La Mecatrónica en México, Vol. 4, No. 2, páginas 65 – 76, Mayo 2015
Disponible en línea en www.mecamex.net/revistas/LMEM
ISSN en trámite, 2014 Derechos de autor y derechos conexos, Asociación Mexicana de Mecatrónica A.C

Desarrollo de un Sistema de Detección de Movimiento basado en Flujo Óptico en Raspberry Pi

Silva Blanco Gonzalo Elias[1], Avecilla Rangel Francisco Javier[1], Rivas Araiza Edgar Alejandro[2], Toledano Ayala Manuel[2], Pedraza Ortega Jesus Carlos[1], Ramos Arreguín Juan Manuel[1]

[1]Facultad de Informática, Universidad Autónoma de Querétaro,
[2]Facultad de Ingeniería, Universidad Autónoma de Querétaro,
jramos@mecamex.net

Resumen

El presente proyecto muestra una alternativa para desarrollar un sistema para la detección de movimiento, utilizando sistemas embebidos de bajo costo basados en microprocesador. Se realiza la implementación de un algoritmo para la detección de flujo óptico. El flujo óptico se caracteriza básicamente por la detección de movimiento en una secuencia continua de imágenes o en un video en particular. De esta manera, es posible que se detecte movimiento en una zona donde no debe existir actividad en un horario específico. El sistema que se va a desarrollar se basa en el uso de sistemas embebidos basados en microprocesador, en este caso, una tarjeta Raspberry Pi, el cual cuenta con una cámara de 5 Mega-pixeles, para la adquisición de imágenes o video, así como la instalación de Python y las librerías de OpenCV. Se analizan diversas opciones de algoritmos de flujo óptico para determinar el que puede ser implementado en la tarjeta mencionada. Una vez que el sistema detecta movimiento, se adquiere la imagen del lugar y es almacenada para un análisis posterior.

Palabras clave: Sistemas embebidos, Raspberry Pi, flujo óptico, Raspberry Pi, Python, openCV.

1. Introducción

El flujo óptico visto desde un punto de vista de visión artificial es una herramienta algorítmica que ha probado ser un componente de vital importancia en tareas de visión por computadora. Entre las cuales destacan detección y cálculo de tiempos de colisión, segmentación del movimiento, seguimiento, extracción de fondo, odometría visual, compresión de video, entre otras tareas, que bien pueden ser usadas tanto en el dominio de la inspección industrial, la robótica y misiones espaciales.

Para el hombre la capacidad de recibir a información través de sus sentidos, las imágenes, impresiones o sensaciones para conocer algo, alude al término percepción. La percepción visual es el medio más importante puesto que permite tener conocimiento espacial de un entorno relativamente amplio, a una resolución considerablemente alta, permitiendo procesar escenas dinámicas de forma natural, desde la estimación del movimiento relativo entre objetos, movimiento propio, orientación, discriminación entre distintos objetos e incluso en reconocimiento de objetos.

Con la finalidad de reproducir sistemas capaces de analizar el movimiento de objetos dentro de una escena, se han propuesto múltiples algoritmos de estimación del movimiento, y también existen

La Mecatrónica en México, Vol. 4, No. 2, páginas 65 – 76, Mayo 2015
Disponible en línea en www.mecamex.net/revistas/LMEM
ISSN en trámite, 2014 Derechos de autor y derechos conexos, Asociación Mexicana de Mecatrónica A.C

diversas formas para clasificarlos. Una propuesta de clasificación la da Tabatabai [1], donde una primera clasificación es la existente entre técnicas de detección del movimiento 3D y técnicas de detección del movimiento 2D. Dentro de cada técnica existen distintos métodos que a su vez, algunos de ellos, pueden sub-clasificarse, y al mismo tiempo se complementa con la clasificación propuesta por Barrón [2] tal como se muestra en la figura 1.

$$\text{Análisis del movimiento} = \begin{cases} 3D \\ 2D \begin{cases} \text{Flujo óptico} \begin{cases} \text{Basados en gradiente} \begin{cases} \text{Con restricciones locales} \\ \text{Con restricciones globales} \end{cases} \\ \text{Basados en correlación} \begin{cases} \text{Correspondencia entre puntos} \\ \text{Correspondencia entre regiones} \end{cases} \\ \text{Basados en frecuencia} \begin{cases} \text{Orientación de fase} \\ \text{Orientación frecuencia} \end{cases} \\ \text{Basados en energía} \end{cases} \\ \text{Correlacionales} \begin{cases} \text{Correspondencia entre puntos} \\ \text{Correspondencia entre regiones} \\ \text{Correspondenciaentre fase} \end{cases} \\ \text{Diferenciales} \begin{cases} \text{Imágenes de diferenciales acumuladas} \\ \text{Substracción de fondo} \end{cases} \\ \text{Frecuenciales (Fourier)} \\ \text{Recursivos} \\ \text{Estocásticos} \end{cases} \end{cases}$$

Figura 1.- Clasificación de las distintas técnicas para el análisis del movimiento.

