UNIVERSIDAD POLITÉCNICA SALESIANA
SEDE QUITO

CARRERA:
INGENIERÍA ELECTRÓNICA

Trabajo de titulación previo a la obtención del título de: INGENIERO ELECTRÓNICO

TEMA:
DISEÑO E IMPLEMENTACIÓN DE UN DETECTOR DE BORDES DE IMÁGENES USANDO MICROCONTROLADOR

AUTOR:
JUAN FRANCISCO ORDÓÑEZ PÁEZ

TUTOR:
LUIS GERMÁN OÑATE CADENA

Quito, abril de 2016

CESIÓN DE DERECHOS DE AUTOR

Yo Juan Francisco Ordóñez Páez, con documento de identificación N° 171832558-0, manifiesto mi voluntad y cedo a la Universidad Politécnica Salesiana la titularidad sobre los derechos patrimoniales en virtud de que soy autor del trabajo de titulación intitulado: DISEÑO E IMPLEMENTACIÓN DE UN DETECTOR DE BORDES DE IMÁGENES USANDO MICROCONTROLADOR, mismo que ha sido desarrollado para optar por el título de: Ingeniero Electrónico, en la Universidad Politécnica Salesiana, quedando la Universidad facultada para ejercer plenamente los derechos cedidos anteriormente.

En aplicación a lo determinado en la Ley de Propiedad Intelectual, en mi condición de autor me reservo los derechos morales de la obra antes citada. En concordancia, suscribo este documento en el momento que hago entrega del trabajo final en formato impreso y digital a la Biblioteca de la Universidad Politécnica Salesiana.

Nombre: Juan Francisco Ordóñez Páez
Cédula: 171832558-0
Fecha: abril del 2016

DECLARATORIA DE COAUTORÍA DEL DOCENTE TUTOR

Yo declaro que bajo mi dirección y asesoría fue desarrollado el proyecto técnico, DISEÑO E IMPLEMENTACIÓN DE UN DETECTOR DE BORDES DE IMÁGENES USANDO MICROCONTROLADOR, realizado por Juan Francisco Ordóñez Páez, obteniendo un producto que cumple con todos los requisitos estipulados por la Universidad Politécnica Salesiana, para ser considerados como trabajo final de titulación.

Quito, abril del 2016

……

Luis Germán Oñate Cadena

Cédula de identidad: 171215740-1

ÍNDICE

INTRODUCCIÓN **1**

CAPÍTULO 1 ..... **3**

1.1 Antecedentes 3

1.2 Objetivos 4

1.2.1 Objetivo General: 4

1.2.2 Objetivos Específicos: 4

1.3 Justificación 4

1.4 Alcance 5

CAPÍTULO 2 ..... **6**

2.1 Algoritmo de Sobel 6

2.1.1 Ejemplo del Cálculo de una matriz de 3x3 aplicando Sobel 7

2.2 Cuadro comparativo entre tecnologías ARM y DSPIC 10

2.3 Tecnología ARM 12

2.4 Modelo RGB 15

2.5 Histogramas 16

2.6 Método de correlación 18

CAPÍTULO 3 ..... **19**

3.1 Tarjeta STM32F429I-DISCO 19

3.1.1 Microcontrolador STM32F429I 19

3.1.2 LCD 2.4" QVGA TFT 20

3.1.3 Cámara OV7670 21

3.2 Diagramas Esquemáticos 22

3.2.1 Diagrama de Bloques del Funcionamiento del Detector 22

3.2.2 Diagrama esquemático de conexión de tarjeta y cámara 24

3.3 PCB de la placa para conexión de la cámara 25
3.4 Diagramas de Flujo 26
3.4.1 Diagrama de flujo para capturar la imagen 26
3.4.2 Diagrama de flujo del filtro Sobel 28
3.4.3 Diagrama de flujo para mostrar la imagen filtrada 29
3.4.4 Diagrama de flujo del algoritmo de Sobel en Matlab 31
CAPÍTULO 4 34
4.1 Procesamieto de imágenes con el detector y contraste Matlab 34
4.2 Resultados obtenidos tanto en la tarjeta y en software mediante la utilización del filtro Sobel..... 36
4.3 Resultados del método de correlación entre las imágenes tanto de la tarjeta de detección de bordes y software usando Matlab. 46
CONCLUSIONES 48
RECOMENDACIONES 50
REFERENCIAS 51

ÍNDICE DE FIGURAS

Figura 1. Matriz final aplicada al filtro 9

Figura 2. Matriz para el color rojo 10

Figura 3. Arquitectura Cortex-M4 15

Figura 4. Formatos RGB 888 y 565 16

Figura 5. Representación de un histograma variable vs frecuencia.....17

Figura 6. Panel frontal tarjeta STM32F429I-DISCO 19

Figura 7. Microcontrolador STM32F429-DISCO 20

Figura 8. Pantalla LCD 2.4¨ QVGA TFT.....21

Figura 9. Cámara OV7670 22

Figura 10. Diagrama de bloques del funcionamiento del detector 23

Figura 11. Diagrama esquemático de conexión con la alimentación a la tarjeta 32F429I-DISCO 24

Figura 12. Diagrama esquemático de conexión de pines de la tarjeta al módulo de cámara OV7670.....24

Figura 13. Pines de control del algoritmo de funcionamiento del detector.....25

Figura 14. Diagrama PBC 26

Figura 15. Diagrama de flujo para capturar la imagen. 27

Figura 16. Diagrama de flujo del filtro Sobel.....29

Figura 17. Diagrama de flujo para mostrar la imagen 31

Figura 18. Diagrama de flujo del filtro Sobel en Matlab 33

Figura 19. Procesamiento de imágenes con el detector de bordes y software Matla...35

Figura 20. Resultado de imágenes figura 19.a en tarjeta y software Matlab 37

Figura 21. Resultado de imágenes de un circuito impreso figura 19.d en tarjeta y software Matlab…… 39

Figura 22. Resultado de imágenes de figuras geométricas figura 21.j de un circuito impreso figura 19.g en tarjeta y software Matlab. 41

Figura 23. Resultado de imágenes de una impresora figura 19.g en tarjeta y software Matlab.....43

Figura 24. Resultado de imágenes de un edificio figura 19.m en tarjeta y software Matlab. 45

ÍNDICE DE TABLAS

Tabla 1. Cuadro comparativo entre: STM32F429I-DISCO y DSPIC 11

Tabla 2. Arquitecturas ARM 14

Tabla 3. Resultados de correlación entre imágenes obtenidas 46

ÍNDICE DE ECUACIONES

Ecuación 2.1 Matrices Kernel para el calculo del algoritmo de Sobel..……...…..…6

Ecuación 2.2 Combinación de la matriz horizontal y vertical…….....….…....…6

Ecuación 2.3 Dirección del gradiente de la Ecuación 2.2…….....……..…6

Ecuación 2.4 Mascara de Sobel para la matriz vertical.....……...,...…7

Ecuación 2.5 Mascara de Sobel para la matriz horizontal.....……...…...…7

Ecuación 2.6 Correlación de imágenes…….....…...……18

ÍNDICE DE ANEXOS

Anexo 1. Código para el análisis de resultados con histogramas 53

Anexo 2. Código para el análisis de resultados método de correlación.....53

Anexo 3. Código para la transmisión serial de las imágenes hacia el computador 54

Anexo 4. Código para el filtro Sobel en el microcontrolador STM32F429I…...……55

RESUMEN

Dentro del procesamiento digital de imágenes juega un papel importante analizando ciertas imágenes, que contengan bordes falsos. Los bordes falsos originan un problema al momento de realizar el análisis de filtrado de una imagen digital, siendo el procesamiento el momento crucial de reconstrucción de una imagen y suele verse afectado por la pérdida de información importante. El proyecto que se presenta a continuación se basa en el diseño de un hardware para el análisis de procesamiento digital de imágenes, en el cual mediante la implementación del algoritmo de Sobel para la detección de bordes de imágenes, usando el módulo de desarrollo STM32F429I-DISCO basado en el microcontrolador STM32f429 logre determinar la detección de bordes de cualquier imagen. Como resultado del análisis cuantitativo mediante el método de correlación al comparar las figuras en las cuales se aplicó el filtro Sobel, para obtener los bordes usando un microcontrolador STM32F429I-DISCO en la tarjeta de detección y el software Matlab, se obtuvo valores aproximados a 0.8 para tres imágenes, ademas otros valores mayores a 0.6 que indican que las imágenes son semejantes.

ABSTRACT

Within digital image processing plays an important role by analyzing certain images containing false edges. False edges cause a problem when performing the analysis filtering a digital image processing being the crucial moment of reconstruction of an image and is often affected by the loss of important information. The project presented below is based on the design of hardware for analyzing digital image processing, in which by implementing the algorithm Sobel to detect edges of images, using the module development STM32F429I-DISCO based on the microcontroller STM32f429 achieve determine the edge detection of any image. As a result of quantitative analysis by the method of correlation by comparing the figures in which the Sobel filter is applied to obtain the edges using a STM32F429I-DISCO microcontroller card detection and Matlab software, approximate 0.8 values was obtained for three images, plus other values greater than 0.6 indicate that the images are similar.

INTRODUCCIÓN

Dentro del ámbito del procesamiento digital de imágenes, se encuentran las técnicas del proceso del filtrado de imágenes, con lo que se pretende analizar el procesamiento digital con el fin de mejorar la comprensión de las imágenes enviadas por un medio de transmisión lo que optimiza la utilización del ancho de banda y una más eficiente comunicación al realizar el procesamiento de la imagen. La detección de bordes reduce significativamente la cantidad de datos y la información inútil, mientras que preserva las propiedades estructurales importantes en una imagen. Hay un número extremadamente grande de algoritmos de detección de bordes disponibles, cada uno diseñado para ser sensible a ciertos tipos de bordes. Para mejorar las técnicas de filtrado en imágenes se implementó un detector de bordes de imágenes usando un microcontrolador STM32F429I, mediante la utilización de una de las técnicas de filtrado como el algoritmo de Sobel. Lo que se busca es optimizar el problema principal que tiene una imagen como los bordes falsos lo que produce líneas delgadas y gruesas produciendo el fenómeno que se conoce como ruido de la imagen. Los bordes caracterizan límites y son por lo tanto un problema de importancia fundamental en el procesamiento de imágenes.

A continuación se describe cada uno de los capítulos dentro de este proyecto: en el primer capítulo hace referencia a los antecedentes, objetivos generales y específicos planteados para el proyecto de titulación, así mismo la justificación y el alcance del detector de bordes de imágenes.

El capítulo 2 describe el algoritmo de Sobel que se utilizó para la implementación del detector de bordes de imágenes, se analiza la tecnología que utiliza el microcontrolador y la arquitectura ARM, además se estudia los algoritmos empleados para la construcción del filtro Sobel tanto en la tarjeta de detección de bordes como en el software Matlab.

Para el capítulo 3 se detalla las herramientas y equipos usados con el fin de diseñar e implementar el detector de bordes de imagen, se hace referencia a la tarjeta utilizada, los distintos diagramas de conexión y diagramas de flujos para el funcionamiento del detector.

Finalmente el capítulo 4 aborda las pruebas realizadas en el presente proyecto, mediante la comparación de las imágenes filtradas con el microcontrolador STM32F429I y con el Software Matlab, utilizando del algoritmo de Sobel.

CAPÍTULO 1

A continuación se describe los antecedentes, objetivos generales y específicos planteados para el proyecto de titulación, así mismo la justificación y el alcance del detector de bordes de imágenes.

1.1 Antecedentes

Una de las técnicas para reconstrucción de una imagen son las variaciones de intensidades en los bordes, lo que se conoce como detección de bordes, aunque una imagen carezca de texturas o sombras se logra reconocer el objeto tan solo con sus siluetas. (Martínez & Martínez, 2012)

Los detectores de bordes de imágenes son herramientas fundamentales para el procesamiento de imagen y la visión por computadora, con lo que respecta a las investigaciones que se han hecho anteriormente sobre el procesamiento de imágenes en el Ecuador se puede citar al siguiente trabajo de la Universidad Politécnica Salesiana, Sede Cuenca en la cual reposa el trabajo de Alba Raquel Paguay Paguay y Pedro Remigio Urgilés Ortiz titulado "Recuperación de imágenes mediante extracción de blobs (Binary Large Objects, objetos binarios grandes) aplicando el operador Laplaciano de Gauss, el kernel Guassiano y desarrollo de un prototipo". En este proyecto muestra cómo se puede recuperar una imagen a partir de blobs o lo que se conoce como base de datos aplicando ciertos algoritmos de detección de bordes de imágenes. (Paguay & Urgilés, 2012)

En la Escuela Superior Politécnica del Litoral (ESPOL), se realizó la investigación de los autores José Moreira Quiroz, Vladimir Valencia Delgado y Patricia Chávez Delgado denominado "Implementación de un algoritmo para la detección y conteo de células en imágenes microscópicas". Esta investigación se basa en el desarrollo de una aplicación para el conteo de células en imágenes obtenidas con microscopio, en la cual se hace posible la utilización de dos algoritmos (métodos) de segmentación de una imagen y de dos de los

principales operadores Canny y Sobel para detectar bordes en imágenes. (Moreira, Valencia, & Chávez, 2009)

1.2 Objetivos

1.2.1 Objetivo general:

- Diseñar y construir un detector de bordes de imágenes usando un microcontrolador mediante el algoritmo de Sobel.

1.2.2 Objetivos específicos:

- Analizar el algoritmo Sobel de detección de bordes de imágenes.
- Simular el algoritmo de Sobel para detección de imágenes en Matlab.
- Diseñar y construir un sistema de procesamiento digital (DSP) de imágenes para la detección de bordes.
- Implementar el algoritmo de detección de bordes Sobel en el sistema DSP.
- Comparar los resultados de detección de los bordes sobre las imágenes adquiridas con respecto al simulador.

1.3 Justificación

La entidad que se beneficia con este Proyecto de titulación es la Universidad Politécnica Salesiana, en específico los estudiantes de la carrera Ingeniería Electrónica, debido a que contarán con una herramienta de estudio para el procesamiento digital de imágenes y la detección de bordes con algoritmos que permitan un mejor manejo de la imagen para posibles proyectos en telecomunicaciones como el caso de la telefonía y la televisión digital.

Además, se habrá abordado una línea de investigación en el área de electrónica, lo cual sería un aporte básico para futuras publicaciones en donde sea necesario el uso de detectores

de imágenes, para que los estudiantes cuenten con herramientas de procesamiento de datos e imágenes, que luego sean enviados por algún medio de trasmisión.

1.4 Alcance

El proyecto se basa en obtener los bordes de cualquier imagen, para lo cual se usa una tarjeta STM32F429I-DISCO que permite realizar lo siguiente:

- Obtener una imagen mediante un hardware de cámara OV7670.
- Procesarla a través del microcontrolador STM32F429I-DISCO, haciendo uso del algoritmo de Sobel.
- Mostrar los bordes de la imagen ya procesada sobre una pantalla gráfica LCD de 2.4¨.

CAPÍTULO 2

A continuación se describe el algoritmo para la detección de bordes, la tecnología utilizada para el procesamiento de la imagen y la arquitectura ARM, además de los histogramas y el método de correlación.

