UNIVERSIDAD MICHOACANA
DE SAN
NICOLÁS DE HIDALGO

FACULTAD DE INGENIERÍA
ELÉCTRICA

Tratamiento de imágenes en un procesador embebido.

TESIS

Que para obtener el título de

INGENIERA EN ELECTRÓNICA

Presenta

Dayana Vanesa González Salazar

Asesor

M.C. Alberto Carlos Salas Mier

Octubre, 2017
Morelia, Michoacán.

Agradecimientos

- Quiero dar gracias a mi Dios, que sin su gracia no podría estar en este punto de mi vida.

- Agradezco infinitamente a mis padres Martín y Bertha por su apoyo incondicional, por la educación, su comprensión, su amor, por soportar mi ausencia, por tener paciencia en los días malos y cuidar de mí cuando fue necesario.

- También doy gracias a mis profesores quiénes tuvieron la confianza en mí y mis capacidades académicas.

- Agradezco a mi asesor, profesor y amigo el M.C. Alberto Carlos Salas Mier por su dedicación, formación y paciencia para dirigirme en este trabajo y en mi desarrollo académico.

- A mis tíos y tías que me apoyaron a lo largo de este camino.

- Por último agradezco a los lectores de este trabajo.

Índice general

Índice de figuras

Índice de tablas

Resumen

Este trabajo esta motivado por la necesidad actual en la optimización y automatización de procesos y el gran impulso que el procesamiento de señales y la aplicación de visión computacional han tenido en algunas áreas como la industria, la medicina, militar, espacial.

La tesis presenta algunos de los fundamentos matemáticos básicos utilizados para el procesamiento de señales, como álgebra lineal, análisis de Fourier y probabilidad y estadística, los cuales son el soporte para el desarrollo del trabajo .

Se describen las herramientas de software y hardware utilizadas: OpenCV, herramienta de software que proporciona una gran cantidad de funciones enfocadas al procesamiento de imágenes y visión computacional, y el entorno Linux Ubuntu; y el dispositivo DE1_SoC el cual cuenta con un sistema en chip que contiene dos secciones: el HPS (Sistema de Procesador en Hardware) y la FPGA (Matriz de Compuertas Programables), que proporcionan ventajas en comparación con un microcontrolador o un ordenador de escritorio.

Finalmente, se implementan dentro del procesador embebido: un algoritmo de mejoramiento de imágenes y otros dos que permiten obtener características especificas de una imagen.

Palabras clave: Dispositivo embebido, Sistema en Chip (SoC), FPGA, HPS, Procesamiento de imÃągenes, OpenCV.

Abstract

Current needs in process optimization and automation as well as the great impulse that signal processing and computer vision have had in some areas such as industry, medicine, military, space, among others, have motivated this research.

This thesis presents some of the mathematical foundations used for signal processing, such as linear algebra, Fourier analysis and probability and statistics, which are the support for the development of the work.

It describes the software tools and hardware used: OpenCV library provides a lot of functions focused on image processing and computer vision, and the Linux Ubuntu; the DE1_SoC device which is an on-chip system that contains two sections: HPS (Hardware Processor System) and FPGA (Field Programmable Gate Array), that offers advantages compared to a microcontroller or a desktop computer.

Finally, an image enhancement algorithm and two feature extraction ones are implemented within the embedded processor.

A mi abuelita Felipa Pacheco Saucedo, por sus sabios consejos y dejarme la enseñanza de afrontar la vida con valentía.

A mis padres, que me han enseñado a no rendirme jamás.

CAPÍTULO 1

Introducción.

1.1. Introducción

El ojo humano es un sistema receptor que nos permite percibir a gran detalle la forma, color y algunas otras características de cualquier objeto en nuestro entorno; es bajo el mismo principio que una cámara realiza la captura de imágenes; la imagen obtenida por un dispositivo de captura, es únicamente una representación de la escena original, ya que el ruido del canal de transmisión y características del propio dispositivo, como defectos en el sensor o errores de conversión, limitan la generación de una imagen idéntica. Para mejorar una imagen tomada por algún dispositivo se hace uso del procesamiento digital de imágenes que nos permite hacer un análisis a profundidad de dichas señales capturadas.

A continuación se proporciona una clasificación de las operaciones que el procesamiento de imágenes realiza [10]:

- *Representación y modelado de imágenes.-* En este nivel, se realizan las operaciones de muestreo y cuantificación, es decir la representación de una escena continua por medio de una serie de datos discretos; para esto se requieren los desarrollos del teorema de muestreo y de transformación de Fourier
- *Mejoramiento de imágenes.-* Consiste en realzar ciertos detalles de la imagen capturada para un análisis posterior. Algunos de los ejemplos de mejoramiento de imagen son: el brillo, mejoramiento en bordes, filtrado de ruido, entre otros. El mejoramiento de imágenes hace uso de algoritmos, herramientas computacionales y operaciones matemáticas como la convolución, la transformada discreta de Fourier, entre otras.

- *Restauración de imágenes.-* Este nivel se encarga de la remoción o minimización de la descomposición conocida de una imagen, un ejemplo es la reducción de la de borrosidad de una imagen (deblurring). Se hace uso de algunos métodos, ya sean probabilistas o deterministas, que permiten la restauración de una imagen.

- *Análisis de imágenes.-* Este nivel en procesamiento de imágenes se encarga de recuperar información especifica de una imagen para su posterior análisis o incluso su aplicación a algunas áreas como la medicina. Algunos de los algoritmos ya desarrollados son: la detección de bordes de una imagen, las características de forma, detección de algún cuadro en particular, entre otros.

- *Reconstrucción de imágenes.-* Debido a las deficiencias de un dispositivo de captura de imágenes se puede tener la ausencia de uno o más segmentos en éstas, por lo que el procesamiento de imágenes en este nivel se encarga de realizar una estimación que contrarreste el defecto por medio de algoritmos que permitan la reconstrucción más aproximada a esta sección sin perder la relación con la imagen. Algunos de los problemas de reconstrucción de imágenes se presentan en el área médica, por ejemplo en los rayos X donde su principio está basado en la proyección de un haz a través de un objeto que permite tener una vista de este desde distintos ángulos; para algunos de ellos, no se tiene información disponible por lo que es necesario realizar un estimación de estos segmentos.

- *Compresión de datos de imágenes.-* La cantidad de datos asociados con la información visual es tan grande que su almacenamiento requeriría una enorme capacidad de almacenamiento. Aunque la capacidad de varios medios de almacenamiento son sustanciales, sus velocidades de acceso suelen ser inversamente proporcional a esta capacidad. Las técnicas de compresión de datos de imagen se preocupan por la reducción del numero de bits requeridos para almacenar o transmitir imágenes sin ninguna perdida apreciable de información. Las aplicaciones de la compresión de datos son principalmente en la transmisión y almacenamiento de información.
  La aplicación de compresión de datos también es posible en el desarrollo de algoritmos rápidos donde el número de operaciones requeridas para implementar un algoritmo se reduce al trabajar con los datos comprimidos.

1.2. Reseña histórica

El procesamiento de imagen y la visión computacional han tenido un gran desarrollo en los últimos años ya que permiten optimizar procesos con ayuda de nuevas tecnologías como los procesadores de computadoras.

La reseña histórica que a continuación se presenta se ha acotado sólo a los trabajos realizados en la Facultad de Ingeniería Eléctrica de la Universidad Michoacana de San Nicolás de Hidalgo, a partir del año 2010 se encuentran tesis de interés con la orientación al tema de procesamiento de señales. En este año se encuentra la tesis titulada "Reconocimiento inteligente de flujo vehicular basado en el procesamiento de imágenes" [11] en la que se describe el problema del flujo vehicular en la ciudad, se realizan distintas pruebas con imágenes y la detección de vehículos, e implementar en un semáforo inteligente según la cantidad de vehículos detectados. En 2013 se encuentran tres tesis: la primera titulada "Indices de proximidad en el reconocimiento de voz" [12] en la que se prueban distintos indices de proximidad y la similitud entre objetos para encontrar una palabra en una base de datos. La siguiente tesis de este año titulada "Diseño e implementación de un dispositivo detector de profundidad de apoyo para personas invidentes" [13] donde hacen uso de herramientas muy conocidas como es el sensor Kinect para la detección de imágenes y una *barebone* BeagleBone para el procesamiento de las señales obtenidas. Otra de las tesis de este año es "Aplicación de visión con LabView en el seguimiento de linea para un móvil" [14] en el que implementan un prototipo que podrá detectar, rastrear y seguir el patrón de una linea a través de de una *webcam* recabando los datos en el software **IMAQ Visión** enviándolos a un microprocesador y haciendo uso del entorno de desarrollo gráfico LabView para el análisis, adquisición y despliegue de los datos. Por ultimo en el año 2015 se encuentran dos tesis la primera titulada "Robot móvil autónomo seguidor de linea con Raspberry Pi y biblioteca de visión computacional (OPENCV)" [15] en la que se realiza el prototipo de un robot móvil autónomo seguidor de linea mediante visión computacional utilizando una microcomputadora Raspberry Pi, la librería OpenCV y un Arduino para implementar un controlador PID. Y por ultimo la tesis titulada "Sistema de visión computacional para el conteo de personas heurística para el seguimiento de objetos" [16] en la que se presentan dos heurísticas nuevas para el seguimiento de objetos basadas en el contorno, para el diseño e implementación de un sistema para el conteo de personas por medio de visión computacional.

1.3. Objetivo

El objetivo de este trabajo es probar tres algoritmos: el algoritmo CLS, el algoritmo de Canny y el algoritmo de Hough, los cuales se encuentran en distintos niveles de procesamiento de imagen, con ayuda del esquema de procesamiento propuesto, constituido por las partes de Hardware y Software:

- **Hardware:** Un dispositivo SoC (Sistema en chip) de Altera, ahora Intel; este consta de dos secciones: la sección del HPS y la sección del FPGA, que permite tener un sistema versátil en comparación con un ordenador de escritorio o un microcontrolador,

- **Software:** La biblioteca OpenCV que cuenta con un amplio contenido de algoritmos específicos para el procesamiento de imágenes y visión computacional.

1.4. Motivación

Este trabajo es motivado por el interés de profundizar aun más en los temas estudiados en las asignaturas: procesamiento digital de señales, programación de computadoras, álgebra lineal, así como probabilidad y estadística, cursadas durante la licenciatura.

El procesamiento de imágenes y la visión computacional son procesos computacionales en desarrollo continuo y han tenido un avance exponencial en los últimos años. Estos permiten eliminar imperfecciones, mejorar imágenes con problemas por movimiento o desenfoque, localizar figuras u objetos en una imagen, etc. La contribución de dichos temas pueden ser aplicados en un sinfín de proyectos dentro de las áreas de robótica móvil, vehículos autónomos, análisis de imágenes aéreas, análisis de imágenes médicas, reconocimiento de personas, gestos, objetos, vídeojuegos, entre otros.

1.5. Esquema de la tesis

A continuación se da una descripción breve de los capítulos que conforman la tesis.
En el **Capítulo 2** se describe los fundamentos teóricos del procesamiento de señales. En el **Capítulo 3** es detallado el entorno de software y hardware donde se desarrollo el trabajo. En el **Capítulo 4** se presentan las pruebas realizadas en el dispositivo y los resultados obtenidos del trabajo finalizado. Y por último en el **Capítulo 5** se muestran las conclusiones, así como posibles trabajos futuros.

CAPÍTULO 2

Marco Teórico

2.1. Procesamiento digital de Señales [1]

Definición 1. **Señal.-** Una señal es cualquier cantidad que aporte información acerca de algún determinado fenómeno físico. Las señales son siempre dependientes de una o mas variables: tiempo, coordenadas espaciales, etc [17].

Las señales son representadas por una regla de correspondencia, regularmente conocida como función; de esta manera $x(t)$ es una función de la variable independiente t, la cual usualmente involucra al tiempo. La gráfica de una función es una imagen que resulta cuando cada punto perteneciente a ésta es colocada en el plano, esta representación es sumamente útil para obtener visualmente el concepto de la función; sin embargo, existe un límite para realizar esta tarea.

Los *espacios de señales* son un marco de trabajo más estructurado desarrollado para manipular la señal en una forma general; en éstos, una señal es considerada un punto o entidad.

Las señales pueden ser clasificadas en analógicas o digitales, periódicas o aperiódicas, de magnitud acotada o no acotada, acotadas en tiempo o no acotadas en tiempo, de potencia o de energía, estocásticas o determinísticas.

Como un ejemplo, $I(t, x, y)$ representa una función multivariante de tiempo continuo la cual expresa la intensidad de brillo de un punto (pixel) en una pantalla de televisión negro y blanco, dependiendo del tiempo t y las coordenadas espaciales (x, y).

Junto con el concepto de señal, se revisa el concepto de sistema, definido aquí en términos de procesamiento de señales:

Definición 2. **Sistema de procesamiento de señales.-** El sistema de procesamiento de señales es un dispositivo en la forma de hardware físico o software computacional el cual es estimulado por una colección de datos ruidosos con el objetivo de extraer información acerca de una cantidad de interés.

Cualquier sistema puede ser clasificado como lineal o no-lineal, invariante en el tiempo o no, estable o inestable en un sentido especifico, etc.

Una señal bidimensional $I_s\left(x,y\right)$ representa la intensidad de brillo de un pixel de alguna imagen fija de coordenadas espaciales (x,y), el subíndice s denota la salida del sistema $\mathcal{H}$ ante la respuesta de la señal $I_i\left(x,y\right)$ la cual representa la intensidad de brillo de un pixel de alguna imagen fija con coordenadas espaciales (x,y) y el subíndice i denota la entrada esto es:

$$I_s(x,y) = \mathcal{H}\left\{I_i(x,y)\right\} \tag{2.1}$$

Definición 3. **Sistema Lineal.-** El Sistema $\mathcal{H}$ es lineal si y sólo si la respuesta del sistema ante múltiples entradas combinadas, es la misma que la combinación de las respuestas individuales del sistema:

$$I_s(x,y) = \mathcal{H}\left\{\alpha_1 I_{i1}(x,y) + \alpha_2 I_{i2}(x,y)\right\} = \alpha_1\mathcal{H}\left\{I_{i1}(x,y)\right\} + \alpha_2\mathcal{H}\left\{I_{i2}(x,y)\right\} \tag{2.2}$$

La ecuación 2.2 Se conoce como el teorema de superposición [10].

Aunque la mayoría de los sistemas realmente no son lineales, pueden hacerse algunas simplificaciones para considerarlos lineales; esto porque el análisis matemático de un sistema lineal es más simple que el análisis de un sistema no lineal; sin embargo, como la potencia computacional de los dispositivos existentes crece, los desarrollos no-lineales están ganando atención.

2.1.1. Vectores y Matrices

Una herramienta matemática que permite la manipulación eficiente de señales multidimensionales es el *vector*. Un vector es una secuencia ordenada unidimensional la cual representa algunos datos, por ejemplo, sea $\mathbf{x}$ algún vector de tamaño K, esto se representa

$$\mathbf{x} = \text{vec}\left\{x_k\right\}_{k=1}^{K} = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_K \end{bmatrix} \tag{2.3}$$

Una matriz es una representación de una secuencia bidimensional. Por ejemplo, una imagen I_m o la señal de intensidad de brillo $I(x, y)$ de las coordenadas espaciales (x, y) presentado previamente; cualquier matriz $\mathbf{A}$ de dimensiones $M \times K$ se expresa como:

$$\mathbf{A} = \text{matr}\left\{a_{m,k}\right\}_{m=1,k=1}^{M,K} = \begin{bmatrix} a_{1,1} & \cdots & a_{1,k} & \cdots & a_{1,K} \\ \vdots & \cdots & a_{m,k} & \cdots & \vdots \\ a_{M,1} & \cdots & a_{M,k} & \cdots & a_{M,K} \end{bmatrix} \tag{2.4}$$

Para cualquier matriz $\mathbf{A}, \mathbf{B}, \mathbf{C}$ y vector $\mathbf{x}$,$\mathbf{y}$, La tabla 2.1 muestra algunas propiedades de las matrices.

Operación	**Definición**	**Descripción**
Matriz	$\mathbf{A} = \text{matr}\left\{a_{m,k}\right\}_{m=1,k=1}^{M,K}$	Matriz $\mathbf{A}$ de dimensiones $M \times K$
Transpuesta	$\mathbf{A^T} = \text{matr}\left\{a_{k,m}\right\}_{k=1,m=1}^{K,M}$	Dada la matriz $\mathbf{A}$ de dimensiones $M \times K$, su transpuesta es la matriz $\mathbf{A^T}$ de dimensiones $K \times M$, con cada elemento en la posición k, m inicializando el elemento en la posición m, k de la matriz original
Conjugado complejo	$\mathbf{A}^* = \text{matr}\left\{a_{m,k}^*\right\}_{m=1,k=1}^{M,K}$	Dada una matriz $\mathbf{A}$ de $M \times K$-dimensiones, cada m, k elementos de la matriz $\mathbf{A}^*$ de $M \times K$-dimensiones es el complejo conjugado de m, k elementos de la matriz original
Continúa en la siguiente página		

Tabla 2.1 – Continúa de la página anterior

Operación	**Definición**	**Descripción**
Conjugado Transpuesto	$\left(\mathbf{A}^T\right)^* = \text{matr}\left\{a^*_{k,m}\right\}^{K,M}_{k=1,m=1}$	Obtiene una matriz $\mathbf{A}$ de $M \times K$-dimensiones, cada elemento k, m de la matriz $\left(\mathbf{A}^T\right)^*$ de $K \times M$-dimensiones es el complejo conjugado del elemento m, k de la matriz original. Si la matriz $\mathbf{A}$ es igual a su conjugado transpuesto, este es llamado *hermitiano*.
Identidad	$\mathbf{I} = \text{matr}\left\{i_{m,k}\right\}^{M,K=M}_{m=1,k=1} = 1;$ $\forall \quad m = k$	La matriz identidad es una matriz cuadrada con elementos iguales a 1 en la diagonal principal; 0 fuera de la diagonal principal.
Suma de matrices	$\mathbf{C} = \mathbf{A} + \mathbf{B}$ $= \text{matr}\left\{c_{m,k}\right\}^{M,K}_{m=1,k=1}$ $= \text{matr}\left\{a_{m,k}\right\}^{M,K}_{m=1,k=1}$ $+ \text{matr}\left\{b_{m,k}\right\}^{M,K}_{m=1,k=1}$	La suma de matrices puede ser realizada cuando se suman ambas matrices de dimensiones iguales; el resultado es la suma elemento con elemento de los sumandos
Multiplicación escalar	$\alpha\mathbf{A} = \alpha\text{matr}\left\{a_{m,k}\right\}^{M,K}_{m=1,k=1}$	Multiplicación del valor escalar por cada uno de los elementos de la matriz.

