Congr. Int. en Ing. Electrónica. Mem. ELECTRO, Vol. 40, pp. 231 - 236, Oct 2018, Chihuahua, Chih. México
http://electro.itchihuahua.edu.mx/memorias_electro/MemoriaElectro2018.zip
ISSN 1405-2172

DISEÑO DE RED IOT BASADA EN COMPUTACIÓN EN EL BORDE PARA TRASMITIR MOVIMIENTO EN SISTEMA DE VISIÓN

Juan A. Ramírez Quintana[1], Luis M. Morales Pérez[2], Alma D. Corral Sáenz[1], Mario I. Chacón Murguía[1]
Laboratorio de Percepción Visual con Aplicaciones en Robótica[1], Laboratorio de innovación de Intel[2]
Tecnológico Nacional de México/I.T. Chihuahua
Ave. Tecnológico No. 2909, 31310, Chihuahua, Chih., México, Tel. (614)2026511
jaramirez@itch.edu.mx, lmmoralespe@hotmail.com, adcorral@itch.edu.mx, mchacon@itch.edu.mx

RESUMEN

El Internet de las cosas (IoT) es una tecnología que ha mejorado significativamente la funcionalidad de sistemas de procesamiento de video. Sin embargo, la mayoría de estos sistemas se basan en *computación en la nube*, los cuales se ven limitados en tiempos de respuesta, transmisión y ancho de banda. Este artículo presenta el desarrollo de una novedosa arquitectura de red IoT basada en el modelo *computación en el borde* que, a diferencia del *computación en la nube,* el almacenamiento y procesamiento de datos se realizan en un dispositivo móvil/fijo que son el borde de la red, reduciendo tiempos de respuesta y trasmisión de datos. La implementación se realizó en Python con Raspberry Pi como plataforma de borde, donde el video es adquirido mediante cámaras o drones, y procesado y trasmitido a servidores *socket*. Como resultado, se obtuvo un sistema con velocidad aceptable en comparación a sistemas de vigilancia comerciales CCTV.

Palabras clave: Internet de las cosas, Procesamiento de video, Computación en el borde

ABSTRAC

The internet of things is a technology that allows the development of network protocols with the capability of transfer and process video, improving the functionality of video processing and surveillance systems. However there are some challenges to these networks, which are the time response and bandwidth, because most of the IoT systems for video processing are based on "Cloud Computing". Then, the present paper explains the development of a IoT network closer to the client that consist on a novel video processing architecture based on "Edge Computing". The implementation was made on Python with a raspberry Pi as Platform where the frames from a Camera or a drone are computed and then transfer through Socket Servers. The results show that the IoT vision system has acceptable time response in comparison with commercial CCTV devices.

Keywords: internet of things, Video processing, Edge computing

1. INTRODUCCIÓN

El Internet de las cosas (IoT) es una tecnología de redes en la cual varios dispositivos de uso cotidiano son conectados a en red para obtener control e información que permita a los usuarios acceder a diversos recursos desde cualquier lugar por medio de Internet. Visión por computadora y el monitoreo de sistemas de vigilancia inteligentes son de los principales campos de investigación y desarrollo beneficiados por IoT ya que permite tener acceso a la información de un video desde distintos puntos que puede ser útil para sistemas de seguridad o para desarrollo de terapias que requieran del monitoreo de un paciente con padecimientos como epilepsia o trastorno de estrés postraumático [1]. Para estas aplicaciones, se requiere una respuesta en tiempo real que es posible con *edge computing* o computación en el borde (el procesamiento es cercano al usuario). En la literatura se reportan diversos trabajos sobre procesamiento de video e IoT, la mayoría de ellos se enfocan a detectar movimiento para propósitos de seguridad o sistemas inteligentes. Por ejemplo, en [2] se realizó un análisis exitoso donde se revisó la posibilidad de combinar visión por computadora e IoT para realizar sistemas comerciales. Para ello, se propone un algoritmo de detección de personas y se utilizó Midgar una plataforma para pasar datos a la nube. En [3] se propone un esquema basado en IoT y visión por computadora para detectar rostros, utilizando una Raspberry. En [4] se describe una alarma de seguridad que detecta movimiento mediante diferencias entre cuadros, la cual permite al usuario ver remotamente la actividad y recibir mensajes con IoT y una Raspberry. En [5] propone un algoritmo basado en filtros Kalman para localizar un objeto en movimiento mediante IoT.

