Revista Ingeniería y Región
2013(10): 41-53

Diseño e implementación de algoritmo para el procesamiento de imágenes en sistemas embebidos

Design and implementation of algorithm for images processing in embedded systems

José Salgado Patrón, Luis Vásquez Díaz y Mauricio Vidal Solano

Resumen

En este artículo se presenta el diseño y la implementación de un sistema de reconocimiento de colores, detección de bordes y seguimiento de color rojo por medio de un algoritmo de visión artificial, para lograr esto se tiene en cuenta que el hardware usado será un sistema embebido de bajo costo y fácil adquisición, en este caso se utilizarán la FPGA SPARTAN 3AN y la RASPBERRY PI , el lenguaje de programación usado para la creación de algoritmos es VHDL en la FPGA y PYTHON en el caso de la RASPBERRY PI.

En el proceso de adquirir las imágenes se necesitan 2 tipos de cámaras, su principal característica es la compatibilidad con los sistemas embebidos usados. Una vez adquirida la imagen se aplican técnicas de umbralización y segmentación, de los diferentes métodos existentes se toman en cuenta el gradiente y el operador sobel. Para poder ver los resultados en ambos casos se usan los puertos que cada una de las tarjetas presenta, en el caso de la FPGA se configura la salida de video VGA y se conecta un monitor a dicha salida, y en el caso de la tarjeta RASPBERRY PI se tienen varios puertos para mostrar el resultado, se puede acceder a la interfaz gráfica de la tarjeta mediante un puerto HDMI, puerto Ethernet y puerto RCA.

Palabras claves: procesamiento embebido; visión artificia; Raspberry p; Spartan 3an.

Abstract

This project will develop a color recognition system, edge detection and red color monitoring by an artificial vision algorithm, to achieve this, it is taken in mind that the hardware used is an embedded system inexpensive and readily available, in this case is used the FPGA SPARTAN 3AN and RASPBERRY PI, the programming language used to create the algorithms are VHDL for FPGA and PYTHON for RASPBERRY PI.

In the acquiring process the images are needed 2 camera kinds, its main feature is support for embedded systems used. Once acquired, the image thresholding and segmentation techniques are applied, of the different existing methods takes into account the gradient and the sobel operator. To show the results in both cases are used ports each card, in the FPGA case is configured video output and a VGA monitor is connecting to said output , and for Raspberry PI card will have multiple ports to show , it can access the card graphical interface via an HDMI port , Ethernet port and RCA port.

Keywords: embedded processing; computer vision; Raspberry P; Spartan 3an.

1 Ingeniero Electrónico, Docente Universidad Surcolombiana Neiva. Avenida Pastrana Carrera 1a. Jose.salgado@usco.edu.co
2 Ingeniero Electrónico, Universidad Surcolombiana Neiva. Avenida Pastrana Carrera 1a. luisvd007@ingenieros.com
3 Ingeniero Electrónico, Universidad Surcolombiana Neiva. Avenida Pastrana Carrera 1a. mauriviso2@hotmail.com

1. Introducción

El procesamiento de imágenes es un conjunto de técnicas utilizadas para manipular la información contenida en una imagen (Vélez, J. 2011). Para realizar una tarea específica, algunas de estas técnicas se utilizan para detección de objetos, biometría, inspección y control de calidad en procesos industriales, visión artificial etc. Son muchos los programas y herramientas desarrollados para realizar estas tareas, y se han logrado muy buenos resultados utilizando computadoras pero muy pocos ejemplos se pueden encontrar en sistemas embebidos.

Existen numerosas definiciones de sistemas embebidos, algunos lo conocen como cualquier dispositivo que incluye un computador programable, pero en sí mismo no es un computador de propósito general (Wolf, W., 2008), otros cambian el concepto de computador programable y lo definen como un sistema electrónico que contiene un microprocesador o microcontrolador, sin embargo, no pensamos en ellos como un computador (Morton, T,.2000), En particular, un microprocesador es un componente LSI (Large Scale Integration) que realiza una gran cantidad de funciones o tareas en una sola pieza de circuito integrado (Zazks, R., 1981). Para ampliar el concepto a un nivel más práctico, se puede decir que es una combinación de hardware y software para realizar una tarea específica, es importante resaltar que el software que se ejecuta en el sistema embebido está limitado por algunas características importantes del hardware: memoria, procesamiento, consumo de energía, puertos de entrada y salida.