Continúa en la siguiente página

Tabla 2.1 – Continúa de la página anterior

Operación	**Definición**	**Descripción**
Multiplicación matricial	$\mathbf{C} = \mathbf{A} * \mathbf{B}$ $= \{c_{m,k}\}_{m=1,k=1}^{M,K}$ $= \sum_{j=1}^{J} a(m,j) * b(j,k)$	La multiplicación matricial entre las matrices $\mathbf{A}, \mathbf{B}$ puede ser realizada sí y solo si el número de columnas del primer operando es igual al número de renglones del segundo; cada elemento m,k del resultado de la matriz es el producto interno del m-esimo renglón del primer multiplicando y la k-esima columna del segundo multiplicando. En general, la multiplicación de matices no es conmutativa.
Producto interno de un vector	$[\mathbf{x}, \mathbf{y}] = \sum_{k=1}^{K} x_k^+ * y_k$	Para el vector **x**,**y** de K-dimensiones
Inversa	$\mathbf{A}^{-1}\mathbf{A} = \mathbf{A}\mathbf{A}^{-1} = \mathbf{I}$	Para cada matriz invertible solo existe una inversa.
Traza	Tr $\{\mathbf{A}\} = \sum_{k}^{K} a_{k,k}$	
Lema **ABCD**	$(\mathbf{A} + \mathbf{BCD})^{-1} = \mathbf{A}^{-1} + \mathbf{A}^{-1}\mathbf{B}$ $\left(\mathbf{C}^{-1} - \mathbf{DA}^{-1}\mathbf{B}\right)^{-1}\mathbf{DA}^{-1}$	

Tabla 2.1: Vectores y propiedades de matrices.

Las matrices circulantes y Toeplitz son dos matrices usadas ampliamente en el procesamiento de señales.
La matriz Toeplitz es una matriz cuadrada que tiene elementos constantes a lo largo de la diagonal principal y la subdiagonal [10]; ellas describen la transformación de un sistema lineal invariante en el tiempo unidimensional y las matrices de correlación de secuencia estacionaria. Si la matriz **A** es una matriz Toeplitz [18], estos elementos son definidos por:

$$\mathbf{A} = \{a_{m,k}\}_{m=1,k=1}^{M,K=M} \tag{2.5}$$

donde

$$\{a_{m,k}\}_{m=1;k=1}^{M,K=M} = \{a_{m-k}\}_{m=1;k=1}^{M,K=M} \tag{2.6}$$

esto es:

$$\mathbf{A} = \begin{bmatrix} a_0 & a_{-1} & a_{-2} & \cdots & a_{-M+1} \\ a_1 & a_0 & a_{-1} & \cdots & \vdots \\ a_2 & a_1 & a_0 & \cdots & \vdots \\ \vdots & \cdots & \ddots & \cdots & \vdots \\ a_{M-1} & \cdots & \cdots & a_1 & a_0 \end{bmatrix} \tag{2.7}$$

Las Matrices Circulantes son aquellas en las cuales los renglones (o columnas) son un corrimiento circular del renglón (o columna) anterior [10], i.e., si **B** es una matriz circulante

$$\mathbf{B} = \begin{bmatrix} b_0 & b_1 & b_2 & \cdots & b_{N-1} \\ b_{N-1} & b_0 & b_1 & \cdots & \vdots \\ b_{N-2} & b_{N-1} & b_0 & \cdots & \vdots \\ \vdots & \cdots & \ddots & \cdots & \vdots \\ b_1 & b_2 & \cdots & b_{N-1} & b_0 \end{bmatrix} \tag{2.8}$$

2.1.2. Espacio de señales

En esta subsección, el concepto de espacio de señales se presenta tal como en [8], [17], [19]; algunas funciones continuas usadas arbitrariamente en la presentación del tema, como $v(x)$ o

$u(t)$ serán denotados como v o u por motivos de simplicidad, una vez que han sido introducidas.

Un espacio de señales es una herramienta matemática que representa un espacio geométrico M-dimensional, donde M puede ser infinito, con M ejes mutuamente ortoganales etiquetados con los elementos del conjunto $V = \{\varphi_n\}_{m=1}^{M}$. Este conjunto de elementos forman la *base* del espacio de señales Ahora, V es un espacio de señal:

$$V = \text{span } \{\varphi_n\}_{m=1}^{M} \tag{2.9}$$

El espacio de señal esta conformado con la operación del producto interno, y este proporciona una forma de obtener un par de métricas: la norma y la distancia. Dados $v(x)$ y $w(x)$, donde x es cualquier variable independiente:

- Producto interno:
$$[v, w] = \int_X vw^* dx \tag{2.10}$$
donde X es la región donde esta definida la función.

- Norma l_2:
$$\|v\|_{l_2} = \sqrt{[v, v]} \tag{2.11}$$

- Distancia:
$$d(v, w) = \|v - w\|_{l_2} \tag{2.12}$$

Existe otro conjunto, $\{\vartheta_m\}_{m=1}^{M}$, ortonormal para el conjunto ya mencionado, el cual es conocido como base dual

$$[\vartheta_i, \varphi_k] = \delta_{i,k} \tag{2.13}$$

De esta manera, una señal continua $v(x) \in \mathbf{V}$ puede ser expresada como una combinación lineal que compone la base de la señal:

$$v(x) = \sum_{m}^{M} v_m \varphi_m(x) \tag{2.14}$$

La ecuación 2.14 llamada la serie Generalizada de Fourier. El coeficiente v_m se obtiene por el producto interno:

$$v_m = [v, \vartheta] = \int_X v(x)\vartheta^*(x)dx \tag{2.15}$$

Esto permite que cualquier señal pueda ser representada por un conjunto reducido de coeficientes (los primeros), así M sera finito; entonces el vector de coeficientes de orden M, $\mathbf{v} = \text{vec } \{v_m\}$ es la representación discreta de la señal continua $v(x)$. Como consecuencia, la señal es manipulable por computadoras y procesadores digitales de señales.

Para finalizar esta subsección, la definición del operador lineal es obtenido de [9]; esta es fundamental para proporcionar el mecanismo que permite analizar la relación entre diferentes clases de señales.

Definición 4. **Operador Lineal.-** Dados dos espacios U, V un operador lineal $\mathcal{O} : U \to V$ es cualquier función en U con valores en V, esto cumple las propiedades de linealidad.

Las series de aproximación proporcionan una poderosa herramienta que realiza la implementación del *kernel* de transformación en un sistema discreto. La siguiente discusión se presenta en [8].

Dadas dos series de aproximación

$$v(x) = \sum_n^N v_n \phi_n(x); \qquad u(y) = \sum_l^L u_l \theta_l(y); \tag{2.16}$$

con las propiedades ortonormales

$$[\phi_n, \phi_j] = \delta_n j, \quad [\theta_l, \theta_m] = \delta_l m \tag{2.17}$$

Los coeficientes de expansión son obtenidos por el producto interno, como se muestra en 2.15:

$$v_n = [v, \phi_n], u_l = [u, \theta_l] \tag{2.18}$$

Dados los operadores lineales $\mathcal{S} : V \to U$, la ecuación de observación

$$u = Sv + n \tag{2.19}$$

Representación de la ecuación de observación en forma vectorial (Ecuación 2.20).

$$\mathbf{u} = \mathbf{S}\mathbf{v} + \mathbf{n} \tag{2.20}$$

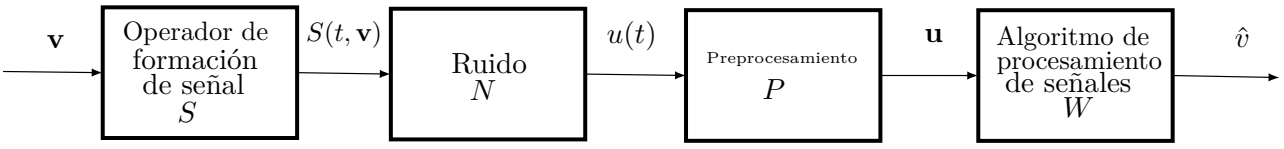

FIGURA 2.1: Proceso de transformación de información

dada la matriz de operador de formación de sistema (matriz de forma de señal, SFO) **S** definida por

$$S_{ln} = [S\phi_n, \theta_l] \tag{2.21}$$

Demostración

$$\begin{aligned} u_l =& [u, \theta_l] \\ =& [(Sv + n), \theta_l] \\ =& \left[(S(\sum v_n\phi_n) + n), \theta_l\right] \\ =& v_n [S\phi_n, \theta_l] + [n, \theta_l] \\ =& S_{ln}v_n + n_l \end{aligned}$$

donde $S_{ln} = [S\phi_n, \theta_l]; l = 1, 2, \cdots, L; n = 1, 2, \cdots, N$.

2.1.3. Fenomenología del problema de procesamiento de señales

El proceso de transformación de información para la detección o estimación del problema mostrado en la Figura 2.1, consta de: primera etapa, el vector de información original **v**, el cual es la representación discreta de la señal de información continua $v(x)$, que depende de la variable continua x; segunda etapa, transformación por el operador de formación de señal S representada por el sensor o el sistema de adquisición (y usualmente es un operador de convolución); tercera etapa, el operador de ruido N agregado para algunas estadísticas especificas (distribución, media y covarianza); cuarta etapa, proceso de muestreo a la señal ruidosa la cual permite obtener una señal discreta y posteriormente procesar por sistemas digitales; finalmente, diseño de un algoritmo para ser utilizado en la descomposición de datos y obtener una estimación $\hat{\mathbf{v}}$ de la señal original **v**, bajo algunos criterios óptimos.

Definición 5. **Problema inverso y directo.-** Dos problemas se dicen que son inversos uno del otro si la formulación de cualquiera de ellos requiere un conocimiento total o parcial del otro.

De acuerdo con esta definición, es arbitrario cual de los dos problemas se denomina problema directo y cual se refiere al problema inverso. Pero desde los puntos de vista de la física y la ingeniería, sin embargo, es costumbre distinguir entre estos dos problemas. En la practica de procesamiento de señales (SP), es costumbre encontrar la solución de uno de estos dos problemas basados en el conocimiento del fenómeno del modelo que fue estudiado anteriormente. Desde las siguientes consideraciones físicas, el problema en el cual el objetivo es predecir la salida del sistema basado en un conjunto de entradas, un modelo y un conjunto de parámetros dados, referido usualmente como el problema directo. Esto es obvio en ingeniería, tal definición está basada en la coincidencia de la dirección del flujo de la señal física en el sistema (desde la entrada hasta la salida) con la dirección que se sigue cuando se deriva una solución del problema (de nuevo desde la entrada hasta la salida).
Seria lógico referir el problema en el cual la dirección de una solución va en sentido contrario como el problema inverso.

El desarrollo anterior esta basado en el modelo siguiente:

$$u(y) = s(y) + n(y); \qquad y \in Y, \tag{2.22}$$

donde

- $u(y)$ Es la señal de datos observados dependiente de $y \in Y$,
- $s(y)$ Es la señal de transformación después el operador de formación de señal $\mathcal{S} : U(X) \to V(Y)$, definida por:

  $$s(y) = \int_X S(y,x)v(x)dx; \; x \in X, y \in Y \tag{2.23}$$

  donde $S(y,x)$ es el kernel del operador $\mathcal{S}$,
- $n(y)$ es ruido aleatorio con algunas estadísticas especificadas.

Usando la teoría presentada previamente (Ecuación (2.16)-(2.21)), (2.22) se convierte en:

$$\mathbf{u} = \mathbf{S}\mathbf{v} + \mathbf{n} \tag{2.24}$$

El objetivo del ingeniero en procesamiento de señales es el diseñar un operador $\mathbf{W}$ que cuando se aplica al vector de datos recopilados $\mathbf{u}$, se encuentra una estimación optima del vector original $\mathbf{v}$ ($\hat{\mathbf{v}}$) bajo algún sentido estadístico o no estadístico.

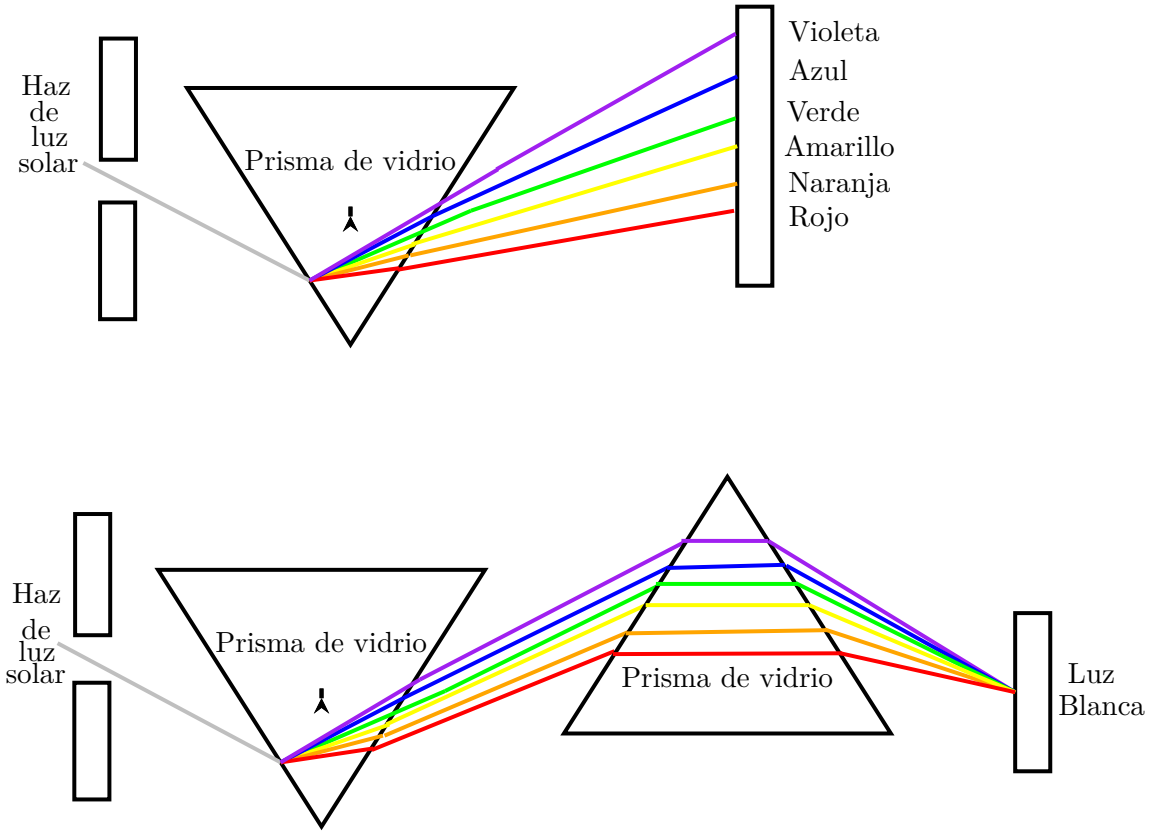

FIGURA 2.2: Análisis y síntesis de la luz blanca (luz solar) utilizando prismas de vidrio.

2.2. Análisis de Fourier

El análisis de Fourier es una herramienta matemática muy útil en el procesamiento de señales, en este caso para el procesamiento de señales bidimensionales, esta consiste en la descomposición de señales en una serie finita o infinita de senos y cosenos, para señales periódicas, en el caso de las series de Fourier, y aperiódicas de energía finita, para el caso de la transformada de Fourier, cumpliendo así una de las propiedades de los sistemas LTI.

La serie y la transformada de Fourier son muy útiles para el análisis de cualquier señal de energía finita (señales aperiódicas) ya que permite realizar una fragmentación de dicha señal para poder simplificar el análisis.

Si descomponemos una forma de onda en componentes sinusoidales de la misma forma que un prisma separa la luz blanca en diferentes colores Figura 2.2, la suma de estas componentes sinusoidales proporciona la forma de onda original. Por el contrario si falta cualquiera de estas componentes, el resultado será una señal diferente [20].

Para explicar las ecuación de la serie trigonométrica de Fourier es necesario conocer las funciones ortogonales descritas en la siguiente subsección.