Actualmente, el esquema de IoT más extendido para visión es computación en la nube (CN), el cual se refiere a que el procesamiento del video se realiza en un servidor remoto [6] [7]. Para CN, se emplean protocolos de compresión para la transferencia de video como H264, el cual agrupa la información redundante para disminuir la cantidad de información a transferir [8] [9]. Sin embargo, la latencia se incrementa al aumentar el número de datos a transmitir debido a que las redes físicas CN de Internet están conformadas de múltiples repetidores, compuertas y servidores [10]. Por ello, se han realizado estudios que concluyen que el incremento de los dispositivos implicará que la infraestructura existente en algunos años no sea suficiente para soportar la demanda de conectividad a Internet [11]. De esta manera, las redes que procesan información de cámaras de vigilancia basadas en CN experimentaran problemas significativos de latencia con el paso el tiempo por la cantidad de información a procesar [12] [13].

Sin embargo, en aplicaciones donde se requiere una respuesta en tiempo real con videos de alta calidad [14], se puede recurrir a computación en el borde (CB), lo cual consiste en realizar las tareas de procesamiento masivo desde el cliente [15] [16] [17] [18] [19]. En esta forma de procesamiento, la

adquisición del video y el procesamiento se realiza directamente desde la cámara y un dispositivo para trasmitir solo los resultados a Internet [20] como detección de movimiento, presencia, entre otros [21]. Esto puede reducir el consumo de ancho de banda hacia la nube [22] y como consecuencia, se mejora los tiempos de latencia. Por lo tanto, en este trabajo se propone un sistema de computación en el borde escalable para procesar y trasmitir video terrestre y aéreo en una red IoT que utiliza la plataforma la Raspberry Pi 3 y servidores socket, los cuales son un tipo de servidor que permite el intercambio de información entre cliente y servidor de manera bidireccional mediante la transferencia de datos en cadena. Esto para realizar un esquema de procesamiento cercano al cliente que permita aplicaciones como vigilancia de espacios cerrados para seguridad o el cuidado de pacientes con problemas de epilepsia o trastorno de estrés postraumático entre otras [23]. Hacia la nube no hay trasmisión de datos.

El resto del artículo se organiza de la siguiente manera. La Sección 2 describe la arquitectura en desarrollo. En la Sección 3 se analizan los resultados obtenidos y en la Sección 4 se presentan las conclusiones.

2. DESARROLLO DEL SISTEMA

En la figura 1 se muestra el esquema general del sistema, donde la entrada es una cámara Raspicam modelo 1.3 o bien un punto de acceso de un vehículo aéreo no tripulado (drone) Ar-Drone v2.0. Ambas entradas envían la información al dispositivo que va a hacer el procesamiento en borde, el cual es una Raspberry Pi modelo 3B donde se procesa el video para detectar si existe movimiento y si es así, definir cuáles son los pixeles que cambian. También, se realizan diversas operaciones de pre procesamiento para mejorar la calidad de los resultados del video.

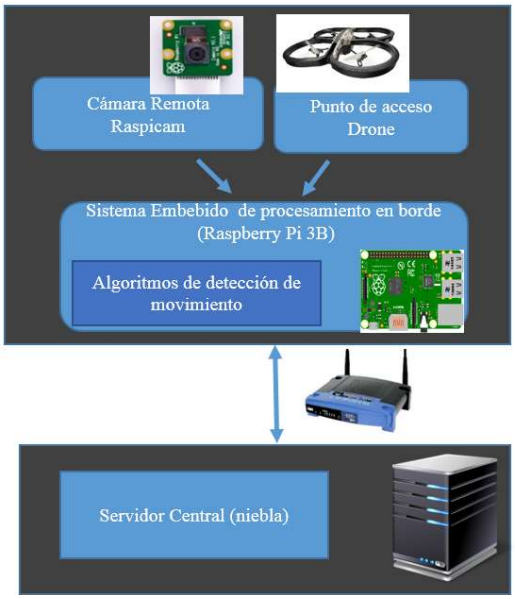


Figura 1. Diagrama de sistema.