2.1 Algoritmo de Sobel

El algoritmo de Sobel utilizado para desarrollar el detector de bordes de imágenes, esencialmente analiza cada píxel de una imagen con la finalidad de procesarla mostrando el resultado en el vector gradiente correspondiente a la norma del mismo, calculando la variación máxima de un punto analizado en el cual se evaluará el mayor cambio posible de un color oscuro a un color claro. (CALOT, 2008)

El filtro Sobel se forma con la convolución de dos matrices. Una matriz vertical y otra horizontal, que originan dos imágenes para calcular aproximaciones mediante las derivadas, matemáticamente el operador Sobel utiliza dos matrices kernels de 3x3 como muestra la Ecuación 2.1

$$G_x = \begin{bmatrix} -1 & 0 & +1 \\ -2 & 0 & +2 \\ -1 & 0 & +1 \end{bmatrix} . A \; ; \quad G_y = \begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix} . A \qquad \text{Ecuación 2.1}$$

Combinando la matriz horizontal y vertical se obtiene la magnitud del gradiente como muestra la Ecuación 2.2

$$G = \sqrt{{G_x}^2 + {G_y}^2} \qquad \text{Ecuación 2.2}$$

Una vez obtenida la magnitud se puede calcular así mismo la dirección del gradiente como muestra la Ecuación 2.3

$$\theta = \arctan\left(\frac{G_y}{G_x}\right) \qquad \text{Ecuación 2.3}$$

2.1.1 Ejemplo del Cálculo de una matriz de 3x3 aplicando Sobel

A continuación se detalla el proceso para la obtención de las matrices para el cálculo del método de Sobel, la máscara que se utiliza para la obtención de las matrices kernels 3x3, en el siguiente cálculo matemático se detalla un ejemplo para el primer color (rojo) que se va analizar dentro del protocolo RGB.

Declaración de matrices

$$G_x = \begin{bmatrix} -1 & 0 & +1 \\ -2 & 0 & +2 \\ -1 & 0 & +1 \end{bmatrix} \qquad G_y = \begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix}$$

$$G_{pixel} = \begin{bmatrix} Z1 & Z2 & Z3 \\ Z4 & Z5 & Z6 \\ Z7 & Z8 & Z9 \end{bmatrix} \qquad G_{pixel} \begin{bmatrix} 8 & 9 & 1 \\ 3 & 2 & 6 \\ 1 & 2 & 4 \end{bmatrix}$$

Aplicar la máscara de Sobel Ecuación 2.4 y Ecuación 2.5

$$G_x = (Z7 + 2.Z8 + Z9) - (Z1 + 2.Z2 + Z3) \qquad \text{Ecuación 2.4}$$

$$G_y = (Z3 + 2.Z8 + Z9) - (Z1 + 2.Z4 + Z7) \qquad \text{Ecuación 2.5}$$

Se encuentra la matriz horizontal Gx

$$G_x = (1 + 2(2) + 4) - (8 + 2(9) + 1)$$

$$G_x = (1 + 4 + 4) - (8 + 18 + 1)$$

$$G_x = (9) - (27)$$

$$G_x = -18$$

Se encuentra la matriz vertical Gy

$$G_y = (1 + 2(2) + 4) - (8 + 2(3) + 1)$$

$$G_y = (1 + 4 + 4) - (8 + 6 + 1)$$

$$G_y = (9) - (15)$$

$$G_y = -6$$

Se obtiene la magnitud del gradiente con la combinando la matriz horizontal y vertical.

$$G = \sqrt{G_x{}^2 + G_y{}^2}$$

$$G = \sqrt{(-18)^2 + (-6)^2}$$

$$G = \sqrt{324 + 36}$$

$$G = \sqrt{360}$$

$$G = 18.9734$$

El valor G obtenido es la aproximación del gradiente aplicado a la convolución de las dos matrices, la matriz tanto horizontal como la matriz vertical, este valor se lo reemplaza en cada color de la escala de grises del modelo RGB, siendo en este caso el primer valor obtenido para el color rojo, lo que se busca es descomponer cada uno de estos colores. Al final se busca obtener cuanto color se tiene para el rojo, verde y azul respectivamente.

La figura 1 muestra la matriz aplicada al filtro, en la figura 1.a se detalla la imagen original adquirida por el módulo de cámara OV7670 se escoge un píxel central, en la figura 1.b se observa como la imagen con el filtro aplicado, este proceso se realiza sobre los píxeles de una zona predefinida o lo que se conoce como el análisis de la vecindad de píxeles, por lo que se escoge los píxeles vecinos al punto central. En la figura 1.c se observa el resultado de la operación del proceso en el píxel analizado, este resultado no es más que la convolución del filtro con la imagen original, la imagen de salida es la suma de los píxeles de la imagen original multiplicado por el peso correspondiente de las máscaras de Sobel, donde el resultado de estos productos será el valor del píxel de la imagen final.

Matriz final aplicada al filtro

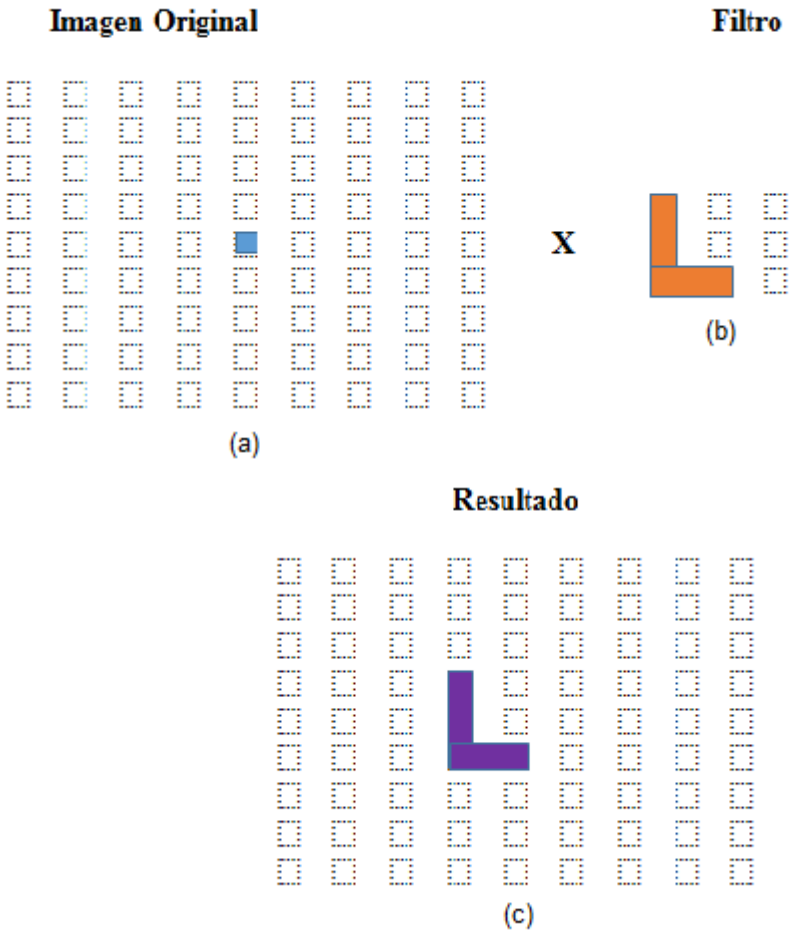


Figura 1. Matriz final aplicada al filtro
(a) Imagen Original
(b) Filtro o Kernel de Sobel
(c) Resultado de la convolución del filtro aplicado a la imagen original

Elaborado por: Ordóñez, J

La figura 2 muestra la matriz para el color rojo, se observa que la imagen digital es una matriz rectangular. Cada píxel toma un valor entre 0 y 255, el cual determina su color, este valor esta codificado con un número específico de bits. A continuación se muestra la representación para el primer color RGB que se va utilizar posteriormente para el análisis del filtro Sobel, en este caso el color rojo se utilizó en el ejemplo 2.1.1 para una matriz 3x3 aplicando el kernel de Sobel.

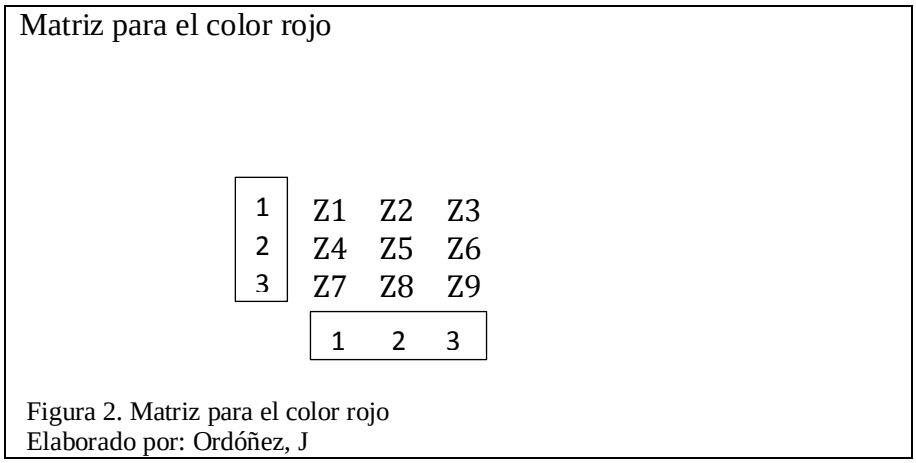


Figura 2. Matriz para el color rojo
Elaborado por: Ordóñez, J

2.2 Cuadro comparativo entre tecnologías ARM y DSPIC

La tabla 1 indica las principales características técnicas de los microcontroladores STM32F429I-DISCO y DSPIC30F4013, las diferencias van más allá del tipo de fabricante, se observa principalmente un procesador con tecnología ARM respecto al otro, se indica las memorias Flash, SRAM y memoria de programa mayores a las del microcontrolador DSPIC30F4013. Por otra parte se indica el número de instrucciones que pueden soportar ambos microcontroladores por segundo, el STM32F429I-DISCO tiene 225 MIPS y el DSPIC30F4013 que cuenta con 30 MIPS, la diferencia de 195 MIPS entre ambos.

La gran capacidad que posee el microcontrolador STM32F429I tiene algunas diferencias respecto al DSPIC30F4013, la tecnología ARM cuenta con una mayor capacidad en lo que se refiere a memoria. El microcontrolador STM32F429I cuenta con memoria flash de 2 Mbytes mayor que la memoria del microcontrolador DSPIC30F4013 con memoria flash de 2 Kbytes, en cuanto a la memoria SRAM el microcontrolador STM32F429I tiene 64 Mbits y el microcontrolador DSPIC30F4013 tiene 2048 Bytes. La memoria de programa para el microcontrolador STM32F429I es de 256 Kbytes y del microcontrolador DSPIC30F4013 es de 48 Kbytes. En el funcionamiento del núcleo, la arquitectura ARM dispone de 31 registros de propósito general de 32 bits, de los cuales 16 bits son utilizados en la memoria del

programa, el resto de los bits son para acelerar el procesamiento de excepciones (control de flujo de saltos o efecto no deseado dentro de la ejecución de un programa), los especificadores de registro de las instrucciones ARM direccionan cualquiera de esos 16 registros visibles (registros más utilizados). Los procesadores ARM disponen de otras características raramente vistas en otras arquitecturas RISC, como el direccionamiento relativo al PC y los modos de direccionamiento pre y post incrementados. (Navarro, Herrera, & Rodriguez, Minix 3 Sobre Arquitectura ARM, 2009)

Tabla 1.
Cuadro comparativo entre STM32F429I-DISCO y DSPIC

Parámetros técnicos

CARACTERÍSTICAS	**32F429IDISCOVERY**	**EasydsPIC4A**
Microcontrolador	STM32F429I-DISCO	DSPIC30F4013
Fabricante:	STMicroelectronics	MikroElektronika
Núcleo	ARM Cortex M4	DSPIC
Memoria Flash	2 Mbytes	2 Kbytes
Memoria SRAM	64 Mbits	2048 Bytes
Capacidad de memoria de programa	256 Kbytes	48 Kbytes
Anchura de bus de datos	32 bit	16 bit
Millones de instrucciones por segundo	225 MIPS	30 MIPS
LCD	2.4" QVGA TFT LCD	LCD & Graphic LCD 128x64
Pines	80 de propósito general	40 de propósito general
Dimensión de tarjeta	119.30 mm x 66 mm	265 mm x 220 mm

Nota: Ordóñez, J
Fuente: (MikroElektronika, 2015) (MOUSERELECTRONICS, 2015)

2.3 Tecnología ARM

La tecnología ARM es una arquitectura potente gracias a su gran capacidad de 32 y 64 bits, su evolución ha sido muy notoria contando en la actualidad con microprocesadores en dispositivos móviles y aplicaciones de vanguardia gracias a su bajo consumo y costo, sin perder versatilidad. Los microcontroladores y microprocesadores que cuentan con arquitectura ARM tienen módulos de hardware sencillo pero muy buenas capacidades de software, además de bajo consumo de energía. (Caprile, 2012)

Básicamente ARM es una arquitectura RISC (Reduced Instruction Set Computer), a la cual se le implementó mejoras para lograr alto rendimiento, se mantuvo de la arquitectura RISC las siguientes caracteristicas:

- Arquitectura de carga y Almacenamiento
- Instrucciones de longitud fija de 32 bits
- Formatos de instrucciones de 3 direcciones

Mientras que se implementó sobre la arquitectura de RISC características tales como:

- Todas las instrucciones se ejecutan en un ciclo de reloj.
- Modos de direccionamiento simples.
- Control sobre la unidad aritmética ALU.
- Modos de direccionamiento con incremento y decremento automático.
- Carga y almacenamiento de múltiples instrucciones.
- Ejecución condicional de todas las instrucciones.
- Set de instrucciones ortogonal, regular o simétrico.
- Técnica pipeline.
- Excepciones vectorizadas.
- Extensión Thumb.

Para mejorar la densidad de código compilado ARM utiliza extensiones, para este propósito se incluyó el modo Thumb, al encontrarse en este modo el procesador ejecuta instrucciones de 16 bits, en otras palabras Thumb es incidente al mejorar el rendimiento al cargar menos código, disminuyendo la carga en memoria RAM de poca capacidad al usar 16 bits siendo esta una forma para lograr simplificar las instrucciones ARM de 32 bits frecuentemente utilizadas. (Navarro, Herrera, & Rodriguez, Minix 3 sobre Arquitectura ARM, 2009) La arquitectura de ARM es licenciada, anteriormente esta empresa se dedicó a desarrollar los procesadores pero hoy en día ya no los fabrican, más bien ofrecen distintos tipos de licencias tanto sobre sus conjuntos de instrucciones como sobre sus arquitecturas. (Firtec, 2015)

La tecnología ARM consta de varias arquitecturas ampliamente difundidas tales como: ARMv4T, ARMv5TEJ, ARMv6, ARMv7. En el presente proyecto se utilizó la arquitectura ARMv7 denominado también Cortex-M, dentro de esta arquitectura existen subdivisiones que abarcan los Cortex-R, Cortex-M4 y Cortex-A, el Cortex-M4, este último se utilizó para la ejecución del presente proyecto, las diferencias entre arquitecturas se puede observar en la tabla 2.

Tabla 2.
Arquitecturas ARM

Arquitecturas ARM			
ARQUITECTURA	**CPU Clock**	**Núcleo ARM**	**CARACTERÍSTICAS**
ARMv4T	Hasta 100Mhz	ARM7TDMI ARM9T	Gran velocidad A/D & D/A
ARMv5TEJ	Hasta 250Mhz	ARM926EJ ARM7EJ	Amplia selección con periféricos: LCD, MCI, USB
ARMv6	Hasta 150Mhz	CORTEX-M0 ARM 11	Implementación FPGA, núcleo del CPU integra Control de sistema de interrupciones.
ARMv7	Hasta 400Mhz	CORTEX-R CORTEX-M4 CORTEX-A	Extensiones Thum2.