2.2.1. Propiedades de las funciones seno y coseno: Funciones ortogonales.

Un conjunto de funciones $\phi_k(t)$ es ortogonal en un intervalo $a < t < b$ si para dos funciones cualesquiera $\phi_m(t)$ y $\phi_n(t)$ pertenecientes al conjunto $\phi_k(t)$ [21], cumple:

$$\int_a^b \phi_m(t)\,\phi_n(t)\,dt = \begin{cases} 0 & : \text{ para } m \neq n \\ r_n & : \text{ para } m = n \end{cases} \tag{2.25}$$

Considérese, por ejemplo, un conjunto de funciones sinusoidales; mediante el cálculo elemental se puede demostrar que

$$\int_{-\frac{T}{2}}^{\frac{T}{2}} \cos m\omega_0 t dt = 0 \quad m \neq 0 \tag{2.26}$$

$$\int_{-\frac{T}{2}}^{\frac{T}{2}} \sin m\omega_0 t dt = 0 \quad \forall\, m \tag{2.27}$$

$$\int_{-\frac{T}{2}}^{\frac{T}{2}} \cos m\omega_0 t \cos n\omega_0 t dt = \begin{cases} 0 & : \text{ para } m \neq n \\ \frac{T}{2} & : \text{ para } m = n \neq 0 \end{cases} \tag{2.28}$$

$$\int_{-\frac{T}{2}}^{\frac{T}{2}} \sin m\omega_0 t \sin n\omega_0 t dt = \begin{cases} 0 & : \text{ para } m \neq n \\ \frac{T}{2} & : \text{ para } m = n \neq 0 \end{cases} \tag{2.29}$$

$$\int_{-\frac{T}{2}}^{\frac{T}{2}} \sin m\omega_0 t \cos n\omega_0 t dt = 0 \quad \forall\, m, n \tag{2.30}$$

$$donde \quad \omega_0 = \frac{2\pi}{T} \tag{2.31}$$

Estas relaciones demuestran que las funciones

$$\{1, \cos\omega_0 t, \cos 2\omega_0 t, ..., \cos n\omega_0 t, ..., \sin\omega_0 t, \sin 2\omega_0 t, ..., \sin n\omega_0 t, ...\}$$

forman un conjunto de funciones ortogonales en el intervalo $-\frac{T}{2} < t < \frac{T}{2}$.

Ecuación de la serie trigonométrica de Fourier

$$f(t) = \frac{1}{2}a_0 + \sum_{n=1}^{\infty} a_n \cos n\omega_0 t + b_n \sin n\omega_0 t \tag{2.32}$$

Utilizando las relaciones de ortogonalidad se puede evaluar los coeficientes a_n y b_n para la serie trigonométrica de Fourier [21]

$$a_0 = \frac{2}{T}\int_{-\frac{T}{2}}^{\frac{T}{2}} f(t)\, dt \tag{2.33}$$

$$a_n = \frac{2}{T}\int_{-\frac{T}{2}}^{\frac{T}{2}} f(t) \cos n\omega_0 t dt \quad n = 1, 2, 3... \tag{2.34}$$

$$b_n = \frac{2}{T}\int_{-\frac{T}{2}}^{\frac{T}{2}} f(t) \sin n\omega_0 t dt \quad n = 1, 2... \tag{2.35}$$

Serie exponencial de Fourier

$$f(t) = \sum_{n=-\infty}^{\infty} c_n e^{jn\omega_0 t} \tag{2.36}$$

$$c_n = \frac{1}{T}\int_{-\frac{T}{2}}^{\frac{T}{2}} f(t)\, e^{-j\omega_0 k t} dt \tag{2.37}$$

2.2.2. Análisis de Señales en el dominio de la frecuencia - Transformada de Fourier [2]

Anteriormente se expuso la representación de cualquier señal periódica mediante la serie de Fourier en un intervalo de tiempo. Esta representación es válida en todo el tiempo, desde $-\infty$ hasta $+\infty$, si la señal es periódica. Se extiende ahora esta representación para el caso de las señales aperiódicas. Supóngase que ahora $f(t)$ es una señal aperiódica, es decir un solo pulso que no se vuelve a repetir (Figura 2.3 a)) y se desea representar a $f(t)$ mediante la serie exponencial de Fourier de tal manera que la representación sea válida en todo el tiempo desde $-\infty$ hasta $+\infty$. Para esto se hace el siguiente razonamiento: con base en el pulso $f_A(t)$ de la Figura 2.3 a), se construye la señal periódica $f(t)$ de la Figura 2.3 b). Se sabe que la serie exponencial de Fourier que representa a $f(t)$ es válida en todo el eje de tiempo desde $-\infty$ hasta $+\infty$ y que esta validez es independiente del valor de T. Así, al incrementar el valor del periodo T los pulsos de

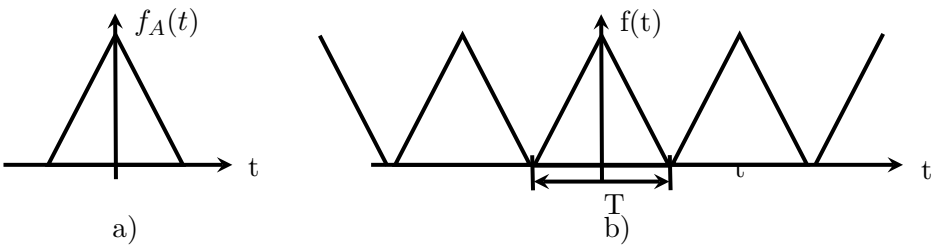

FIGURA 2.3: Señal aperiódica

$f(t)$ se separan cada vez más y el límite, cuando $T \to \infty$ la señal periódica es también la señal aperiódica $f_A(t)$; es decir:

$$f_A(t) = \lim_{T\to\infty} f(t) \tag{2.38}$$

Esto es, que en el límite las dos señales son iguales y por lo tanto tienen la misma serie de Fourier, válida en todo el eje real del tiempo. En esta forma, mediante el proceso de límite haciendo que $T \to \infty$, es posible pasar de la señal periódica a la aperiódica. Resta ahora estudiar como se modifica la expresión matemática del espectro, qué forma adoptan los coeficientes F_n y cómo se transforma la serie exponencial de Fourier con este proceso de límite. Para esto se utilizan las ecuaciones de la serie exponencial de Fourier 2.36 y 2.37.

Es importante recordar que $\omega_0 = \frac{2\pi}{T}$ es el esparcimiento de las componentes de frecuencia en el espectro de $f(t)$. Las ecuaciones anteriores también se pueden escribir como:

$$f(t) = \frac{1}{T} \sum_{n=-\infty}^{\infty} c_n e^{jn\omega_0 t} \tag{2.39}$$

$$c_n = \int_{-\frac{T}{2}}^{\frac{T}{2}} f(t)\, e^{-j\omega_0 kt} dt \tag{2.40}$$

y como $T = \frac{2\pi}{\omega_0}$ de 2.39

$$f(t) = \frac{1}{2\pi} \sum_{n=-\infty}^{\infty} c_n e^{jn\omega_0 t} \omega_0 \tag{2.41}$$

Cuando $T \to \infty$, $\omega_0 \to 0$ por lo que se le puede representar como $\Delta\omega$. Es decir a medida que T aumenta aparecen más armónicas en el espectro y, en el límite cuando $T \to \infty$, el espectro c_n

se convierte en función continua de $n\omega_0$ que ahora se transforma en variable continua, es decir, $n\omega_0 \to \omega$ y $c_n \to F(\omega)$. Así,

$$F(\omega) = \lim_{T\to\infty} c_n \tag{2.42}$$

En resumen, cuando $T \to \infty$
$n\omega_0 \to \omega$
$c_n \to F(\omega)$
$f(t) \to f_A(t)$

Así, de 2.38 y 2.41

$$f_A(t) = \lim_{T\to\infty} \frac{1}{2\pi} \sum_{n=-\infty}^{\infty} c_n e^{jn\omega_0 t} \Delta\omega \tag{2.43}$$

El segundo miembro de esta ecuación constituye la definición de la integral de Riemann. Por lo tanto,

$$f_A(t) = \frac{1}{2\pi} \int_{-\infty}^{\infty} F(\omega) e^{j\omega t} d\omega \tag{2.44}$$

De de las ecuaciones 2.40 y 2.43 obtenemos

$$F(\omega) = \lim_{T\to\infty} \int_{-\frac{T}{2}}^{\frac{T}{2}} f(t) e^{-jn\omega_0 t} dt \tag{2.45}$$

$$F(\omega) = \int_{-\infty}^{\infty} f_A(t) e^{-j\omega t} dt \tag{2.46}$$

La ecuación 2.44 constituye la representación de $f_A(t)$, la señal aperiódica, en términos de señales exponenciales en todo el intervalo de tiempo $(-\infty, +\infty)$. Como se ve, se trata ahora de una suma continua (integral) de exponenciales complejas de frecuencia ω. La amplitud de las componentes es proporcional a $F(\omega)$, por lo que $F(\omega)$ constituye el espectro de frecuencia de $f_A(t)$; se le llama función de densidad espectral o simplemente el espectro de $f(t)$. $F(\omega)$ se calcula con la ecuación 2.46 y, es ahora una función continua de ω; hay un número infinito de armónicas en el espectro de $f_A(t)$. $F(\omega)$ se conoce, matemáticamente, como transformada de Fourier de $f_A(t)$. La ecuación 2.44 es la transformada inversa de Fourier de $F(\omega)$, es decir $f_A(t)$ es la transformada inversa de Fourier de $F(\omega)$. Las ecuaciones 2.44 y 2.46 se conocen como par de transformadas de Fourier; simbólicamente se les representa como:

FIGURA 2.4: Sistema Lineal Invariante en el Tiempo.

$$F(\omega) = \mathcal{F}[f_A(t)] \quad y \quad f_A(t) = \mathcal{F}^{-1}[F(\omega)] \tag{2.47}$$

Así,

$$\mathcal{F}[f(t)] = F(\omega) = \int_{-\infty}^{\infty} f_A(t)\, e^{-j\omega t} dt \tag{2.48}$$

$$\mathcal{F}^{-1}[F(\omega)] = f_A(t) = \frac{1}{2\pi} \int_{-\infty}^{\infty} F(\omega)\, e^{j\omega t} d\omega \tag{2.49}$$

La transformada directa, ecuación 2.48, establece que conociendo $f_A(t)$ basta con someterla a la operación indicada (integral de Fourier) para encontrar su espectro $F(\omega)$; es decir, especificada $f_A(t)$ en el tiempo, se puede calcular su representación en el dominio de la frecuencia. Por el contrario, la ecuación 2.49 establece que si lo que se especifica es el espectro, operándolo como lo indica el segundo miembro de 2.49 es posible encontrar la función del tiempo correspondiente.

2.3. Convolución

La ecuación 2.50 se conoce comúnmente como integral de convolución [22]. De este modo, tenemos como resultado fundamental que la salida de cualquier sistema *LTI* de tiempo continuo es la convolución de la entrada $x(t)$ con la respuesta al impulso $h(t)$ del sistema. La figura 2.4 ilustra la definición de la respuesta al impulso $h(t)$ y la relación de la ecuación 2.50

$$s * h = \int_{-\infty}^{\infty} s(\tau)\, h(t - \tau)\, d\tau \tag{2.50}$$

La convolución indica que un sistema LTI de tiempo continuo se caracteriza completamente por su respuesta al impulso.

Propiedades de la convolución:

- Conmutativa
- Asociativa
- Distributiva

[21] La función 2.52 se conoce como la función de correlación entre las funciones $f_1(t)$ y $f_2(t)$.

$$R_{12}(\tau) = \int_{-\infty}^{\infty} f_1(t) f_2(t-\tau)\, dt \tag{2.51}$$

$$R_{21}(\tau) = \int_{-\infty}^{\infty} f_2(t) f_1(t-\tau)\, dt \tag{2.52}$$

2.4. Correlación

La función de correlación $R_{12}(\tau)$ y $R_{21}(\tau)$ suministra una medida de la similitud o independencia entre las funciones $f_1(t)$ y $f_2(t)$, en función del parámetro τ (el desplazamiento de una función con respecto a la otra). Si la función de correlación es cero para todo valor de τ, entonces se dice que las funciones no están correlacionadas.
Si $f_1(t)$ y $f_2(t)$ son idénticas, entonces la función de correlación $R_{12}(\tau)$ se denomina función de autocorrelación de $f_1(t)$.

$$R_{11}(\tau) = \int_{-\infty}^{\infty} f_1(t) f_1(t-\tau)\, dt \tag{2.53}$$

Ahora

$$R_{12}(\tau) = \int_{-\infty}^{\infty} f_1(t) f_2(t-\tau)\, dt = \int_{-\infty}^{\infty} f_1(t+\tau) f_2(t)\, dt \tag{2.54}$$

Según la ecuación 2.54 , se observa que la correlación es indiferente si se desplaza la función $f_1(t)$ en una cantidad τ en la dirección negativa, o si se desplaza la función $f_2(t)$ en la misma cantidad, en la dirección positiva.

2.5. Probabilidad y estadística [3]

2.5.1. Experimentos aleatorios

Los experimentos en la ciencia e ingeniería son importantes ya que, en algunas pruebas, no se es capaz de determinar o controlar el valor exacto de las variables así que el resultado variará de un experimento a otro, aunque la mayoría de las condiciones sean las mismas. Estos experimentos se conocen como *aleatorios*.

2.5.2. Espacio de muestreo

Un conjunto S que consiste de todas las salidas posibles de un experimento aleatorio es llamado *espacio de muestreo*, y cada salida es llamado punto muestra (en el espacio de muestreo S). Frecuentemente habrá más de un espacio de muestreo que pueda describir los resultados de un experimento, pero normalmente sólo hay uno que proporcionará la mayor información.

A continuación se describen algunos tipos de espacios:

- **Espacio de muestreo finito.** Contiene un número finito de puntos.
- **Espacio de muestreo infinito contable.** Contiene una gran cantidad de puntos como los números naturales $1, 2, 3, ...$
- **Espacio de muestreo infinito no contable.** Contiene muchos puntos en un intervalo del eje x como $0 <= x <= 1$. Otro ejemplo es la función del espacio de señales (función del espacio, ejemplo: la realización de algún proceso aleatorio)

2.5.3. Evento

Un evento $\mathcal{A}$ es un subconjunto del espacio de muestreo S, este es un conjunto de salidas posibles. Si las salidas del experimento son un elemento de $\mathcal{A}$, se dice que el evento $\mathcal{A}$ ha ocurrido.

Un evento *simple* o *elemental*.- Es un evento que consiste en puntos individuales de S.

Un evento *verdadero* o *evento cierto*.- Es el espacio de muestreo S ya que siempre debe ocurrir un elemento de S.

Un *evento imposible* o *nulo*.- Es con frecuencia llamado conjunto vacío $\varnothing$ porque un elemento de $\varnothing$ no puede ocurrir.

Utilizando las operaciones sobre conjuntos, se pueden definir otros eventos en S. Por ejemplo, si $\mathcal{A}$ y $\mathcal{B}$ son eventos en S, las operaciones suma, producto y complemento producen los siguientes eventos:

1. Suma lógica o unión de $\mathcal{A}$ y $\mathcal{B}$: por definición, $\mathcal{A} \cup \mathcal{B}$ es el evento: cualquiera $\mathcal{A}$ o $\mathcal{B}$ o ambos.
2. Producto lógico o intersección de $\mathcal{A}$ y $\mathcal{B}$: por definición, $\mathcal{A} \cap \mathcal{B}$ es el evento: ambos $\mathcal{A}$ y $\mathcal{B}$.
3. Complemento lógico o complemento de $\mathcal{A}$: por definición, es el evento: no $\mathcal{A}$.
4. Diferencia lógica: Por definición, $\mathcal{A} - \mathcal{B} = \mathcal{A} \cap \overline{\mathcal{B}}$ es el evento: $\mathcal{A}$ pero no $\mathcal{B}$.
5. Eventos mutuamente excluyentes: son dos eventos $\mathcal{A}$ y $\mathcal{B}$ para los cuales sus conjuntos son disjuntos en $\mathcal{S}$: $\mathcal{A} \cap \mathcal{B} = \{\varnothing\}$. Esto significa que no pueden ocurrir ambos

En cualquier experimento aleatorio hay incertidumbre en el evento en particular ocurrirá o no. Como medida de la posibilidad, o *probabilidad*, con la cual se puede esperar que el evento ocurra, es conveniente asignar un número entre 0 y 1.

Si se esta seguro que el evento ocurrirá, se dice que es una probabilidad de 100 % o 1.

Si se esta seguro que el evento no ocurrirá, se dice que es una probabilidad de cero.

Si, por ejemplo, la probabilidad es 1/4, sera cierto que hay un 25 % de posibilidad de que esto ocurra y un 75 % de posibilidad de que no ocurrirá. Equivalentemente, se dice que las probabilidades de que esto ocurra son 75 % a 25 %, o 3 a 1.

Estos son dos enfoques de estimación de probabilidad de un evento:

1. **Enfoque clásico (axioma)**. Si un evento puede ocurrir en n diferentes salidas de un número total de N posibles maneras, y todas estas son igualmente probables, entonces la probabilidad del evento es considerado n/N.

2. **Enfoque frecuencial**. Si después de N repeticiones de un experimento, donde N es muy grande, un evento se observa que ocurre n de estos, entonces la probabilidad del evento es n/N.

Ambos enfoques tienen inconvenientes fundamentales, porque la expresión "igualmente probables" con un "número grande" involucra incertidumbre.