Si se detecta movimiento, entonces la red trasmite los datos del video mediante Internet por red local empleando tecnologías de servidores socket que permiten comunicación bidireccional más eficiente y para finalmente llegar al servidor central de la red local donde se pueden realizar operaciones más complejas.

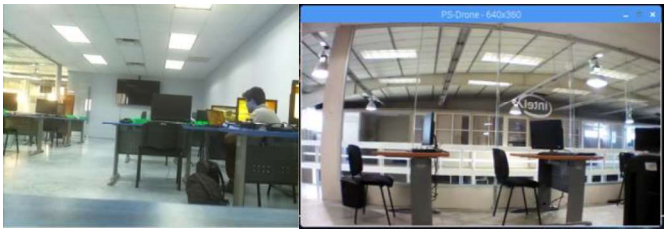
2.1. Bloque de adquisición (cliente)

La adquisición de imágenes es el primer bloque y está ubicado en el lado del cliente. Como se observa en la Figura 1, este bloque consta de dos módulos: el de entrada que es una cámara que adquiere video y el sistema embebido basado en la Raspberry Pi, el cual recibe el video cuadro por cuadro y realiza los procesamientos más básicos empleando la librería Open-CV (Detección de movimiento, seguimiento, filtrado, etc.).

La cámara Raspicam tiene como características una resolución de 5 megapíxeles (2592x1944), velocidades de muestreo de hasta 30 cuadros por segundo (fps *frames per second*) e interfaz de conectores flexibles DSI-MIPI (*Display Serial Interface Movile Industry Processor Interface*). La cámara optimiza el proceso de adquisición mediante el muestreo directo con una Unidad Grafica de Procesamiento (GPU) que toma capturas del entorno a diferencia de emplear una cámara por interfaz USB donde el procesador requiere tiempo para capturar la información. Para conectar la cámara al módulo del sistema embebido se emplea la librería de adquisición llamada Picamera para adquirir imágenes con Python. En la Figura 2a se puede observar un cuadro generado por la Raspicam.

El drone es el AR-Drone-2.0 y cuenta con dos cámaras de alta definición incorporadas en el fuselaje (Frontal e Inferior) además la comunicación a este se realiza por medio de Wifi AP (*Wifi Access point*) para emplearse con dispositivos móviles. En este caso se emplea para monitoreo y seguimiento de objetivos. La conexión del drone hacia el modulo del sistema embebido es por medio de Wifi donde el vehículo posee un servidor Interno Administrado por un Microcontrolador embebido con corteza ARM A8 TMS320DMC64x. Las capturas de la cámara y los comandos de vuelo se realizan por medio de un DSP. Para controlar el drone, es necesario un dispositivo externo que se programe en Python y permita conectividad Wifi. El módulo que permite la comunicación con el drone está basado en la librería "*ps-drone*" de Python que incorpora las funciones para analizar movimiento, orientación de objetos y puede adquirir video con un formato de compresión (H264). La operación de captura se realiza con la tarea *vCapture*, la cual adquiere el video decodificado y lo despliega como una pantalla con la función *imshow()* de *Open-CV*. Para la sincronía entre la transferencia de la información y la captura se emplea una tarea de transferencia que se basa en la función *Frame_Send()* que recibe información de un buffer circular procedente de otra tarea que realiza el procesamiento de imágenes. La imagen resultante de este proceso se trasmite al sistema embebido de

procesamiento de borde. La figura 2b muestra un ejemplo de capturas que se trasmiten al sistema embebido desde el drone.



a) b)

Figura 2. Capturas de las cámaras de entrada. a) Captura de la Raspicam. b) Captura del drone.

