ongr. Int. en Ing. Electrónica. Mem. ELECTRO, Vol. 42 pp. 190-195, Oct 2020, Chihuahua, Chih. México
http://electro.itchihuahua.edu.mx/memorias_electro/MemoriaElectro2018.zip
ISSN 1405-2172

Algoritmo de Detección de Movimiento en Secuencias de Video Basado en Arquitectura de Redes Neuronales Tradicionales

AUTORES
Luis X. Nevárez Ochoa, Mario I. Chacón-Murguía, Juan A. Ramírez Quintana
Laboratorio de Percepción Visual con Aplicaciones en Robótica
Instituto Tecnológico de Chihuahua
División de Estudios de Posgrado e Investigación
Ave. Tecnológico No. 2909, 31310, Chihuahua, Chih., MÉXICO
6141052807
luisxavier21@hotmail.com, mchacon@ieee.org, jaramirez@itchihuahua.edu.mx

RESUMEN.

El desarrollo de algoritmos para la detección de movimiento en secuencias de video es una parte muy importante en el análisis de escenarios mediante sistemas de visión artificial. Estos algoritmos generalmente son diseñados para correr en computadoras o PCs. Hoy en día, el número de sistemas embebidos, así como su capacidad de procesamiento ha ido aumentando, y con ello, su rango de aplicaciones. En el campo de la visión por computadora se han explorado algoritmos que puedan implementarse en tarjetas de desarrollo, como la Raspberry Pi, y tomar ventaja de su pequeño tamaño y capacidades computacionales. En el presente trabajo, se realizó el rediseño de dos algoritmos para la detección de movimiento, originalmente desarrollados en Matlab, para ser implementados en una Raspberry. La evaluación de las nuevas versiones reporta un comportamiento similar al de sus homólogos en PCs y, en ciertas condiciones, con mejor rendimiento a bajas resoluciones.

Palabras Clave: Detección de movimiento, Sistemas embebidos, Raspberry Pi.

ABSTRACT.

The development of motion detection algorithms in video sequences is a paramount aspect for analysis of scenarios by artificial vision systems. These algorithms are manly designed for PC architectures. Nowadays the number of embedded systems has been increasing, as well as their processing capacity and range of applications. In the field of computer vision algorithms that can be implemented on development cards, such as Raspberry Pi, have been explored, taking advantage of its small size and computational capabilities. This work presents the re-design of two algorithms for motion detection, originally developed in Matlab, to be implemented in a Raspberry. The evaluation of these new version of the algorithms report a behavior similar to their PC counterparts, and even, with better performance at low resolutions.

Keywords: Motion Detection, Embedded Systems, Raspberry Pi.

1. INTRODUCCIÓN

Desde los inicios de la computación se han buscado maneras en las que los dispositivos electrónicos puedan interactuar con el medio que los rodea, resultando en la creación de sensores que dan capacidades similares a las nuestras o a las de otras especies.

En la actualidad, existe un amplio campo de trabajo donde la visión por computadora encuentra utilidad. Existen diversos laboratorios e individuos alrededor del mundo que se encuentran investigando y desarrollando nuevos algoritmos de visión que no se quedan en operaciones básicas, sino que también pueden realizar actividades de aprendizaje y pueden dotar al sistema de la capacidad para mejorarse continuamente.

1.1. Análisis del estado del arte.

En la literatura podemos encontrar casos de implementación de algoritmos de visión sobre dispositivos de bajo consumo energético [1]-[3], en otros casos se implementan para sistemas de seguridad CCTV para el seguimiento de personas utilizando *Deep Learning* [4], [5]. Para la detección de objetos dinámicos mediante sistemas de visión se pueden utilizar redes neuronales [6], [7]. Por otro lado, hay quienes optan por emplear la lógica difusa para el reconocimiento del entorno en secuencias de imágenes o videos, tal como se realiza en [8]. Otra técnica de detección de movimiento es el algoritmo *Mean Shift*, el cual es un modelo estadístico descrito en [9], [10], con la diferencia de que en [10] se utiliza un algoritmo del tipo *Tracking-Learn-Detection* basado en *Mean Shift*.