2.5.4. El axioma de probabilidad

Suponiendo que se tiene un espacio de muestreo S discreto, todos los subconjuntos corresponden a un evento, pero si S no es discreto, solo los subconjuntos especiales corresponden a un evento (medible). A cada evento A en la clase de evento C, el número real asociado es $Pr(A)$. Entonces Pr es llamada *función de probabilidad* y la probabilidad de un evento A es $Pr(A)$, si los siguientes axiomas se cumplen:

1. Para cada evento $\mathcal{A}$ en la clase C, $Pr(\mathcal{A}) \geq 0$
2. Para el evento seguro $\mathcal{S}$ en la clase C, $Pr(S) = 1$
3. Para eventos mutuamente excluyentes $\mathcal{A}_1$, $\mathcal{A}_2$,... en la clase C, $Pr(\mathcal{A}_1 \cup \mathcal{A}_2 \cup \cdots) = Pr(\mathcal{A}_1) + Pr(\mathcal{A}_2) + ...$

De estos axiomas, se extraen varios teoremas:

1. Para algún evento $\mathcal{A}$
   $0 \leq Pr(\mathcal{A}) \leq 1$
2. Si $\mathcal{A}_1 \subset \mathcal{A}_2$ entonces $Pr(\mathcal{A}_1) \leq Pr(\mathcal{A}_2)$
3. $Pr(\varnothing) = 0$
4. Si $\bar{\mathcal{A}}$ es el complemento de $\mathcal{A}$, $Pr(\bar{\mathcal{A}}) = 1 - Pr(\mathcal{A})$
5. Si $\mathcal{A} = \mathcal{A}_1 \cup \mathcal{A}_2 \cup \cdots \cup \mathcal{A}_N$ con $\mathcal{A}_N$ exclusivos, entonces
   $Pr(\mathcal{A}) = Pr(\mathcal{A}_1) + Pr(\mathcal{A}_2) + \cdots + Pr(\mathcal{A}_N)$
6. Si $\mathcal{A}$ y $\mathcal{B}$ son cualquier evento, $Pr(\mathcal{A} \cup \mathcal{B}) = Pr(\mathcal{A}) + Pr(\mathcal{B}) - Pr(\mathcal{A} \cap \mathcal{B})$
7. Para cualquier evento $\mathcal{A}$ y $\mathcal{B}$, $Pr(\mathcal{A}) = Pr(\mathcal{A} \cap \mathcal{B}) + Pr(\mathcal{A} \cap \bar{\mathcal{B}})$

Todos estos teoremas pueden ser representados fácilmente usando diagramas de Venn. En la Figura 2.5 se muestra un ejemplo de este tipo de diagramas.

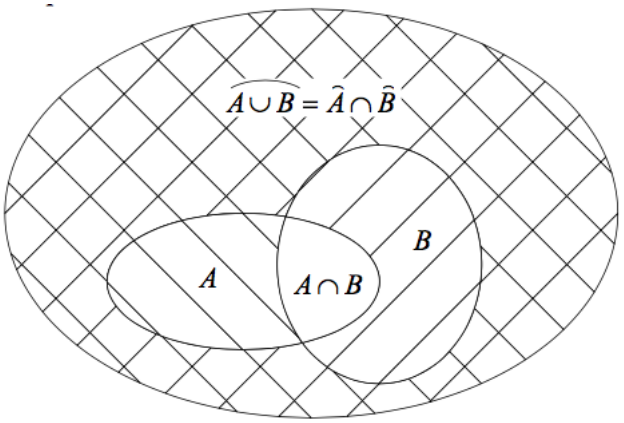

FIGURA 2.5: Diagrama de Venn.

2.5.5. Probabilidad Condicional

Sean dos eventos $\mathcal{A}$ y $\mathcal{B}$ tal que $Pr(\mathcal{A}) \geq 0$. $Pr(\mathcal{B}|\mathcal{A})$ denota la probabilidad de $\mathcal{B}$ dado que $\mathcal{A}$ ha ocurrido. Puesto que $\mathcal{A}$ ha ocurrido, ahora éste es el nuevo espacio muestral, remplazando a $\mathcal{S}$. En este nuevo espacio muestral, por definición $Pr(\mathcal{A} \cap \mathcal{B}) = Pr(\mathcal{A})Pr(\mathcal{B}|\mathcal{A})$, lo que da la definición de probabilidad condicional:

$$Pr(\mathcal{B}|\mathcal{A}) = \frac{Pr(\mathcal{A} \cap \mathcal{B})}{Pr(\mathcal{A})}$$

CAPÍTULO 3

Descripción del marco de hardware y software utilizado.

3.1. OpenCV

OpenCV (Open Source Computer Version) [23], es un conjunto de bibliotecas de algoritmos escritos en C/C++ para realizar tareas de procesamiento de imagen y visión computacional.

Estos algoritmos se utilizan para detectar rostros, detectar objetos, seguir la trayectoria de movimiento, extraer modelos 3D de objetos, entre otras aplicaciones.

Trabaja bajo las plataformas:

- Windows
- Linux
- Android
- MacOS

Tiene interfaz con los lenguajes de programación:

- C++
- C
- Python
- Java

- Matlab

Utiliza los compiladores:

- Microsoft visual C++
- MinGW
- GCC

en este trabajo se utilizará bajo el entorno de desarrollo de Code::Blocks.

Ya que OpenCV es utilizado en distintas plataformas se optó por la plataforma de Linux para un dispositivo Cyclone V con núcleo ARM Cortex A9. Esto permite aprovechar algunas de las ventajas que un dispositivo embebido nos ofrece como la versatilidad para su configuración, la posibilidad de reducir el tiempo de proceso, etc.

3.1.1. Estructuras y operaciones básicas en OpenCV [4]

3.1.1.1. Tipo de datos

OpenCV utiliza algunos tipos de datos conocidos en C, llamados datos primitivos de OpenCV que son: unsigned char, bool, signed char , unsigned short, signed short, int, float, double, o tuplas (secuencia ordenada de elementos) de valores de uno de estos tipos, donde todos los valores de la tupla tienen el mismo tipo.

OpenCV cuenta con una amplia cantidad de estructuras básicas para el procesamiento de imágenes y visión computacional, en seguida se describen algunas de las estructuras utilizadas en el trabajo.

- Clase `Mat`: La clase `Mat` representa una matriz numérica de un solo canal o multi-canal. Puede usarse para almacenar vectores y matrices reales o complejos, imágenes en escala de grises o en color, campos vectoriales, histogramas entre otros.
- Clase `Size_`: La clase de modelo `Size_` sirve para especificar y proporcionar el tamaño de una imagen o rectángulo. La clase incluye dos miembros llamados ancho y alto (cantidad de columnas y la cantidad de renglones).
- Estructura `Mat::clone`: Crea una copia completa de la matriz y los datos subyacentes.
- Clase `Vec`: Modelo de clase para vectores numéricos cortos, un caso especial de matriz.

3.1.1.2. Constructores de la clase Mat

La clase `Mat` cuenta con una gran cantidad de constructores de matrices, a continuación se presentan algunas de ellas. [4]

```
Mat::Mat
Mat::Mat()
Mat::Mat(int renglones, int columnas, tipo de dato [bits-tipo de dato-canales])
Mat::Mat(tamaño, tipo de dato [bits-tipo de dato-canales])
Mat::Mat(int renglones, int columnas, tipo de dato [bits-tipo de dato-canales],
        void* Puntero a los datos de usuario(un arreglo por ejemplo))
```

Para utilizar los constructores de la clase `Mat` se manejaron los tipos de datos `32F` (32 bits de tipo flotante) y `8U` (8 bits unsigned char), se sabe que una imagen a color esta constituida por tres canales, que corresponden a los colores rojo, verde y azul (RGB). En el trabajo se lee una imagen a color y se convierte a escala de grises, donde existe solo un canal por ser monocromático por lo que se asigna `C1`.

3.1.1.3. Operaciones básicas

A continuación se muestra una lista de algunas operaciones matriciales con las que cuenta OpenCV, estas pueden ser combinadas con expresiones complejas (A y B representan matrices (de clase `Mat`), s es un escalar (scalar), α para un escalar real (doble)):

- **Adición, sustracción y negación:**

  `A+B, A-B, A+s, A-s, s+A, s-A, -A+`

- **Multiplicación y división por elemento:**

  `A.mul(B), A/B, alfa/A`

- **Multiplicación matricial:** `A*B`
- **Transpuesta:** `A.t()`
- **Producto cruz y producto punto:**

  `A.cross(B) y A.dot(B)`

- **Operaciones lógicas bit a bit:**

  `A opLog B, A opLog s, s opLog A, ~A,`

  donde opLog es un operador logico: `&, |, ^`

Las operaciones básicas utilizadas en este trabajo fueron la multiplicación, la inversa y la transpuesta de una matriz usando los operadores de la siguiente manera

```
Mat matriz = matriz.t()
```

```
matriz A * matriz B
```

3.1.1.4. Operaciones en matrices [5]

OpenCV cuenta con una gran cantidad de funciones que operan en las matrices, a continuación se mencionan algunas de las operaciones que maneja la biblioteca; algunas de ellas se utilizan en el trabajo.

- Valor absoluto `abs`: Calcula el valor absoluto de cada elemento de la matriz. La matriz de salida tiene el mismo tamaño y el mismo tipo que la entrada

  ```
  Mat Matriz-de-salida abs(Matriz)
  ```

- Inversa de una matriz `invert`: Encuentra la inversa o pseudo-inversa de una matriz. La función invierte la matriz de entrada y almacena el resultado en la matriz de salida. Cuando la matriz de entrada es singular o no cuadrada, la función calcula la matriz pseudo-inversa (la matriz de salida) para que la norma ($\mathbf{src} * \mathbf{dst} - \mathbf{I}$) sea mínima, donde $\mathbf{I}$ es una matriz de identidad.

  ```
  double invert(Matriz-de-entrada,Regreso-de-la-inversa-de-la-matriz)
  ```

- Rotación `flip`: Rota una matriz 2D alrededor de los ejes verticales, horizontales o ambos.

  ```
  void flip(Matriz-de-Entrada,Matriz-de-Salida, int flipCode)
  ```

  FlipCode es un indicador para especificar cómo invertir la matriz;

- `flipCode=0` significa voltear alrededor del eje x
- `flipCode>0` el valor positivo significa voltear alrededor del eje y y
- `flipCode<0` el valor negativo significa voltear alrededor de ambos ejes.

- Números aleatorios normalmente distribuidos `randn`: rellena la matriz de salida con números aleatorios normalmente distribuidos con el vector de media especificada y la matriz de desviación estándar. Los números aleatorios generados se recortan para ajustarse al rango de valores del tipo de datos de la matriz de salida.

```
void randn(Matriz-de-salida,media-de-los-numeros-aleatorios-generados,
            desviacion-estandar-de-los-numeros-aleatorios-generados)
```

3.1.1.5. Atributos públicos

Al tener elementos de una clase, no se tiene la posibilidad de acceder a todos los datos, por lo que `Mat` tiene los siguientes datos de acceso público:

```
MatAllocator *  allocator asignador personalizado
      uchar *  data   puntero a los datos
const uchar *  dataend
const uchar *  datalimit
const uchar *  datastart  campos auxiliares utilizados en
locateROI y adjustROI
          int  dims  dimensiones de la matriz >= 2
          int  flags
          int  rows  numero de renglones cuando la matriz
           es de mas de 2 dimensione
          int  cols  numero de columnas cuando la matriz
           es de mas de 2 dimensione
      MatSize  size
      MatStep  step
    UMatData *  u interacción con UMat
```

Estas propiedades permiten obtener las características de la matriz, un ejemplo del uso de `rows` y `cols` para obtener la cantidad de renglones y columnas de la matriz:

```
int matriz.rows();
int matriz.cols();
```

o para obtener el tamaño de una matriz se utiliza la propiedad `size` de la siguiente manera:

```
regresa-el-tamaño-de-matriz   matriz.size();
```

3.1.1.6. Cargar y mostrar una imagen [6]

Para cargar y mostrar una imagen OpenCV cuenta con una serie de funciones específicas para cada una de estas tareas, para cargar y mostrar una imagen.

- La función `imread`, carga la imagen del nombre especificado por el primer argumento. El segundo argumento especifica el formato de la imagen. Esto podría ser:

`CV\_LOAD\_IMAGE\_UNCHANGED (<0).`	Carga la imagen tal como está
`CV\_LOAD\_IMAGE\_GRAYSCALE (0).`	Carga la imagen en escala de grises.
`CV\_LOAD\_IMAGE\_COLOR (> 0).`	Carga la imagen en el formato RGB.

  ```
  cv::imread ("Dirección-donde-se-encuentra-la-imagen-capturada",
              Especificación-de-color-(B/N o RGB))
  ```

- Función `namedWindow` crea una ventana OpenCV, para ello, debe especificarse el nombre y cómo debe manejar el cambio de la imagen, tamaño:

  - `WINDOW\_AUTOSIZE` es el único admitido, el tamaño de la ventana ocupará el tamaño de la imagen que muestra y no se permite cambiar el tamaño.
  - `WINDOW\_NORMAL` La imagen se redimensionará según el tamaño de la ventana actual. Mediante el uso de operador `|`, también necesita especificar si desea que la imagen mantenga su relación de aspecto `(WINDOW\_KEEPRATIO)` o no `(WINDOW\_FREERATIO)`.

  ```
  namedWindow ("Nombre-de-la-ventana", tamaño-de-la-ventana)
  ```

- La función `imshow`, para actualizar el contenido de la ventana OpenCV con una nueva imagen, donde se especifica el nombre de la ventana OpenCV para actualizar y mostrar la imagen que se usará durante la operación

  ```
  imshow ( "Nombre-de-la-ventana-previamente-creada", Imagen-que-se-desea-ver)
  ```

3.2. Linux [7]

Linux es un sistema operativo, es un software que administra todos los recursos de hardware asociado con una laptop, el escritorio o un sistema embebido. Simplemente el sistema operativo maneja la comunicación entre el software y el hardware.

El sistema operativo se compone de las siguientes partes:

- **Cargador de arranque:** Es el software que administra el proceso de arranque de la computadora.
- **Kernel (núcleo):** El kernel es el núcleo del sistema y administrador del CPU, memoria, y dispositivos periféricos.
- **Daemons:** Procesos en segundo plano como el sonido,impresoras, etc.
- **Shell:** Es el interprete de comandos que permite acceder a los servicios del sistema operativo.
- **Servidor de gráfico:** Es el subsistema que muestra los gráficos en la computadora.
- **Entorno de escritorio:** Es el componente del sistema donde el usuario interactúa con el sitema de una forma sencilla.
- **Aplicaciones:** Son programas utilizados como herramientas que realizan una tarea específica. Son sencillas de instalar y en Linux se ofrecen miles de ellas para una mejor experiencia para el usuario.

Algunas de las razones por las que se utiliza Linux

- Linux es un software distribuido libremente
- En la actualidad su instalación es sencilla
- Es estable: no tiene problemas por malware, virus o desaceleraciones aleatorias de la computadora.

Las versiones de Linux son llamadas distribuciones y puede encontrarse la indicada según las necesidades del usuario. Estas distribuciones pueden ser descargadas libremente y ser grabadas en un disco o guardadas en una memoria USB, para instalarlo (en las maquinas que se deseen).

Algunas de las distribuciones mas conocidas de Linux son:

- Ubuntu Linux
- Linux Mint
- Arch Linux
- Deepin
- Fedora
- Debian
- openSUSE

3.3. Introducción al Sistema de Procesador en Hardware (HPS).

El dispositivo Cyclone V es un sistema en chip (SoC) [24], compuesto por dos secciones distintas: la sección donde se encuentra el HPS (Hard Processor System) y una sección donde se encuentra un FPGA (Field Programmable Gate Array), la primera puede constar de un solo núcleo o de dos núcleos ARM Cortex A9. La figura 3.1 muestra un esquema de este dispositivo.

La arquitectura HPS integra un amplio conjunto de periféricos que reducen el tamaño de la tarjeta e incrementan el rendimiento del sistema.

El dispositivo SoC cuenta con terminales E/S (de entradas y salidas) exclusivos de la sección de entramado de la FPGA.

El HPS consta de los siguientes tipos de módulos:

- Unidad de microprocesador (subsistema de unidad de microprocesos MPU) con procesadores ARM Cortex-A9 MPCore® ya sean simples o dobles
- Controlador de memoria flash.
- Subsistema del controlador SDRAM.
- Interconexión del sistema.
- Memorias en chip.
- Periféricos de soporte.

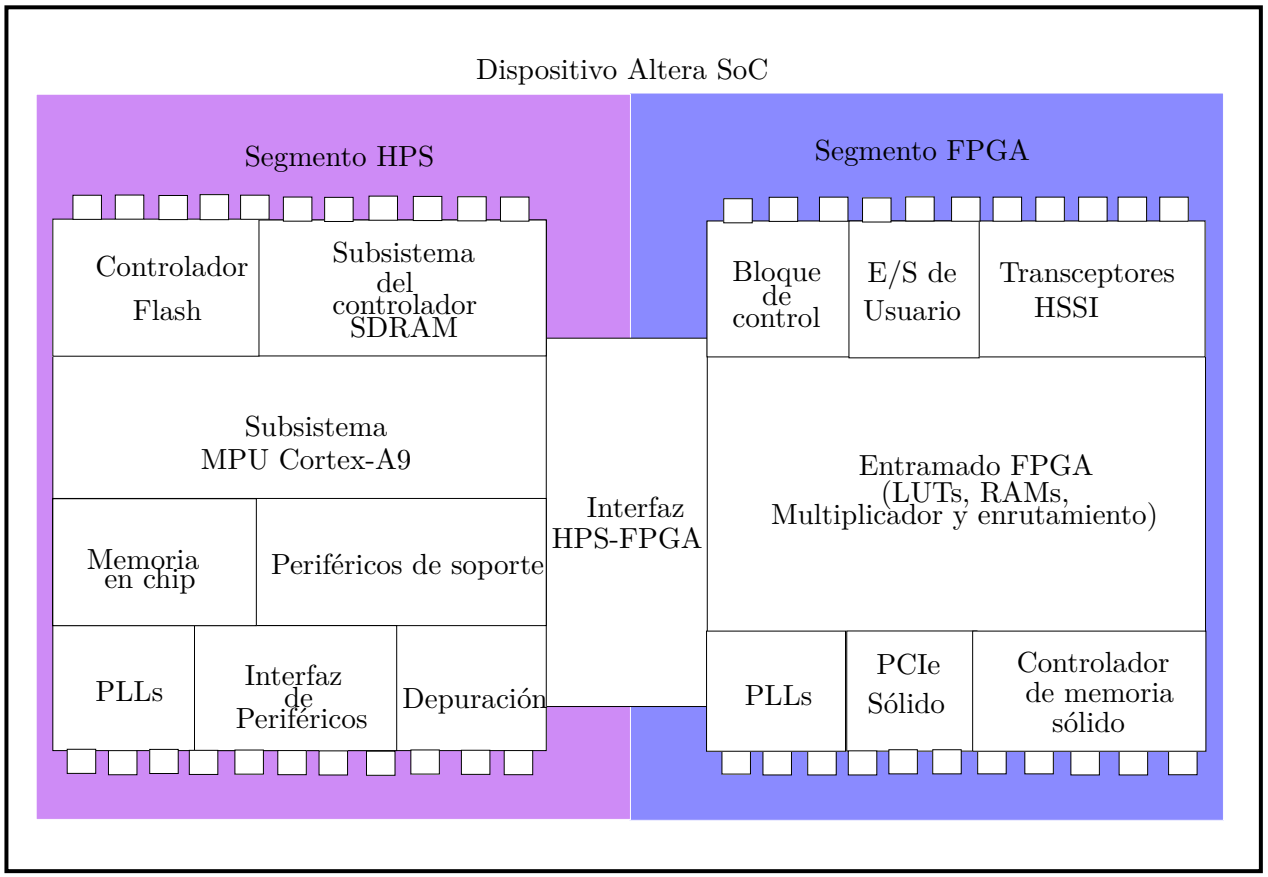

FIGURA 3.1: Dispositivo Altera SoC

- Interfaz de periféricos.
- Componentes de depuración
- Lazos de bloqueo de fase (PLL).