2.2. Detección de movimiento del lado del cliente

Una vez que se obtiene la entrada, el siguiente paso es procesar cada cuadro en la Raspberry Pi modelo 3B. Para ello se desarrolló un algoritmo de detección de movimiento que consiste en una diferenciación de cuadros basado en [24] [25] y se realiza de la siguiente forma:

1. Adquisición de cada cuadro con la Raspicam o la cámara integrada del drone frontal como fuentes.

2. Dimensionar el cuadro con el módulo Imutils de Python, de esta forma se obtiene un cuadro en el espacio de color RGB $F_{RGB}(x,y,t)$ donde (x,y) se refiere a la coordenada de cada pixel y t es el índice de tiempo de cada cuadro.

3. Para facilitar los cálculos en la Raspberry, se obtiene la componente de luminancia dada por:

$$F(x,y,t) = \left(F_R(x,y,t) + F_G(x,y,t) + F_B(x,y,t)\right)/3 \quad (1)$$

4. Para obtener la diferenciación temporal, se implementa un filtro de promediado para eliminar el ruido causado por la adquisición [25]. Esta operación está dada por:

$$P(x,y,t) = \sum_{m=-\infty}^{\infty}\sum_{n=-\infty}^{\infty} F(m-x,n-y,t)H(m,n) \quad (2)$$

Donde $H(m,n)$ es un filtro de suavizado gaussiano dado por:

$$H(m,n) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{m^2+n^2}{2\sigma^2}} \quad (3)$$

Donde, m es el alto y va de $m=1,\ldots,11$, n el ancho en y va de $n=1,\ldots,11$, σ es la desviación estándar que para este caso es uno ya que las diferencias mínimas en los pixeles no son mayores a una desviación estándar. El tamaño del filtro definió mediante experimentación basada en la eliminación de variaciones y cambios de iluminación producidos por el mismo ambiente del laboratorio.

5. Para terminar la diferenciación temporal, se realiza la siguiente operación:

$$D(x,y,t) = \|P(x,y,t) - P_o(x,y,t)\| \quad (4)$$

Donde $P_o(x,y)$ es el cuadro de referencia que se obtiene aplicando las ecuaciones (1) a (3) a $F_{RGBO}(x,y)$, el cual se obtiene de la captura anterior de la cámara del sistema embebido.

6. Para mejorar la localización de los pixeles que cambiaron sus valores derivados del movimiento, se realiza una dilatación en $P(x,y)$ dada por:

$$\Lambda(x,y,t) = \mathrm{D}(x,y,t) \oplus A \quad (5)$$

Donde A es el kernel de dilatación binario descrito por:

$$A \subseteq B \Rightarrow A \oplus K \subseteq B \oplus K \quad (6)$$

7. Para definir si el video se trasmite o no, se obtiene el área de los contornos mediante:

$$\Delta = \begin{cases} 1, & \sum_x \sum_y \Lambda(x,y,t) > Th \\ 0, & \text{otra forma} \end{cases} \quad (7)$$

Th se define como un ajuste de sensibilidad definido por el usuario, para este caso $Th = 30$. Si Δ es uno, se trasmite el video hacia el servidor central los pixeles $F_{\Delta}(x,y)$ que cambian su valor dados por:

$$F_{\Delta}(x,y) = \{F_{RGB}(x,y,t) | \Lambda(x,y,t) > Th\} \quad (8)$$

Es decir, $F_{\Delta}(x,y)$ es el conjunto de pixeles de color de $F_{RGB}(x,y,t)$ que cambian sus valores debido al movimiento.

Si es el primer cuadro del video, este pasa completo al servidor. Después del primer cuadro, pasan únicamente los pixeles $F_{\Delta}(x,y)$, los cuales se deben "Serializar", es decir, se debe realizar una transformación de la representación de un objeto en Python a su representación en Cadena, ya que la comunicación en redes se realiza por cadena. Para realizar esta representación *cPickle* de Python, el cual es un módulo que realiza la conversión y esta optimizado ya que las funciones están implementadas en C, véase figura 3.