El flujo óptico, es definido como el movimiento aparente de un patrón de brillo sobre una superficie y/o borde en una escena, causado por el movimiento relativo entre un observador y la escena. Este concepto fue introducido por primera vez en la década de 1925 por el psicólogo Von H. Helmholtz [3] quien lo atribuía principalmente a la percepción de las variaciones de la imagen en la retina y, finalmente, fue publicado por el psicólogo estadounidense James J. Gibons [4] indicando su relación directamente con el patrón óptico y haciendo referencia a la estructura poseída por la luz en un punto de observación en movimiento. En un concepto bio-inspirado es el cambio estructurado de patrones de luz sobre la retina que conduce a una impresión del movimiento de la imagen visual proyectada, tal como se presenta en la figura 2.

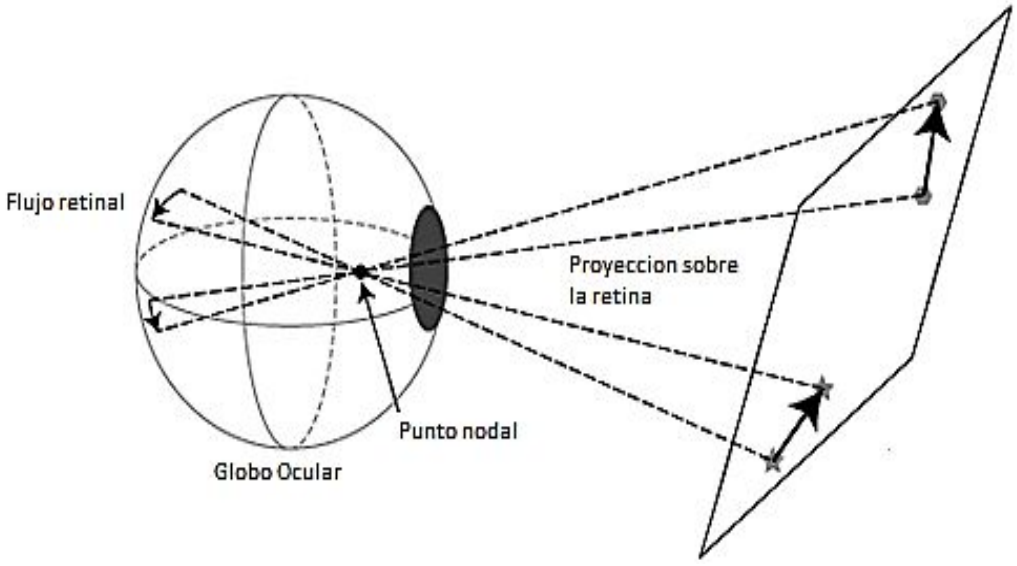

Figura 2.- Desplazamiento de dos características visuales en un plano y el globo ocular.

La Mecatrónica en México, Vol. 4, No. 2, páginas 65 – 76, Mayo 2015
Disponible en línea en www.mecamex.net/revistas/LMEM
ISSN en trámite, 2014 Derechos de autor y derechos conexos, Asociación Mexicana de Mecatrónica A.C

De manera análoga, si la luz en la estructura es muestreada espacial y temporalmente, y resulta en una secuencia de imágenes como las presentadas en la figura 3, es posible determinar los vectores de desplazamiento de dicho patrón.

El hecho es que, se defina como se defina, el flujo óptico es la mejor medida del movimiento en el espacio bidimensional. El cálculo del flujo óptico es importante para un sistema artificial y para un organismo vivo. El único problema es que el flujo óptico, sigue siendo un campo de investigación buscando nuevas estrategias y técnicas para su cálculo [5], [6]. La razón es que los sistemas de visión artificial todavía no han logrado calcularlo con precisión elevada en poco tiempo.

Actualmente existe una amplia base de conocimiento acerca de métodos para el cómputo del flujo óptico. Estos pueden ser clasificados en cuatro grupos principalmente: aquellos basados en derivadas, correlación, energía y fase. Sin embargo, el cómputo del flujo óptico así como de algunos otros algoritmos, sufre de alta complejidad computacional, por lo que aplicaciones robóticas y espaciales requieren el uso de sistemas de cómputo de alto costo. Para este tipo de implementaciones es común el uso de PCs, Procesadores Digitales de Señales (Digital Signal Processing) y Microcontroladores. En los últimos años el desarrollo de hardware, ha hecho posible el uso de microprocesadores con un alto poder de cómputo, los cuales son empleados en sistemas embebidos como tarjetas de desarrollo basados en microprocesadores, dispositivos lógicos reprogramables FPGA y dispositivos móviles.