El HPS incorpora Propiedad Intelectual (IP) de terceros de varios proveedores.
El HPS de procesador doble soporta multiprocesos simétricos (SMP) y multiprocesos asimétricos (AMP).

El segmento de la FPGA del dispositivo esta compuesto por:

- Sección de entramado de la FPGA.
- Bloque de control (CB).
- PLLs.

- Transceptores de Interfaz serie de alta velocidad (HSSI), (dependiendo del dispositivo).
- Controladores físicos PCI Express (PCI-e).
- Controladores de memoria física.

La sección del HPS y la sección de la FPGA se comunican entre sí a través de interfaces de bus que unen los dos segmentos.

Al encender el dispositivo, el HPS puede arrancar desde múltiples fuentes, incluyendo la memoria flash externa. La FPGA puede ser configurada a través de la sección del HPS o por un dispositivo soportado externamente.

El HPS y el segmento de FPGA del dispositivo tienen sus propios pines, los cuales no se comparten entre el HPS y el entramado de la FPGA, pero sí se puede acceder a recursos del FPGA desde el HPS, estos pines de E/S están configurados por software de arranque ejecutados por el subsistema MPU dentro del segmento del HPS. El software que es ejecutado en el HPS tiene acceso a los registros de control del administrador del sistema para asignar los pines de E/S desde el segmento del HPS a los módulos disponibles de este segmento. Los pines de E/S de la FPGA se configuran mediante una imagen de configuración a través del HPS o cualquier fuente externa soportada por el dispositivo.

El subsistema de la MPU puede arrancar desde la memoria flash del dispositivo conectado a los pines del HPS. Cuando el segmento de la FPGA es configurado por una fuente externa, el subsistema de la MPU puede arrancar desde el dispositivo de la memoria flash disponible en parte del segmento de la FPGA del dispositivo.

El HPS y el segmento de la FPGA del dispositivo pueden tener suministros de energía externa separada e independientemente del encendido. Se puede encender el HPS sin encender el segmento de la FPGA del dispositivo. Sin embargo para encender la FPGA, el HPS debe ser activado o encendido al mismo tiempo que el segmento de la FPGA. Se puede apagar el segmento de la FPGA del dispositivo mientas el HPS esta encendido.

3.3.1. Caracteristicas del HPS

Los modulos principales son:

- MPU subsistema con un único núcleo o doble núcleo ARM Cortex-A9 MPCore processor.
- Controlador de acceso directo a memoria de propósito general (DMA).
- Dos controladores USB 2.0 activos (OTG).
- Controlador flash NAND.
- Controlador flash del cuadrante SPI.
- Controlador de seguridad digital/tarjeta multimedia (SD/MMC)
- Dos Controladores maestros de interfaz periférica en serie (SPI).
- Dos controladores esclavos SPI.
- Cuatro controladores de circuitos integrados. (I^2C).
- 64KB en chip de RAM.
- 64KB en chip de arranque ROM.
- Dos puertos UART.
- Cuatro timers.
- Dos timers whatchdog.
- Tres interfaces E/S de propósito general (GPIO).
- Dos controladores de área de red del controlador (CAN).
- Componentes de depuración ARM Coresight:
  - Puerto de acceso a depuración (DAP).
  - Unidad de interfaz de puerto de rastreo (TPIU).
  - Macroceldas de rastreo del sistema.
  - Ruta de rastreo incorporado (ETR).
  - *Embedded cross trigger* (ECT).
- Administrador del sistema.
- Administrador de reloj.
- Administrador de reset.
- Administrador de escaneo.
- Administrador de FPGA.
- SDRAM

3.3.2. Soporte de *endianness.*

El término "Endian" se refiere a la forma en que los números binarios de bytes múltiples son almacenados en la memoria. Estos números binarios pueden ser almacenados en formato de:

- **Big Endian**: (MSB LSB)
- **Little Endian**: (LSB MSB)

donde MSB se refiere al byte más significativo y LSB se refiere al byte menos significativo.

El HPS es un sistema nativo Little-endian. Todos los esclavos del HPS trabajan bajo el sistema Little endian.

El procesador maestro se configura por medio de software para establecer la interpretación de datos como: Little endian, big endian, o byte invariante (BE8). Todos los maestros, incluyendo la interfaz USB 2.0, son Little endian. Los registros del subsistema MPU y de la caché de L2 son Little endian independientemente del modo endian de la CPU.

Nota: Altera recomienda que se use solo el sistema Little endian.

Para acceder a los periféricos y al DMA enlazadas en memoria, el procesador está fijado al modo BE8, el software debe convertir al sistema endianness. El procesador proporciona instrucciones para intercambiar la ruta de bytes para distintos tamaños de datos.

El núcleo ARM Cortex-A9 MPU soporta una sola instrucción para cambiar el sistema endianness del procesador y proporciona las instrucciones REV y REV16 para intercambiar el endianness de bytes o media palabra de datos (half-words) respectivamente.

El controlador DMA de ARM es software configurable para realizar un intercambio de ruta de bytes durante una transferencia.

3.3.3. Interconexión del sistema.

Los componentes del Sistema de Procesos en Hardware (HPS) se comunican entre sí con otro segmento del dispositivo SoC, mediante la interconexión del sistema que consiste de los siguientes bloques:

- Interconexión del nivel principal 3 (L3).
- Buses de nivel 4 (L4).

La interconexión del sistema se implementa con la interconexión de red ARM CoreLink (NIC-301), está interconexión de red proporciona una base para un alto rendimiento del sistema de interconexión HPS basado en los protocolos de ARM:

- AMBA (arquitectura avanzada de bus del microcontrolador),
- AXI (Intefaz Avanzada eXtensible),
- AHB (Bus avanzado de alto rendimiento), y
- APB (Protocolo de Bus periferico Avanzado).

La interconexión del sistema implementa una multicapa, una arquitectura no bloqueante que soporta multiples tareas simultaneas entre maestros y esclavos, incluyendo el subsistema MPU Cortex-A9. El sistema de interconexión proporciona cinco buses independientes L4 para el control de acceso y registros de estado (CSRs) de periféricos, administradores, y controladores de memoria.

3.3.4. Características de la interconexión del sistema.

Las siguientes interconexiones soportan dispositivos periféricos de alto rendimiento que tienen las siguientes características:

- Ancho de datos principal interno de 64 bits.
- Prioridad maestro programable arbitraria de un solo ciclo.
- Segmentación completa para evitar bloqueo del maestro.
- Control programable para la liberación de transacción del búfer FIFO.

- Compilación ARM TrustZone, con funciones de seguridad adicionales configurables por el maestro.
- Múltiples buses independientes L4.

3.3.5. Arquitectura de la interconexión del sistema.

La interconexión L3 es un entramado del conmutador parcialmente conectada en la cual no todos los maestros pueden acceder a todos los esclavos.
Internamente, la interconexión L3 es particionada dentro de los siguientes subconmutadores:

- Interconexión L3
- Uso de interconexión de alto rendimiento para la transferencia de datos de 64 bits.
- Funciona con hasta la mitad de la frecuencia del reloj principal del MPU.
- Provee al maestro con una conexión de retardo bajo para: los puentes AXI, la memoria en chip , la SDRAM, y el administrador de FPGA.
- Conmutador periférico maestro L3.
- Se utiliza para conectar periféricos de la memoria del maestro a la interconexión.
- Ancho de datos de 32-bits
- Funciona con hasta la mitad de la frecuencia del reloj de interconexión.
- Conmutador periférico esclavo.
- Se utiliza para proporcionar acceso a interfaces de nivel 3 y 4 de esclavos a maestros del periférico maestro e interconexiones.
- Ancho de datos.
- Cinco buses L4 independientes.

El maestro L3 y esclavo periférico son conmutados con barras transversales completamente conectadas. La interconexión L3 es una barra transversal parcialmente conectada.

3.3.6. Puentes HPS-FPGA

En esta sección se describen los puentes en el Sistema de Procesos en Hardware (HPS) usados para comunicar datos entre el entramado de la FPGA y la lógica HPS. Los puentes usan la Arquitectura de Buses de Microcontroladores Avanzados (AMBA) y el protocolo de Interface eXtensible Avanzada (AXI), que están basados en la Interconexión de red AMBA (NIC-301). Ver tabla 3.1.

El HPS contiene los siguientes puentes de HPS-FPGA:

- Puentes FPGA a HPS
- Puente HPS a FPGA
- Puente ligero de HPS a FPGA

3.3.6.1. Características de los puentes HPS-FPGA

Los puentes HPS-FPGA permiten al maestro, en el entramado de la FPGA, comunicarse con los esclavos en la lógica del HPS y viceversa. Por ejemplo, si se implementa memoria o periféricos en el entramado de la FPGA, los componentes del HPS como la MPU pueden acceder a ellos. Componentes implementados en el entramado de la FPGA, como el procesador Nios, pueden acceder a memoria y a periféricos en la lógica de HPS.

Características	**Puente FPGA a HPS**	**Puente HPS a FPGA**	**Puente Ligero HPS a FPGA**
Soporte de protocolo de la interfaces AMBA AXI3	Sí	Sí	Sí
Continúa en la siguiente página			

Tabla 3.1 – Continúa de la página anterior

Características	**Puente FPGA a HPS**	**Puente HPS a FPGA**	**Puente Ligero HPS a FPGA**
Implementa el cruce de reloj y gestiona la transferencia de datos a través de los dominios de reloj en la lógica HPS y el entramado FPGA	Sí	Sí	Sí
Realiza la conversión de ancho de datos entre la lógica del HPS y el entramado FPGA	Sí	Sí	Sí
Permite la configuración del ancho de la interfaz de la FPGA en el momento de la instancia	Sí	Sí	No

Tabla 3.1: Características de los puentes HPS-FPGA

Cada puente consiste de un par maestro-esclavo con una interfaz expuesta al entramado de la FPGA y la otra expuesto a la lógica HPS. El puente de la FPGA a HPS expone una interfaz de esclavo AXI que se puede conectar a las interfaces maestro AXI o interfaces Avalon-MM en el entramado de la FPGA. El puente HPS a FPGA y el puente ligero exponen una interfaz de maestro AXI que puede conectar una interfaz esclavo AXI o Avalon-MM en el entramado de la FPGA.

3.3.6.2. Puentes HPS-FPGA y el sistema de integración

El puente del HPS a FPGA es controlado por el conmutador principal del nivel 3 (L3), y el puente ligero HPS a FPGA es dominado por el conmutador periférico esclavo L3.

El puente FPGA a HPS es controlado por el conmutador principal L3, el cual permite que cualquier maestro implementado en el entramado de la FPGA acceda a la mayoría de los esclavos

en el HPS. Por ejemplo, el puente de la FPGA a HPS puede acceder al Puerto de Acelerador de Coherencia (ACP) del subsistema MPU Cortex-A9 para realizar accesos lógicos a la memoria caché SDRAM.

Los tres puentes contienen registros de visualización de programadores globales (GPV). Los registros GPV controlan el comportamiento del puente. El acceso a los registros GPV de los tres puentes se proporciona a través del puente ligero de HPS a FPGA.

3.4. Proceso de configuración de Linux en la tarjeta DE1_SoC.

A continuación se describe paso a paso el proceso de configuración de Linux en la tarjeta DE1_SoC. Este proceso fue realizado a partir de la descripción en [25], proporcionado por el fabricante del dispositivo.

3.4.1. Arrancar Linux en la tarjeta DE1_SoC.

Para programar una imagen Linux a una tarjeta micro SD se utiliza la herramienta libre llamada Win32DiskImager.exe en una maquina con el sistema operativo Windows.

Especificaciones del micro SD:

- Capacidad: Mínimo de 4GB
- Rapidez: Clase 4 (al menos. Ver tabla 3.2.)

Clase	**Velocidad mínima de escritura de datos en serie**	**Aplicaciones**
Clase 2	2 MB/s	Grabación de Vídeo SD
Continúa en la siguiente página		

Tabla 3.2 – Continúa de la página anterior

Clase	**Velocidad mínima de escritura de datos en serie**	**Aplicaciones**
Clase 4	4 MB/s	Vídeo de alta definición (por sus siglas en inglés HD) incluyendo grabación HD (de 720p a 1080p/1080i)
Clase 6	6 MB/s	Vídeo de alta definición (por sus siglas en inglés HD) incluyendo grabación HD (de 720p a 1080p/1080i)
Clase 10	10 MB/s	Full HD (1080p) grabación de vídeo y grabación consecutiva fija HD (bus de datos de alta velocidad)
UHS Clase de velocidad 1	10 MB/s	Transmisión en tiempo real y archivos grandes de vídeo HD (bus UHS)
UHS Clase de velocidad 3	30 MB/s	Archivos de vídeo 4K (bus UHS)

Tabla 3.2: Clase de velocidad en memorias SD[9]

3.4.1.1. Pre-construcción de la imagen en la tarjeta SD.

La Pre-construcción binaria se transmite como un archivo nombrado DE1_SoC. Este archivo de la imagen en la tarjeta SD contiene todos los archivos que se necesitan para arrancar Linux en la tarjeta DE1_SoC.

Archivos que se obtienen de la imagen:

- Pre-cargador SPL, es un cargador secundario de programa bootloader que se encarga de cargar el sistema operativo.
- U-boot (Universal Bootloader).
- Bloqueo del árbol de dispositivos.
- Núcleo de Linux.
- Sistema raíz de archivos de Linux.

El archivo de imagen para la tarjeta SD necesita ser programado para una tarjeta micro SD antes de que pueda usarse.

Los siguientes pasos muestran cómo programar el micro SD en una máquina con Windows usando Win32DiskImager.exe.

1. Conexión de la tarjeta micro SD a la PC con Windows.
2. Ejecución de Win32DiskImager.exe.
3. Selección del archivo de la imagen para la tarjeta micro SD.
4. Pulsar en “Escribir” para iniciar la escritura del archivo de la imagen a la tarjeta micro SD. Esperar hasta que la imagen esté escrita correctamente.

3.4.1.2. Ejecución de Linux Ubuntu en la tarjeta DE1_SoC

Esta sección muestra cómo iniciar el entorno de escritorio Linux Ubuntu en la tarjeta DE1_SoC. Linux Ubuntu es una version comprimida de Linux. Esto es un desarrollo extremadamente rápido y reduce el consumo de energía del entorno de escritorio. Este entorno de escritorio (Linux

Ubuntu) maneja menos recursos de CPU y menos memoria RAM que otros entornos de escritorio.

La configuración del hardware para arrancar el escritorio Linux Ubuntu en DE1_SoC necesita de los periféricos listados a continuación:

- Un monitor VGA.
- Un teclado USB.
- Un mouse USB.
- Una tarjeta micro SD con capacidad mínima de 8GB.

3.4.2. Procedimiento de configuración de Linux Ubuntu.

1. Asegurar el MSEL [4:0] en "0000" para la configuración de HPS en la FPGA. Ver figura 3.2
2. Se conecta el mouse y el teclado USB a los conectores USB (J7 y J8) y el monitor VGA al conector VGA (J9) en la tarjeta DE1_SoC.
3. Se inserta la tarjeta micro SD, con la imagen Linux Ubuntu, dentro de la tarjeta DE1_SoC.
4. Se enciende la tarjeta DE1_SoC.
5. Se debe esperar hasta visualizar el logotipo que se muestran en el monitor VGA cuando se arranque Linux.
6. Mientras la interfaz inicia la sesión, se puede elegir mas de un tipo de "root" cuando este requiere el nombre del usuario se presiona "Enter" en el teclado, y se escribe como contraseña "terasic" , y se presionar una vez mas "Enter". Realizadas las instrucciones anteriores se puede iniciar sesión en Linux Ubuntu.

3.5. Configuración de OpenCV en la plataforma Linux.

Para la configuración de la herramienta OpenCV en la arquitectura ARM se debe realizar una compilación cruzada y se debe contar con las siguientes herramientas dentro de Linux [26]:

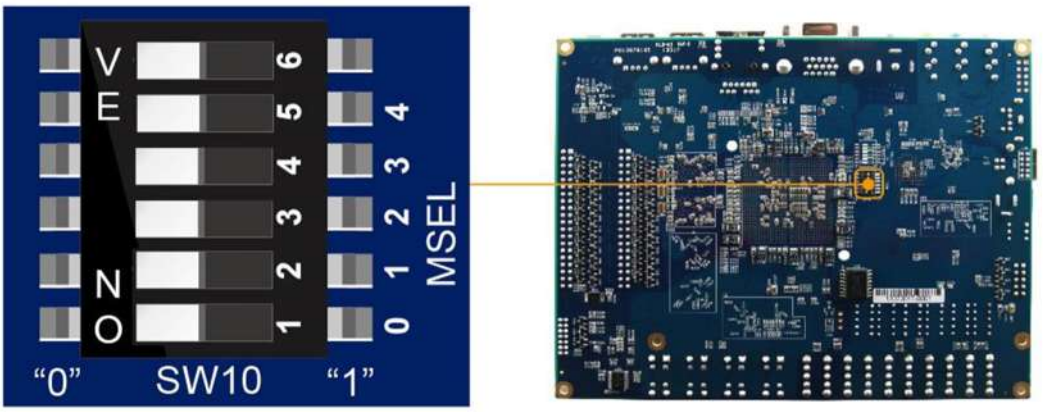

FIGURA 3.2: MSEL, ajuste del modo de configuración de la tarjeta DE1-SoC.