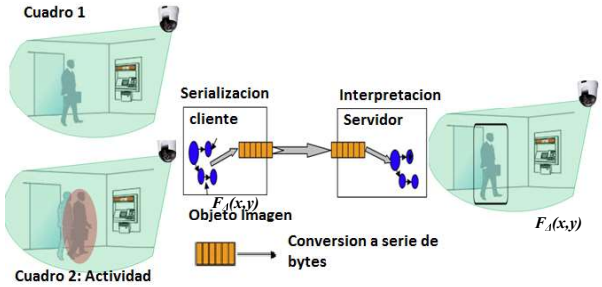


Figura 3. Representación del procedimiento de serializacion.

Mediante este algoritmo, la red tiene la posibilidad de trasmitir únicamente la información necesaria cuando es necesario. El propósito de esta trasmisión es enviar la información hacía otras estaciones que reciban el mensaje sobre algún cambio en la secuencia de video que no se considere. Esto puede ser para encender una alarma y que el usuario del otro lado de la red visualice el video durante la presencia de movimiento.

2.3. Servidor central

Se considera al servidor en red Local (LAN) como el servidor central y se encarga de procesar la información procedente de diferentes Módulos de adquisición. Debido a que el sistema IoT es una red es escalable, el servidor central puede enlazar varios módulos de adquisición embebidos (clientes) como se ve en la Figura 4.

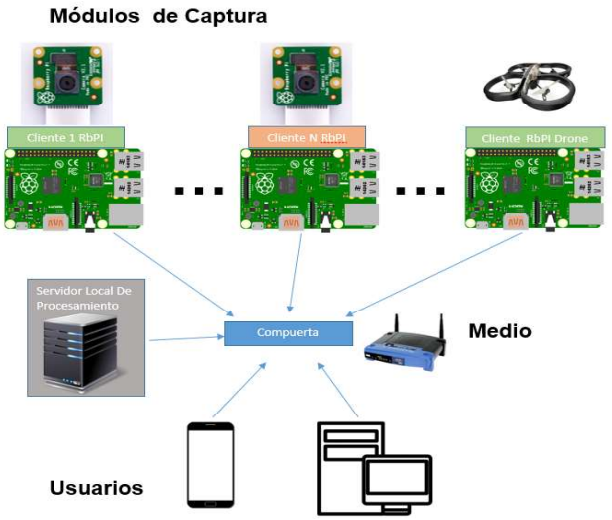


Figura 4. Esquema de procesamiento centralizado. No se trasmite información hacia la nube.

Cada cliente conectado la red realiza el procesamiento discutido en la sección 2.1. Por consiguiente, el servidor recibe la información de los pixeles de cada cuadro de cada cliente y los procesa o trasmite según sea el caso. Esta información se despliega en una interfaz gráfica como la que se observa en figura 5 en el servidor LAN donde se cuenta con más recursos computacionales para llevar a cabo operaciones más complejas por ejemplo una red neuronal o retrasmitir a sistema de procesamiento en borde. Cuando se recibe la información determinada como relevante por el algoritmo de movimiento (Se detectó un cambio fuera del umbral) se reducen la cantidad de información redundante (Información repetitiva sin gran cambio) evitando consumir ancho de banda esto ya que no se transfieren capturas cuando se considera que no hay actividad.

2.4. Sistema de red IoT

Considerando los clientes y el servidor, la arquitectura de toda la red tiene un conjunto de características de cliente y servidor que se pueden resumir en:

Cliente

Captura de información. Drone o cámara Raspicam.

Procesamiento en tiempo real. Procesamiento y transferencia de información de forma concurrente mediante tareas por la librería *Threads* en Python.

Administración de excepciones. En caso de errores de desconexión, existe protección para mantener el programa en funcionamiento. Para configurar esto se emplean estatutos *Try except* en Python.

Conectividad a una red por un medio. La Raspberry Pi cuenta con WiFi y Ethernet, el Wifi se emplea para la comunicación con el drone mientras que el Ethernet se emplea para comunicar con el ruteador y el servidor.

