSN 0124 - 7492. Disponible en: <http://revistas.usbbog.edu.co/index.php/ingenium/article/view/375/292>. Fecha de acceso: 05 Sep. 2016

[5] Color Models: RGB, HSV, HSL. 28 de mayo de 2016, de - Sitio web: https://en.wikibooks.org/wiki/Color_Models:_RGB,_HSV,_HSL.

[6] F. Horst. (22 March 2010). RGB Cube Show lowgamma cutout. Mayo 28 de 2016, de - Sitio web: https://commons.wikimedia.org/wiki/File:RGB_Cube_Show_lowgamma_cutout_b.png

[7] L. Moreno. (-). Modelos de Color. 28 de mayo de 2016, de desarrollador web Sitio web: www.desarrolloweb.com/articulos/1483.php.

[8] J. D. Gibson. (1999). Handbook of Image and Video Processing. Austin, Texas: al Bovik.

[9] Kharar, Mohali, Punjab, (2011). "Morpholgical image processing". India.Dept. of ECE, Doaba Institute of Engineering and Technology.

[10] (11 de octubre de 2011). Seven grayscale conversion algorithms. 28 de mayo de 2016, de Tanner Helland (dot) com Sitio web: www.tannerhelland.com/3643/grayscale-image-algorithm-vb6/

[11] F. H. Martínez, F. Martínez, H. Montiel, «Identificación visual sobre sistema embebido para navegación robótica autónoma», Ingenium, vol. 15, n. º 29, pp. 71-84, mayo, 2014.

[12] Recuperado de: www.pccomponentes.com/Raspberry_pi_3_modelo_b.html vista por última vez 28 mayo 2016

[13] Toshiba satélite. Recuperado de: www.notebookcheck.org/typo3temp/_processed_/csm_toshiba_satellite_l350_gesamtklein_1c605ee2d7.jpg

[14] A. Marzal, I. Garcia. (2002). Introducción al análisis de algoritmos. pp: 1-7.