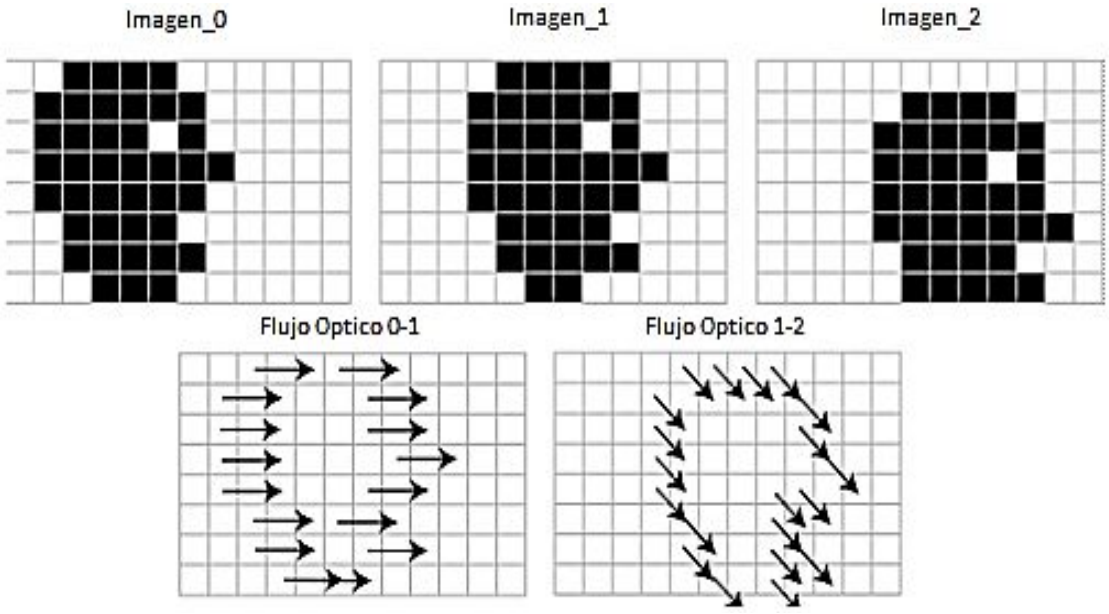

Figura 3. Secuencia de tres imágenes, donde el flujo óptico es representado por el contorno de los pixeles entre el cuadro 0 y 1 así como el cuadro 1 y 2.

El presente trabajo presenta el resultado de haber implementado un algoritmo proyecto, busca hacer uso de las tecnologías de sistemas embebidos y dispositivos móviles para implementar algoritmos de flujo óptico, y detectar movimiento en áreas donde en ciertos momentos no debe existir movimiento. Los equipos comerciales son extremadamente costosos, y con el uso de sistemas embebidos y sistemas móviles, se pueden tener sistemas con un menor costo.

2. Desarrollo

En esta sección se presenta el desarrollo del trabajo, comenzando por una evaluación de métodos que puedan ser implementados en la tarjeta Raspberry Pi, de acuerdo a sus limitaciones.

La Mecatrónica en México, Vol. 4, No. 2, páginas 65 – 76, Mayo 2015
Disponible en línea en www.mecamex.net/revistas/LMEM
ISSN en trámite, 2014 Derechos de autor y derechos conexos, Asociación Mexicana de Mecatrónica A.C

2.1 Evaluación de técnicas de flujo óptico.

Es posible evaluar diversos métodos comparándolos uno respecto de otro en términos de precisión y velocidad de computo [2], [7], en alguna secuencia de imágenes conocida tal como ‘‘Rubik’s cube’’, ‘‘Hamburg Taxi’’, ‘‘Translating Tree’’ o “Yosemit with clouds” presentadas en la figura 4, o bien referirnos a una base propuesta como punto de referencia [7], la cual presenta secuencias de imágenes de textura oculta, sintética, tipo estéreo y de cámara de alta velocidad y es presentada en la figura 5.

En el desarrollo de algoritmos de Flujo Óptico se considera lo siguiente:

Precisión. Dentro de la literatura uno de los principales puntos a evaluar para cualquier tipo de técnica para el cálculo del flujo óptico es el desempeño de los valores obtenidos, en particular Baker [7] refina y extiende la metodología de evaluación propuesta por Barron [2] en términos de: medidas de desempeño y estadísticas calculadas.