Figura 5. Interfaz del servidor con la cámara de la computadora, video procedente de una Raspicam y video procedente del drone.

Transferencia

Un Ancho de buffer optimizado para la transferencia determinado dado por:

$$L = 3BR \tag{9}$$

Donde B es la cantidad de Bytes a trasmitir, R es la resolución del cuadro y L el ancho mínimo de paquetes de salida (Bytes). Mediante el cálculo de ancho del buffer, se evita consumir tiempo extra en la lectura del buffer. Por ejemplo, para un cuadro de resolución de 640x480 RGB (3 canales) se obtiene $L = 3 * 1 * 640 * 480 = 921{,}600\ Bytes$. Esto quiere decir que se requiere trasmitir 921,600 Bytes para enviar el cuadro.

Servidor

Capacidad de procesamiento. Se implementa en una computadora de escritorio o bien un servidor remoto que cuenta con al menos cuatro veces las capacidades de uno de los sistemas embebidos basados en Raspberry Pi.
Capacidad de atención a múltiples clientes. El servidor puede atender múltiples solicitudes mediante un servidor por tareas con la librería *Threads* de Python.
Administración de excepciones de desconexión espontanea o fallo de transferencia. El servidor puede tolerar el fallo de desconexión de un cliente con el uso de bloques *Try Catch* en Python. Tal como se mostró en la figura 4, se permite la conexión al sistema de *N* clientes cada uno de los cuales procesan las capturas y determina cuales de estas son relevantes y cuales no evitando ocupar tiempo por parte del servidor para leerlo pero manteniendo el enlace entre cliente y servidor, en este caso la comunicación solo es en red local (no hay interacción con la nube).

3. RESULTADOS OBTENIDOS

Se realizaron pruebas de validación con el objetivo de evaluar la velocidad de respuesta en tiempo real. En la Tabla 1 se muestran los tiempos de retardo promedio obtenidos para procesar capturas de una cámara. Las pruebas se realizaron con base en la información obtenida de las bases de datos de *ImageProcessingPlace.com* y *sipi.usc.edu* [26]. La Tabla 1 muestra además los resultados para el promedio de tiempo al trasmitir según la resolución de cada cuadro. Se puede observar que los tiempos son menores a un segundo, mientras que un sistema comercial tiene tiempos mayores a un segundo, debido a la trasmisión y procesamiento de datos en la nube. Adicional a ello, los sistemas comerciales requieren de la incorporación de sensores de movimiento basados en sensores PIR (Sensor infrarrojo pasivo), mientras que el sistema propuesto tienen algoritmos de visión permiten la detección automática de movimiento que ahorra horas de revisión por parte del usuario. Por ejemplo al sistema SIERA EASY kit 8 el cual posee una unidad de almacenamiento de 1 TB para 8 cámaras, tiene un tiempo de grabación completo de 20 días que el usuario deberá revisar.

Tabla 1. Promedios de retardo de procesamiento y transferencia para el Modelo Inicial de un servidor en cada cliente sobre una Raspbery Pi 3 modelo B comparado con un sistema comercial este incorpora un sistema de visión.

Resolución	Procesamiento (Seg)	Transferencia (Seg)	Total (Seg)
630x430	0.55637	0.19841	0.75478
360x280	0.58944	0.19763	0.78707
360x240	0.79211	0.10388	0.89599
720x480	0.68522	0.25259	0.93781
320x240	0.4282	0.14863	0.57683

Adicionalmente a estas pruebas de tiempo y resolución, se realizaron pruebas de calentamiento del sistema ya que en operación continúa el procesador se calienta y eso afecta el rendimiento y desempeño. En la figura 6 se aprecia que el calentamiento se incrementa exponencialmente respecto a la temperatura ambiente. Las muestras se tomaron del sensor interno del procesador que toma como referencia la temperatura en grados Celsius del sistema. El muestreo es de 1 Hz, durante dos minutos con una temperatura ambiente de 24 grados, obteniendo un límite de 62.3 grados que supera la temperatura recomendada del fabricante; Las pruebas realizadas son evidencia de la necesidad de la implementación de un disipador o un ventilador para estabilizar la temperatura.