A nivel mundial se han realizado varias aplicaciones importantes de sistemas embebidos de visión artificial una de ellas en el instituto de ciencias y tecnología de Buenos Aires, (Raponi, M., 2009), se realizó un procesamiento de imágenes para rehabilitación visual basándose en una tarjeta de desarrollo BEAGLEBOARD y software libre. El primer sistema embebido reconocido fue el sistema de guía de Apolo desarrollado por el laboratorio de desarrollo del MIT (Instituto Tecnológico de Massachusetts) para las misiones Apolo hacia la luna. Cada vuelo hacia la luna tenía dos de estos sistemas. La función era manejar el sistema de guía inercial de los módulos de excursión lunar. En un comienzo fue considerado como el elemento que más riesgo presentaba en el proyecto Apolo. Este sistema de cómputo fue el primero en utilizar circuitos integrados y utilizaba una memoria RAM magnética, con un tamaño de palabra de 16 bits. El software fue escrito en el lenguaje ensamblador propio y constituía en el sistema operativo básico, pero capaz de soportar hasta ocho tareas simultáneas, (Computer Weekly, 2012)

En la actualidad con la nueva generación de microcontroladores de 32 bits, DSPs y FPGAs se logran velocidades de procesamiento muy altas y se pueden emplear en sistemas de visión artificial aplicados a múltiples tareas. En este proyecto se aplicarán técnicas de procesamiento de imágenes basadas en una FPGA (Xilinx Productos, 2011) y en una SBC llamada RASPBERRY PI (Raspberry Pi, 2012) que reciben la imagen directamente de una cámara y mostrarán los resultados por medio de una pantalla LCD o en su defecto en un monitor VGA.

2. Metodología

En el proceso de cumplir los objetivos propuestos por el proyecto se buscó desarrollar un algoritmo que pueda desarrollar la captura, procesamiento y visualización de la imagen como se muestra en la Figura 1, los requisitos básicos para el desarrollo del software son:

- Tomar imágenes de resolución mayor o igual a 320x240 usando cámara web o módulos con cámara.
- Aplicar métodos para detección de color y bordes por medio de umbralización, segmentación y máscaras.
- Visualizar los resultados usando un puerto de salida de video que pueda ser conectado a una pantalla.

2.1 Desarrollo del Algoritmo en Spartan 3an

El algoritmo a desarrollar en el proyecto debe permitir controlar la FPGA para realizar la adquisición de la imagen del módulo OV7670 y mostrarla en un monitor VGA, el lenguaje de programación es en VHDL y se definirán 3 etapas básicas para realizar el código como se ve en la Figura 2. En la primera etapa se adquiere la

imagen, este bloque se encarga de generar las señales de control para la cámara y recibe el valor del pixel. En la segunda etapa se procesa la imagen, este bloque toma la imagen en formato de 8 bits y aplica la segmentación del objeto basándose en umbrales de color. La tercera etapa se encarga de generar la señal VGA para la visualización de la imagen. A continuación se verá una descripción de las etapas que se utilizaron para este proyecto:

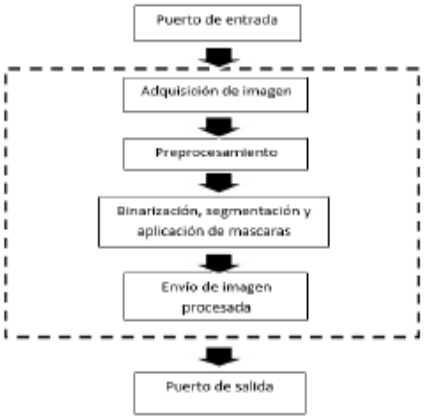

Figura 1. Descripción general del software.