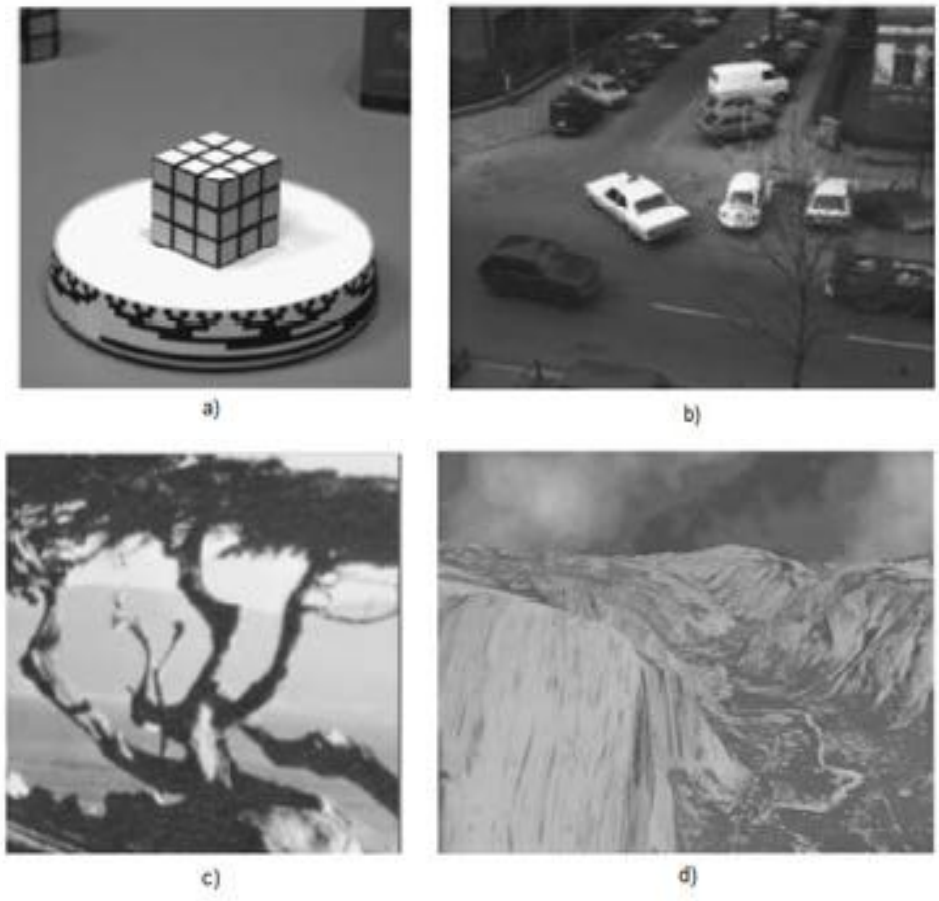

Figura 4. Secuencias de imágenes comúnmente empleadas para la evaluación de algoritmos de flujo óptico a) Cubo Rubik, b) Hamburg Taxi, c) Translating Tree, d) Yosemit with cloud.

La Mecatrónica en México, Vol. 4, No. 2, páginas 65 – 76, Mayo 2015
Disponible en línea en www.mecamex.net/revistas/LMEM
ISSN en trámite, 2014 Derechos de autor y derechos conexos, Asociación Mexicana de Mecatrónica A.C

Figura 5. Secuencias de imágenes contenidas en Middlebury database.

Error angular (AE). Es la medida más comúnmente usada para medir el desempeño del flujo óptico. Es calculado entre un vector de flujo estimado (u, v) y un vector de referencia (u_r, v_r). Se calcula mediante el producto punto de los vectores, dividiendo por el producto de sus distancias, y calculando la inversa del coseno, ecuación (1). Esta medición fue introducida por Barrón [2].

$$AE = \cos^{-1}\frac{1 + u * u_r + v * v_r}{\sqrt{1 + u^2 + v^2}\sqrt{1 + {u_r}^2 + {v_r}^2}} \tag{1}$$

Error de punto final (EE). Es posible medir el error absoluto en un punto final de flujo, usado por Otte & Nagel [9](1994) y definido en la ecuación (2).

$$EE = \sqrt{(u - u_r)^2 + (v - v_r)^2} \tag{2}$$

Interpolación (IE). Para hacer uso de interpolación de imágenes Szeliski introduce el error de interpolación para ser la raíz cuadrada media (RMS) diferencia entre una imagen de referencia I_r y la imagen interpolada estimada I, ecuación (3).

$$IE = [\frac{1}{N}\sum_{(x,y)}(I(x,y) - I_r(x,y))^2]^{\frac{1}{2}} \tag{3}$$

Donde N es el número de pixeles. Para imágenes en color, es necesario considerar la norma $L2$ del vector de diferencias RGB.

Interpolación normalizada (NE). El error de interpolación normalizada entre la imagen interpolada y la imagen de referencia es denotado por la ecuación (4).

$$NE = [\frac{1}{n}\sum_{(x,y)}\frac{(I(x,y) - I_r(x,y))^2}{||\nabla I_r(x,y)||^2 + \in}]^{\frac{1}{2}} \tag{4}$$

Estadísticas Calculadas. Se puede realizar variantes de las mediciones anteriores, como es el promedio de los errores angulares en la imagen completa, y se obtiene una medida global llamada error medio angular (AAE) para la imagen. También se puede definir AAE como medición de una secuencia (SAAE) así como su respectiva desviación estándar (SD) [7].

La Mecatrónica en México, Vol. 4, No. 2, páginas 65 – 76, Mayo 2015
Disponible en línea en www.mecamex.net/revistas/LMEM
ISSN en trámite, 2014 Derechos de autor y derechos conexos, Asociación Mexicana de Mecatrónica A.C

Velocidad. Un punto a resaltar en la evaluación de cada método es el tiempo de computo necesario para la estimación del flujo óptico, para lo cual es necesario reproducir cada método e implementarlo sobre una misma plataforma a fin de obtener resultados lo más exactos para su comparación, una aproximación a esto es la presentada por Baker [7] para los cuales se usa la secuencia Urban contenida en Middlebury database, sin embargo, no se encuentran normalizados para un mismo entorno de programación, velocidad de PC, numero de núcleos o aceleración de hardware, sino más bien son tratados como una guía muy aproximada de la complejidad computacional inherente a los algoritmos.