- Compilador GCC
- Editor Cmake
- GIT
- GTK+2.x
- pkg-config

3.5.0.1. Configurar el compilador GCC.

Haciendo uso de los comandos de Linux:

- `sudo`: permite al usuarios ejecutar programas con los privilegios de seguridad del super-usuario.
- `apt-get` instala/informa sobre los paquetes resolviendo las dependencias, los paquetes que instala los obtiene de Internet.

```
# sudo apt-get install gcc
```

`build-essential`: es un paquete que incluye un metapaquete incluye un conjunto mínimo necesario para poder realizar compilaciones de los paquetes que se consideran esenciales para la creación de paquetes Debian.

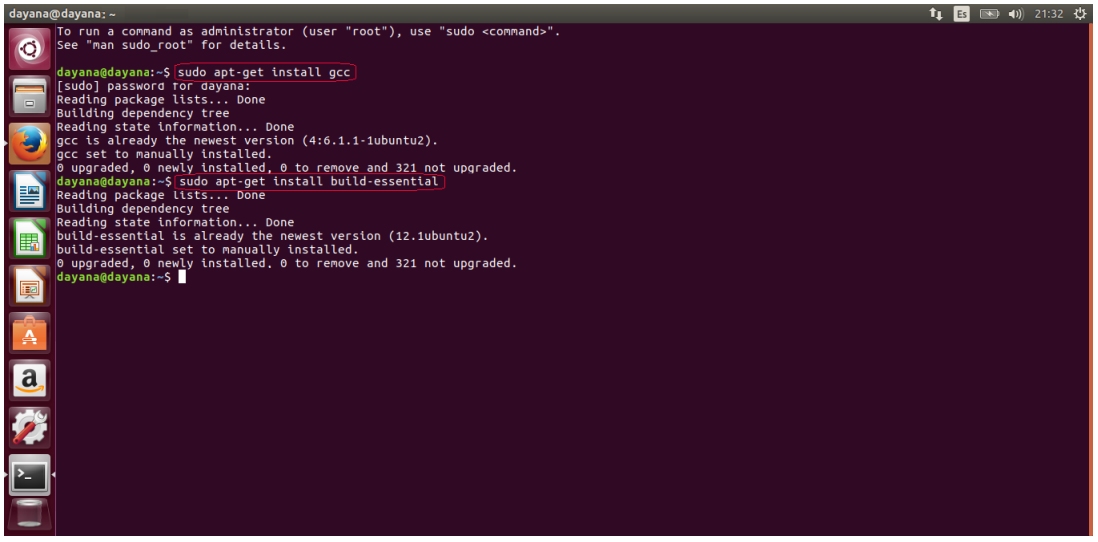

FIGURA 3.3: Instalación del compilador GCC en Linux.

```
# sudo apt-get install build-essential
```

3.5.0.2. Configurar CMake (Cross platform Make)

CMake es una herramienta multiplataforma de generación o automatización de código.

Para configurar esta herramienta se utilizaron los siguientes comandos desde la terminal de Linux. Figura 3.5.

```
# sudo add-apt-repository ppa:george-edison55/cmake-3.x
```

```
# sudo apt-get update
```

```
# sudo apt-get install cmake
```

En la figura 3.6 se muestra el comando para actualizar e instala las nuevas versiones de los paquetes respetando la configuración.

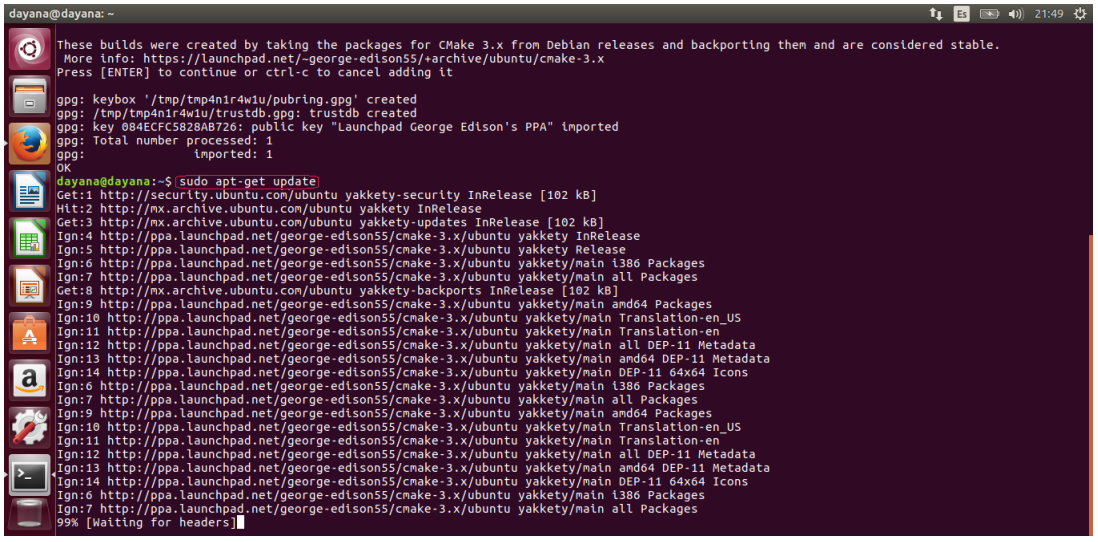

FIGURA 3.4: Actualización de la lista de paquetes disponibles y sus versiones en Linux.

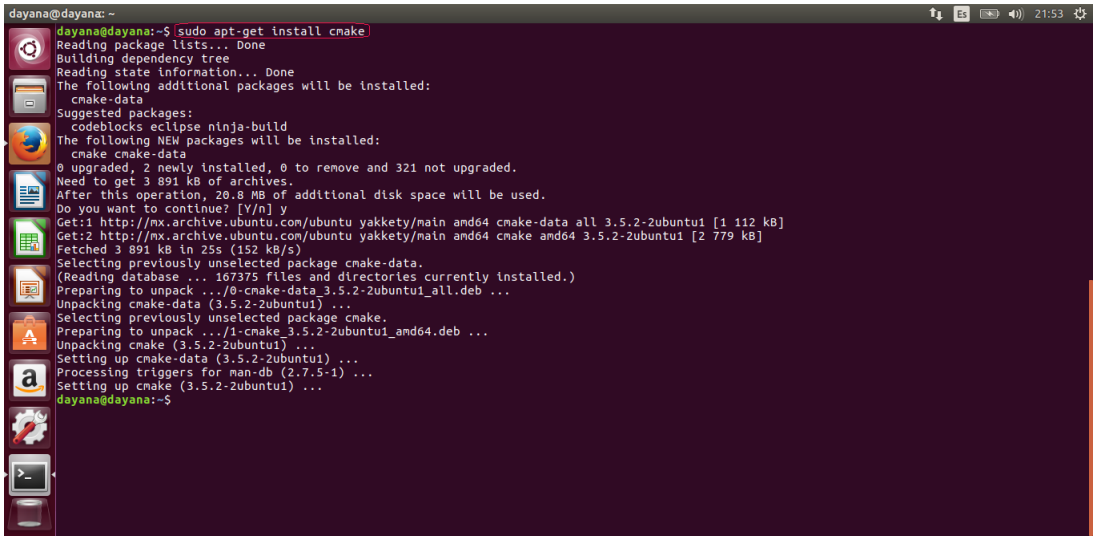

FIGURA 3.5: Instalación de la herramienta CMake.

```
# sudo apt-get upgrade
```

3.5.0.3. Configurar GIT: software de control de versiones.

`git` es un software de control de versiones diseñado pensando en la eficiencia y la confiabilidad del mantenimiento de versiones de aplicaciones cuando éstas tienen un gran número de archivos de código fuente.

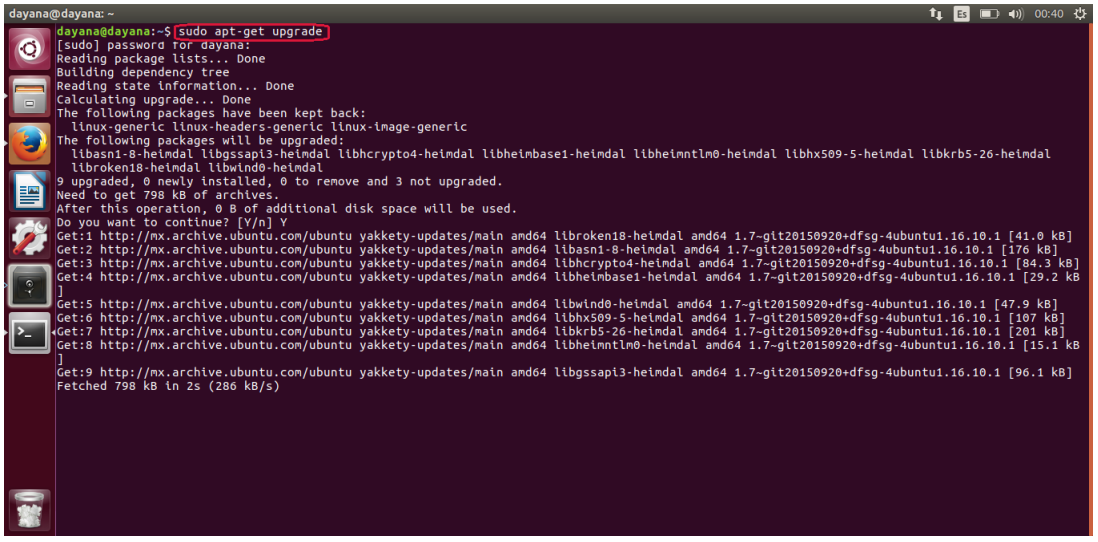

FIGURA 3.6: Actualización de paquetes.

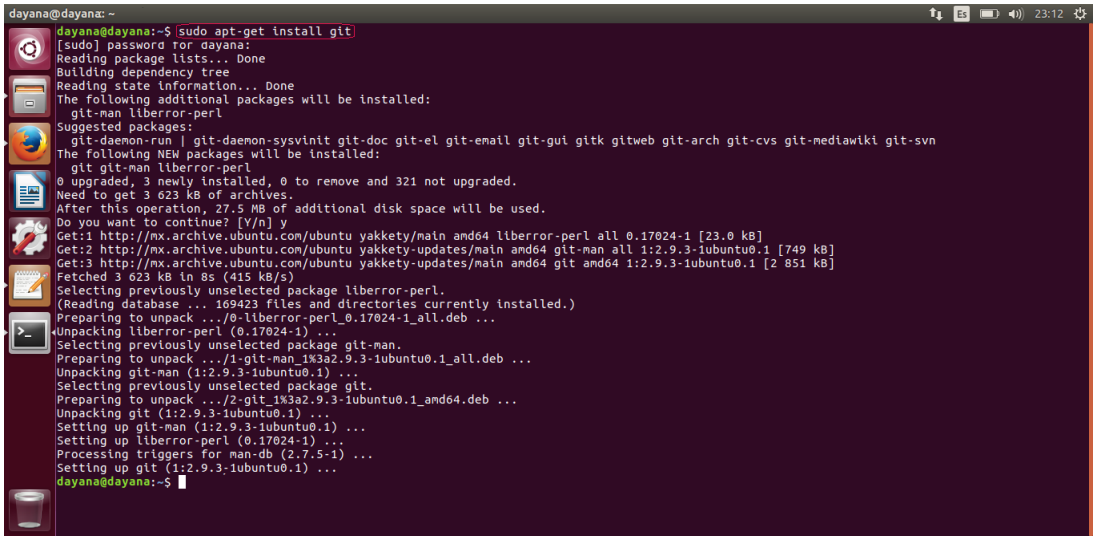

FIGURA 3.7: Instalación del control de versiones GIT

```
# apt-get install git
```

3.5.0.4. Configurar gtk+2.0 (The GIMP Toolkit)

GTK es un conjunto de bibliotecas multiplataforma para desarrollar interfaces gráficas de usuario (GUI).

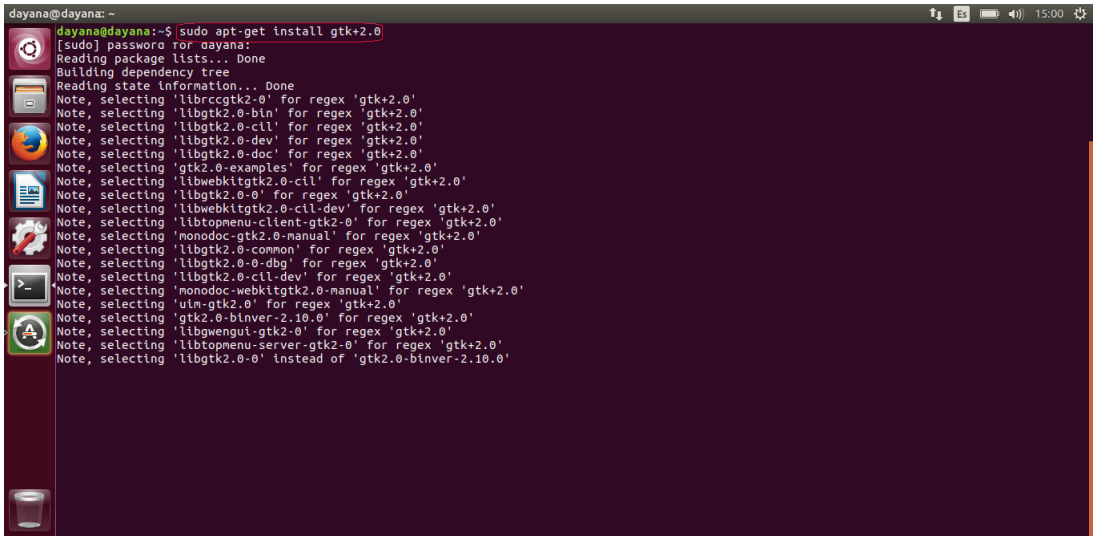

FIGURA 3.8: Instalación del conjunto de bibliotecas multiplataformas GTK.

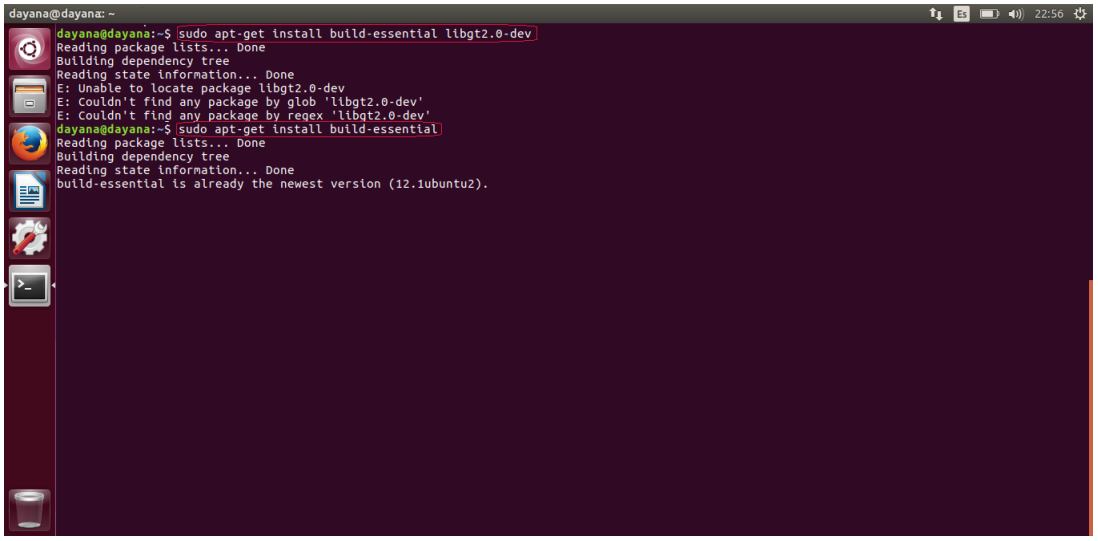

FIGURA 3.9: Instalación de las dependencias de OpenCV

```
# sudo apt-get install gtk2.0
```

```
# sudo apt-get install build-essential libgtk2.0-dev
```

`libgtk2.0-dev` es un archivos de desarrollo para la biblioteca de interfaces de usuario GTK+. Este es el backend (capa de acceso a datos,) por defecto para highgui.