Nota: Ordóñez, J
Fuente: (Capel, 2008)

La arquitectura Cortex-M4 se aprecia en la figura 3, con sus diferentes componentes y detallando cada uno. Dentro de esta arquitectura se encuentra ISA (Instruction Set Architecture) compuesta por el tipo de datos, el tipo de instrucciones, los registros el modo de direccionamiento y el manejo de excepciones El pipeline hace referencia a la segmentación en sí, permitiendo acelerar y aumentar el rendimiento dentro del sistema, descomponiendo las ejecuciones en varias etapas, contando con 3 etapas de predicción de salto. Las unidades de soporte de memoria, permiten al CPU acceder a las memorias, básicamente corresponden a un hardware conformado por un gran número de circuitos integrados, realizando funciones como traducción de direcciones lógicas a físicas, protección de memorias. (Dual & Arm, 2012)

Arquitectura Cortex-M4

Figura 3. Arquitectura Cortex-M4
Elaborado: Ordóñez, J

2.4 Modelo RGB

El modelo RGB analiza la composición de los colores primarios de una imagen como: rojo, verde y azul, basados en la síntesis aditiva por lo que representa un color con la suma de los colores primarios antes mencionados. Para representar la intensidad de color RGB en una imagen normalmente se puede medir cada una de estas componentes en un intervalo de 0 a 255, es decir para el color rojo se tendrá (R=255, G=0, B=0); el verde con (R = 0, G = 255, B = 0) y el azul con (R = 0, G = 0, B = 255), por lo que el resultado será en cada caso un mismo color pero diferente intensidad lo que se conoce como colores monocromáticos. (Escolano, Cazorla, Alfonso, Colomina, & Lozano, 2013).

En la figura 4 se aprecian los formatos más usados que son el RGB 888 y el RGB 565. El formato RGB 888 es el formato más común, cada píxel es almacenado en 24 bits es decir tanto el rojo, verde y azul utilizan 8 bits. Esto significa que la intensidad de cada luz puede ir de 0 a 255, donde 0 es ausencia de luz, y 255 es la máxima intensidad, usando las diferentes

combinaciones se puede llegar a tener unos 16 millones de colores diferentes. Usualmente es usado para contenidos sin comprimir es decir en imágenes de fondo que contienen una gran cantidad de colores, que no podrían ser representados con 16 u 8 bits. La diferencia entre formatos se basa principalmente en los bits asignados para cada matriz, asumiendo esto para el formato RGB 565 la matriz roja tiene 5 bits, la matriz verde almacena 6 bits y la matriz azul tiene 5 bits, lo que se traduce en un ahorro de memoria sacrificando la cantidad de colores que se pueden formar. Al RGB 565 se lo conoce también como el formato 65K ya que se pueden lograr 65536 colores con todas la posibles combinaciones, este formato hace énfasis en el color verde ya que el ojo humano es mucha más sensible a dicho color. (Aparicio's, 2015)

Formatos RGB 888 y RGB 565

Formato RGB 888	**Formato RGB 565**
	

Figura 4. Formatos RGB 888 y 565
Fuente: (Ayala)

2.5 Histogramas

El histograma es un gráfico de barras verticales en el cual el ancho de cada barra corresponde con el rango del intervalo, y la altura de la barra corresponde al número de puntos dentro del intervalo. (Dr Macías, 2005)

Básicamente el histograma es una representación gráfica en forma de barras, donde la superficie de cada barra es proporcional a la frecuencia de los valores representados, como se puede ver en la figura 5 en el eje vertical se representan las frecuencias, y en el eje horizontal los valores de las variables, en donde este tipo de representaciones en forma de barras se puede usar para interpretar las variaciones de una cierta cantidad de datos.

Representación de un histograma variable vs frecuencia

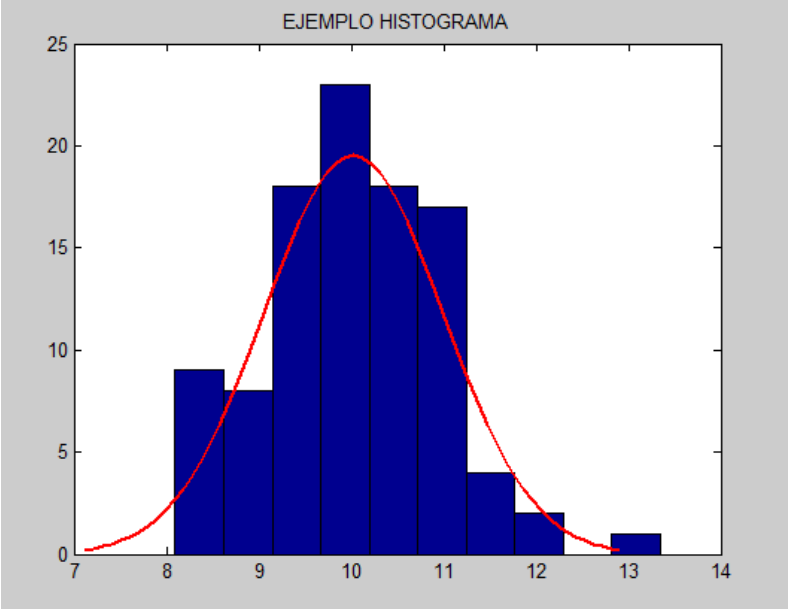


Figura 5. Representación de un histograma variable vs frecuencia.
Fuente: Ordóñez, J

Para calcular un histograma se debe seguir los siguientes pasos: primero se tiene que determinar el rango de datos que se va usar para el análisis, es decir el dato mayor menos el dato menor, luego se debe obtener el número de clases en donde la referencia debe ser aproximada a la raíz cuadrada del número de datos. En el eje de las abscisas se forman los

rectángulos que tiene como base la amplitud del intervalo, mientras que por la altura se tiene la frecuencia absoluta de cada intervalo. (Vitutor, 2016)

2.6 Método de correlación

La correlación es una operación parecida a la convolución, en la cual el valor de un píxel de salida se calcula como la suma ponderada de los píxeles vecinos. La correlación está dada por la siguiente Ecuación 2.6. (Esqueda & Palafox, 2005).

$$g(x,y) = h(x,y) \text{ o } f(x,y) = \sum_{i=-\infty}^{\infty} \sum_{j=\infty}^{\infty} f * (i,j)h(x+i, y+j) \quad \text{Ecuación 2.6}$$

El cálculo de correlación se utiliza para encontrar las diferencias y similitudes de una imagen, es decir busca los pixeles iguales o parecidos para identificar si las imágenes analizadas tienen relación o no sobre un patrón analizado. (Esqueda & Palafox, 2005).

De forma estadística la correlación es medida por lo que se denomina coeficiente de correlación (r). Su valor numérico varía de 1 a -1. El coeficiente r = 0 indica que no hay relación entre las imágenes comparadas, cuando el coeficiente r = 1 representa una relación que indica que las dos imágenes son iguales, un valor cercano del coeficiente r= 1 indica que las imágenes son semejantes (Pedroza & Dicovskyi, 2006)

CAPÍTULO 3

El presente capítulo detalla las herramientas y equipos usados para diseñar e implementar el detector de bordes de imagen, se hace referencia al hardware de la tarjeta utilizada, el diagrama esquemático de las placas de circuitos electrónicos construidas y los diagramas de flujo.

3.1 Tarjeta STM32F429I-DISCO

La tarjeta STM32F429I-DISCO como se puede observar figura 6 muestra el panel frontal del kit de desarrolló, cuenta con un núcleo Cortex-M4F de 32 bits. Además incorpora un sensor de movimiento ST MENS, una pantalla de visualización TFT, un conector USB, tres leds indicadores y dos pulsadores. Las principales características se pueden ver en el enlace:

http://www.st.com/st-webi/static/active/cn/resource/technical/document/user_manual/DM00093903.pdf

(MOUSERELECTRONICS, 2015)

Panel frontal tarjeta STM32F429I-DISCO

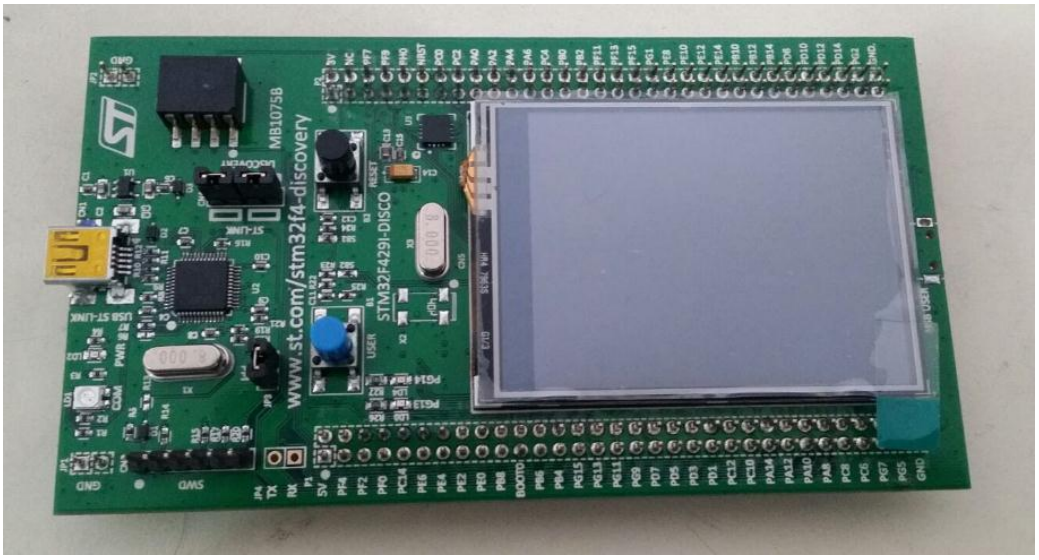


Figura 6. Panel frontal tarjeta STM32F429I-DISCO

Elaborado por: Ordóñez, J

3.1.1 Microcontrolador STM32F429I

El microcontrolador STM32F429I está incluido en la tarjeta de desarrollo, la hoja de datos se pueden encontrar en el siguiente enlace: http://www.st.com/web/en/resource/technical/document/datasheet/DM00071990.pdf , en la Tabla 1 se muestra el microcontrolador. (MOUSERELECTRONICS, 2015)

Microcontrolador STM32F429I-DISCO

Figura 7. Microcontrolador STM32F429I-DISCO

Elaborado por: Ordóñez, J

3.1.2 LCD 2.4" QVGA TFT

Como se aprecia en la figura 8, la pantalla lcd contiene un display TFT de 2.4 pulgadas, con una resolución de 320 x 240 y con modelo de color RGB, las principales especificaciones se pueden consultar en el siguiente enlace: http://www.embeddedartists.com/sites/default/files/docs/3.2_inch_QVGA_TFT_Color_LCD_Users_Guide-Version_2.1_Rev_A.pdf

(Displaytech, 2015)

Pantalla LCD 2.4¨ QVGA TFT

Figura 8. Pantalla LCD 2.4¨ QVGA TFT

Elaborado por: Ordóñez, J

3.1.3 Cámara OV7670

La figura 9 muestra la cámara OV7670, la cual es un sensor de imágenes que proporciona las funcionalidades de una cámara VGA, opera a un máximo de 30 fotogramas por segundo con una resolución de 0.3 megapíxeles. Es usada en diferentes dispositivos tales como: teléfonos celulares, computadoras, cámaras digitales, etc. Las características principales se pueden consultar en: http://web.mit.edu/6.111/www/f2015/tools/OV7670_2006.pdf (Aparicio's, 2015)

Cámara OV7670

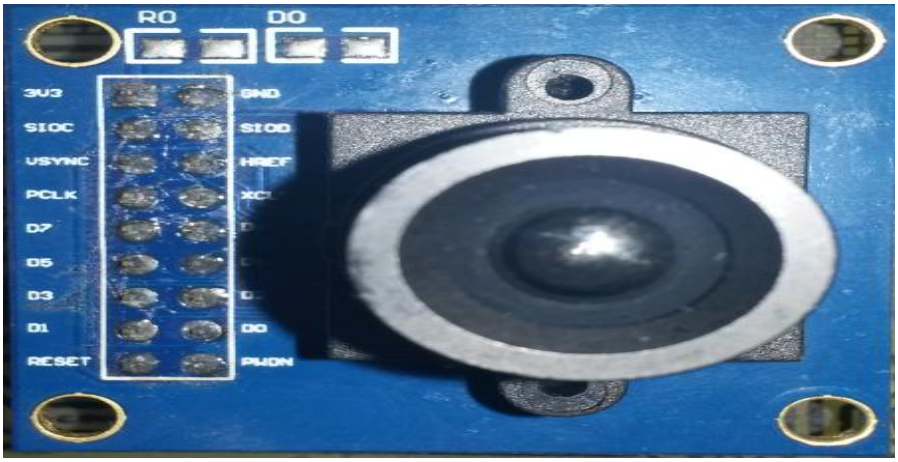


Figura 9. Cámara OV7670

Elaborado por: Ordóñez, J

3.2 Diagramas Esquemáticos

A continuación se describe los diagramas esquemáticos que se diseñaron para este proyecto de titulación.

3.2.1 Diagrama de Bloques del Funcionamiento del Detector

En la figura 10 se puede observar el diagrama de bloques del detector de bordes de imágenes, el equipo está formado por: la fuente de alimentación, el microcontrolador,la pantalla TFT , la cámara y el dispositivo transmisión serial.

- La conexión USB aparte de ser usada para la Tx y Rx de datos, provee energía necesaria para el funcionamiento de la tarjeta formado parte del bloque de alimentación.
- El bloque para la adquisición de imágenes, comprende la cámara OV7670 la cual es responsable de capturar las imágenes que posteriormente serán procesadas.

- El procesamiento de imágenes, corresponde al bloque comprendido por el microcontrolador STM32F429I-DISCO, el cual se encarga de realizar los algoritmos para procesar la imagen que seguidamente será visualizada.
- El bloque para visualizar la imagen ya procesada es una pantalla TFT, donde se puede observar los bordes de la imagen que fue adquirida por la cámara.
- En la etapa de comunicación serial, se muestra el modulo implementado para enviar las imágenes desde el microcontrolador STM32F429I hasta la PC.

Diagrama de bloques del funcionamiento del Detector

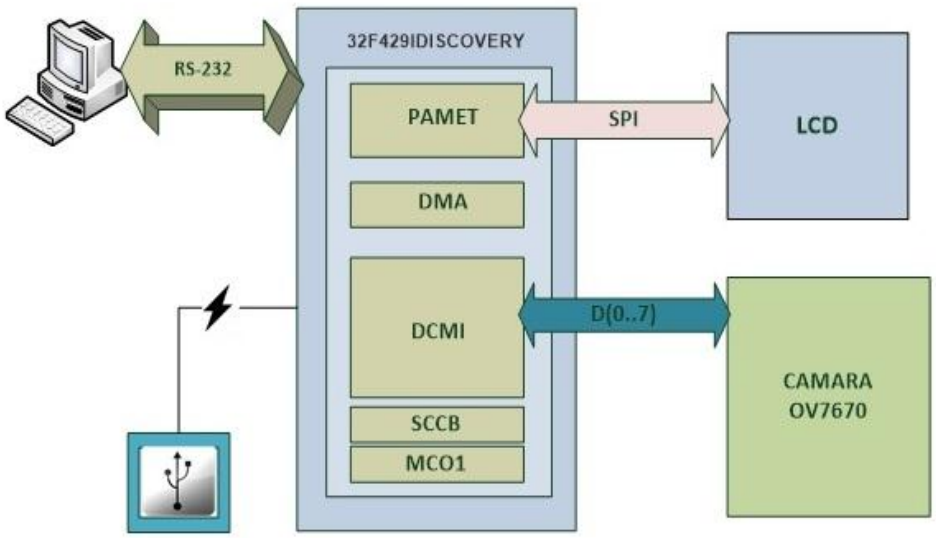


Figura 10. Diagrama de bloques del funcionamiento del detector

Elaborado por: Ordóñez, J

3.2.2 Diagrama esquemático de conexión de tarjeta y cámara

La figura 11, muestra el circuito de conexión con la alimentación de la tarjeta de desarrollo STM32F429I-DISCO, este diagrama se diseñó para futuras aplicaciones debido a que la tarjeta se alimenta con un cable USB, que viene incorporada en la placa del microcontrolador.