2.2 Flujo óptico método seleccionado (Horn & Schunk, 1981)

El Flujo Óptico cuenta con dos vectores desconocidos u y v en una sola ecuación lineal. Por lo anterior, es necesario introducir ciertas restricciones adicionales a fin de poder calcular estos vectores. Horn & Schunck [8] introducen una ecuación adicional de restricción de suavidad. Esta segunda restricción es desarrollada en base a la observación a menudo en secuencias de video, los pixeles vecinos a un pixel de la imagen están sujetos a un movimiento similar al del pixel mismo. Este comportamiento resulta en un vector de campo de suave variación y una ecuación de restricción de la suavidad. El término de restricción es obtenido definiendo una función de costo adecuado. El primer componente de este costo es para representar el monto total de cambio en la imagen como una función de la posición de la imagen (x, y). Calculado como el cuadrado de las magnitudes de los gradientes de los vectores de flujo óptico y dado por la ecuación (5).

$$\varepsilon_s^2 = u_x^2 + u_y^2 + v_x^2 + v_y^2 \tag{5}$$

Otro componente de la función de costo es acerca de la constancia de iluminación sobre una misma secuencia de imágenes, ecuación (6).

$$\varepsilon_b = I_x u + I_y v + I_t \tag{6}$$

El problema puede ser reformulado como una minimización combinada de la función de costo calculado sobre la imagen completa, ecuación (7).

$$\varepsilon^2 = \iint (\varepsilon_b^2 + \alpha^2 \varepsilon_s^2) dx dy \tag{7}$$

Aquí, $\propto$ es un término de peso usado para ajustar la contribución relativa de los dos términos en la función de costo. Es posible ajustar la suavidad de el campo de flujo óptico mediante el ajuste de $\propto$. Incrementando el valor de este parámetro se incrementara el peso del termino de suavidad en la función de costo y por lo tanto, el algoritmo dará lugar a un campo de flujo óptico más suave. Este parámetro juega un papel importante solo para áreas donde el gradiente de iluminación es pequeño, previniendo ajustes aleatorios para la estimación de la velocidad de flujo ocasionados por ruido en la estimación de las derivadas. Esta formulación acepta que durante la minimización, la iluminación se verá minimizada pero no será exactamente cero. Este no es un problema debido a que en la práctica los valores de iluminación cambian muy levemente para ruido o cuantización de errores.

Los valores del par (u, v) que minimizan la función de costo presentada en (7) son el vector de campo del flujo óptico. Esto puede ser reformulado en la Ecuación (8).

$$\begin{aligned} I_x^2 u + I_x I_y v &= \propto^2 \nabla^2 u - I_x I_t \\ I_x^2 I_y u + I_y v &= \propto^2 \nabla^2 v - I_y I_t \end{aligned} \tag{8}$$

En la práctica, la solución es obtenida por el método iterativo de Gauss-Seidel [12], ecuación (9).

La Mecatrónica en México, Vol. 4, No. 2, páginas 65 – 76, Mayo 2015
Disponible en línea en www.mecamex.net/revistas/LMEM
ISSN en trámite, 2014 Derechos de autor y derechos conexos, Asociación Mexicana de Mecatrónica A.C

$$u^{n+1} = \bar{u}^n - I_x \frac{(I_x \bar{u}^n + I_y \bar{v}^n + I_t)}{\left(\propto^2 + I_x^2 + I_y^2\right)}$$
$$v^{n+1} = \bar{v}^n - I_y \frac{(I_x \bar{u}^n + I_y \bar{v}^n + I_t)}{\left(\propto^2 + I_x^2 + I_y^2\right)} \quad (9)$$

Donde n representa el número de iteración. Para el desarrollo computacional de la Ecuación (9), los gradientes y LaPlacianos pueden ser estimados numéricamente. Esto es posible mediante diversas formas. Para el cálculo del gradiente, Horn & Schunck [8] utiliza una diferenciación de primer orden en un arreglo de ocho pixeles mostrado en la figura 6a. El Laplaciano es estimado mediante la substracción del valor en un punto promedio, ponderando los valores de los pixeles vecinos, mediante el uso de una máscara de 3x3 mostrada en la figura 6b, para el cálculo de los promedios locales de vectores de flujo óptico, donde los subíndices (i, j, k) representan los números de fila, columna e imagen respectivamente. Haciendo uso de la máscara presentada en la figura 6a, donde el cálculo de I_x puede ser obtenido como en la Ecuación (10), de igual forma para los gradientes I_y y I_t.

$$I_x = \frac{1}{4}(I_{i,j+1,k-1} - I_{i,j,k-1} + I_{i+1,j+1,k-1} - I_{i+1,j,k-1} + I_{i,j+1,k} - I_{i,j,k} + I_{i+1,j+1,k} - I_{i+1,j,k}) \quad (10)$$