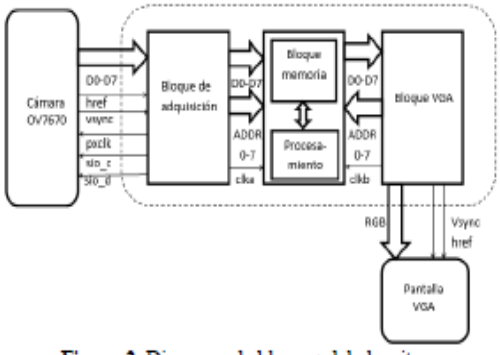

Figura 2. Diagrama de bloques del algoritmo.

2.1.1 Bloque de adquisición. Este bloque se encarga de generar la señal de reloj para el módulo de la cámara OV7670, recibe las señales de sincronización horizontal y vertical y captura el valor de 8 bits correspondiente al pixel, adicionalmente puede realizar cambios a la configuración de la cámara por medio de comunicación I2C.

Puertos y señales de datos para la adquisición de la imagen:

- **d** es el valor de pixel que se toma de la cámara, este valor oscila entre 0 y 255 (8 bits).
- **Dout** puerto de salida que se genera en el bloque para ser enviado a la memoria, este valor oscila entre 0 y 255 (8 bits).
- **addr** puerto de dirección de memoria para guardar los pixeles de la imagen
- **we** habilita escritura de la memoria, esta señal se mantiene en alto durante la escritura de la imagen en memoria.

2.1.2 Bloque de memoria. Esta memoria en particular es un bloque generado por medio de una herramienta de XILINX llamada **CORE Generator** basta con ingresar la longitud del dato y la cantidad de datos a guardar, adicionalmente se puede escoger el tipo de memoria, para mejor manejo del algoritmo se configuró la memoria

como una RAM de doble puerto, es decir que por un puerto se escribe el dato y por el otro puerto se lee el dato. Como se muestra en la Figura 3.

Figura 3. Bloque de memoria.

Variables del Bloque de memoria:

- Clka: señal de reloj de la memoria puerto A
- Wea: señal que habilita escritura
- Addra: dirección donde se escribe el dato
- Dina: dato a escribir en memoria
- Clkb: señal de reloj de la memoria Puerto B
- Addrb: dirección del dato a leer
- Doutb: dato leído

2.1.3 Bloque VGA. Este bloque se encarga de generar la señal de video para visualizar los resultados del procesamiento en un monitor VGA. El bloque lee la imagen desde la memoria a través de los puertos *frame_addr y frame_pixel*, descompone los datos de pixel en sus componentes RGB para poder ser enviados como *vga_green, vga_red, vga_blue* y generar las señales para la sincronización vertical y horizontal *vga_vsync* y *vga_hsync,* como se muestra en la Figura 4.

En esta FPGA cada color es compuesto por 4 bits en total son 12 bit necesarios para representar un pixel en la pantalla pero la cámara entrega 8 bits por pixel, así que se debe ajustar este valor de forma que no afecte demasiado. Las señales de sincronismo horizontal y vertical (*hsync y vsync*) se generan por medio de contadores haciendo un barrido por toda la pantalla, primero se desplazan columnas y luego filas hasta llegar al valor máximo, teniendo en cuenta que la resolución VGA es 640x480 pixeles.

Figura 4. Diagrama VGA.

2.1.4 Bloque de procesamiento

- **Binarización de la imagen.** El formato de imagen YUV que utiliza la cámara es muy sensible a los cambios de luz, esta característica se puede aprovechar para detección de movimiento y segmentación en la imagen, después de realizar ensayos con diferentes fuentes de luz y colores se detectó que la luz de color rojo contrasta bastante con el resto de la imagen, en especial en el componente Y donde se muestra un mayor incremento. Mediante pruebas se determinó un valor umbral para este componente, binarizando la imagen de tal forma que solo muestre el segmento rojo que detecta, como se indica en la Figura 5.