Diagrama esquemático de conexión con la alimentación a la tarjeta STM32F429I-DISCO

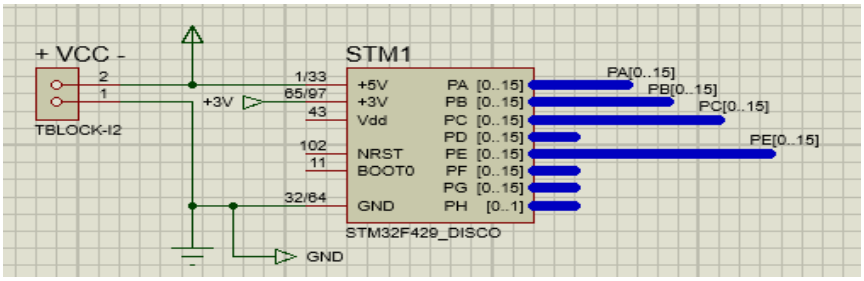


Figura 11. Diagrama esquemático de conexión con la alimentación a la tarjeta STM32F429I-DISCO.
Elaborado por: Ordóñez, J

La figura 12, indica los pines de conexión del módulo de cámara OV7670, con cada uno de los respectivos pines de control de la tarjeta,

Diagrama esquemático pines de conexión del módulo de cámara OV7670 a la tarjeta.

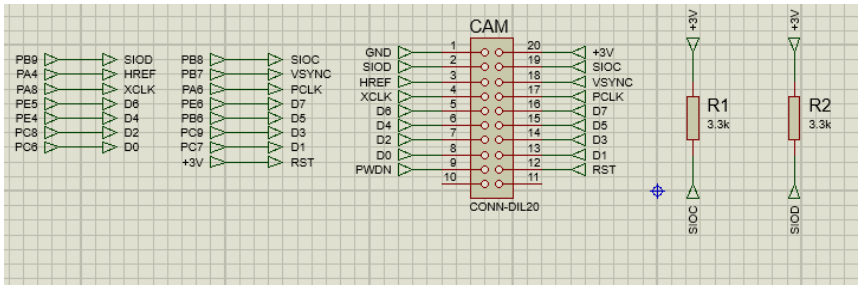


Figura 12. Diagrama esquemático pines de conexión del módulo de cámara OV7670 a la tarjeta.
Elaborado por: Ordóñez, J

La figura 13 muestra los pines utilizados para los dos botones que controlan el algoritmo de funcionamiento del detector de bordes de imágenes.

Pines de control del algoritmo de funcionamiento del detector

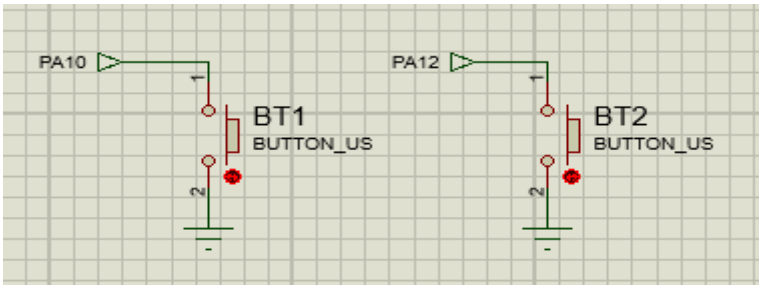


Figura 13. Pines de control del algoritmo de funcionamiento del detector.
Elaborado por: Ordóñez, J

3.3 PCB de la placa para conexión de la cámara

El circuito impreso que se muestra en la figura 14, se diseñó para conectar la cámara con el microcontrolador, para sostener mecánicamente al dispositivo final. Esta placa conecta todos los pines del microcontrolador, la fuente alimentación USB y otros periféricos. Se observa los conectores para: la cámara (CAM, los pines de la tarjeta STM32F429I-DISCO (STM1), la fuente de alimentación y los dos botones que se utilizaron el primero para tomar la foto y el segundo para aplicar el detector de bordes a la foto tomada (BT1 y BT2).

Diagrama PCB

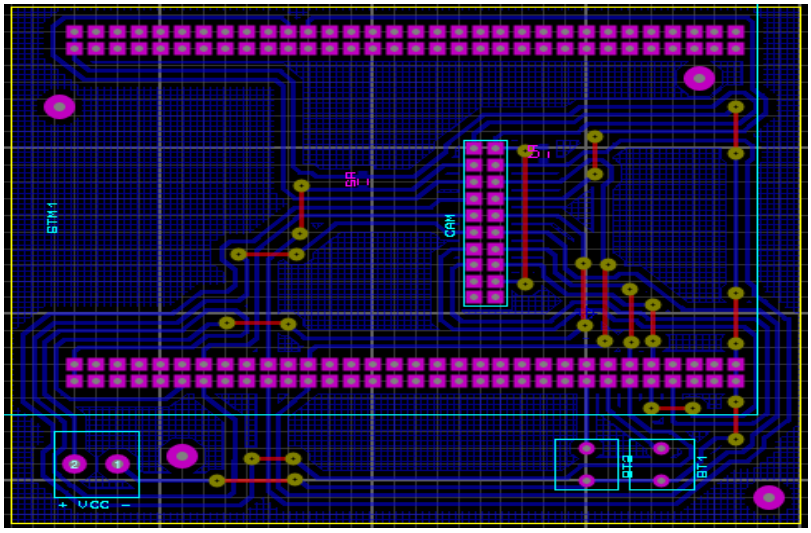


Figura 14. Diagrama PBC.
Elaborado por: Ordóñez, J

3.4 Diagramas de Flujo

A continuación se describen los diagramas de flujo del algoritmo de Sobel que fue implementado en hardware y en el software Matlab.

3.4.1 Diagrama de flujo para capturar la imagen

El diagrama se muestra en la figura 15 y se describe a continuación:

1. Inicio del proceso: este paso indica el punto de partida del algoritmo de funcionamiento.
2. Inicialización hardware de la cámara: se realiza toda la declaración de los periféricos que se conectan físicamente del hardware al software.
3. Configuración del módulo de cámara OV7670: Configuración de botón 2 (BT2) para capturar la fotografía: en este paso se declara el botón para que realice el proceso de captura de la imagen, si el estado de este botón se pone en 1 empezara el proceso de

captura, si el estado se pone en 0 se tomará un lapso de tiempo y volverá a la configuración de la cámara.

4. Empieza el protocolo de comunicación para la configuración de la cámara, si el estado se ejecuta captura la fotografía y se mostrará en la pantalla LCD.
5. presiona el botón 2 el cual inicia el protocolo del algoritmo de Sobel para la detección de imágenes envió de la imagen mediante el puerto serial al PC.
6. Se detiene el proceso si el estado no se ejecuta o muestra la imagen la pantalla TFT, este proceso se realiza cuando se ejecuta la acción, si el estado no completa el proceso regresa a su estado original, la configuración de la cámara.

Diagrama de flujo para capturar la imagen

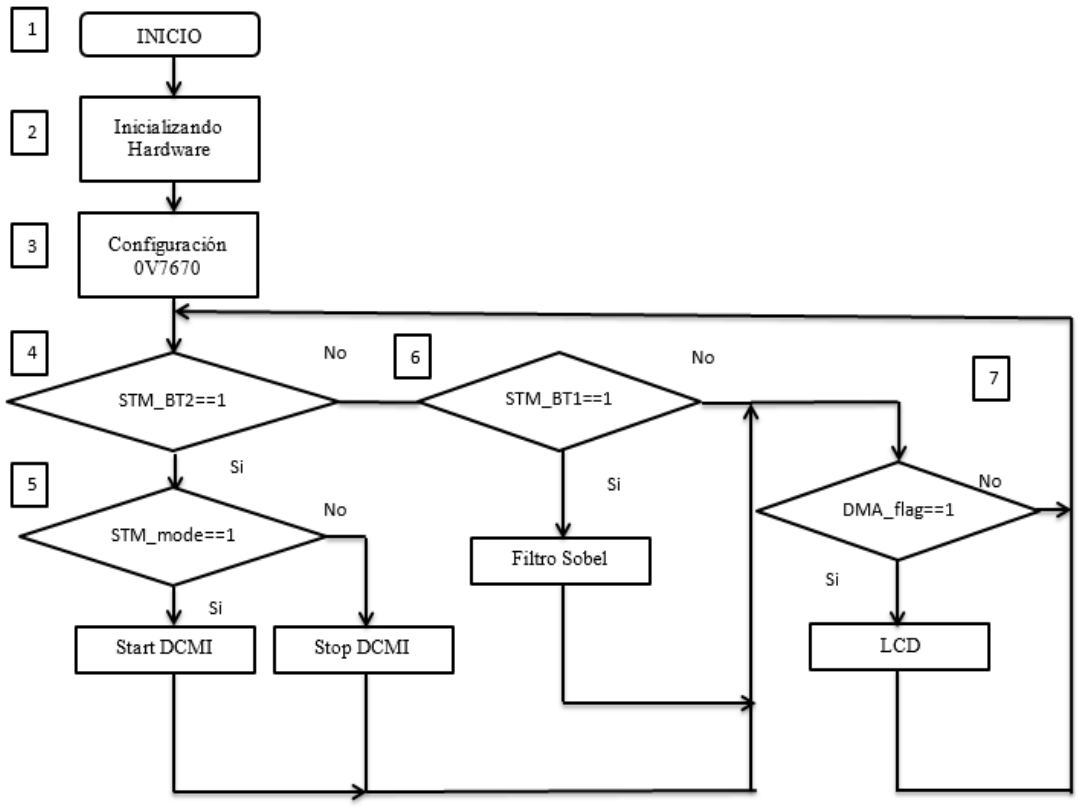


Figura 15. Diagrama de flujo para capturar la imagen.

Fuente: (Machala, 2015)

3.4.2 Diagrama de flujo del filtro Sobel

Para el filtrado de una imagen es necesario realizar la combinación de dos aproximaciones al gradiente, una aproximación vertical y una aproximación horizontal, el resultado de combinar ambos gradientes será la aproximación al gradiente de una imagen original el cual es la imagen filtrada, en el siguiente diagrama se detalla el proceso para obtener la imagen final. El diagrama se muestra en la figura 16.

1. Inicio del proceso: este paso indica el punto de partida del algoritmo de funcionamiento.
2. Imagen original: Para analizar el proceso de filtrado de una imagen se debe escoger la imagen a la cual se le va a realizar el proceso del algoritmo de Sobel.
3. Declarar la matriz gradiente horizontal y vertical: se determina las matrices para obtener las aproximaciones a las derivadas una Gx para la matriz horizontal y una Gy para la matriz vertical. Los bordes verticales se identifican haciendo uso del operador gradiente horizontal, en contraste los bordes horizontales se identifican por medio del uso del gradiente vertical para luego ser combinados.
4. Combinación de los dos gradientes: en este paso se realiza la convolución de ambas matrices para aplicar el algoritmo de Sobel.
5. Suavizado de la imagen antes de obtener la imagen final: se obtendrá directamente una imagen con reducido número de variaciones de píxeles que pueden afectar la imagen final, este proceso se lo realiza principalmente para reducir el ruido o detalles que no sean necesarios en la imagen.
6. Imagen filtrada: es la imagen final aplicada el algoritmo de Sobel para la detección de bordes.
7. Finalización: se termina el proceso del filtro Sobel.

Diagrama de flujo del filtro Sobel

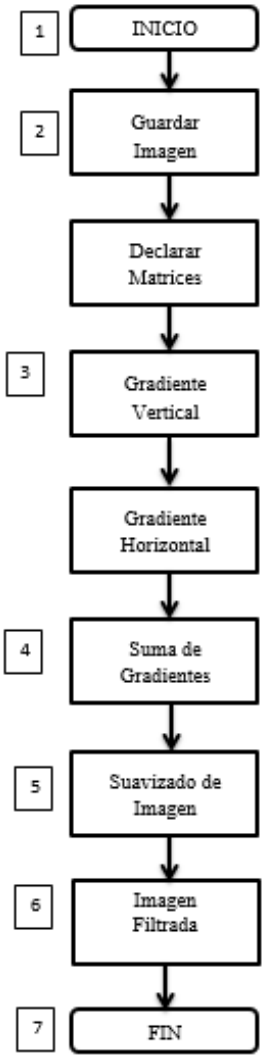


Figura 16. Diagrama de flujo del filtro Sobel
Elaborado: Ordóñez, J

3.4.3 Diagrama de flujo para mostrar la imagen filtrada

Una vez obtenida la imagen es necesaria mostrarla en una pantalla, para presentarla se ejecutan dos acciones que se guardan en el espacio de memoria del microcontrolador, si la acción se ejecuta se pondrá en estado 1 y guardará la imagen, por otro lado si la acción no se realiza quedará en estado 0 y el proceso volverá a comenzar como lo indica la figura 17.

1. Imagen capturada: esta imagen es adquirida con el módulo de cámara OV7670.

2. Inicialización en el acceso directo a memoria: permite que se guarde la imagen en un espacio que el microcontrolador dispone para luego obtener la imagen captura sobre la pantalla LCD.
3. Presentación de la imagen: una vez que la imagen ha sido guardada se ejecuta una acción que permite mostrar esta imagen alojada en el DMA (direct memory access), principalmente permite el acceso directo a memoria, es decir ayuda a no sobrecargar al CPU del microcontrolador, llenándolo de instrucciones para realizar funciones de rutina, pasando a visualizar la imagen directamente en la pantalla LCD.
4. Mostrar la imagen y guardar en el espacio de memoria: aquí se puede visualizar cuando se ejecuta la acción, esta guarda la imagen y la muestra esperando un determinado tiempo antes de presentarla.
5. Proceso cuando no se muestra la imagen y vuelve a la configuración original: en esta instancia se declara el proceso cuando la acción no se ejecuta el estado es 0, el programa espera un tiempo para nuevamente regresar a la configuración de la cámara.
6. Finalización del proceso para mostrar la imagen.

Diagrama de flujo para mostrar la imagen

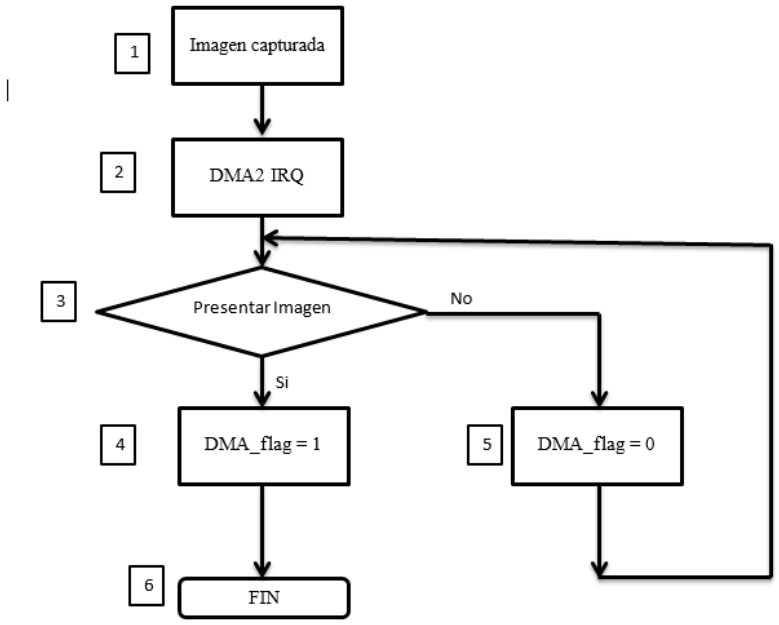


Figura 17. Diagrama de flujo para mostrar la imagen
Fuente: (Machala, 2015)

3.4.4 Diagrama de flujo del algoritmo de Sobel en Matlab

El proceso de la figura 18, muestra un algoritmo para la obtención de la imagen filtrada en el software Matlab, el cual utiliza el mismo principio para obtener la imagen mediante el operador Sobel. Este algoritmo guarda cualquier imagen en una variable, se declara las matrices para el filtro y finalmente se obtiene la imagen filtrada. Este algoritmo fue necesario para entender el método de Sobel una vez realizado el proceso de digitalización de la imagen en la computadora, siendo más fácil utilizar programas computacionales que ayudan entender el método de detección de bordes aplicado en el procesamiento digital de imágenes.