El presente trabajo describe el rediseño de dos algoritmos de detección de movimiento originalmente escritos en Matlab para su implementación en un sistema embebido. Estos algoritmos están documentados en [11] y [12]. Una explicación más detallada sobre el funcionamiento de cada uno podrá encontrarse en la Secciones 2.1.1 y 2.1.2. La metodología empleada y mejoras realizadas se presentan en la sección 2.2. Finalmente, una comparativa entre los algoritmos propuestos en la literatura y los desarrollados para este artículo se encontrarán en la sección 3.

2. DESARROLLO

2.1. Algoritmo de Mediana Aproximada.

El algoritmo de Mediana Aproximada es una técnica de sustracción y actualización de fondo para la detección de

objetos dinámicos en escenarios con cambios paulatinos. Por la manera en la que es implementado, la actualización se encuentra en un ciclo permanente, independientemente de si existe movimiento en la escena o no, por lo que la cantidad de recursos que utiliza el sistema es casi constante. El algoritmo consta de tres etapas. La primera es la inicialización del modelo de fondo, que puede ser el primer cuadro que se capture al ejecutar el programa. La segunda etapa se refiere a la sustracción del fondo de referencia con el siguiente cuadro de video para determinar si existió movimiento o no. Por último, la tercera etapa lleva a cabo la actualización del modelo de fondo, donde se puede variar la velocidad con la que se absorben nuevos objetos. El comportamiento de esta última etapa se encuentra modelado en la ecuación (1):

$$f_b(x,y,t+1) = \begin{cases} f_b(x,y,t) + \dfrac{1}{\sigma} & si\ \ I(x,y,t) > f_b(x,y,t) \\ f_b(x,y,t) - \dfrac{1}{\sigma} & si\ \ I(x,y,t) < f_b(x,y,t) \end{cases} \quad (1)$$

donde $f_b(x,y,t+1)$ es el modelo de fondo en el siguiente instante de tiempo, $f_b(x,y,t)$ es el modelo de fondo actual, σ una constante que indica la velocidad a la que se refrescará el modelo de fondo y $I(x,y,t)$ es el cuadro de video actual.

Para obtener los pixeles que corresponden al movimiento se obtiene una matriz de diferencia por medio de la ecuación (2):

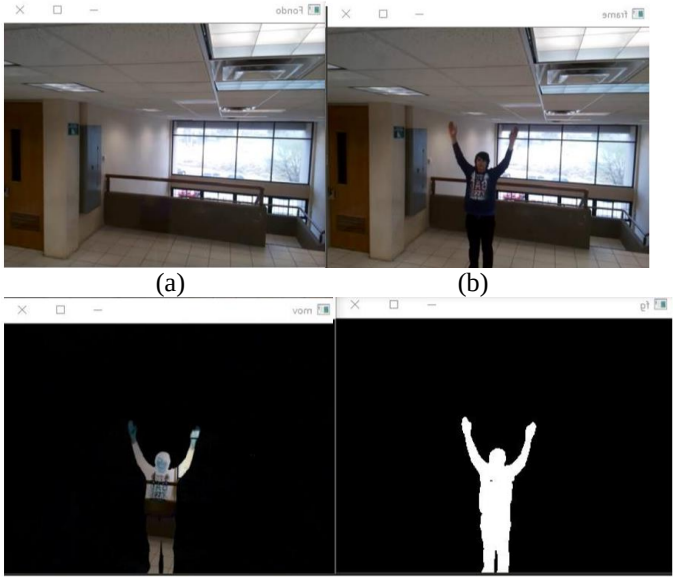
$$S_f(x,y,t) = |I(x,y,t) - f_b(x,y,t)| \quad (2)$$

$S_f(x,y,t)$ es una matriz de enteros positivos que indica los pixeles que fueron distintos entre el cuadro actual y el modelo de fondo.