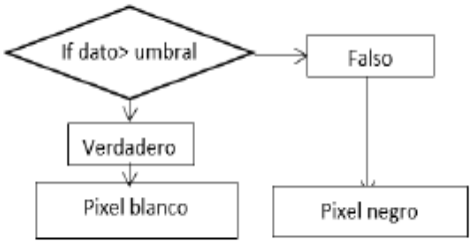

Figura 5. Diagrama de binarización.

- **Parametrización del objeto en la imagen.** Después de segmentar la imagen se procede a marcar en qué posición se encuentra el objeto que se está detectando, esto se hace a medida que se están leyendo los valores del pixel. Se almacena el valor de los contadores horizontal y vertical en una variable que se utilizara posteriormente para dibujar un objeto en la pantalla.
- **Marcación del movimiento.** A partir de las coordenadas tomadas en el proceso anterior se dibuja un rectángulo amarillo de 20x20 que siga el objeto segmentado de la siguiente manera (Figura 6):

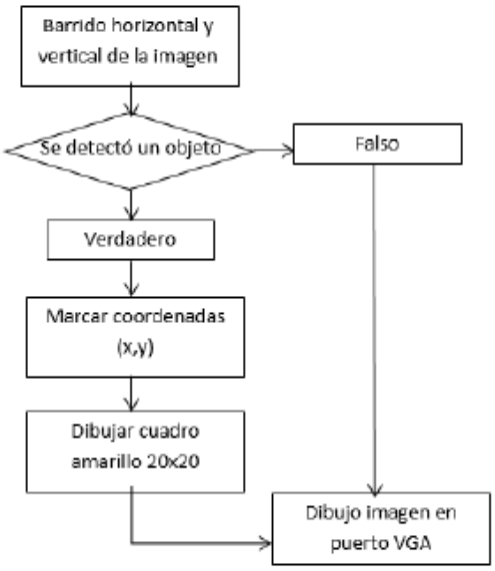

Figura 6. Diagrama de marcación.

- **Seguimiento.** Como ya se dispone de un parámetro para hacer seguimiento a la imagen, este se puede utilizar para controlar un servomotor y hacer seguimiento al haz de luz roja. Para esto es necesario generar la señal de control para el servomotor que se indican en la Figura 7.

El movimiento del servomotor está controlado por el ancho del pulso enviado, la frecuencia de dicho pulso es de 50 Hz y el ancho del pulso varia de 0,7ms a 2ms para tener un movimiento de 0 a 180° en sentido horizontal. Se utilizan contadores para generar la señal de control (*ancho_pulso*) y guardar la posición del pixel en la imagen, cuando el objeto se encuentre aproximadamente en el centro de la imagen, es decir entre 380 y 420 pixeles, el

servomotor se quedará quieto, cuando el objeto se mueva hacia la izquierda, el ancho del pulso tendrá que incrementarse. Cuando el objeto se mueva a la derecha, el ancho del pulso debe disminuirse (Figura 8).

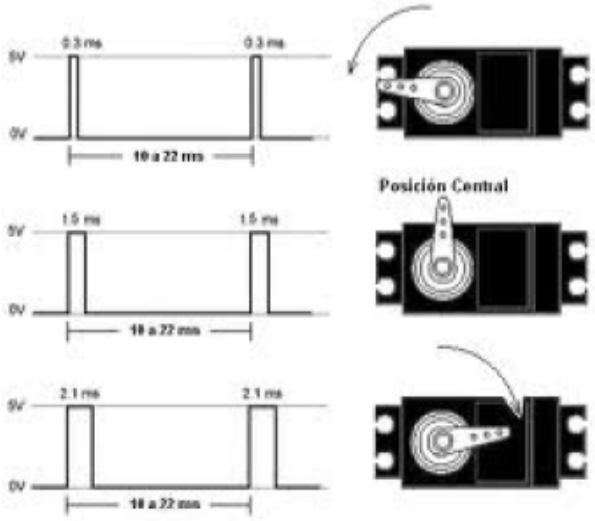

Figura 7. Servomotor.