1. Inicio del proceso: como todo algoritmo muestra el principio del procedimiento a realizarse.
2. Guardar la imagen en una variable: se guarda la imagen en la variable Lena, debido a que se puede cargar directamente y guardarse en un formato para leer los datos de la matriz de la imagen.
3. Declaración de matrices kernels horizontal y vertical: para realizar el algoritmo de Sobel es necesario declarar las dos matrices la Gx y la Gy.
4. Combinación de gradientes horizontal y vertical: se realiza la convolución de estas dos matices para obtener las aproximaciones del gradiente y su módulo.
5. Imagen filtrada: esta imagen es la obtenida después de aplicar el filtro Sobel en Matlab.
6. Finalización: se termina el proceso del filtro Sobel.

Diagrama de flujo del filtro Sobel en Matlab

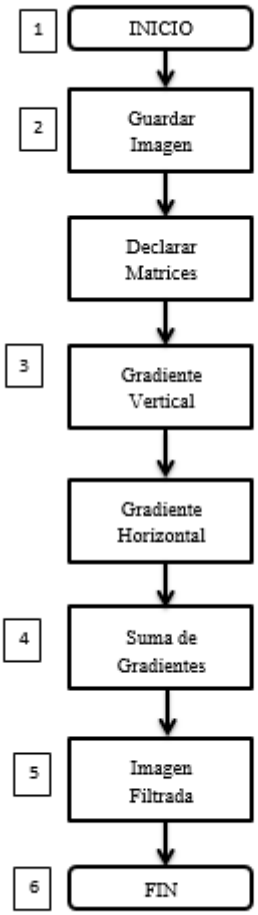


Figura 18. Diagrama de flujo del filtro Sobel en Matlab
Elaborado: Ordóñez, J

CAPÍTULO 4

El presente capítulo aborda las pruebas realizadas con el detector de bordes de imágenes y el software Matlab, haciendo uso de diferentes imágenes y verificando los resultados obtenidos con la tarjeta de desarrollo STM32F429I-DISCO y su comparación con el algoritmo de Sobel implementado en Matlab.

4.1 Procesamiento de imágenes con el detector y contraste con el software Matlab

La figura 19, muestra los resultados de varias imágenes aplicando el filtro de Sobel, en la primera columna se observa la imagen original en formato RGB, en la segunda columna se observa la detección de los bordes de la imagen realizada con la tarjeta STM32F249I-DSICO y la tercera columna el filtrado de la imagen mediante el algoritmo de Sobel desarrollado en Matlab. Se realizó distintas pruebas, tanto en rostros de personas, también en diferentes objetos, figuras geométricas y un edificio. Para el análisis de los resultados se utilizó dos procesos comparar las imágenes filtradas en la tarjeta de desarrollo y en Matlab. Mediante análisis cualitativo y cuantitativo, para verificar las diferencias y similitudes entre las imágenes comparadas.

Con la implementación de histogramas se tiene una primera impresión a simple vista de los resultados comparados de forma cualitativa, es decir la representación gráfica de los dos casos de los bordes de imágenes obtenidos. Para la comparación de forma cuantitativa se usó el método de correlación entre dos imágenes, determinando su similitud mediante el coeficiente de correlación.

Procesamiento de imágenes con el detector de bordes y software Matlab

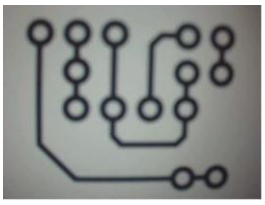
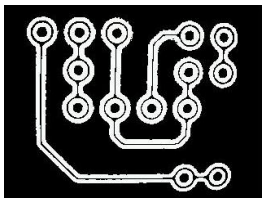
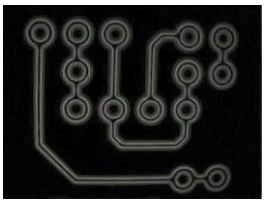
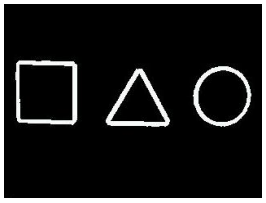
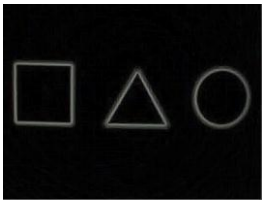
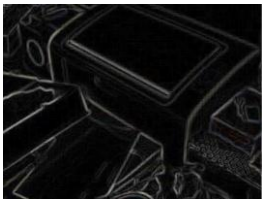
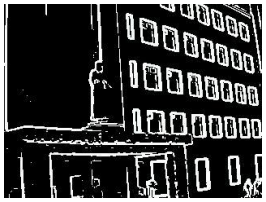
Imagen original	**Filtro en Tarjeta**	**Filtro en Matlab**
a) 	b) 	c) 
d) 	e) 	f) 
g) 	h) 	i) 
j) 	**k)** 	l) 
m) 	**n)** 	o) 

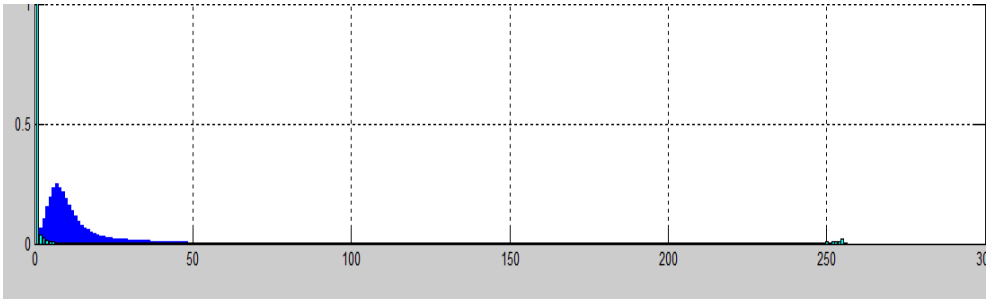
Figura 19. Procesamiento de imágenes con el detector de bordes y software Matlab.
Elaborado: Ordóñez, J

4.2 Resultados obtenidos tanto en la tarjeta y en software mediante la utilización del filtro Sobel.

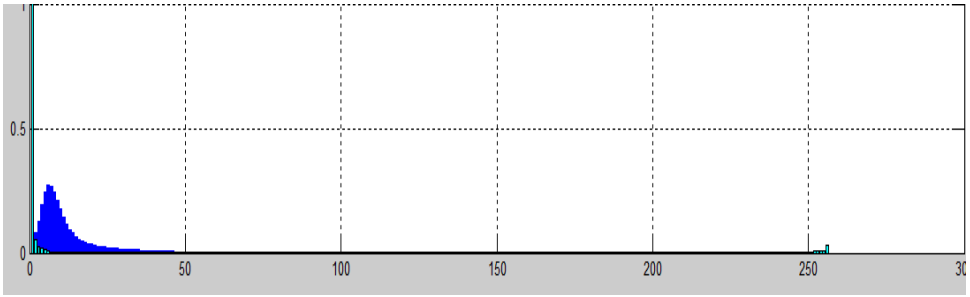
En la figura 19.a, muestra mi rostro en el cual se comparó las dos imágenes resultantes tanto en la tarjeta de deteción de bordes como software Matlab, con la ayuda de histogramas y el metodo de correlación se obtuvo los siguientes resultados: Para el analisis de histogramas se descompone a la imagen en el protocolo RGB, luego se uso el software Matlab para la elaboración de los mismos, de esta manera se puede ver la diferencia de forma cualitativa en la concentración de color que tiene una imagen con respecto a la otra. La figura 20, indica la concentración de color en cada imagen, es decir que tanto de color Rojo, Verde y Azul se tiene aplicando el filtro que se obtuvo en la tarjeta de detección de bordes y el software Matlab aplicando Sobel, el color rojo con bastante aproximación tanto en la tarjeta como en software, el color verde y azul se detalla una minima diferencia en el software Matlab por lo que se puede decir que la variación de colores es mejor en las imágenes analizadas para el filtro implementado en Matlab.

Resultado de imágenes figura 19.a en tarjeta y software Matlab

a) Contraste Software y tarjeta en Matlab color rojo



Contraste Software y tarjeta en Matlab color verde



b) Contraste Software y tarjeta en Matlab color azul

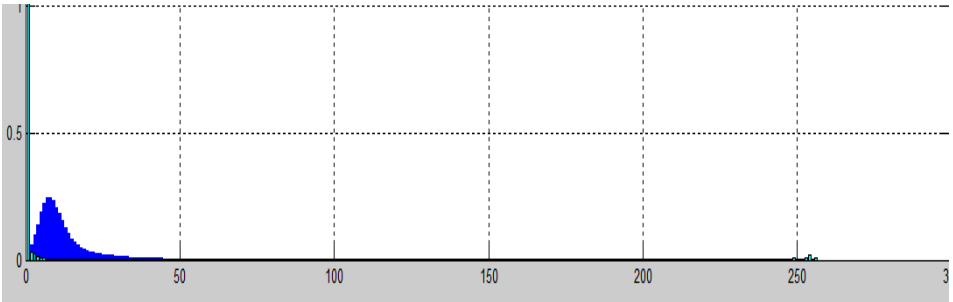
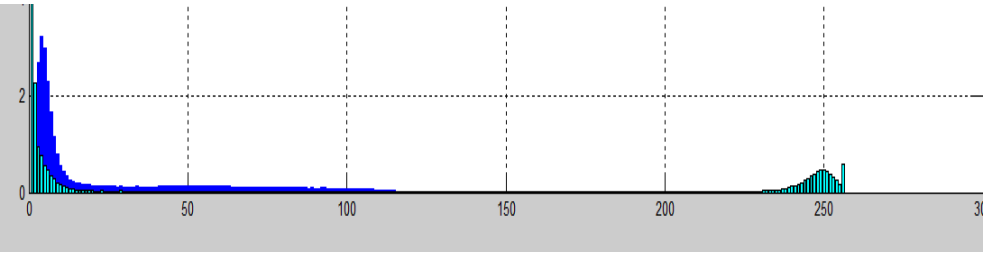


Figura 20. Resultado de imágenes figura 19.a en tarjeta y software Matlab
Elaborado: Ordóñez, J

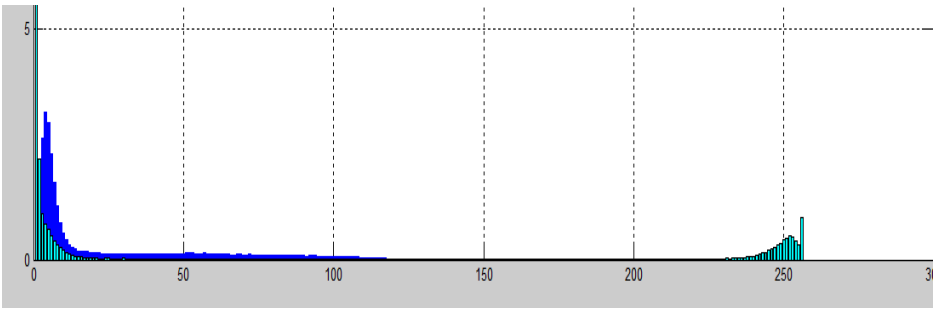
Para la figura 21, se muestra la relacion cualitativa de la imagen resultante de un circuito impreso figura 19.d, en donde se analiza el procesamiento tanto de la figura obtenida en la tarjeta de detección de bordes y en el software Matlab. El resultado de comparar ambas imágenes mediante un histograma indica que el procesamiento en el software matemático Matlab tiene mayor concetración de colores que en la tarjeta de detección de bordes, ya que existe una mayor distribución de datos en los colores primarios del protocolo RGB aplicando el filtro Sobel para los dos casos comparados. La figura 21, indica que para el color rojo, verde y azul se tiene igual concentración de colores para al analisis del filtro el software Matlab, y muy poca concentración de colores en la tarjeta de detección de bordes de imágenes.

Resultado de imágenes de un circuito impreso figura 19.d en tarjeta y software Matlab

a) Contraste Software y tarjeta en Matlab color rojo



b) Contraste Software y tarjeta en Matlab color verde



c) Contraste Software y tarjeta en Matlab color azul

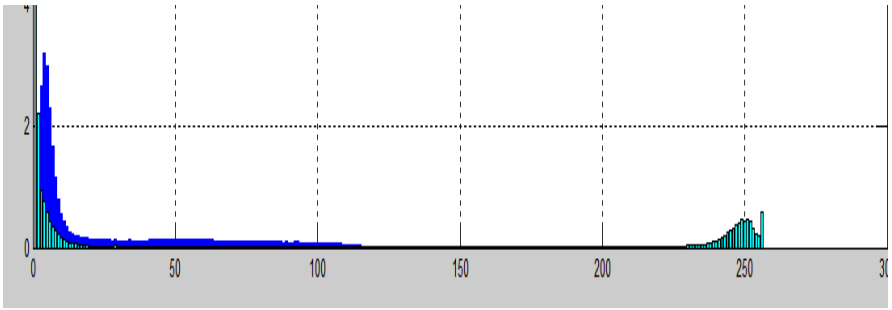
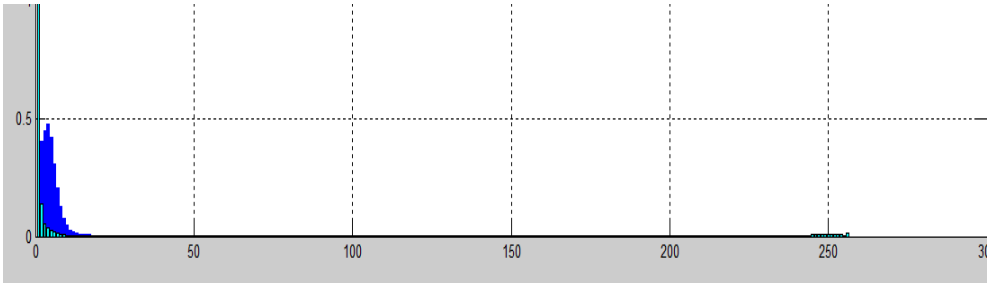


Figura 21. Resultado de imágenes de un circuito impreso figura 19.d en tarjeta y software Matlab.
Elaborado: Ordóñez, J

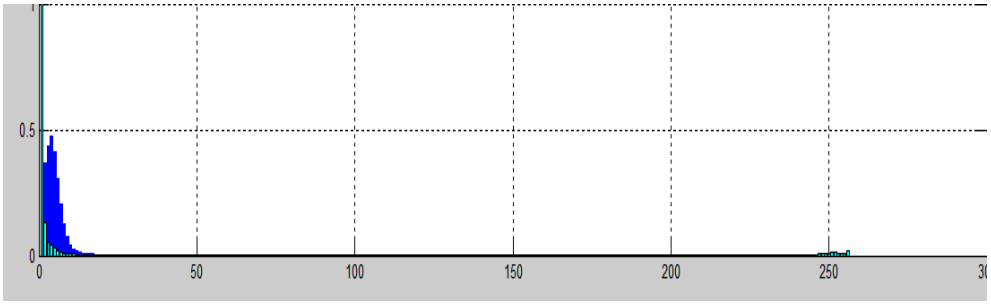
La figura 22, muestra la relacion cualitativa de la imagen resultante de figuras geométricas figura 19.g, en donde se observa el procesamiento tanto de la figura obtenida en la tarjeta de detección de bordes y software Matlab. Para los resultados de forma cualitativa de los histogramas analizados en el figura 22, se tiene una mayor concentración de colores en el software Matlab, debido a que es mucho mas visible la tonalidad de colores para el rojo, verde y azul siendo mas significativa que los resultados que muestra la tarjeta de detección de bordes de imágenes utilizando el microcontrolador STM32F429I-DISCO.