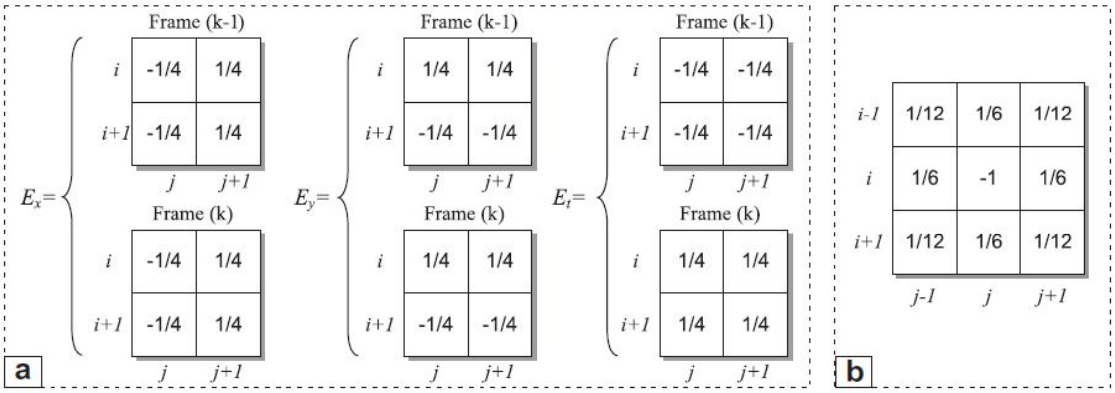

Figura 6. (a) Diferenciación de primer orden de ocho pixeles. (b) Matriz de peso para estimación de promedios locales.

2.3 Adaptación del algoritmo en entorno de procesamiento de imágenes.

Una vez considerado el algoritmo y considerando su naturaleza se prosigue a su implementación en el entorno seleccionado, para lo cual se plantea el diagrama a bloques presentado en la figura 7. El bloque principal representa una arquitectura general de procesamiento de imágenes, mediante una etapa de adquisición, procesado y etapa final, la cual representa una interpretación del movimiento en la escena, como se muestra en la figura 8.

La Mecatrónica en México, Vol. 4, No. 2, páginas 65 – 76, Mayo 2015
Disponible en línea en www.mecamex.net/revistas/LMEM
ISSN en trámite, 2014 Derechos de autor y derechos conexos, Asociación Mexicana de Mecatrónica A.C

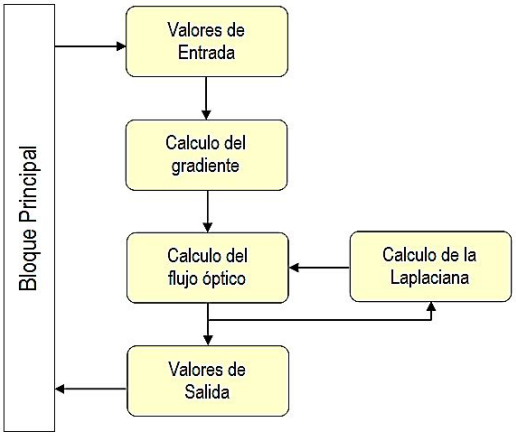

Figura 7. Diagrama a bloques del algoritmo seleccionado.

Figura 8. Etapas que constituyen un sistema para el procesamiento de imágenes.

2.4 Implementación en Sistema Embebido basado en Microprocesador

Para este propósito se utiliza la tarjeta Raspberry Pi, la cual se muestra en la figura 9. Se utiliza el lenguaje de programación Python, y librerías OpenCV para la adquisición de las imágenes, así como el procesamiento de las imágenes.

Figura 9. Tarjeta Raspberry Pi, para implementar el algoritmo de Horn & Schunk.