Para que el ruido que presentan de manera natural las cámaras no afecte el rendimiento del algoritmo y provoque falsos positivos, se utiliza un umbral que discriminará los valores bajos de la matriz de diferencia. El resultado es otra matriz de valores lógicos que indican la presencia de movimiento:

$$f_g(x,y,t) = \begin{cases} 1 & si \quad S_f(x,y,t) > \tau \\ 0 & si \quad S_f(x,y,t) < \tau \end{cases} \quad (3)$$

donde $f_g(x,y,t)$ es la imagen con los objetos en movimiento y τ es el umbral para que este sea detectado. Una ilustración que representa el funcionamiento del sistema es la que se observa en la Figura 1. La Figura 1a muestra el modelo de fondo, la Figura 1b el cuadro que se procesa, la Figura 1c, una representación gráfica de la matriz de diferencia $S_f(x,y,t)$ y la Figura 1d, el movimiento que fue detectado y almacenado en $f_g(x,y,t)$.



(a) (b) (c) (d)

Figura 1. a) Modelo de fondo. b) Cuadro actual. c) matriz de diferencia. d) Movimiento detectado.

2.2. Algoritmo Pi3SOM

El segundo algoritmo desarrollado es el denominado Pi3SOM el cual, a diferencia del de Mediana Aproximada, lleva a cabo la actualización del fondo por medio de neuronas que procesan los valores HSV de cada píxel. Este modelo consta de dos fases: la inicialización del modelo de fondo y la actualización del fondo. El fondo inicial es el primer cuadro de la secuencia de video; la actualización del fondo se lleva a cabo por medio de una red neuronal artificial tipo SOM modificada. Cada neurona de la SOM recibe la información de los componentes HSV del píxel correspondiente a su posición espacial. Al inicio del algoritmo, los pesos de las neuronas serán los mismos que los componentes HSV del cuadro de referencia. En cuanto a los parámetros de aprendizaje, se definieron valores constantes, lo mismo aplicó para los valores de umbral que se utilizarían para la discriminación de movimiento. A continuación, se explican las fases del algoritmo.

La primera fase es el cálculo de las distancias euclidianas entre el modelo de fondo y el cuadro de la secuencia de video actual.

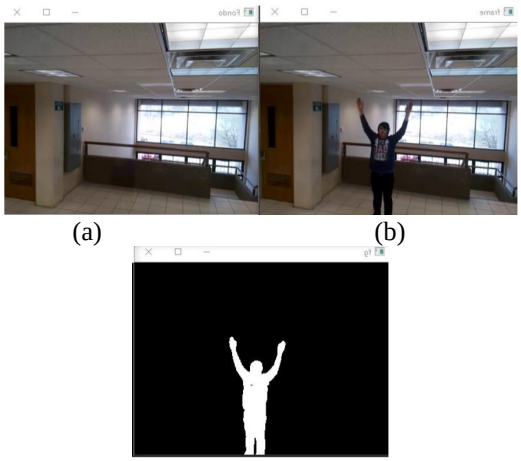
$$\begin{aligned} e_{ps}(x,y,t) = [[&p(x,y,t)^V\ p(x,y,t)^S \cos(\ p(x,y,t)^H, \\ &p(x,y,t)^V p(x,y,t)^S \sin(\ p(x,y,t)^H), \\ &p(x,y,t)^V - \\ &S(x,y,t)^V S(x,y,t)^S \cos(S(x,y,t)^H), \\ &S(x,y,t)^V S(x,y,t)^S \sin(S(x,y,t)^H), \\ &S(x,y,t)^V]] \end{aligned} \quad (4)$$

donde $p(x,y,t)^V, p(x,y,t)^S$ y $p(x,y,t)^H$ corresponden a los valores V, S y H del cuadro actual y $S(x,y,t)^V, S(x,y,t)^S$ y $S(x,y,t)^H$, a los del modelo de fondo.