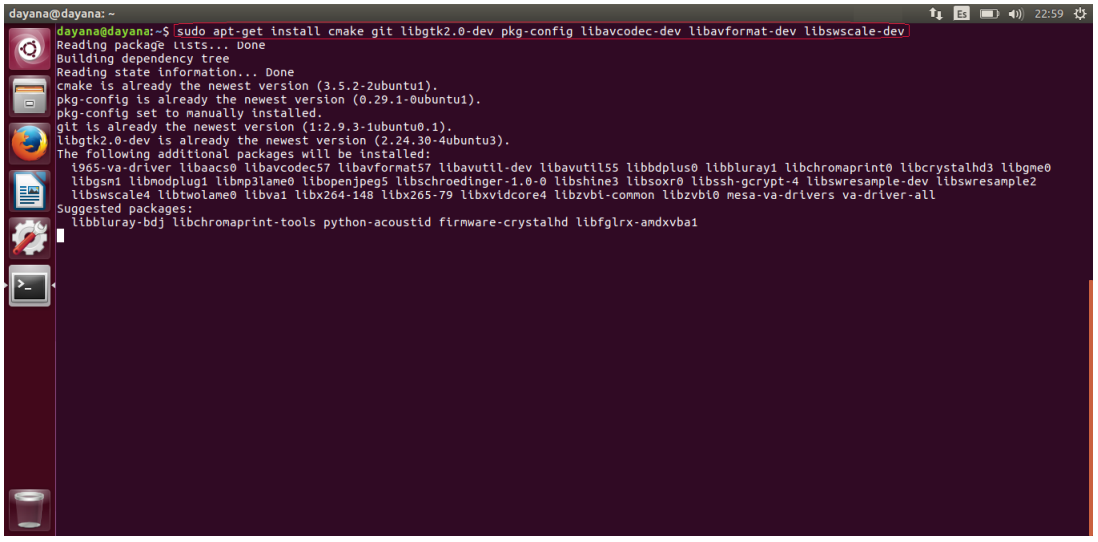

FIGURA 3.10: Instalación de pkg-config.

```
# sudo apt-get install cmake git libgtk2.0-dev
       pkg-config libavcodec-dev libavformat-dev libswscale-dev
```

3.5.1. Instalación de OpenCV.

Una vez instalados los paquetes necesarios se descarga el código de OpenCV, que es obtenido desde GitHub según los comandos siguientes.

```
# git clone https://github.com/opencv/opencv.git
```

```
# cd opencv
```

3.5.1.1. Compilación

Ahora se genera el makefile que define qué partes de OpenCV necesitan ser compiladas, se crea el directorio `release` y se entra a este directorio:

```
# mkdir release
# cd release
```

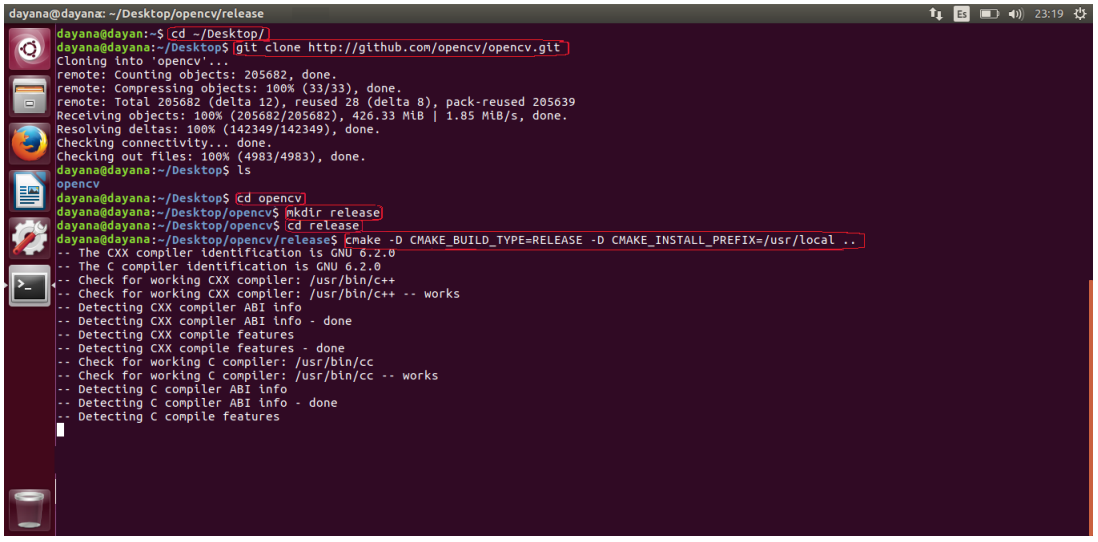

FIGURA 3.11: Instalación de OpenCV desde el repositorio.

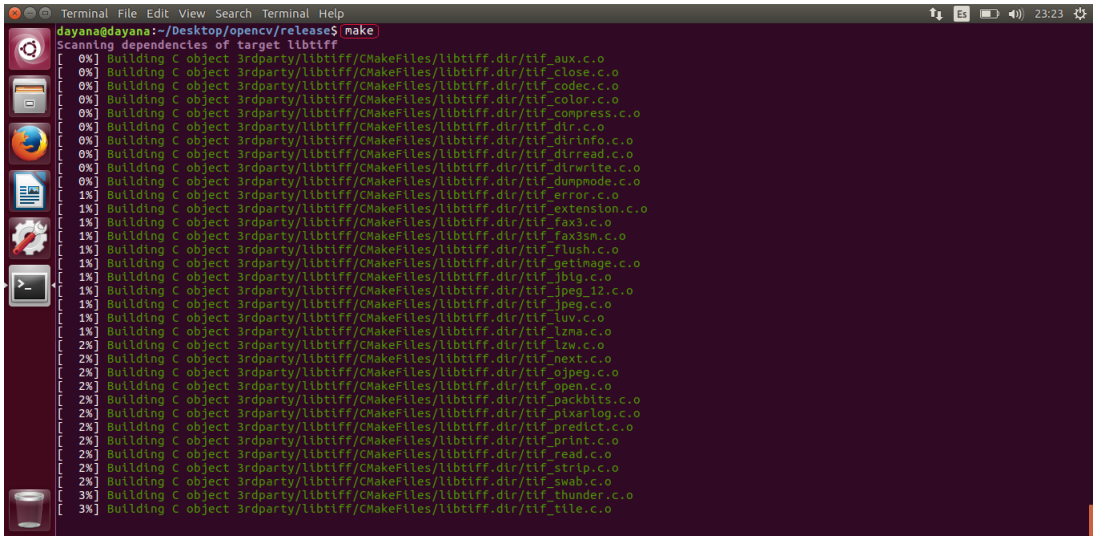

FIGURA 3.12: Compilación de OpenCV.

Se configura todo con lo que se va a trabajar de esta biblioteca y se genera el makefile:

```
# cmake -D CMAKE\_BUILD\_TYPE=RELEASE -D CMAKE\_INSTALL\_PREFIX=/usr/local ..
```

Se comprueba que esto no genere ningún error. En caso de que sí, se retrocede y se instala los paquetes extra y se ejecuta cmake de nuevo. Ahora se teclea make:

```
# make
```

Por último, para instalar la biblioteca se ejecuta:

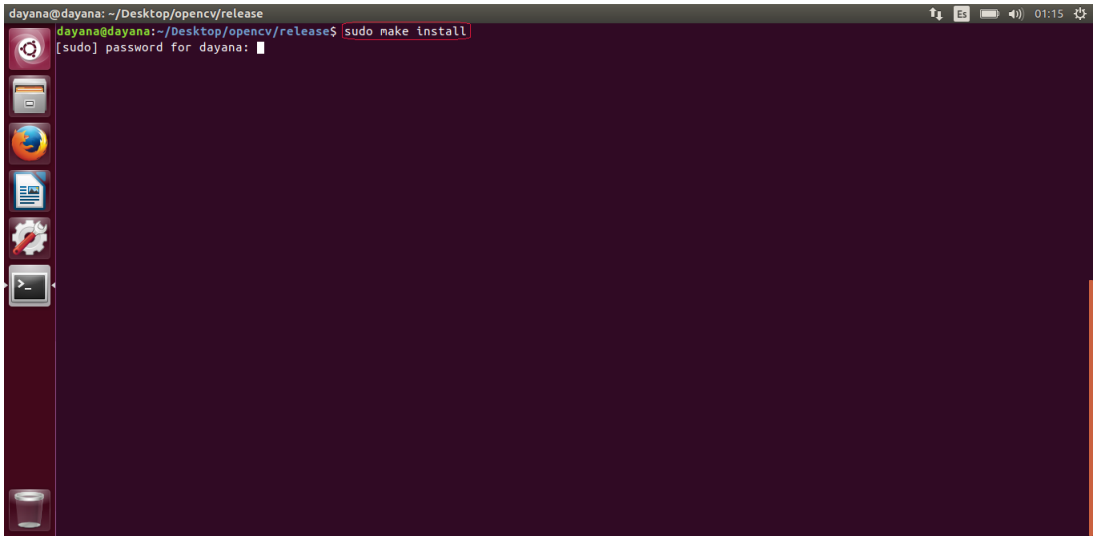

FIGURA 3.13: Instalación de OpenCV.

```
# sudo make install
```

3.6. Configuración de Code::Blocks para Linux.

Code :: Blocks es un Entorno de Desarrollo Integrado (IDE) libre de C, C++ y Fortran construido para satisfacer las necesidades más exigentes de sus usuarios. Está diseñado para ser muy extensible y completamente configurable [27].

Se instala Code::blocks a partir de la interfaz para `APT` llamada *aptitude*.

Aptitude muestra una lista de paquetes de software y permite al usuario elegir de modo interactivo cuáles desea instalar o eliminar. Dispone de un poderoso sistema de búsqueda que utiliza patrones de búsqueda flexibles, que facilitan al usuario entender las complejas relaciones de dependencia que puedan existir entre los paquetes.

```
# sudo apt-get install aptitude
```

```
# sudo aptitude
```

A continuación se muestra la apariencia de la interfaz aptitude.

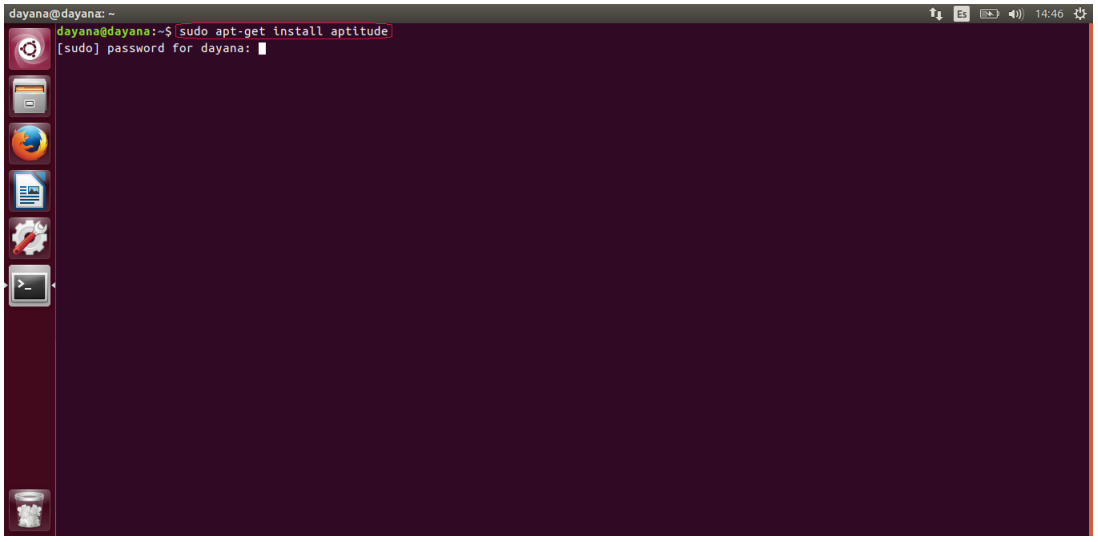

FIGURA 3.14: Instalación de la interfaz aptitude.

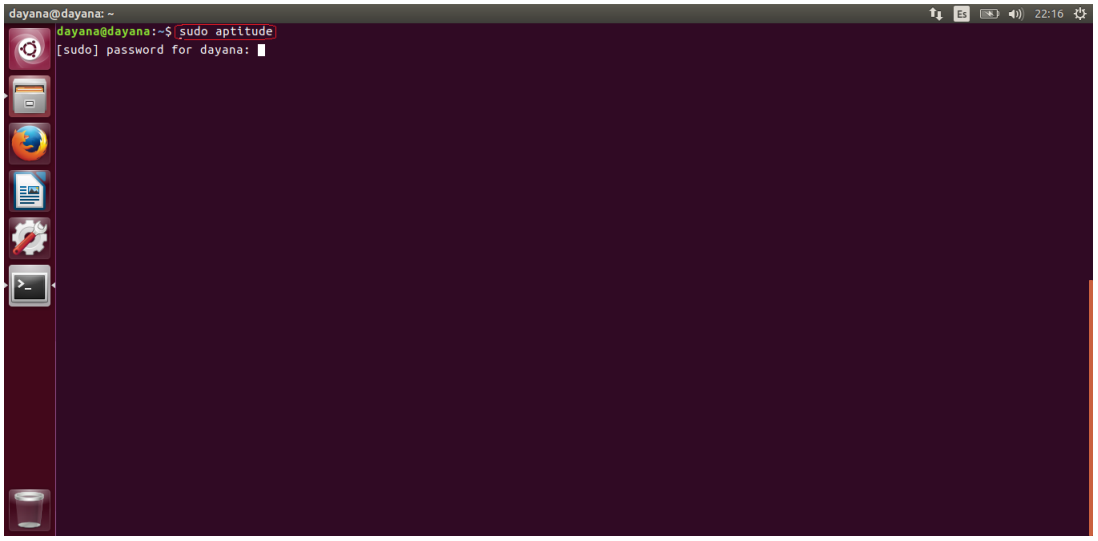

FIGURA 3.15: Ejecución de la interfaz aptitude.

Se selecciona la lista con el nombre `New Packages` se inserta el símbolo: "/" para desplegar una nueva ventana de búsqueda.

Se coloca el nombre del paquete que se deseaba instalar: `codeblocks`. Se presiona en "n" hasta encontrar el paquete que puede ser instalado (en color verde se indican los paquetes que pueden instalarse).

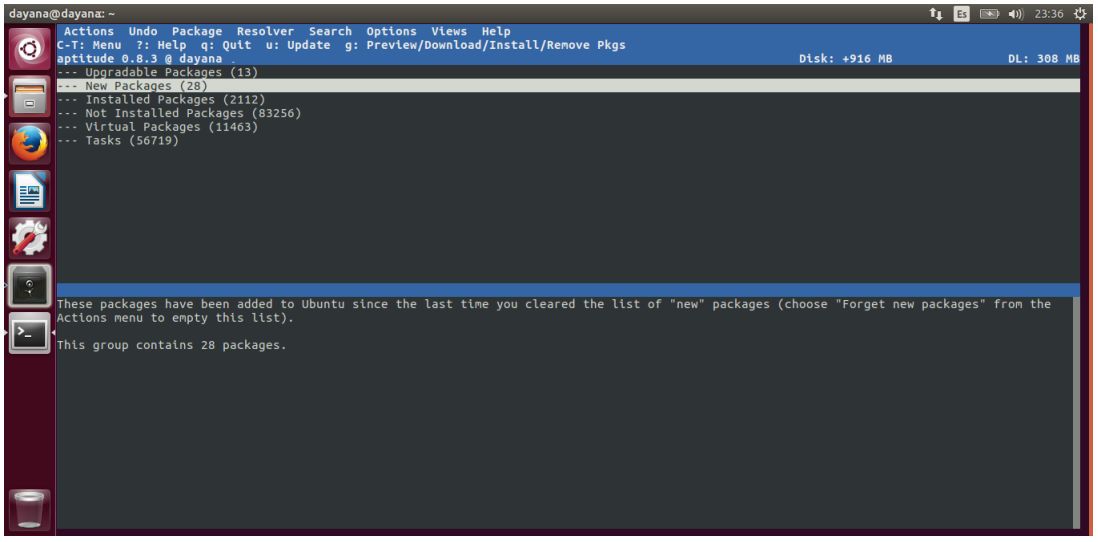

FIGURA 3.16: Vista de la ventana de la interfaz aptitude.

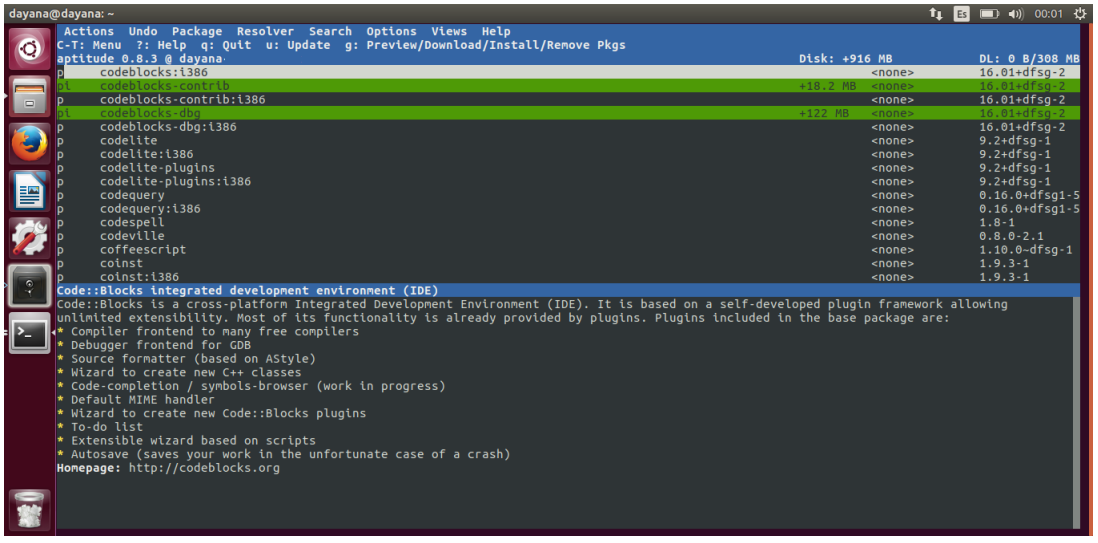

FIGURA 3.17: Vista de los paquetes que pueden ser instalados.

Se selecciona `codeblocks-contrib` y se presiona "+" y "g".