El siguiente código es utilizado para la implementación del algoritmo de Horn & Schunk en la tarjeta Raspberry Pi.

```
import cv2
import sys
import numpy as np

def Pixel(img, i, j):
    i = i if i >= 0 else 0
    j = j if j >= 0 else 0
    i = i if i < img.shape[0] else img.shape[0] - 1
```

```
    j = j if j < img.shape[1] else img.shape[1] - 1
    return img[i, j]

def xDer(img1, img2):
    res = np.zeros_like(img1)
    for i in xrange(res.shape[0]):
        for j in xrange(res.shape[1]):
            sm = 0
            sm += Pixel(img1, i,     j + 1) - Pixel(img1, i,     j)
            sm += Pixel(img1, i + 1, j + 1) - Pixel(img1, i + 1, j)
            sm += Pixel(img2, i,     j + 1) - Pixel(img2, i,     j)
            sm += Pixel(img2, i + 1, j + 1) - Pixel(img2, i + 1, j)
            sm /= 4.0
            res[i, j] = sm
    return res

def yDer(img1, img2):
    res = np.zeros_like(img1)
    for i in xrange(res.shape[0]):
        for j in xrange(res.shape[1]):
            sm = 0
            sm += Pixel(img1, i + 1, j    ) - Pixel(img1, i, j    )
            sm += Pixel(img1, i + 1, j + 1) - Pixel(img1, i, j + 1)
            sm += Pixel(img2, i + 1, j    ) - Pixel(img2, i, j    )
            sm += Pixel(img2, i + 1, j + 1) - Pixel(img2, i, j + 1)
            sm /= 4.0
            res[i, j] = sm
    return res

def tDer(img, img2):
    res = np.zeros_like(img)
    for i in xrange(res.shape[0]):
        for j in xrange(res.shape[1]):
            sm = 0
            for ii in xrange(i, i + 2):
                for jj in xrange(j, j + 2):
                    sm += Pixel(img2, ii, jj) - Pixel(img, ii, jj)
            sm /= 4.0
            res[i, j] = sm
    return res

averageKernel = np.array([[ 0.08333333,  0.16666667,  0.08333333],
                          [ 0.16666667,  0.        ,  0.16666667],
                          [ 0.08333333,  0.16666667,  0.08333333]], dtype=np.float32)

def average(img):
    return cv2.filter2D(img.astype(np.float32), -1, averageKernel)

def translateBrute(img, u, v):
    res = np.zeros_like(img)
    u = np.round(u).astype(np.int)
    v = np.round(v).astype(np.int)
    for i in xrange(img.shape[0]):
        for j in xrange(img.shape[1]):
            res[i, j] = Pixel(img, i + v[i, j], j + u[i, j])
    return res

def hornShunckFlow(img1, img2, alpha):
    img1 = img1.astype(np.float32)
    img2 = img2.astype(np.float32)
```

```
    Idx = xDer(img1, img2)
    Idy = yDer(img1, img2)
    Idt = tDer(img1, img2)

    u = np.zeros_like(img1)
    v = np.zeros_like(img1)

    #100 iterations enough for small example
    for iteration in xrange(100):
        u0 = np.copy(u)
        v0 = np.copy(v)

        uAvg = average(u0)
        vAvg = average(v0)
        u = uAvg - 1.0/(alpha**2 + Idx**2 + Idy**2) * Idx * (Idx * uAvg + Idy * vAvg
+ Idt)
        v = vAvg - 1.0/(alpha**2 + Idx**2 + Idy**2) * Idy * (Idx * uAvg + Idy * vAvg
+ Idt)

    return u, v

if __name__ == '__main__':
    img1c = cv2.imread(sys.argv[1])
    img2c = cv2.imread(sys.argv[2])
    img1g = cv2.cvtColor(img1c, cv2.COLOR_BGR2GRAY)
    img2g = cv2.cvtColor(img2c, cv2.COLOR_BGR2GRAY)

    u, v = hornShunckFlow(img1g, img2g, 0.1)
    imgRes = translateBrute(img2g, u, v)
    cv2.imwrite('res.png', imgRes)
    print img1g
    print translateBrute(img2g, u, v)
```

En la figura 10 se muestra el sistema completo para detección de movimiento, donde se utiliza una cámara, y una pantalla adicional a la tarjeta.

El sistema está diseñado para realizar el análisis entre dos imágenes previamente seleccionadas, las cuales son analizadas utilizando el método Horn & Schunk, y de la matriz resultante se analizan los vectores para determinar los niveles de movimiento. A continuación, se presentan los resultados de la investigación.

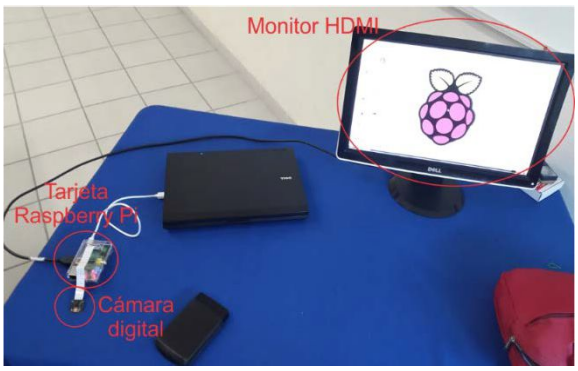

Figura 10. Sistema Raspberry Pi, detección de movimiento utilizando el algoritmo de Horn & Schunk.

La Mecatrónica en México, Vol. 4, No. 2, páginas 65 – 76, Mayo 2015
Disponible en línea en www.mecamex.net/revistas/LMEM
ISSN en trámite, 2014 Derechos de autor y derechos conexos, Asociación Mexicana de Mecatrónica A.C

3. Resultados

Se desarrolló una aplicación en un sistema embebido basado en microcontrolador, de nombre comercial Raspberry Pi. El sistema embebido utiliza un sistema operativo basado en Linux, y el desarrollo de la aplicación ha sido desarrollado en lenguaje Python. Cuando la cámara detecta movimiento, se toma una fotografía y la imagen es almacenada en la memoria, para que, en un momento dado, pueda ser analizada posteriormente. También se ha dejado la opción al usuario de activar una alarma sonora, la cual puede ser programada en la misma aplicación. La figura 11 muestra imágenes de los resultados de la detección de movimiento.

Figura 11. Imágenes obtenidas al detectar movimiento.