Una vez obtenida la distancia entre los elementos, se procede a determinar la existencia de movimiento en la escena. Es necesario saber si ese píxel detectado como movimiento no es parte de una sombra. Para dicha tarea, se empleó una regla que permite al sistema clasificar si el píxel en cuestión puede considerarse como movimiento y no es una sombra proyectada por el objeto (5):

$$Si\ e_{ps}(x,y,t) > th_1\ \&\ |p(x,y,t)^V - S(x,y,t)^V| > th_2$$
$$Entonces, f_g(x,y,t) = 0 \quad (5)$$

donde th_1 es una constante de umbral para considerar la existencia de movimiento; $p(x,y,t)^V$ y $S(x,y,t)^V$ son información del cuadro actual y del modelo de fondo, respectivamente; th_2 es la constante que define si el pixel es parte de una sombra o no, finalmente, $f_g(x,y,t)$ es una matriz lógica que contiene lo detectado como movimiento. La Figura 2a muestra el modelo de fondo, la Figura 2b el cuadro siendo procesado, y la Figura 2c las áreas donde se detectó movimiento.



(a) (b)

(c)

Figura 2. a) Modelo de fondo. b) Cuadro procesado. c) Detección de movimiento.

Finalmente, se procede a la actualización de los pesos *W* de las neuronas que representan el modelo de fondo y, de esta manera, generar un nuevo fondo actualizado, eq. (7) y (8). Para esto, se utiliza la información contenida en todos los pixeles que no fueron considerados como movimiento y las tasas de aprendizaje (*lr*1 y *lr*2), las cuales determinan el grado de actualización, dentro de una ventana de 3x3. El modelo tiene un parámetro para la neurona central y otro para las ocho neuronas vecinas:

$$\Delta W(x,y) = lr1 * |[p(x,y) - S(x,y)] < th_1| \quad (6)$$
$$\Delta W(x,y-1) = lr2 * [p(x,y-1) - S(x,y-1)]$$
$$\Delta W(x,y+1) = lr2 * [p(x,y+1) - S(x,y+1)]$$
$$\Delta W(x+1,y) = lr2 * [p(x+1,y) - S(x+1,y)]$$
$$\Delta W(x-1,y) = lr2 * [p(x-1,y) - S(x-1,y)]$$
$$\Delta W(x+1,y-1) = lr2 * [p(x+1,y-1) - S(x+1,y-1)]$$
$$\Delta W(x+1,y+1) = lr2 * [p(x+1,y+1) - S(x+1,y+1)]$$
$$\Delta W(x-1,y-1) = lr2 * [p(x-1,y-1) - S(x-1,y-1)]$$
$$\Delta W(x-1,y+1) = lr2 * [p(x-1,y+1) - S(x-1,y+1)] \quad (7)$$
$$W(x,y,t+1) = W(x,y,t) + \Delta W(x,y) \quad (8)$$

2.3. Implementación en el sistema embebido.

Debido a las características de los sistemas embebidos se decidió modificar los algoritmos de la sección anterior para que fueran ejecutables en dichos dispositivos, con el fin de aprovechar sus especificaciones y colocarlo en un área con tráfico de personas y, así, realizar un procesamiento a tiempo real. El dispositivo utilizado fue la tarjeta Raspberry Pi 3b+ debido a su gran comunidad de usuarios, la cantidad de dispositivos con los que puede interactuar y el conocimiento previo en su manejo. La cámara utilizada es la Raspicam v2. El lenguaje de programación utilizado fue Python. A pesar de contar con muchas funciones disponibles, no todas realizan las tareas requeridas para este desarrollo, por lo que fue necesaria la modificación del código original en Matlab y el desarrollo de nuevas funciones. Otro aspecto que se trabajó fue la elaboración de una interfaz de usuario, GUI por sus siglas en inglés, que permitiera ejecutar ambos algoritmos con facilidad. Tal interfaz de usuario se muestra en la Figura 3

Figura 3. Interfaz de usuario.

2.4. Funcionalidad en IoT.

Otra de las ventajas de utilizar el sistema embebido Raspberry Pi, es la conectividad a internet de manera nativa bajo el protocolo del Internet de las cosas, IoT por sus siglas en ingles. Entre los programas que el sistema operativo de la tarjeta incluye se encuentra *VNC server*, un software que otorga control sobre el dispositivo siempre que se encuentre conectado a Internet. Otra opción que tiene el usuario para ejecutar el algoritmo es por medio de un Túnel X, en el cual únicamente las partes graficas del programa desarrollado se despliegan en la computadora que solicitó la conexión. Esto permite un mejor funcionamiento en redes con poco ancho de banda y mala velocidad de transferencia.