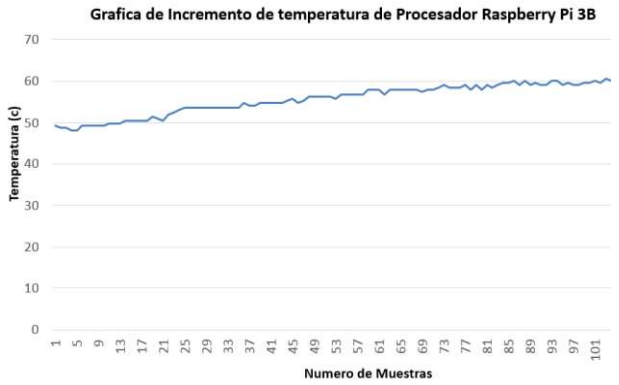


Figura 6 Grafica de Temperatura grados Celsius contra segundos para un porcentaje de uso de CPU de 38 % al interrumpir el funcionamiento del ventilador.

4. CONCLUSIONES

En este artículo se presentó la implementación de un sistema IoT para trasmitir video procesado en el borde mediante un sistema embebido. La entrada es una cámara Raspicam o la cámara de un drone y el dispositivo para hacer procesamiento en el borde es una Raspberry Pi 3 modelo B. El procesamiento realizado es la detección de movimiento para el monitoreo de la seguridad o realización de terapia. Con base en los resultados, se considera que los tiempos de procesamiento y trasmisión de la red IoT propuesta son menores que los sistemas comerciales ya que el servidor central es LAN. Esto se debe a que de acuerdo con las pruebas, la mayoría de los tiempos de procesamiento y trasmisión fueron menores de un segundo, mientras que en un sistema comercial son mayores a un segundo debido a la latencia del acceso a la nube. Solo se debe tener un poco de cuidado con el calentamiento del procesador, para lo cual es necesaria la instalación de disipadores.

Como trabajo futuro se pretende la implementación de un Host remoto para el despliegue de la información y ser accesible por medio de un Navegador Web en cualquier punto con acceso a internet.

AGRADECIMIENTOS

Los autores agradecen a Tecnológico Nacional de México por el apoyo TecNM 6418.18-P para la realización de este trabajo

REFERENCIAS

[1] W.A. Hauser, J.F. Annegers, L.T. Kurland, "Prevalence of epilepsy in Rochester, Minnesota: 1940–1980", Epilepsia, Vol. 32, No. 4, 1991, pp. 429-445.

[2] C. González, D. Meana, B.C. Pelayo, J.M. Cueva, N. García; "Midgar: Detection of people through computer vision in the Internet of Things scenarios to improve the security in Smart Cities, Smart Towns, and Smart Homes", Future Generation Computer Systems, Vol. 76, No. 1, Nov. 2017, pp. 301-313.

[3] I. Aydin; N.A. Othman, "A new IoT Combined Face Detection of People by Using Computer Vision for Security Application", International Artificial Intelligence and Data Processing Symposium, Malatya, Turkey, 2017, pp 1-6.

[4] A.N. Ansari, M. Sedky, N. Sharma, A. Tyagi, "An Internet Of Things Approach for Motion Detection using Raspberry Pi", International Conference on Intelligent Computing and Internet of Things (ICIT), Harbing, 2015, pp 131-134.

[5] J. Guo, H. Zhang, Y. Sun, R. Bie, "Square –root unscented Kalman filtering-based localization and tracking in the internet of things", Personal and Ubiquitous Computing, Vol. 18 No. 4, 2014, pp. 987-996.

[6] J.G. Quílez, "Sistema web de visualización de video en straming mediante el empleo de una Raspberry Pi", 2017, Universidad Pública de Navarra, Tudela.

[7] A.N. Ansari, M. Sedky, N Sharma, A. Tyagi, "An Internet of things approach for motion detection using Raspberry Pi", Proceedings of 2015 International Conference on Intelligent Computing and Internet of Things, Harbin China, 2015, pp. 131-134.