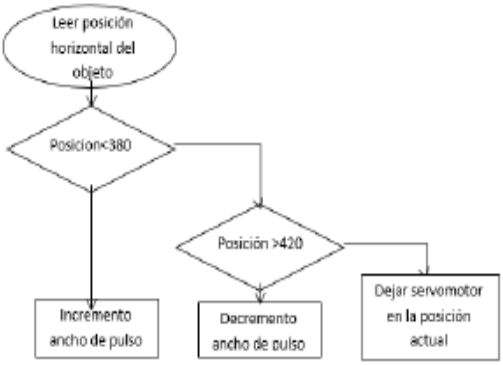

Figura 8. Diagrama seguimiento.

2.2 Desarrollo del algoritmo en Raspberry Pi

En el proyecto la meta final es la detección de bordes en imágenes y la detección de color, para poder lograr los objetivos deseados, se debe hacer uso de librerías que no se encuentran instaladas, para ello el sistema operativo facilita una ventana de terminal que funciona de la misma manera del sistema operativo Ubuntu, las librerías a usar son IMGPROC (dedicada a imágenes) y MATH (dedicada a operación de matrices y operaciones matemáticas), una vez las librerías se encuentren correctamente instaladas, se procede a capturar imágenes por medio de una webcam, se debe estar seguro de que la cámara es compatible con la tarjeta, de lo contrario el compilador no podrá acceder a las imágenes que la cámara logre adquirir.

2.2.1 Algoritmo de detección de bordes. Para llevar a cabo este algoritmo se debió seguir un orden, este orden se vio repartido en pasos que se van a explicar a continuación.

Paso 1: lo primero que se hace es declarar las librerías que se van a usar.

```
from imgproc import *
import math
```

Paso 2: la captura de imagen por la webcam, la palabra Camera hace alusión a una variable creada en la librería llamada IMGPROC que invocará la webcam reconocida por el sistema operativo, X y Y son el tamaño de resolución a tener en cuenta para el procesamiento.

```
camera = Camera(x,y)
```

Paso 3: se genera la apertura de visor de la imagen, la función VIEWER hace referencia a una variable creada en la librería *IMGPROC*, que creara un visor inicialmente vacío.

```
viewer = Viewer(x, y, "deteccion borde")
```

Paso 4: al grabar la imagen, se debe tener en cuenta un par de imágenes para ajustar la cámara así esta será una imagen que evite en lo posible tener errores, es solo necesario si no se ha trabajado antes con la cámara.

```
img = camera.grabImage()
```

Paso 5: para lograr la detección de bordes se debe crear una imagen vacía del mismo tamaño de la original, con el fin de almacenar en ella el resultado que se va a obtener.

```
img_borde = Image(x, y)
```

Se muestra la imagen capturada en el visor que se creó, este paso se puede omitir si solo se desea ver los resultados finales.

```
viewer.displayImage(img)
```

Paso 6: lo que se hace a continuación es iterar sobre cada pixel en la imagen, la imagen se convoluciona con el operador de Sobel. La convolución se realiza moviendo el valor del operador a través de la imagen, un píxel a la vez. Cada píxel y sus vecinos están ponderados por un valor correspondiente a su intensidad, y se suman para producir un nuevo valor, este valor se acumulará para posteriormente calcular el valor del gradiente.

```
for x in range(1, img.width - 1):
    for y in range(1, img.height - 1):
        r, g, b = img[x - 1, y - 1]
        inte = (r + g + b)
        Gx += -inte
        Gy += -inte
```

Paso 7: calcular el valor del gradiente, esto se logra aplicando el teorema de Pitágoras.

```
grad = math.sqrt((Gx * Gx) + (Gy * Gy))
```

Paso 8: normalizar la longitud del gradiente, el máximo valor posible del gradiente será 4328, este número lo podemos calcular de la siguiente manera:

Si R=G=B=255, entonces la variable *inte* será 765, teniendo en cuenta que se almacena el valor en las cuatro direcciones posibles y que todos los valores sean máximos, el valor acumulado será 3060 el valor del gradiente en ese caso será $(\sqrt{(2 * (3060)^2)} = 4328$. La manera de normalizar el gradiente es dividir este por el valor más alto y multiplicar por 255, esto se convierte en un numero entero para poder manejarlo más fácil.

```
grad = grad / 4328 * 255
grad = int(grad)
```

Paso 9: dibujar la longitud de la gradiente en una imagen de bordes, almacenando cada uno de los valores obtenidos.

```
img_borde[x, y] = grad, grad, grad
```

Paso 10: mostrar imagen de la detección de borde que se obtiene.

```
viewer.displayImage(img_borde)
```

2.2.2 Algoritmo de detección de color. Para este algoritmo se ha tenido en cuenta que el color que se detecte va a ser el rojo, por lo siguiente se debe crear 3 variables constantes a las cuales se llamaran variables de referencia, y entre las que van a estar R (rojo con mayor valor para su fácil detección), G (verde) y B (azul), en este algoritmo se encuentran pasos similares al anterior, por lo tanto no se repetirán, es el caso del paso 1 hasta el paso 5.

Paso 6: creación de las constantes de referencia para la comparación, la constante umbral es un valor que se varía dependiendo de la luminosidad, y su valor se logra del resultado de un método heurístico.

```
ref_red = 192
ref_green = 80
ref_blue = 80
umbral = 96
```

Paso 7: iterar sobre cada pixel de la imagen y extraer sus componentes principales.

```
for x in range(0, img.width):
    for y in range(0, img.height):
        red, green, blue = img[x, y]
```

Paso 8: comparar el color de cada pixel con su referencia.

```
d_red = ref_red - red
d_green = ref_green - green
d_blue = ref_blue – blue
```

Paso 9: calcular la longitud de la línea representada por la diferencia de color, que está dada por $\sqrt{((\text{Rojo}^2) + (\text{Verde}^2) + (\text{Azul}^2))}$. Si la línea representada es mayor que el umbral no dibujar nada. Si la línea representada es menor que el umbral, dibujar un punto de color blanco.

```
length = math.sqrt( (d_red * d_red) + (d_green * d_green) + (d_blue * d_blue) )
        if length > umbral:
            img[x, y] = 0, 0, 0
        else:
            img[x, y] = 255, 255, 255
```

Paso 10: mostrar imagen de la detección de color que se obtiene.

```
viewer.displayImage(img)
```

3. Resultados

Luego de implementar el algoritmo en la tarjeta de desarrollo SPARTAN 3AN se verificaron las señales de control y datos en todas las etapas del algoritmo. Primero se midieron las señales de la etapa de captura de la imagen, luego se midieron las señales de salida del módulo VGA, al confirmar que todas las señales estuvieran presentes, se hizo la prueba verificando el resultado en un monitor VGA.

Para poder recorrer toda la imagen obtenida por la cámara fue necesario hacer un barrido, este barrido es solo autorizado por una señal que se llamó Pclk que posee una frecuencia de 12.5Mhz como observa en la Figura 9.

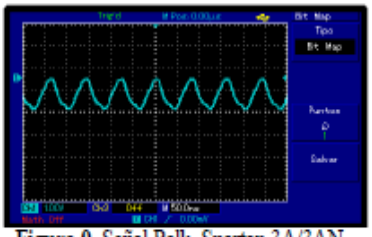

Figura 9. Señal Pclk. Spartan 3A/3AN.

La señal llamada Vsync de un periodo de 3 milisegundos que se observa en la Figura 10, se usa para recorrer las columnas desde la posición 0 y se va incrementando el recorrido.

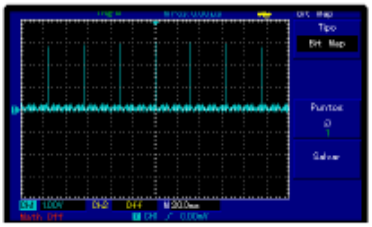

Figura 10. Señal Vsync. Spartan 3A/3AN.