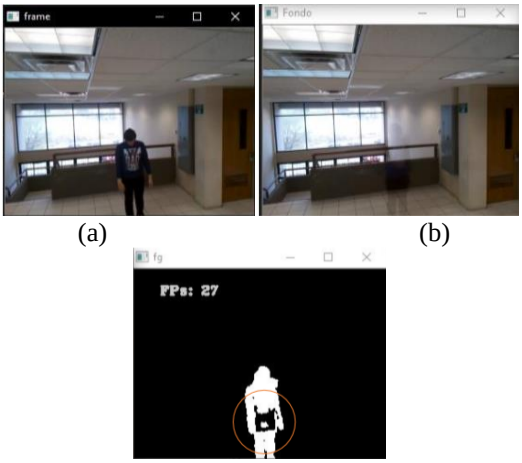
2.5. Rendimiento.

Una vez que los algoritmos, Mediana Aproximada y Pi3SOM, pasaran por una etapa de pruebas y ajustes, se midieron sus rendimientos en distintas plataformas. El primer algoritmo que se analizó fue el de Mediana Aproximada. Este se ejecutó en tres dispositivos distintos: PC, Raspberry Pi y la tarjeta embebida, Up Board. El algoritmo se evaluó usando tres resoluciones de videos diferentes y se registró la cantidad de cuadros por segundos, fps por sus siglas en inglés, que era capaz de procesar. Los resultados se muestran en la Tabla 1.

Tabla 1. Velocidad en fps de Mediana Aproximada.

Equipo	300x255	400x300	600x450
Computadora	45~50	40~45	30~34
Raspberry	15~18	10~13	5~7
Up Board	35~46	30~32	15~18

Como era de esperarse, la computadora personal logró un gran rendimiento al obtener 50 fps en bajas resoluciones y mantenerse en los 30 fps en altas resoluciones. La Up Board también obtuvo buenos resultados, oscilando entre el doble y el triple de rendimiento que el obtenido por la Raspberry Pi. Esto es debido a que la Up Board cuenta con muchos más recursos, sin embargo, requiere de una mayor fuente de alimentación para funcionar. Otro de los aspectos a evaluar es el obtenido en la tarea de la detección de movimiento. Se encontró que existen ciertas condiciones óptimas de videos las cuáles, de estar presentes, ocasionan una mejor respuesta del sistema. Entre estos destacan la buena iluminación y la carencia de vibraciones en el soporte de la cámara al capturar el video. En las secuencias de video donde no existía una buena distribución de luz, o que se encontraba muy oscuro, se presentaban problemas de camuflaje, el cual se produce cuando el algoritmo confunde el objeto en movimiento con el fondo y lo clasifica como estático, dando lugar a falsos negativos. Sin embargo, dicho problema no afecta de gran manera los resultados globales puesto que solo ocurre en ciertos escenarios y afecta solo una parte de la detección completa. Este efecto puede observarse en la Figura 4, donde se muestra en un círculo la presencia de camuflaje, Figura 4a muestra el cuadro siendo procesado, Figura 4b el modelo de fondo, Figura 4c la aparición de camuflaje. El otro problema es el que se presenta cuando la tasa de actualización del modelo de fondo es alta y comienza a agregar objetos que se mantuvieron estáticos frente a la cámara durante poco tiempo. Al moverse, generan un fantasma que provoca la aparición de falsos positivos. Tal efecto se muestra en la Figura 5, donde la Figura 5a muestra la imagen siendo procesada, la Figura 5b es el modelo de fondo y finalmente, la Figura 5c, muestra en un círculo una detección fantasma debido a una persona que se detuvo unos instantes frente a la cámara y fue absorbida como parte del fondo. El segundo algoritmo se probó en 3 dispositivos diferentes: una PC, una computadora portátil y en la Raspberry Pi. Los resultados obtenidos, a tres distintas resoluciones, se encuentran contenidos en la Tabla 2.