[8] H. Ochoa, J. Mireles, J. Cota, "Descripción del nuevo estándar de video H. 264 y comparación de su eficiencia de codificación con otros estándares", Ingeniería, investigación y tecnología, Vol. 8, No 3, 2007, pp. 157-180.

[9] M. Dai, P. Baylou, "An efficient algorithm for computation of shape moments from chain codes", Pattern Recognition, Vol. 25, No. 10, 1992, pp. 1119-1128.

[10] D.E. Comer, "Redes de computadoras, Internet e Interredes", Prentice-Hall Hispanoamericana, México, 1997, pp. 4-5.

[11] W. Shi, J. Cao, Q. Zhang, Y. Li and L. Xu, "Edge Computing: Vision and Challenges" in IEEE Internet of Things Journal, Vol. 3, No. 5, octubre 2016, pp. 637-646.

[12] R. Kuschnig, I. Kofler, H. Hellwagner, "Improving Internet Video Streaming Performance by Parallel TCP-Based Request-Response", IEEE Xplore, Austria, 2010.

[13] S. Xie, G.Y. Keung, B. Li, "A Measurement of a Large-scale Peer-to-PeerLive Video Streaming System", 2007 International Conference on Parallel Processing Workshops, Xian, China, 2007.

[14] T. Soyata, R. Muraleedharan, C. Funai, M. Kwon, W. Heinzelman, "Cloud-Vision: Real-time face recognition using a mobile-cloudlet-cloud acceleration architecture", 2012 IEEE Symposium on Computers and Communications, Cappadocia, Turquía, 2012.

[15] Kraijak, Surapon; Tuwanut, Panwit. A survey on IoT architectures, protocols, applications, security, privacy, 2015, real-world implementation and future trends.

[16] M.I. Chacón, "Procesamiento digital de imágenes", Editorial Trillas, México, 2007.

[17] E.A. Sobrado, "Sistema de visión artificial para el reconocimiento y manipulación de objetos utilizando un brazo robot". Pontificia Universidad Católica de Perú, 2003, Lima, Perú.

[18] A. Buchowicz, G. Galinski, "Video Streaming Framework", Journal of Telecommunications, and Information Tecnology, 2011, pp. 102 - 108.

[19] J.W. Bubnicki, M. Churski, P.J, Kuijper, "TRAPPER: an open source web-based application to manage camera trapping projects", Methods in Ecology and Evolution, Vol. 7, No. 10, abril 2016, pp. 1209, 1216.

[20] L.A. Martínez, "Sistema de visión para el equipo de robots autónomos del ITAM", Tesis, Instituto Tecnológico Autónomo de México, 2004, ITAM,

[21] Y. Wang, B.M. Nguyen, H. Fujimoto, Y. Hori, "Vision-based vehicle body slip angle estimation with multi-rate Kalman filter considering time delay" 2012 IEEE International Symposium on Industrial Electronics, Hangzhou China, 2012, pp. 1506-1511.

[22] F. Kuhnt et al, "Robust environment perception for the Audi Autonomous Driving Cup" 2016 IEEE 19th International Conference on Intelligent Transportation Systems (ITSC), Río de Janeiro, Brasil, 2016, pp 1424-1431.

[23] E. G. R. C. S. F. L. W. H. C. Mark Patton, "Uninvited Connections: A study of Vulnerable Devices on the Internet of Things (IoT)", 2014 IEEE Joint Intelligence and Security Informatics Conference, The Hague, Netherlands, 2014, pp. 232-235.

[24] A. Manzanera, J. Richefeu, "A new motion detection algorithm based on Σ–Δ background estimation", Pattern Recognition Letters, Vol. 28, No 3, February 2007, pp. 320-328.

[25] R.A. Peters, "A new algorithm for image noise reduction using mathematical morphology", IEEE transactions on Image Processing, Vol. 4, No. 5, May 1995, pp. 554-568.

[26] Allan Weber "USC Viterbi", University of Southern California, Disponible en: sipi.usc.edu. [Accessed 21 Junio 2018].