Resultado de imágenes de figuras geométricas figura 19.g en tarjeta y software Matlab

a) Contraste Software y tarjeta en Matlab color rojo



b) Contraste Software y tarjeta en Matlab color verde



c) Contraste Software y tarjeta en Matlab color azul

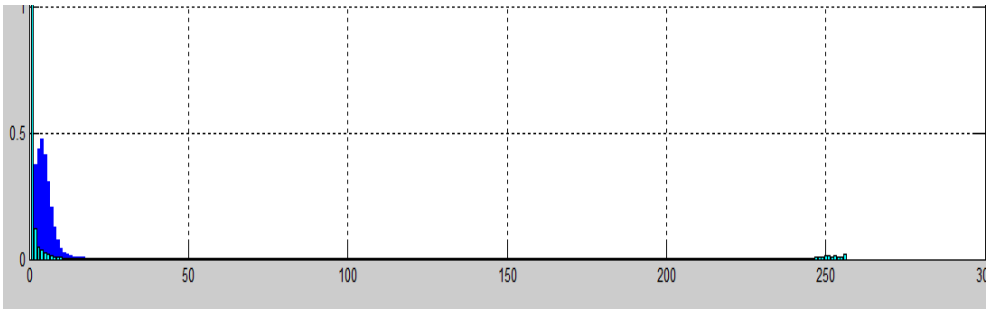
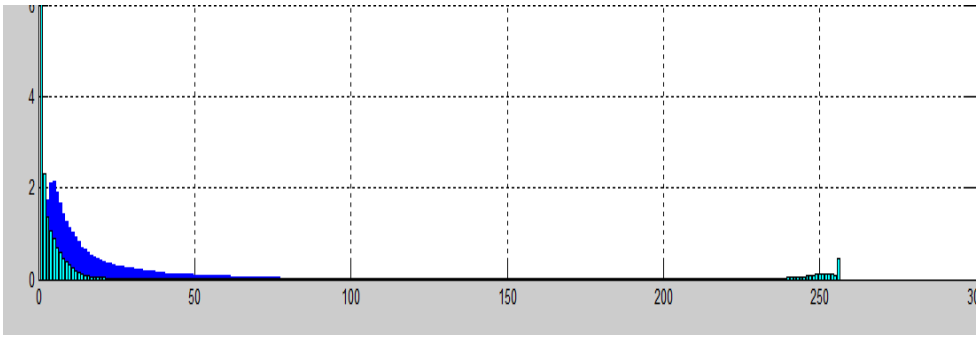


Figura 22. Resultado de imágenes de figuras geométricas figura 19.g en tarjeta y software Matlab.
Elaborado: Ordóñez, J

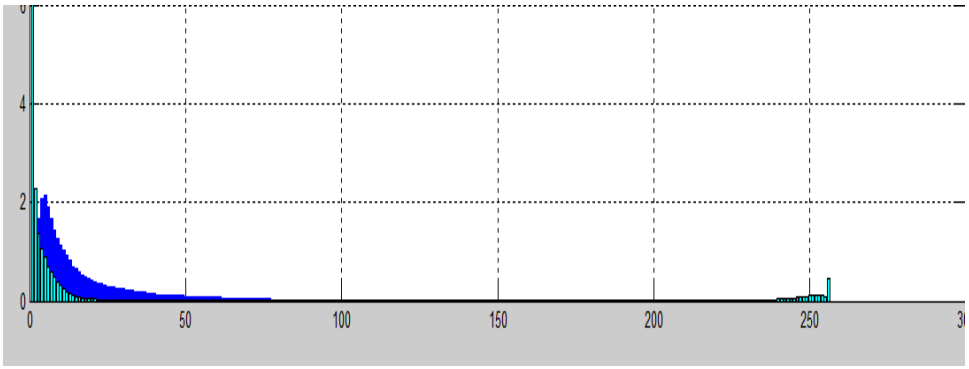
La figura 23, muestra la relación cualitativa de la imagen resultante de una impresora figua 19.j, en donde se observa el procesamiento tanto de la figura obtenida en la tarjeta de detección de bordes y software Matlab. Al realizar la comparación de imágenes de forma cualitativa, indica que los resultados en la tarjeta de detección de bordes es mejor en detalles que la comparada en el software Matlab, pero se tiene una mayor concentración de colores en el software Matlab para el color rojo, verde y azul haciendo que guarden relación en contraste y nitidez comparado con la imagen original.

Resultado de imágenes de una impresora figura 19.j en tarjeta y software Matlab

a) Contraste Software y tarjeta en Matlab color rojo



b) Contraste Software y tarjeta en Matlab color verde



c) Contraste Software y tarjeta en Matlab color azul

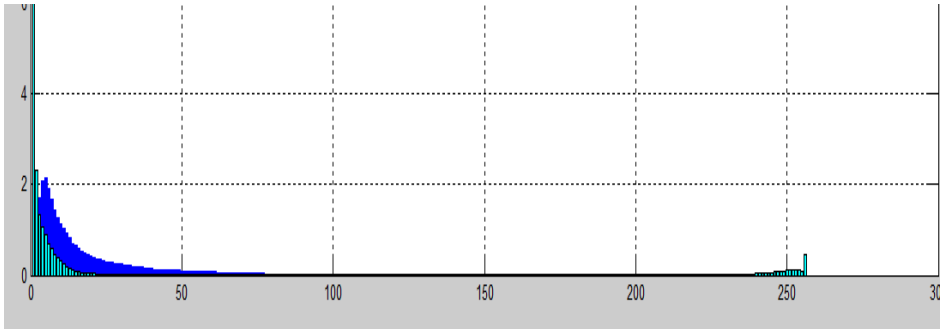
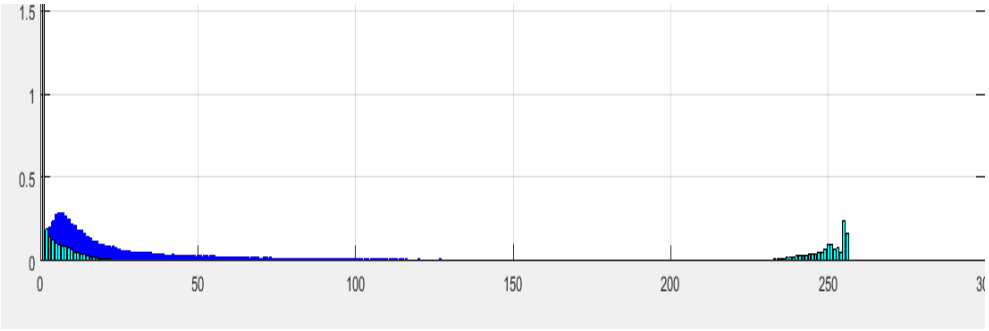


Figura 23. Resultado de imágenes de una impresora figura 19.j en tarjeta y software Matlab.
Elaborado: Ordóñez, J

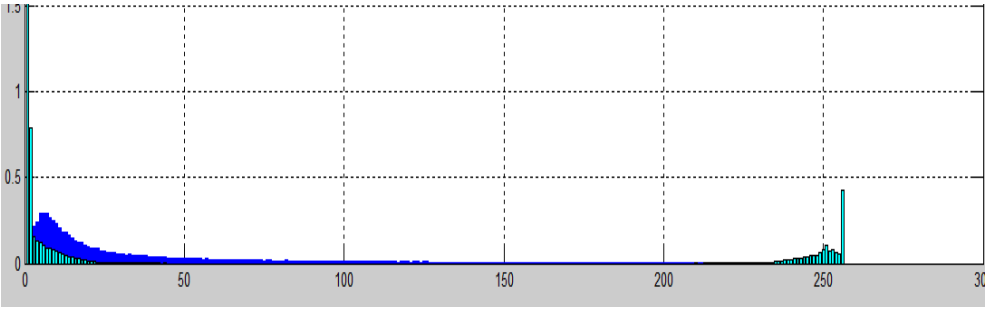
La figura 24, muestra la relación cualitativa de la imagen resultante de un edificio figua 19.m, en donde se observa el procesamiento tanto de la figura obtenida en la tarjeta de detección de bordes y software Matlab. Al realizar la comparación de imágenes de forma cualitativa, indica que los resultados en la tarjeta de detección de bordes es mejor en detalles que la comparada en el software Matlab, pero se tiene una mayor concentración de colores en el software Matlab para el color rojo, verde y azul haciendo que guarden relación en contraste y nitidez comparado con la imagen original.

Resultado de imágenes de un edificio figura 19.j en tarjeta y software Matlab

d) Contraste Software y tarjeta en Matlab color rojo



e) Contraste Software y tarjeta en Matlab color verde



f) Contraste Software y tarjeta en Matlab color azul

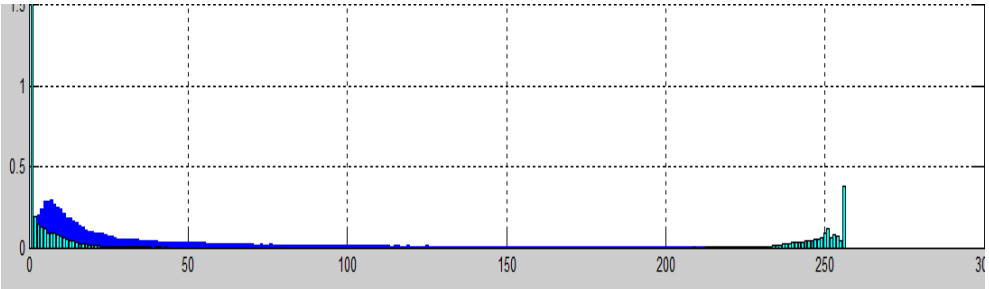


Figura 24. Resultado de imágenes de un edificio figura 19.m en tarjeta y software Matlab.
Elaborado: Ordóñez, J

4.3 Resultados del método de correlación entre las imágenes tanto de la tarjeta de detección de bordes y software usando Matlab.

Para comparar las imágenes de la figura 19 de forma cuantitativa se utilizó el método de correlación para imágenes digitales, con el mismo se obtuvo varios valores como se muestra en la Tabla 3 aplicando la correlación para ambas representaciones resultantes a partir del filtro tanto para las imágenes de la tarjeta de detección de bordes como para las del Software Matlab

Tabla 3.

Resultados de correlación entre imágenes obtenidas

N° Ilustración	Resultado de Correlación
Figura 19.a Rostro Juan Francisco Ordóñez	R= 0.6794
Figura 19.d Circuito Impreso	R= 0.8744
Figura 19.g Figuras Geometricas	R= 0.8700
Figura 19.j Figura impresora	R= 0.7533
Figura 19.m Edificio	R= 0.8659

Nota: Ordóñez, J
Fuente: Software Matlab

Para comparar las imágenes de la figura 19 de forma cuantitativa se utilizó el método de correlación para imágenes digitales, con el mismo se obtuvo varios valores como se muestra en la Tabla 3 aplicando la correlación para ambas representaciones resultantes a partir del filtro tanto para las imágenes de la tarjeta de detección de bordes como para las del Software Matlab.

En los resultados se baso en el parámetro de correlación R el cual va en el rango de 0 a 1, para obtener valores aproximandos siendo los que se acercan a 1 altamente relacionados mientras que los que se llegan a 0 no tienen relación. (Pedroza & Dicovskyi, 2006). El resultado final para la imagen de la figura 19.a obtenido en la Tabla 3, muestra que para la tarjeta de detección de bordes de imágenes como en el software Matlab aplicando el filtro Sobel, la correlación es R= 0.6794 por lo que se puede decir que este valor se aproxima a un rango entre 0.5 y 1, el cual indica que mantiene leve relación entre ambas imágenes correlacionadas. La comparación de los resultados de la figura 19.d de un circuito impreso obtenido en la Tabla 3 arroja un valor de R=0.8744, el cual muestra que el resultado se aproxima a 1, por lo tanto ambas imágenes son semejantes. Mediante la correlación figura 19.g de la imagen de las figuras geométricas obtenidas en la Tabla 3 muestra una relación de R=0.8700 , este valor se aproxima a 1 por lo que son semejantes.En los resultados de la figura 19.j para una impresora se obtuvo un valor de relación de R=0.7533 por lo que indica relacón entre ambas figuras comparadas. Finalmente el resultado de la correlación del ejemplo de un edificio figura 19.m muestra un resultado de R=0.8659, se podra decir que tienen cierta semejanza.

CONCLUSIONES

- Se diseñó e implemento un detector de bordes de imágenes con el fin de analizar imágenes, mediante la utilización de un microcontrolador STM32F429I-DISCO y el módulo de cámara OV7670 aplicando el algoritmo de Sobel, dando como resultado las imágenes observadas en la figura 20.
- En el programa Matlab se realizó el proceso de detección de bordes de imágenes, mediante la implementación del filtro Sobel, de esta manera se logró comparar las imágenes obtenidas en la pantalla TFT de la tarjeta STM32F429I-DISCO con respecto a las imágenes que se obtuvo en el software matemático Matlab.
- Para el análisis de resultados, para la imagen de mi rostro se obtuvo un R=0.6794, del circuito impreso se obtuvo R= 0.8744, en la imagen de las figuras se obtuvo un factor de correlación R= 0.8700, en la imagen de un edificio el coeficiente de correlacion es R= 0.8659, y para la imagen de la impresora se obtuvo un

  R =0.7533, al realizar la correlación de imagenes en la tarjeta de detección de bordes, con respecto a las imagenes obtenidas en el software Matlab se aprecia valores cercanos a 1, se concluye que los resultados arrojados mantienen relación de semejanza entre ambas imágenes comparadas. La tarjeta de procesamiento de imágenes tiene limitaciones debidas a que usa un menor numeor de bits para el desarrollo del algoritmo. La imagen mas similar es la del circuito impreso con un valor de correlación de R= 0.8744.
- Al momento de comparar las dos imágenes en la tarjeta de detección de bordes con respecto al software Matlab, el formato RGB 888 usa los 24 bits en donde se toma 8 bits para cada color, este formato permite una aproximación más exacta en términos de contraste y concentración de color que los 16 bits usados por el formato RGB 565 de la tarjeta de detección implementada, a pesar de la gran capacidad de memoria de la

cual está provisto el microcontrolador se observa un mejor procesamiento en el Software Matlab debido a una mayor cantidad de memoria que procesa un PC.

RECOMENDACIONES

- Con los resultados que se obtuvo de las imágenes aplicadas el filtro en el detector de bordes de imágenes, usando el microcontrolador STM32F429I-DISCO, se puede implementar en futuros proyectos nuevos algoritmos de detección como el operador Canny, el operador Laplaciano del Gaussiano (LoG), o el operador diferencial del Gaussiano (DoG).
- Para mejorar el proceso de detección de bordes de imágenes se puede implementar en futuros proyectos, un análisis espectral de las componentes de frecuencia de la imagen filtrada mediante la utilización del microcontrolador STM32F429I-DISCO.
- Debido a que la resolución de la cámara OV7670 es muy baja, se recomienda que en futuros proyectos se implemente un nuevo módulo de cámara para obtener imágenes con mejor resolución, detalles, etc.

REFERENCIAS

Antonio, J. (s.f.). Control Estadístico de Procesos. Pichincha , Ecuador.

Aparicio's, J. (Viernes 13 de Noviembre de 2015). Embedded Programmer - Atom. Obtenido de http://embeddedprogrammer.blogspot.com/2012/07/hacking-ov7670-camera-module-sccb-cheat.html

Ayala, D. (s.f.). Desarrollo de un sistema autónomo de detección y apuntamiento de objetos de color mediante el uso una cámara de video integrada a un microcontrolador . Quito, Pichincha, Ecuador.

CALOT, E. (Lunes 9 de Noviembre de 2008). Reconocimiento de patrones en imágenes médicas basado en sistemas inteligentes. Obtenido de https://books.google.com.ec/books?id=n7f52mHrs8MC&pg=PA17&lpg=PA17&dq=El+filtro+de+Sobel+parte+de+la+convoluci%C3%B3n+de+dos+matrices+con+la+imagen&source=bl&ots=0OBRhVxQ8W&sig=buqzRKF3_S0OPmWBVv_uqKZHQto&hl=es&sa=X&ved=0CDIQ6AEwCWoVChMIqJzBjc-EyQIVxHE-Ch

Capel. (Septiembre de 2008). Microcontroladores ARM. España.

Capel, D. T. (2008). Microcontroladores ARM. Revista Electrónica Española , 66.