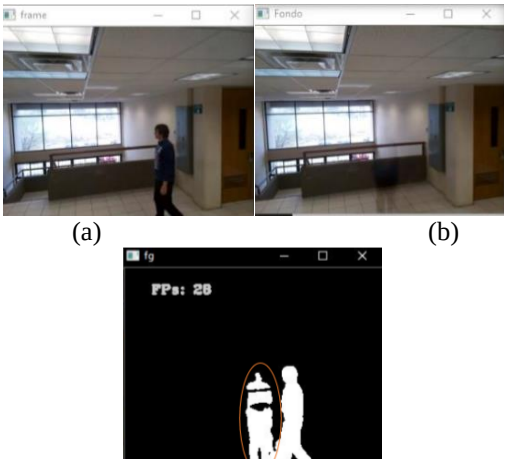


(a) (b) (c)

Figura 4. a) Cuadro procesado. b) Fondo actual. c) Presencia de camuflaje en Mediana Aproximada.

Tabla 2. Rendimiento de PiSOM en fps.

	300x225	400x300	600x450
PC	25~26	14~16	6~7
Laptop	19~22	10~12	4~5
Raspberry	2~3	1	0

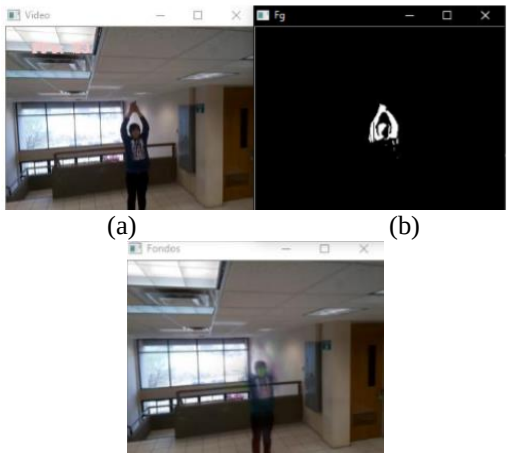


(a) (b) (c)

Figura 5. a) Cuadro procesado. b) Modelo de fondo. c) Falsos positivos en Mediana Aproximada.

Puesto que Pi3SOM conlleva a una mayor carga computacional que el algoritmo de Mediana Aproximada, el rendimiento en fps es menor. Sin embargo, presenta importantes mejoras con respecto al de Mediana Aproximada, tales como: mejor funcionalidad ante fondos dinámicos, una actualización más inteligente y un mejor comportamiento ante presencia de sombras.

En ocasiones se presentaba una absorción prematura del fondo, ilustrada en la Figura 6, donde la Figura 6a es el cuadro siendo procesado, la Figura 6b el modelo de fondo y la Figura 6c la detección realizada. Además, se observó cierto camuflaje en los objetos en movimiento, ilustrado en la Figura 7, donde se muestra el cuadro procesado en la Figura 7a, el modelo de fondo Figura 7b y la detección realizada, Figura 7c.



(a) (b)

(c)

Figura 6. a) Cuadro siendo procesado. b) Modelo de fondo. c)Absorción prematura en Pi3SOM

(a) (b)

(c)

Figura 7. a) Cuadro siendo procesado. b) modelo de fondo. c) Presencia de camuflaje en PiSOM

3. RESULTADOS

Finalmente se procedió a comparar los algoritmos desarrollados en Python, Mediana Aproximada y Pi3SOM, contra sus versiones de Matlab. Para llevar a cabo esta tarea, se trató de ejecutar los algoritmos con las mismas condiciones, ambos en una computadora personal, a distintas resoluciones y se midieron sus fps como punto de comparación.

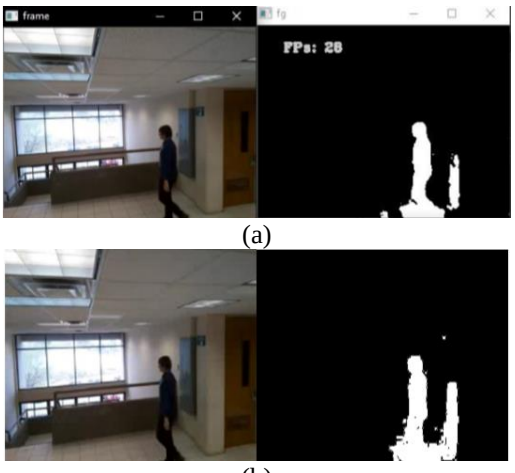
Los resultados obtenidos para el método de Mediana Aproximada están contenidos en la Tabla 3, en donde se muestran la cantidad de cuadros por segundo obtenidos en su versión original de Matlab y los logrados en Python.