Las imágenes de la figura 11, muestran a una persona en posición estática (figura 11a), y cuando procede a moverse, el algoritmo de Flujo Óptico Horn & Schunk implementado en la aplicación y desarrollada en lenguaje Python, detecta cambios en la intensidad de los pixeles que cambian, por lo que el sistema procede a almacenar la imagen (Figura 11b) para un análisis posterior de las incidencias en la zona que está siendo monitoreada. La imagen almacenada tiene formato jpg, con una resolución de 5 MPixeles. Debido a la lentitud de procesamiento del sistema, no se considera la impresión de los vectores de flujo óptico de cada uno de los vectores, simplemente se procesan las magnitudes y de acuerdo a eso se toman las decisiones de movimiento activo.

Se puede apreciar un retardo considerable en la respuesta del sistema a la acción del movimiento, por lo que es necesario mejorar la respuesta para se considere factible su uso como sistema de detección de movimiento. Sin embargo, es importante resaltar que el sistema resulta compacto, por lo que es fácil de ocultar, y el consumo de corriente es bajo, por lo que puede ser alimentado fácilmente con una batería externa. Este sistema tiene un costo aproximado de $1,500.00 MN, sin considerar el monitor de alta definición. Se tiene noticia de que está por salir una versión mejorada de la tarjeta Raspberry Pi, la cual se va a llamar Raspberry Pi 2, que va a contar con una mayor cantidad de memoria RAM, por lo que se considera que esto viene a resolver el problema.

A pesar de que el sistema resulta ser lento en la tarjeta Raspberry Pi, es suficiente para realizar los cálculos necesarios. Las imágenes de la figura 11 fueron pre-procesadas para manejar las mismas en formato de grises y ahorrar de esta manera un poco de memoria. Posteriormente, se configura el

sistema para detectar movimiento, de forma que cuando la magnitud de los vectores característicos del flujo óptico son mayores de un cierto umbral, entonces se concluye que existe movimiento, por lo cual la imagen es almacenada en la memoria del sistema embebido.

4. Conclusiones

Tomando en cuenta que un dispositivo móvil puede resultar costoso a comparación de un sistema embebido, se procede a realizar la implementación del algoritmo Horn & Schunk en un sistema embebido Raspberry Pi, para comprobar el funcionamiento del mismo y saber si el sistema tiene la capacidad de procesamiento adecuado. Se utiliza el lenguaje Python y librerías OpenCV para el desarrollo de la aplicación, obteniendo de esta manera un sistema de monitoreo con detección de movimiento, de manera compacta y económica. La respuesta del sistema embebido es satisfactoria, aun cuando se hace lento debido a la exigencia de cómputo que implica el algoritmo implementado. Cuando se detecta el movimiento, se toma una fotografía y se almacena para que se pueda realizar un análisis posterior.

En estos casos, es importante resaltar que los sistemas tienen la capacidad de realizar alguna acción secundaria, como emitir alarmas sonoras, envío de mensajes SMS en caso de teléfonos inteligentes.

Se espera que con el uso de la tarjeta Raspberry Pi 2, se mejore la respuesta del sistema, de forma que pueda ser utilizado de una manera más continua.

Referencias

[1] Tabatabai, A., R. Jasindchi, and T. "*Naveen. Motion estimation methods for video compression – A review*". Elsevier Science Ltd, J. Franklin Inst., 335B(8):1411–1441. 1998.
[2] Barron, J. and D. Fleet. "*Performance of Optical-Flow Techniques*". International Journal of Computer Vision 12(1): 43-77. 1994.
[3] Von Helmholtz, D. "*Treatise on physiological optics*". Translated from the 3rd German Edition. Vol.I James. Soutthall, JPC Dover Publications. 1925.
[4] Gibson, J. "*The senses considered as perceptual systems*". A symposium on Perception: 230-232. 1967.
[5] Koray, G. and A. Saranli. "*An FPGA based high performance optical flow hardware design for computer vision applications*". Journal of Microprocessors and Mycrosystems, 37: 270-286. 2013.
[6] Raudies, F., R. Gilmore, K. Kretch, J. Franchak and K. Adolph. "*Understanding the Development of Motion Processing by Characterizing Optic Flow Experienced by Infants and their Mothers*". In International Conference on Development and Learning and Epignetic Robotics (ICDL): 1-6. 2012.
[7] Baker, S., D. Scharstein, J. Lewis, S. Roth, M. Black, and S. Richard, "*A database and evaluation methodology for optical flow*". International Journal of Computer Vision 92, 1-31. 2011.
[8] Horn, B. and B. Schunck. "*Determining optical flow*". AI 17: 185–203. 1981.

Año 4, No. 2, Febrero-Mayo 2015, es una publicación cuatrimestral editada por la Asociación Mexicana de Mecatrónica A.C. para dar a conocer artículos originales y especializados en todas las áreas de la Ingeniería Mecatrónica. Certificado de Reserva de Derechos al Uso Exclusivo No. 04-2012-092010534100-102, ISSN: en trámite, ambos otorgados por el Instituto Nacional del Derecho de Autor. Asociación Mexicana de Mecatrónica A.C. se ubica en calle Fonología No. 116, Col. Tecnológico, C.P. 76158, Querétaro, Querétaro, www.mecamex.net/revistas/LMEM/, vinculacion_revista@mecamex.net, Editores responsables: Juan Manuel Ramos Arreguín y José Emilio Vargas Soto.