Caprile, S. R. (2012). Desarrollo con microcontroladores ARM. Buenos Aires: Puntolibro.

Displaytech. (Viernes 13 de Noviembre de 2015). Liquid Crystal Display Modules &amp. Obtenido de http://www.displaytech-us.com/

Dr Macías, J. (2005). Matlab: Una introducción con ejemplos prácticos. Barcelona. España: REVERTÉ.

Dual, C., & Arm, C. (2012). Electrocomponentes S.A. Dual Core ARM Cortex A5-M4 .

Escolano, F., Cazorla, M. A., Alfonso, M. I., Colomina, O., & Lozano, M. Á. (2013). Inteligencia artificial Modelos, Técnicas y Áreas de Aplicación. Madrid: Paraninfo.

Esqueda, J., & Palafox, L. (2005). Fundamentos del procesamiento de imágenes. Fundamentos del procesamiento de imágenes . Baja California, México, México: editorial@info.rec.uabc.mx.

Firtec. (24 de Enero de 2015). Firtec. Obtenido de Arquitectura ARM: http://www.firtec.com.ar/cms/10-notas-tecnicas/10-cortex

Instruments, N. (Lunes 28 de Marzo de 2016). Comunicación Serial. Obtenido de Comunicación Serial: http://digital.ni.com/public.nsf/allkb/039001258CEF8FB686256E0F005888D1

Machala, P. (2015 de Diciembre de 2015). UREL. Obtenido de Individální projekty MPOA: http://www.urel.feec.vutbr.cz/MPOA/2014/cam-ov7670

Martinez, I. M.-I. (Domingo de Diciembre de 2015). Microcontroladores de 32 bits ARM. Obtenido de Microcontroladores de 32 bits ARM: http://www.arrowar.com/iweb/files_registracion/149czozNzoibWljcm9jb250cm9sYWRvcmVzK2RlKzMyK2JpdHMrYXJtLnBkZiI7.pdf

Martínez, L., & Martínez, M. (13 de Marzo de 2012). Detección de Bordes .

MikroElektronika. (Viernes 13 de Noviembre de 2015). MIKROE-606 MikroElektronika Placas e kits de desenvolvimento. Obtenido de PIC/DSPIC MIKROMMB DSPIC33 MULTIMEDIA BOARD: http://www.mouser.com/ds/2/272/mikrommb_dspic33_manual_v100-28348.pdf

Moreira, J., Valencia, V., & Chávez, P. (2009). Implementación de un algoritmo para la detección y conteo de células en imágenes microscópicas. Guayaquil, Guayas, Ecuador.

MOUSERELECTRONICS. (Viernes 13 de Noviembre de 2015). STM32F429I-DISCO STMicroelectronics Placas e kits de desenvolvimento. Obtenido de http://www.mouser.com/ProductDetail/STMicroelectronics/STM32F429I-DISCO/?qs=%2fha2pyFadugIKy%2fFd%2ffuLqZPt6I39%252b11MvJJbLiV2GqqgI%2f%252b4X0BDw%3d%3d

Navarro, A., Herrera, H., & Rodriguez, J. (18 de Septiembre de 2009). Minix 3 sobre Arquitectura ARM. Madrid, España.

Navarro, A., Herrera, H., & Rodriguez, J. (18 de Septiembre de 2009). Minix 3 Sobre Arquitectura ARM. Madrid, España.

Paguay, R., & Urgilés, R. (9 de Febrero de 2012). Recuperación de imágenes mediante extracción de bloops aplicando el operador laplaciano de gauss y el kernel gaussiano y desarrollo de un prototipo. Cuenca, Azuay, Ecuador.

Pedroza, H., & Dicovskyi, L. (2006). Sistema de análisis con SPSS. Managua, Nicaragua : LITONIC.

Vitutor. (17 de Febrero de 2016). Histogramas, Estadistica. Obtenido de Histogramas, Estadistica: http://www.vitutor.com/estadistica/descriptiva/a_6.html

ANEXOS

Anexo 1. Código para el análisis de resultados con histogramas

```
f=imread('PAcorr.tiff');
g=imread('PAcorrmat.tiff');
f_R=f(:,:,1);
g_R=g(:,:,1);
f_R=f(:,:,2);
g_R=g(:,:,2);
f_R=f(:,:,3);
g_R=g(:,:,3);
bar (imhist(g_R) , 'b');
title ('CONTRASTE SOFTWARE Y HARDWARE COLOR AZUL');
hold on
bar (imhist(f_R), 'c')
grid on
```

Anexo 2. Código para el análisis de resultados método de correlación

```
clear all
close all
clc

% Load clean image
s2 = imread('figurasmat.jpg');
s = rgb2gray(s2);
%s = double(s);

% Load clean image
s3 = imread('figuras.jpg');
x = rgb2gray(s3);
%x = double(x);


R = corr2(s,x)
imagesc(x)
colormap(gray)
figure
imagesc(s)
colormap(gray)
```

Anexo 3. Código para la transmisión serial de las imágenes hacia el computador

```
clear all
clc
delete(instrfind({'Port'},{'COM11'}));
%Borrar conexiones previas
%delete(instrfind({'Port'},{'/dev/tty.usbmodem621'}));
%Crear una conexion serie
%s = serial('COM5','BaudRate',115200,'Terminator','CR/LF');
s = serial('COM11','BaudRate',115200,'Terminator','CR/LF');
warning('off','MATLAB:serial:fscanf:unsuccessfulRead');
%Abrir el puerto
fopen(s);
%Inicializar las variables
Nvalores=75684*2; %Cantidad de valores que queremos a leer
%m1=zeros(1,Nvalores);
m1 = 0;
k=0;

disp('ENVIAR');
while k<Nvalores
%Leer el puerto serie
a = fscanf(s,'%d')';
m1=[m1 a];
k=k+1;
end
save('m1');

m1 = m1(2:length(m1));

for i=1:75684
    m2(i) = m1(i*2-1);
    m3(i) = m1(i*2);
end


%m2 = m1(2:length(m1));
r = bitand(m2,hex2dec('F800'));
r = 8*bitshift(r,-11);
for i=1:238
    rr(i,:) = r(318*(i-1)+1:318*(i));
end
g = bitand(m2,hex2dec('07E0'));
g = 4*bitshift(g,-5);
for i=1:238
    gg(i,:) = g(318*(i-1)+1:318*(i));
end
b = 8*bitand(m2,hex2dec('001F'));
for i=1:238
    bb(i,:) = b(318*(i-1)+1:318*(i));
```

```
end
%b = bitshift(r,-11);

A(:,:,1) = uint8(rr);
A(:,:,2) = uint8(gg);
A(:,:,3) = uint8(bb);
imwrite(A,'imSTM32F429.bmp','bmp')
t = imread('imSTM32F429.bmp');
imshow(t);

r = bitand(m3,hex2dec('F800'));
r = 8*bitshift(r,-11);
for i=1:238
   rr(i,:) = r(318*(i-1)+1:318*(i));
end
g = bitand(m3,hex2dec('07E0'));
g = 4*bitshift(g,-5);
for i=1:238
   gg(i,:) = g(318*(i-1)+1:318*(i));
end
b = 8*bitand(m3,hex2dec('001F'));
for i=1:238
   bb(i,:) = b(318*(i-1)+1:318*(i));
end
%b = bitshift(r,-11);

A(:,:,1) = uint8(rr);
A(:,:,2) = uint8(gg);
A(:,:,3) = uint8(bb);
imwrite(A,'imSTM32F429_B.bmp','bmp')
t = imread('imSTM32F429_B.bmp');
figure(2);
imshow(t);
```