La señal Href que tiene un periodo de 125 microsegundos como se observa en la Figura 11, es la encargada de hacer el barrido en las filas.

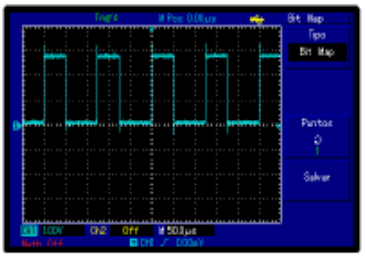

Figura. 11. Señal Href. Spartan 3A/3AN.

Para poder lograr el seguimiento de la luz de color rojo fue necesario que la imagen que se obtuvo se mostrara en formato YUV que era el indicado para poder observar los cambios de luz entre un color y otro, esta imagen en YUV se puede observar en la Figura 12.

Figura. 12. Imagen en formato YUV. SPARTAN 3A/3AN.

Una vez se logró umbralizar de manera exitosa el color que se deseaba ver en la pantalla, se aplicó un marcador de color amarillo que permitió ver el seguimiento de la luz roja, para ver el resultado fue necesario proyectar la imagen mediante un puerto para monitor VGA como se observa en la Figura 13.

Figura. 13. Marcador siguiendo el color SPARTA 3A/3AN.

En el inicio del proyecto se intenta aplicar el operador de sobel a una imagen de color y lograr todos los objetivos del proyecto con la SPARTAN 3A/3AN, debido a que la tarjeta no presento una buena velocidad para trabajar imágenes a color, lo que se logró hacer en el campo de la visión a color fue muy poco, en la Figura 14 se muestra la imagen en tonalidades de gris que se logró obtener.

Figura. 14. Imagen binarizada SPARTAN 3A/3AN.

Para el manejo del servomotor se hizo uso de una señal generada por la FPGA, esta señal debia tener ciertas características para lograr excitar el servomotor de una manera óptima sin cambios bruscos de movimiento, la frecuencia usada y el ancho de pulso de 1,8 milisegundos, que se muestran en la Figura 15 y la Figura 16 respectivamente.

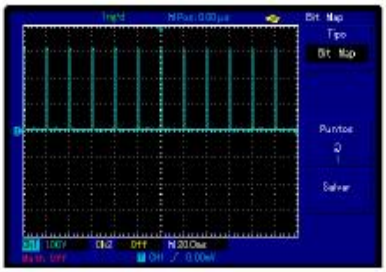

Figura. 15. Señal de control del servomotor SPARTAN 3A/3AN.

Figura 16. Ancho de pulso Spartan 3A/3AN.

Para lograr el movimiento de la cámara en el eje horizontal se fabrica un soporte que va conectado al eje del servomotor y una base metálica que no permite que se caiga en el momento de girar, además aísla el servomotor de cualquier daño externo como se muestra en la Figura 17.

Figura 17. Spartan 3A/3AN y cámara OV7670.

Para visualizar las imágenes obtenidas con la tarjeta Raspberry Pi, se realizó una conexión de escritorio remoto, en ambos casos fue necesario ajustar los umbrales de binarización según la luz presente en el momento. A continuación se muestran algunas capturas de las imágenes y resultados del procesamiento de imágenes en las Figuras 18, 19 y 20.

Figura 18. Captura de imagen con Raspberry Pi mediante la webcam.

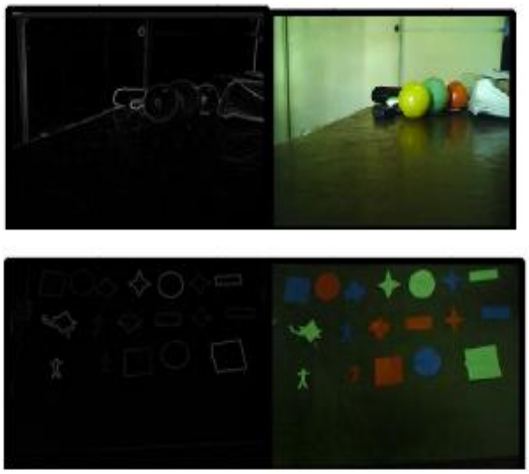

Figura 19. Resultado de la detección de borde de la imagen con Raspberry Pi.