Tabla 3. Comparación de Mediana Aproximada con el algoritmo original, fps.

	Mediana Aproximada original en Matlab	Mediana Aproximada en Python
300x255	4~5	47~54
400x300	3	45~52
600x480	1	19~24

Con base en los datos de la tabla anterior, el algoritmo desarrollado en Python en este trabajo pudo superar por un amplio margen el rendimiento logrado por el método original en Matlab, llegando obtener, en promedio, diez veces mejor rendimiento. Estos resultados, aunados con los datos arrojados en la Tabla 1, nos indican que el método de Mediana Aproximada es un algoritmo que requiere bajos recursos computacionales y puede ser implementado con un buen desempeño tanto en plataformas PC como en sistemas embebidos.

Una de las ventajas que se observó en la comparación de las dos implementaciones es que el algoritmo original suele arrojar mayor cantidad de falsos positivos en situaciones con buena iluminación y en la presencia de sombras, mostrando como el método de Mediana Aproximada desarrollado en Python presenta mejores resultados que su versión original en Matlab. Esto queda evidenciado en la Figura 8, donde Figura 8a muestra el resultado de Mediana Aproximada en Python y Figura 8b el de Mediana Aproximada en Matlab.



(a)

(b)

Figura 8. a) Mediana Aproximada en Python. b) Mediana Aproximada en Matlab.

Para la comparación de Pi3SOM y su versión original en Matlab se llevaron a cabo las mismas pruebas realizadas

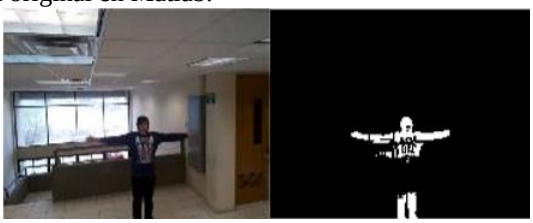
anteriormente. Los resultados obtenidos se encuentran contenidos en la Tabla 4.

Tabla 4. Comparación de Pi3SOM con el algoritmo original en fps.

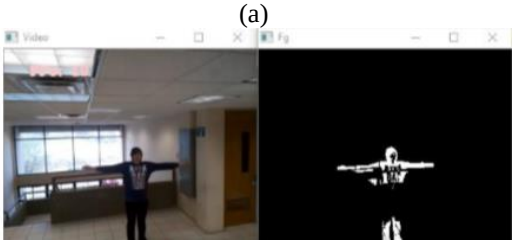
	Versión original en Matlab	Pi3SOM
300x255	10~14	25~26
400x300	10~7	14~16
600x480	4~5	6~7

Gracias a los datos obtenidos, se puede notar como el algoritmo desarrollado en Python, Pi3SOM, logra casi duplicar el rendimiento de la versión original en Matlab. Sin embargo, en resoluciones arriba de los 600x400 pixeles, el nivel de fps que logran procesar se vuelve muy similar.

Gracias a la Tabla 4, podemos notar que el algoritmo Pi3SOM muestra un buen funcionamiento a bajas resoluciones, logrando un procesamiento fluido y eficiente, superando al logrado en su versión original en Matlab. Además, no existen diferencias resaltables en los resultados de la detección de movimiento. Esto queda evidenciado en la Figura 9, donde Figura 9a muestra el resultado del algoritmo Pi3SOM y Figura 9b el de su versión original en Matlab.



(a)



(b)

Figura 9. a) Detección de movimiento de Pi3SOM. b) Detección de movimiento de la versión original en Matlab.