Anexo 4. Código para el filtro Sobel en el microcontrolador STM32F429I-DISCO

```
/*
*
        ============================================================
=============
*   OV7670_control.c
*   (c) 2014, Petr Machala
*
*   Description:
*   OV7670 camera configuration and control file.
*   Optimized for 32F429IDISCOVERY board.
*
*   This program is free software: you can redistribute it and/or modify
```

```
*   it under the terms of the GNU General Public License as published by
*   the Free Software Foundation, either version 3 of the License, or
*   any later version.
*
*   This program is distributed in the hope that it will be useful,
*   but WITHOUT ANY WARRANTY; without even the implied warranty of
*   MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
*   GNU General Public License for more details.
*
*   You should have received a copy of the GNU General Public License
*   along with this program.  If not, see <http://www.gnu.org/licenses/>.
*
*   Camera wiring:
*   3V3         -   3V      ;       GND     -   GND
*   SIOC    -   PB8     ;       SIOD    -   PB9
*   VSYNC -  PB7     ;       HREF    -   PA4
*   PCLK    -   PA6     ;       XCLK    -   PA8
*   D7      -   PE6     ;       D6      -   PE5
*   D5      -   PB6     ;       D4      -   PE4
*   D3      -   PC9     ;       D2      -   PC8
*   D1      -   PC7     ;       D0      -   PC6
*   RESET   -   /       ;       PWDN-   /
*
*
        ============================================================
=============
*/

#include "OV7670_control.h"
#include "tm_stm32f4_usart.h"

// Image buffer
volatile uint16_t frame_buffer[IMG_ROWS*IMG_COLUMNS]; //Imagen tomada desde la
camara

volatile uint16_t sobel_buffer [2][IMG_ROWS];


volatile        uint32_t TC = 0;
volatile        uint32_t RC = 0; //Contador de filas  320x240
volatile        uint32_t CC = 0;        //Contador de columnas      320x240

volatile        int rM = 0;     //contador de filas 3x3
volatile        int cM = 0;     //contador de columnas 3x3
volatile        int cL = 0;     //cantador para 3x3

/*volatile      uint16_t RM [3][3];
volatile        uint16_t GM [3][3];
volatile        uint16_t BM [3][3];
*/
```

```
volatile        uint16_t CM [3][3][3];
volatile        uint16_t auxCM = 0;

const   int Gx[3][3] = {{1,  2,   1},
                {0,  0,   0},
                {-1, -2, -1}};

const int Gy[3][3] = {{1, 0, -1 },
                {2, 0, -2 },
                {1, 0, -1 }};

volatile        double pX [3];
volatile        double pY [3];
volatile        double pT [3];

//volatile      double pC [3][3];

volatile uint16_t i;


const uint8_t OV7670_reg [OV7670_REG_NUM][2] = {
        {0x12, 0x80},           //Reset registers

        // Image format
        {0x12, 0x14},           //QVGA size, RGB mode
        {0x40, 0xd0},           //RGB565
        {0xb0, 0x84},           //Color mode

        // Hardware window
        {0x11, 0x01},           //PCLK settings, 15fps
        {0x32, 0x80},           //HREF
        {0x17, 0x17},           //HSTART
        {0x18, 0x05},           //HSTOP
        {0x03, 0x0a},           //VREF
        {0x19, 0x02},           //VSTART
        {0x1a, 0x7a},           //VSTOP

        // Scalling numbers
        {0x70, 0x3a},           //X_SCALING
        {0x71, 0x35},           //Y_SCALING
        {0x72, 0x11},           //DCW_SCALING
        {0x73, 0xf0},           //PCLK_DIV_SCALING
        {0xa2, 0x02},           //PCLK_DELAY_SCALING

        // Matrix coefficients
        {0x4f, 0x80},           {0x50, 0x80},
        {0x51, 0x00},           {0x52, 0x22},
        {0x53, 0x5e},           {0x54, 0x80},
        {0x58, 0x9e},
```

```
// Gamma curve values
{0x7a, 0x20},       {0x7b, 0x10},
{0x7c, 0x1e},       {0x7d, 0x35},
{0x7e, 0x5a},       {0x7f, 0x69},
{0x80, 0x76},       {0x81, 0x80},
{0x82, 0x88},       {0x83, 0x8f},
{0x84, 0x96},       {0x85, 0xa3},
{0x86, 0xaf},       {0x87, 0xc4},
{0x88, 0xd7},       {0x89, 0xe8},

// AGC and AEC parameters
{0xa5, 0x05},       {0xab, 0x07},
{0x24, 0x95},       {0x25, 0x33},
{0x26, 0xe3},       {0x9f, 0x78},
{0xa0, 0x68},       {0xa1, 0x03},
{0xa6, 0xd8},       {0xa7, 0xd8},
{0xa8, 0xf0},       {0xa9, 0x90},
{0xaa, 0x94},       {0x10, 0x00},

// AWB parameters
{0x43, 0x0a},       {0x44, 0xf0},
{0x45, 0x34},       {0x46, 0x58},
{0x47, 0x28},       {0x48, 0x3a},
{0x59, 0x88},       {0x5a, 0x88},
{0x5b, 0x44},       {0x5c, 0x67},
{0x5d, 0x49},       {0x5e, 0x0e},
{0x6c, 0x0a},       {0x6d, 0x55},
{0x6e, 0x11},       {0x6f, 0x9f},
{0x6a, 0x40},       {0x01, 0x40},
{0x02, 0x60},       {0x13, 0xe7},

// Additional parameters
{0x34, 0x11},       {0x3f, 0x00},
{0x75, 0x05},       {0x76, 0xe1},
{0x4c, 0x00},       {0x77, 0x01},
{0xb8, 0x0a},       {0x41, 0x18},
{0x3b, 0x12},       {0xa4, 0x88},
{0x96, 0x00},       {0x97, 0x30},
{0x98, 0x20},       {0x99, 0x30},
{0x9a, 0x84},       {0x9b, 0x29},
{0x9c, 0x03},       {0x9d, 0x4c},
{0x9e, 0x3f},       {0x78, 0x04},
{0x0e, 0x61},       {0x0f, 0x4b},
{0x16, 0x02},       {0x1e, 0x00},
{0x21, 0x02},       {0x22, 0x91},
{0x29, 0x07},       {0x33, 0x0b},
{0x35, 0x0b},       {0x37, 0x1d},
{0x38, 0x71},       {0x39, 0x2a},
{0x3c, 0x78},       {0x4d, 0x40},
{0x4e, 0x20},       {0x69, 0x00},
```

```
        {0x6b, 0x3a},       {0x74, 0x10},
        {0x8d, 0x4f},       {0x8e, 0x00},
        {0x8f, 0x00},       {0x90, 0x00},
        {0x91, 0x00},       {0x96, 0x00},
        {0x9a, 0x00},       {0xb1, 0x0c},
        {0xb2, 0x0e},       {0xb3, 0x82},
        {0x4b, 0x01},
};

void SobelFilter (int op, int th){
        /* Initialize USART2 at 9600 baud, TX: PA2, RX: PA3 */
        TM_USART_Init(USART2, TM_USART_PinsPack_2, 115200);
        //i=1234;
//  sprintf(STRING_BUF,"%d\n\r",i);
//      TM_USART_Puts(USART2, STRING_BUF);


                for(CC = 1; CC < IMG_COLUMNS-1; CC++){

                        for(RC = 1; RC < IMG_ROWS-1; RC++){
                                auxCM = frame_buffer[(CC)*IMG_ROWS + RC];
                                sprintf(STRING_BUF,"%d\r\n",auxCM);
                                TM_USART_Puts(USART2, STRING_BUF);

                                for (cM = -1; cM < 2; cM++){
        //Completar matriz 3x3 de cada color
                                        for (rM = -1; rM < 2; rM++){
                                                /* Put string to USART */
                                                //TM_USART_Puts(USART2,
"CAMARA\n\r");
                                                auxCM = frame_buffer[(CC +
cM)*IMG_ROWS + RC - rM];

        //sprintf(STRING_BUF,"%d\n\r",auxCM);
                                                //TM_USART_Puts(USART2,
STRING_BUF);

                                                CM [0][rM+1][cM+1] = (auxCM &
0xF800) >> 11; //R
                                                CM [1][rM+1][cM+1]= (auxCM &
0x07E0) >> 5; //G
                                                CM [2][rM+1][cM+1]= auxCM &
0x001F; //B
                                        }
                                }


                                for (cM = 0, pX[0] = 0, pX[1] = 0, pX[2] = 0; cM < 3;
cM++){         //multiplicar matriz 3x3 por Gx
                                        for (rM = 0; rM < 3; rM++){
```

```
                                        pX [0] = pX [0] + CM[0][rM][cM] *
Gx[rM][cM];
                                        pX [1] = pX [1] + CM[1][rM][cM] *
Gx[rM][cM];
                                        pX [2] = pX [2] + CM[2][rM][cM] *
Gx[rM][cM];
                                }
                        }


                        for (cM = 0, pY[0] = 0, pY[1] = 1, pY[2] = 2; cM < 3;
cM++){ //multiplicar matriz 3x3 por Gy
                                for (rM = 0; rM < 3; rM++){
                                        pY [0] = pY [0] + CM[0][rM][cM]*
Gy[rM][cM];
                                        pY [1] = pY [1] + CM[1][rM][cM]*
Gy[rM][cM];
                                        pY [2] = pY [2] + CM[2][rM][cM]*
Gy[rM][cM];
                                }
                        }


                        for(cL = 0; cL < 3; cL++){
                                pT[cL] = sqrt(pow(pX[cL],2) + pow(pY[cL],2));
                        }

                        Delay(20000);
                        if(op == 0){
                                if((((uint16_t)pT[0]) + (((uint16_t)pT[1])>>1) +
((uint16_t)pT[2])) > th){
                                        //sobel_buffer [1][RC] = 0xFFFF;
                                        auxCM = 0xFFFF;
                                        sprintf(STRING_BUF,"%d\r\n",auxCM);
                                        TM_USART_Puts(USART2,
STRING_BUF);
                                        LCD_ILI9341_DrawPixel(RC, CC,
0xFFFF);
                                }else{
                                        //sobel_buffer [1][RC] = 0;
                                        auxCM = 0x0;
                                        sprintf(STRING_BUF,"%d\r\n",auxCM);
                                        TM_USART_Puts(USART2,
STRING_BUF);
                                        LCD_ILI9341_DrawPixel(RC, CC, 0);
                                }
                        }
                        else{
                                //sobel_buffer [1][RC] = ((((uint16_t)pT[0]) <<
11) + (((uint16_t)pT[1]) << 5) + (((uint16_t)pT[2])));
```

```
                                                LCD_ILI9341_DrawPixel(RC, CC,
((((uint16_t)pT[0]) << 11) + (((uint16_t)pT[1]) << 5) + (((uint16_t)pT[2]))));
                                        }
                                        Delay(1000);
                                }
                                /*
                                for(rM = 0; rM < IMG_ROWS; rM++){
                                        frame_buffer[(CC-1)*IMG_ROWS + rM] = sobel_buffer
[0][rM];
                                }

                                for(rM = 0; rM < IMG_ROWS; rM++){
                                        sobel_buffer [0][rM] = sobel_buffer [1][rM];
                                }               */
                        }/*
                        for(rM = 0; rM < IMG_ROWS; rM++){
                                frame_buffer[(IMG_COLUMNS-1)*IMG_ROWS + rM] = 0;
                        }               */
}

void Delay(volatile uint16_t nCount){
  while(nCount--){
  }
}

void MCO1_init(void){
        GPIO_InitTypeDef GPIO_InitStructure;

        RCC_ClockSecuritySystemCmd(ENABLE);

        RCC_AHB1PeriphClockCmd(RCC_AHB1Periph_GPIOA, ENABLE);

        // GPIO config
        GPIO_InitStructure.GPIO_Pin = GPIO_Pin_8;               //PA8 - XCLK
        GPIO_InitStructure.GPIO_Speed = GPIO_Speed_100MHz;
        GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AF;
        GPIO_InitStructure.GPIO_OType = GPIO_OType_PP;
        GPIO_InitStructure.GPIO_PuPd = GPIO_PuPd_UP;
        GPIO_Init(GPIOA, &GPIO_InitStructure);

        // GPIO AF config
        GPIO_PinAFConfig(GPIOA, GPIO_PinSource8, GPIO_AF_MCO);

        // MCO clock source
        RCC_MCO1Config(RCC_MCO1Source_PLLCLK, RCC_MCO1Div_4);
}

void SCCB_init(void){
        GPIO_InitTypeDef GPIO_InitStructure;
        I2C_InitTypeDef I2C_InitStructure;
```

```
  RCC_APB1PeriphClockCmd(RCC_APB1Periph_I2C1, ENABLE);
  RCC_AHB1PeriphClockCmd(RCC_AHB1Periph_GPIOB, ENABLE);

        // GPIO config
        GPIO_InitStructure.GPIO_Pin = GPIO_Pin_8 | GPIO_Pin_9;          //PB8 -
SIOC


                                                //PB9 - SIOD
        GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AF;
        GPIO_InitStructure.GPIO_Speed = GPIO_Speed_2MHz;
        GPIO_InitStructure.GPIO_OType = GPIO_OType_OD;
        GPIO_InitStructure.GPIO_PuPd = GPIO_PuPd_UP;
        GPIO_Init(GPIOB, &GPIO_InitStructure);

        // GPIO AF config
        GPIO_PinAFConfig(GPIOB, GPIO_PinSource8, GPIO_AF_I2C1);
        GPIO_PinAFConfig(GPIOB, GPIO_PinSource9, GPIO_AF_I2C1);

        // I2C config
  I2C_DeInit(I2C1);
  I2C_InitStructure.I2C_Mode = I2C_Mode_I2C;
  I2C_InitStructure.I2C_DutyCycle = I2C_DutyCycle_2;
  I2C_InitStructure.I2C_OwnAddress1 = 0x00;
  I2C_InitStructure.I2C_Ack = I2C_Ack_Enable;
  I2C_InitStructure.I2C_AcknowledgedAddress = I2C_AcknowledgedAddress_7bit;
  I2C_InitStructure.I2C_ClockSpeed = 100000;
        I2C_ITConfig(I2C1, I2C_IT_ERR, ENABLE);
  I2C_Init(I2C1,&I2C_InitStructure);
        I2C_Cmd(I2C1, ENABLE);
}

bool SCCB_write_reg(uint8_t reg_addr, uint8_t* data){
        uint32_t timeout = 0x7FFFFF;

        while(I2C_GetFlagStatus(I2C1, I2C_FLAG_BUSY)){
                if ((timeout--) == 0){
                        return true;
                }
        }

        // Send start bit
        I2C_GenerateSTART(I2C1, ENABLE);

  while( !I2C_CheckEvent(I2C1,I2C_EVENT_MASTER_MODE_SELECT)){
                if ((timeout--) == 0){
                        return true;
                }
  }
```

```
	// Send slave address (camera write address)
  I2C_Send7bitAddress(I2C1, OV7670_WRITE_ADDR, I2C_Direction_Transmitter);

	while(!I2C_CheckEvent(I2C1,
I2C_EVENT_MASTER_TRANSMITTER_MODE_SELECTED)){
		if ((timeout--) == 0){
			return true;
		}
  }

	// Send register address
	I2C_SendData(I2C1, reg_addr);

	while(!I2C_CheckEvent(I2C1,I2C_EVENT_MASTER_BYTE_TRANSMITTED)){
		if ((timeout--) == 0){
			return true;
		}
  }

	// Send new register value
  I2C_SendData(I2C1, *data);

  while(!I2C_CheckEvent(I2C1,I2C_EVENT_MASTER_BYTE_TRANSMITTED)){
		if ((timeout--) == 0){
			return true;
		}
  }

	// Send stop bit
  I2C_GenerateSTOP(I2C1, ENABLE);
	return false;
}

bool OV7670_init(void){
	uint8_t data, i = 0;
	bool err;

	// Configure camera registers
	for(i=0; i<OV7670_REG_NUM ;i++){
		data = OV7670_reg[i][1];
		err = SCCB_write_reg(OV7670_reg[i][0], &data);

		if (err == true)
			break;

		LCD_ILI9341_DrawLine(99+i, 110, 99+i, 130, ILI9341_COLOR_WHITE);
		Delay(0xFFFF);
	}
```

```
	return err;
}
void STOP_CAMERA (void) {
	DCMI_Cmd(DISABLE);
	DMA_Cmd(DMA2_Stream1, DISABLE);
}

void RUN_CAMERA (void) {
	DCMI_Cmd(ENABLE);
	DMA_Cmd(DMA2_Stream1, ENABLE);
}
void DCMI_DMA_init(void){
	GPIO_InitTypeDef GPIO_InitStructure;
  DCMI_InitTypeDef DCMI_InitStructure;
  DMA_InitTypeDef  DMA_InitStructure;
	NVIC_InitTypeDef NVIC_InitStructure;

	RCC_AHB2PeriphClockCmd(RCC_AHB2Periph_DCMI, ENABLE);
  RCC_AHB1PeriphClockCmd(RCC_AHB1Periph_GPIOA, ENABLE);
	RCC_AHB1PeriphClockCmd(RCC_AHB1Periph_GPIOB, ENABLE);
	RCC_AHB1PeriphClockCmd(RCC_AHB1Periph_GPIOC, ENABLE);
	RCC_AHB1PeriphClockCmd(RCC_AHB1Periph_GPIOE, ENABLE);
  RCC_AHB1PeriphClockCmd(RCC_AHB1Periph_DMA2, ENABLE);

	// GPIO config
	GPIO_InitStructure.GPIO_Pin = GPIO_Pin_4 | GPIO_Pin_6;		//PA4 -
HREF


                                        //PA6 - PCLK
  GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AF;
  GPIO_InitStructure.GPIO_Speed = GPIO_Speed_100MHz;
  GPIO_InitStructure.GPIO_OType = GPIO_OType_PP;
  GPIO_InitStructure.GPIO_PuPd = GPIO_PuPd_UP ;
  GPIO_Init(GPIOA, &GPIO_InitStructure);

  GPIO_InitStructure.GPIO_Pin = GPIO_Pin_6 | GPIO_Pin_7;		//PB6 - D5


                                        //PB7 - VSYNC
  GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AF;
  GPIO_InitStructure.GPIO_Speed = GPIO_Speed_100MHz;
  GPIO_InitStructure.GPIO_OType = GPIO_OType_PP;
  GPIO_InitStructure.GPIO_PuPd = GPIO_PuPd_UP ;
  GPIO_Init(GPIOB, &GPIO_InitStructure);

  GPIO_InitStructure.GPIO_Pin = GPIO_Pin_6 | GPIO_Pin_7 | GPIO_Pin_8 | GPIO_Pin_9;
		//PC6 - D0
```

```
                                                    //PC7 - D1


                                                    //PC8 - D2


                                                    //PC9 - D3
  GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AF;
  GPIO_InitStructure.GPIO_Speed = GPIO_Speed_100MHz;
  GPIO_InitStructure.GPIO_OType = GPIO_OType_PP;
  GPIO_InitStructure.GPIO_PuPd = GPIO_PuPd_UP ;
  GPIO_Init(GPIOC, &GPIO_InitStructure);

  GPIO_InitStructure.GPIO_Pin =  GPIO_Pin_4 | GPIO_Pin_5 | GPIO_Pin_6;          //PE4
- D4


        //PE5 - D6


        //PE6 - D7
  GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AF;
  GPIO_InitStructure.GPIO_Speed = GPIO_Speed_100MHz;
  GPIO_InitStructure.GPIO_OType = GPIO_OType_PP;
  GPIO_InitStructure.GPIO_PuPd = GPIO_PuPd_UP ;
  GPIO_Init(GPIOE, &GPIO_InitStructure);

        // GPIO AF config
        GPIO_PinAFConfig(GPIOB, GPIO_PinSource7, GPIO_AF_DCMI);
  GPIO_PinAFConfig(GPIOA, GPIO_PinSource4, GPIO_AF_DCMI);
  GPIO_PinAFConfig(GPIOA, GPIO_PinSource6, GPIO_AF_DCMI);
  GPIO_PinAFConfig(GPIOC, GPIO_PinSource6, GPIO_AF_DCMI);
  GPIO_PinAFConfig(GPIOC, GPIO_PinSource7, GPIO_AF_DCMI);
  GPIO_PinAFConfig(GPIOC, GPIO_PinSource8, GPIO_AF_DCMI);
  GPIO_PinAFConfig(GPIOC, GPIO_PinSource9, GPIO_AF_DCMI);
  GPIO_PinAFConfig(GPIOE, GPIO_PinSource4, GPIO_AF_DCMI);
  GPIO_PinAFConfig(GPIOB, GPIO_PinSource6, GPIO_AF_DCMI);
  GPIO_PinAFConfig(GPIOE, GPIO_PinSource5, GPIO_AF_DCMI);
  GPIO_PinAFConfig(GPIOE, GPIO_PinSource6, GPIO_AF_DCMI);

        // DCMI config
        DCMI_DeInit();
  DCMI_InitStructure.DCMI_CaptureMode = DCMI_CaptureMode_Continuous;
        //DCMI_InitStructure.DCMI_CaptureMode = DCMI_CaptureRate_1of4_Frame;
        //DCMI_InitStructure.DCMI_CaptureMode = DCMI_CaptureRate_1of2_Frame;
```

```
DCMI_InitStructure.DCMI_ExtendedDataMode = DCMI_ExtendedDataMode_8b;
DCMI_InitStructure.DCMI_CaptureRate = DCMI_CaptureRate_All_Frame;
        //DCMI_InitStructure.DCMI_CaptureRate = DCMI_CaptureRate_1of2_Frame;

        DCMI_InitStructure.DCMI_PCKPolarity = DCMI_PCKPolarity_Rising;
        DCMI_InitStructure.DCMI_HSPolarity = DCMI_HSPolarity_Low;
DCMI_InitStructure.DCMI_VSPolarity = DCMI_VSPolarity_High;
DCMI_InitStructure.DCMI_SynchroMode = DCMI_SynchroMode_Hardware;
DCMI_Init(&DCMI_InitStructure);
        DCMI_Cmd(ENABLE);

        // DMA config
DMA_DeInit(DMA2_Stream1);
        DMA_InitStructure.DMA_Channel = DMA_Channel_1;
DMA_InitStructure.DMA_PeripheralBaseAddr = (uint32_t)(&DCMI->DR);
DMA_InitStructure.DMA_Memory0BaseAddr = (uint32_t)frame_buffer;
DMA_InitStructure.DMA_DIR = DMA_DIR_PeripheralToMemory;
DMA_InitStructure.DMA_BufferSize = IMG_ROWS*IMG_COLUMNS*2/4;
DMA_InitStructure.DMA_PeripheralInc = DMA_PeripheralInc_Disable;
DMA_InitStructure.DMA_MemoryInc = DMA_MemoryInc_Enable;
DMA_InitStructure.DMA_PeripheralDataSize = DMA_PeripheralDataSize_Word;
DMA_InitStructure.DMA_MemoryDataSize = DMA_MemoryDataSize_HalfWord;
DMA_InitStructure.DMA_Mode = DMA_Mode_Normal;
DMA_InitStructure.DMA_Priority = DMA_Priority_High;
DMA_InitStructure.DMA_FIFOMode = DMA_FIFOMode_Enable;
DMA_InitStructure.DMA_FIFOThreshold = DMA_FIFOThreshold_Full;
DMA_InitStructure.DMA_MemoryBurst = DMA_MemoryBurst_Single;
DMA_InitStructure.DMA_PeripheralBurst = DMA_PeripheralBurst_Single;
DMA_Init(DMA2_Stream1, &DMA_InitStructure);
        DMA_Cmd(DMA2_Stream1, ENABLE);

        // DMA interrupt
        DMA_ITConfig(DMA2_Stream1, DMA_IT_TC, ENABLE);

        NVIC_InitStructure.NVIC_IRQChannel = DMA2_Stream1_IRQn;
        NVIC_InitStructure.NVIC_IRQChannelPreemptionPriority = 0;
        NVIC_InitStructure.NVIC_IRQChannelSubPriority = 0;
        NVIC_InitStructure.NVIC_IRQChannelCmd = ENABLE;
        NVIC_Init(&NVIC_InitStructure);
}
```