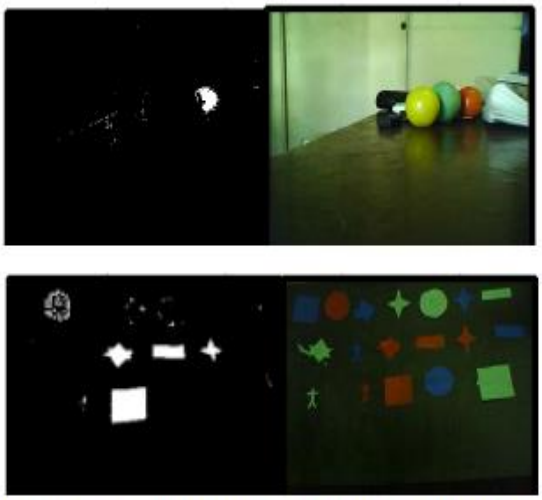

Figura 20. Resultado de la segmentación de color rojo de la imagen con Raspberry Pi.

4. Conclusiones

Con la RASPBERRY PI se logra mayor facilidad para aplicar algoritmos de procesamiento de imágenes, pero al necesitar un sistema operativo instalado no se puede aprovechar todo su potencial para una tarea específica que requiera velocidad y ejecución en tiempo real.

En Colombia, el desarrollo en el área de visión artificial se ve muy limitado por la falta de componentes electrónicos de calidad en la industria para el desarrollo de aplicaciones avanzadas.

La gran ventaja que tiene la FPGA contra otros sistemas embebidos es que los procesos se pueden ejecutar de forma paralela sin que estos afecten la ejecución de otros procesos, los retardos de propagación son mínimos y

dependen en gran parte de la forma de programar y conectar los bloques lógicos, pero la limitación de memoria y conexión de periféricos lo hacen más complicado de usar.

Raspberry Pi es una tarjeta que facilita el manejo de nuevos algoritmos en el área de visión artificial. Se encuentran muchas librerías compatibles con los diferentes sistemas operativos soportados, actualmente se están desarrollando dispositivos periféricos para aumentar las aplicaciones de esta tarjeta.

Para trabajar a una velocidad de 30 fotogramas por segundo aplicando los algoritmos desarrollados se necesita una gran velocidad de procesamiento, las velocidades actuales de los microcontroladores son muy bajas, una alternativa son los dsPIC'S que alcanzan velocidades de hasta 700 Mhz.

Se logra realizar una interfaz entre la tarjeta de desarrollo FPGA y los dispositivos periféricos para la adquisición, procesamiento y visualización de imágenes.

5. Referencias bibliográficas

1. ComputerWeekly, 2012. Apollo 11: The Computers that Put Man on the Moon. Consultado el 4 de febrero de 2012. http://www.computerweekly.com/feature/Apollo-11-The-computers-that-put-man-on-the-moon
2. Computing System. Design". Segunda edición. Miami. 1 -15 p.
3. Morton, T. 2000. Consultado el 25 de febrero de 2012. Embedded Microcontrollers. Prentice Hall.6–10p.
4. Raponi, M. 2009. Consultado el 20 de noviembre de 2011. Procesamiento de imágenes en tiempo real utilizando tecnología embebida. 4-10 p.
5. Raspberry Pi, 2012. Consultado el 23 abril de 2013. http://www.raspberrypi.org
6. Vélez, J. 2011. Consultado el 8 enero de 2012. Visión por Computador. 2 da. Madrid. Págs. 1-22.
7. Wolf, W. 2008. Consultado el 22 de enero de 2012. Computers as Components: Principles or Embedded
8. Xilinx Productos, 2011. Consultado el 15 de febrero de 2012. www.xilinx.com
9. Zazks, R. 1981. Consultado el 18 de diciembre de 2011. Microprocessors: From Chips to Systems. 2-4 p.