4. CONCLUSIONES.

En el presente artículo se llevó a cabo la tarea de modificar dos algoritmos realizados originalmente en Matlab, Mediana Aproximada y SOM-Matlab, para ser ejecutados en un sistema de tipo embebido utilizando el lenguaje Python. Los productos logrados fueron dos nuevos algoritmos en Python que permiten su ejecución en computadoras personales y sistemas embebidos, además de aumentar el rendimiento y reducir la carga computacional necesaria para su funcionamiento. Además, debido a la eficiencia lograda y la utilización del sistema embebido Raspberry Pi, se aprovechan las capacidades brindadas para IoT y es posible utilizar el sistema como una cámara de circuito cerrado con la capacidad de detectar movimiento en video.

Existe un gran campo de desarrollo para los sistemas embebidos, no solo en el campo de la visión por computadora, y se espera que, con el paso de los años, estos dispositivos sean capaces de realizar tareas de procesamiento similares a las que logra una computadora de escritorio común.

5. AGRADECIMIENTOS

Se agradece al Tecnológico Nacional de México/ I.T. Chihuahua por el apoyo brindado para la realización de este trabajo bajo el proyecto 5162.19-P.

6. REFERENCIAS

[1] J. Bedoya Guapacha and S. Mantovanni, "Real time object detection and tracking using the Kalman Filter embedded in single board in a robot", 2017 CHILEAN Conference on Electrical, Electronics Engineering, Information and Communication Technologies (CHILECON), 2017. Available: 10.1109/chilecon.2017.8229695.

[2] A. Soetedjo and I. Somawirata, "Implementation of face detection and tracking on a low cost embedded system using fusion technique", 2016 11th International Conference on Computer Science & Education (ICCSE), 2016. Available: 10.1109/iccse.2016.7581582.

[3] S. S. Kose y J. M.Varvadekar, "Application Development for Video Monitoring System & Motion Detection System using ARM9 Processor." International Journal of Scientific Research Engineering & Technology, vol. 3, nº 4, pp. 806-811, Mumbai, 2014.

[4] J. Park, K. Han and S. Yun, "Intensity classification background model based on the tracing scheme for deep learning based CCTV pedestrian detection", 2018 IEEE 9th International Conference on Mechanical and Intelligent Manufacturing Technologies (ICMIMT), 2018. Available: 10.1109/icmimt.2018.8340453.

[5] A. Dimou, P. Medentzidou, F. Garcia and P. Daras, "Multi-target detection in CCTV footage for tracking applications using deep learning techniques", 2016 IEEE International Conference on Image Processing (ICIP), 2016. Available: 10.1109/icip.2016.7532493.

[6] H. Supreeth and C. Patil, "Moving object detection and tracking using deep learning neural network and correlation filter", 2018 Second International Conference on Inventive Communication and Computational Technologies (ICICCT), 2018. Available: 10.1109/icicct.2018.8473354.

[7] D. Jung, J. Son and S. Kim, "Shot category detection based on object detection using convolutional neural networks", 2018 20th International Conference on Advanced Communication Technology (ICACT), 2018. Available: 10.23919/icact.2018.8323638.

[8] E. Elbaşi, "Fuzzy Logic-Based Scenario Recognition from Video Sequences", Journal of Applied Research and Technology, vol. 11, no. 5, pp. 702-707, 2013. Available: 10.1016/s1665-6423(13)71578-5.

[9] C. Li, C. Zou and L. Zhan, "A mean shift tracking algorithm based on the current statistical model", 2017 IEEE 9th International Conference on Communication Software and Networks (ICCSN), 2017. Available: 10.1109/iccsn.2017.8230309.

[10] T. Xu, C. Huang, Q. He, G. Guan and Y. Zhang, "An improved TLD target tracking algorithm", 2016 IEEE International Conference on Information and Automation (ICIA), 2016. Available: 10.1109/icinfa.2016.7832157.

[11] O. Arias Enríquez, "Análisis de la Marcha Humana Basada en Percepción Visual 2D/Kinect y su Diagnóstico Utilizando Sistemas Difusos", Tesis de Maestría, Chihuahua: DEPI, 2012.

[12] S. González Duarte, "Detección y Seguimiento de Objetos en Secuencias de Video con Fondos Dinámicos", Tesis de Maestría, Chihuahua: DEPI, 2010.