Una ves terminada la instalación se arranca el IDE de Code::Blocks con el comando:

```
# codeblocks
```

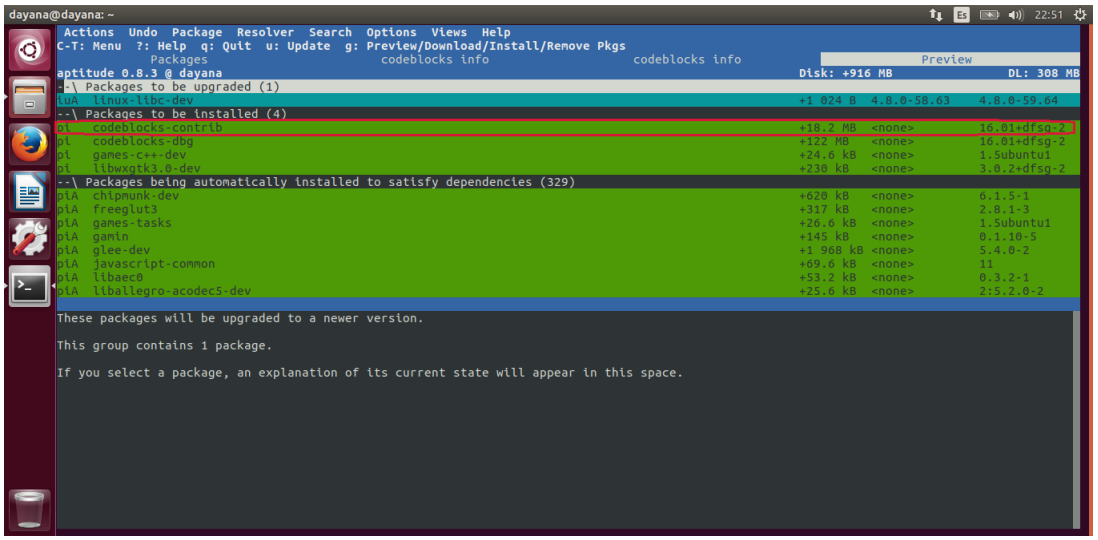

FIGURA 3.18: Vista de las dependencias de Code::Blocks que pueden ser instaladas.

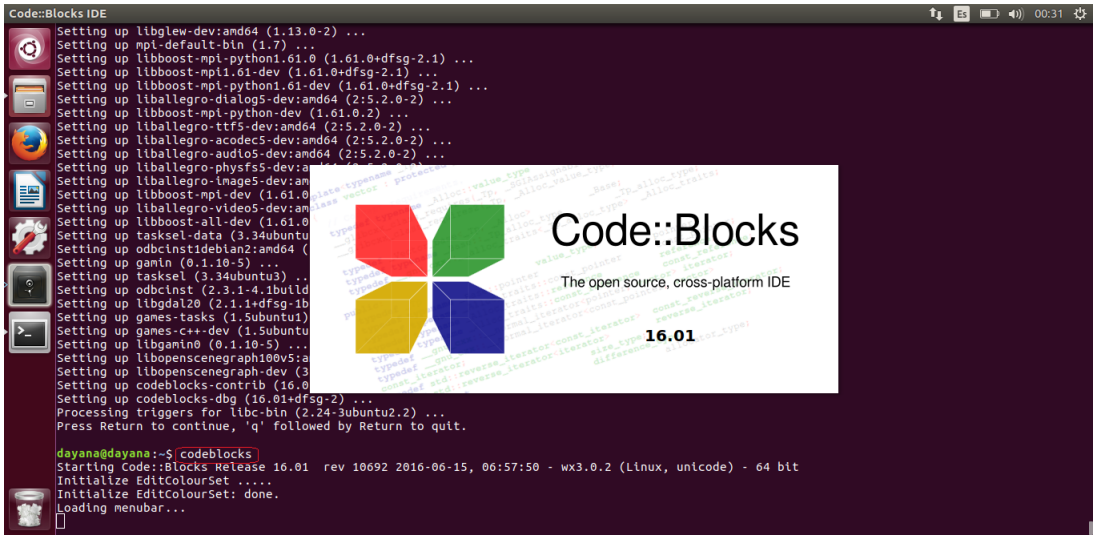

FIGURA 3.19: Arrancando el IDE Code::Blocks.

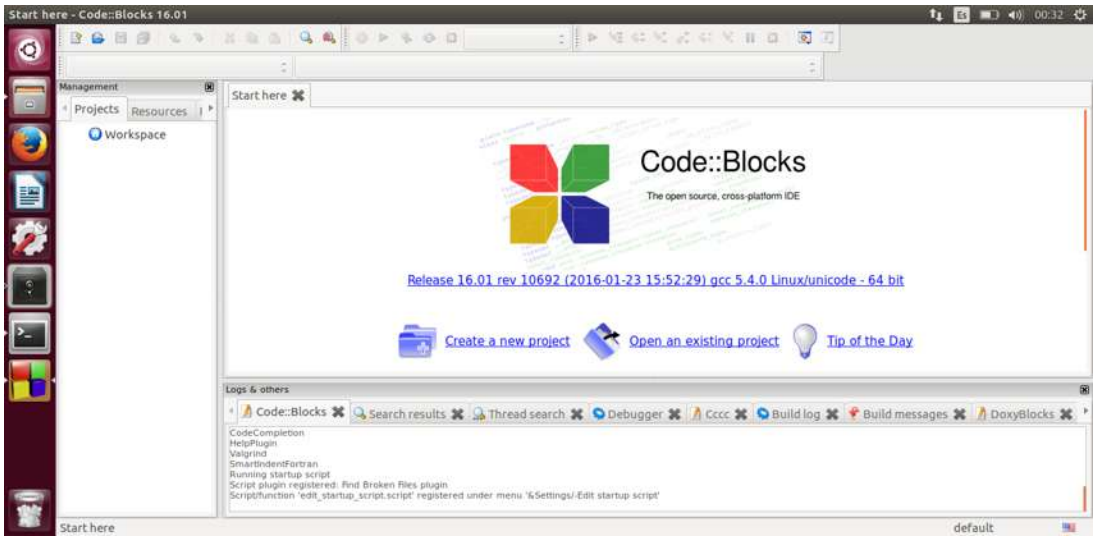

FIGURA 3.20: Vista de la ventana del IDE Code::Blocks.

CAPÍTULO 4

Implementación de algoritmos y prueba en la plataforma de trabajo.

EL objetivo de las pruebas es recuperar una imagen que muestre los bordes de interés de una imagen, para futuras aplicaciones. Esto se realiza con ayuda de distintas etapas del procesamiento de imágenes.

4.1. Definición de los Problemas Bien-Planteado y Mal-Planteado.

El matemático francés Hadamard introdujo el concepto de un problema bien-planteado y mal-planteado [8]:

- Un problema es *bien-planteado [well-posed]* en el sentido de Hadamard si cumple tres condiciones:
  1. **Existencia.** Para cualquier vector de entrada **u**, existe una salida **Wu** que es una solución al problema.
  2. **Unicidad.** Para cualquier par de vectores de entrada **u**, **q**, tenemos **Wu** = **Wq** si y sólo si **u** = **q**.
  3. **Continuidad.** El mapeo es continuo, es decir, para cualquier $e > 0$ existe $r > 0$ tal que la condición $d(u, q) < r$ implica que $d(Wu, Wq) < e$, donde $d(.,.)$ es el símbolo de la distancia entre dos argumentos es sus respectivos espacios.
- Un problema es *mal-planteado [ill-posed] o mal-condicionado [ill-conditioned]* en el sentido de Hadamard si cualquiera de las tres condiciones del problema *bien planteado [well-posed]* no es satisfecha.

Recordando la ecuación 2.24 que representa la ecuación de observación del sistema definida como:

$$\mathbf{u} = \mathbf{S}\mathbf{v} + \mathbf{n} \tag{4.1}$$

Donde:

- **u** son los datos capturados,
- **S** es una matriz con la función de un filtro lineal,
- **v** es la matriz que representa la imagen real y
- **n** es una matriz de ruido gaussiano

Para realizar el proceso y obtener **u** se crea un arreglo en C++ como un vector de siete elementos que representa una señal *sinc*.

$$sinc\,(x) = \begin{cases} \frac{\sin(x)}{x} & : \text{ para } x \neq 0 \\ 1 & : \text{ para } x = 0 \end{cases} \tag{4.2}$$

Con este vector sinc, se forma una matriz Toeplitz que, en este caso, tiene la función de ser un filtro lineal. Se crea un arreglo de una matriz cuadrada de acuerdo a las dimensiones de la matriz de la escena real.

Una vez creada la matriz Toeplitz como un arreglo en C++ se convierte a una clase `Mat`, utilizando uno de los constructores descritos en el capitulo 3 y en [28].

Ahora se requiere crear una matriz de ruido gaussiano de las mismas dimensiones de la matriz de la escena real obtenidas con el operador `size`, para ello, se hace uso de otro constructor de la clase `Mat`.
Esta matriz de ruido **n** contamina la representación de la imagen capturada **Sv**,(simulando, por ejemplo, el ruido térmico o ruido del propio sensor).
Una vez construida la matriz de ruido se utiliza la función de operaciones en matrices `randn` para rellenar la matriz con datos de ruido gaussiano.

4.2. Método de mínimos cuadrados restringidos [8]

Una vez que obtenidas las matrices: Toeplitz, que representa el operador matricial del sistema **S**, la matriz de la escena real (información estadística a priori acerca de la señal) **v** y la matriz de ruido gaussiano **n**, se obtiene la matriz de datos **u** realizando una multiplicación matricial de **Sv** (debido a la forma de nuestra matriz Toeplitz, se realiza una convolución, ver definición de convolución ecuación 2.50) y se suma la matriz de ruido.

Al obtener 4.1 que representa los datos del sistema, al ser procesada por medio de visión computacional puede arrojar valores erróneos, que impidan obtener los bordes y/o alguna información especifica de la escena real, para evitar esto se construye un estimador **W** para **u**, **W** que al ser aplicado a los datos **u**, produce la estimación de la señal de información deseada.

Así se tiene:

$$\hat{\mathbf{v}} = \mathbf{W}\mathbf{u} \tag{4.3}$$

Donde $\hat{\mathbf{v}}$ es la estimación de la imagen
W es el operador de estimación por el método de mínimos cuadrados restringidos (CLS)
u es la matriz de observación del sistema

Información a priori mínima

$$\text{Señal (si es aleatoria)} \quad \mathbf{m_v} =< \mathbf{v} > \quad \text{valor medio conocido} \tag{4.4}$$

$$\text{Ruido} \quad \mathbf{m_n} =< \mathbf{n} >= \mathbf{0} \quad \text{valor media cero} \tag{4.5}$$

$$\mathbf{R_n} =< \mathbf{n}\mathbf{n}^{\mathbf{T}} > \quad \text{covarianza conocida (o desconocida)} \tag{4.6}$$

4.2.1. Idea de regularización descriptiva

La idea de regularización descriptiva determinística (Regularización de Tikhonov [8]) es sustituir el problema mal planteado original por un problema "más cercano" de optimización bien planteado:

$$\hat{\mathbf{v}} = \arg\min_{\mathbf{v}} J(\mathbf{v}) \tag{4.7}$$

La aproximación de Tikhonov, puede ser realizada substituyendo el problema inicial no restringido LS

$$\hat{\mathbf{v}} = \arg\min_{\mathbf{v}} J_1(\mathbf{v}) \tag{4.8}$$

con el estándar funcional de coste del error LS (o el modelo de acuerdo de datos)

$$J_1(\hat{\mathbf{v}}) = \|\mathbf{u} - \mathbf{S}\mathbf{v}\|_{l_2} = [(\mathbf{u} - \mathbf{S}\mathbf{v}), (\mathbf{u} - \mathbf{S}\mathbf{v})] \qquad \text{el error de aproximación LS} \tag{4.9}$$

El problema regularizado de optimización de mínimos cuadrados restringidos de la ecuación 4.7 con el funcional de coste CLS se puede descomponer como:

$$J(\mathbf{v}) = J_1(\mathbf{v}) + \alpha J_2(\mathbf{v}) \tag{4.10}$$

donde

$$J_1(\mathbf{v}) = \|\mathbf{u} - \mathbf{S}\mathbf{v}\|_{l_2} = [(\mathbf{u} - \mathbf{S}\mathbf{v}), (\mathbf{u} - \mathbf{S}\mathbf{v})]$$

es el error de aproximación LS, y

$$J_2(\mathbf{v}) = \|\mathbf{v} - \mathbf{m}_\mathbf{v}\|_{l_2} = [(\mathbf{v} - \mathbf{m}_\mathbf{v}), (\mathbf{v} - \mathbf{m}_\mathbf{v})]$$

es el costo de norma mínima; α es el parámetro de regularización.

4.3. Problema de optimización CLS

La estrategia CLS se formula de la siguiente manera

$$\hat{\mathbf{v}} = \arg\min_{\mathbf{v}\in V} J_1(\mathbf{v}) + \alpha J_2(\mathbf{v}) = \arg\min_{\mathbf{v}\in V} \|\mathbf{u} - \mathbf{S}\mathbf{v}\|_{l_2} + \alpha \|\mathbf{v} - \mathbf{m}_\mathbf{v}\|_{l_2} \tag{4.11}$$

para algún valor fijo elegido del parámetro de regularización α que sirve para ajustar "el grado de libertad" del método CLS.

Casos limitantes:

- Si $\alpha \to 0$, el problema es reducido a su versión no restringida LS mal condicionada;
- Si $\alpha \to \infty$, Los datos medidos $\mathbf{u}$ no son fiables y la solución $\hat{\mathbf{v}}$ puede ser explícitamente obtenida del modelo de conocimiento a priori, es decir $\hat{\mathbf{v}} = \mathbf{m}_\mathbf{v}$

Vector x de valor real $\mathbf{x}$	Vector x de valor complejo $\mathbf{x}$
$\mathrm{J}(\mathbf{x})=\mathbf{a^T x} \Rightarrow \mathbf{g(x)=a}$	$\mathrm{J}(\mathbf{x})=\mathbf{x^+ a} \Rightarrow \mathbf{g(x)=0}$
$\mathrm{J}(\mathbf{x})=\mathbf{x^T A x} \Rightarrow \mathbf{g(x)=a}$	$\mathrm{J}(\mathbf{x})=\mathbf{a^+ x} \Rightarrow \mathbf{g(x)=2A}$
$\mathrm{J}(\mathbf{x})=\mathbf{x^T a x} \Rightarrow \mathbf{g(x)=2ax}$	$\mathrm{J}(\mathbf{x})=\mathbf{x^+ A x} \Rightarrow \mathbf{g(x)=2Ax}$

TABLA 4.1: Reglas para diferenciación de funciones vectoriales de primer y segundo orden.

4.3.1. Solución del problema de optimización de CLS

Podemos ampliar la expresión para la función de coste de CLS en términos de producto punto interno como

$$J(\mathbf{v}) = \text{const} - \mathbf{v}^+\mathbf{S}^+\mathbf{u} - \mathbf{u}^+\mathbf{S}\mathbf{v} + \mathbf{v}^+\mathbf{S}^+\mathbf{S}\mathbf{v} + \alpha(\mathbf{v}^+\mathbf{v} - \mathbf{m}_\mathbf{v}^+\mathbf{v} - \mathbf{v}^+\mathbf{m}_\mathbf{v}) \tag{4.12}$$

con

$$\text{const} = [\mathbf{u},\mathbf{u}] + \alpha\,[\mathbf{m_v},\mathbf{m_v}] = \mathbf{u}^+\mathbf{u} + \mathbf{m}_\mathbf{v}^+\mathbf{m_v} \tag{4.13}$$

agrupando los términos independientes sobre el vector $\mathbf{v}$. $J(\mathbf{v})$ es precisamente una función de segundo orden de valor escalar del vector $\mathbf{v}$. En consecuencia, podemos visualizar la dependencia de $J(\mathbf{v})$ sobre $\mathbf{v}$ como una superficie en forma de tazón con un mínimo único. Para determinar el vector de estimación óptimo $\hat{\mathbf{v}}$ ahora tenemos que seguir la técnica matemática de encontrar el mínimo convexo, p. ej. una función "en forma de tazón".

- Obtener la diferencial de la función del costo aumentado $J(\mathbf{v})$ con respecto al vector deseado $\mathbf{v}$, obtener un vector gradiente e igualarlo a cero, y hallar una solución a la ecuación variacional resultante de la ecuación de estimación $\hat{\mathbf{v}}$

$$g(\mathbf{v}) = \frac{\partial J(\mathbf{v})}{\partial \mathbf{v}} = \mathbf{0} \tag{4.14}$$

Recolección de reglas para diferenciación de funciones de primer y segundo orden con respecto al vector de valor real y valor complejo respectivamente se da en la tabla 4.1.

Usando las reglas dadas en la columna derecha de la Tabla 4.1, se tiene:

$$\mathbf{g(v)} = \frac{\partial J(\mathbf{v})}{\partial \mathbf{v}} = -2\mathbf{S}^+(\mathbf{u} - \mathbf{S}\mathbf{v}) + 2\alpha(\mathbf{v} - \mathbf{m_v}) = \mathbf{0} \tag{4.15}$$

Después, reorganizando términos del producto de la ecuación diferencial modificada

$$(\mathbf{S}^+\mathbf{S} + \alpha\mathbf{I})\mathbf{v} = \alpha\mathbf{m_v} + \mathbf{S}^+\mathbf{u} + (\mathbf{S}^+\mathbf{S}\mathbf{m_v} - \mathbf{S}+\mathbf{S}\mathbf{m_v}) = (\mathbf{S}^+\mathbf{S} + \alpha\mathbf{I})\mathbf{m_v} + \mathbf{S}^+(\mathbf{u} - \mathbf{S}\mathbf{m_v}) \tag{4.16}$$

La solución de la ecuación anterior produce el estimador CLS deseado

$$\hat{\mathbf{v}} = \mathbf{m_v} + (\mathbf{S}^+\mathbf{S} + \alpha\mathbf{I})^{-1}\mathbf{S}^+(\mathbf{u} - \mathbf{S}\mathbf{m_v}) = \mathbf{m_v} + \mathbf{W}(\mathbf{u} - \mathbf{S}\mathbf{m_v}) \tag{4.17}$$

Óptimo operador solución CLS

$$\mathbf{W} = (\mathbf{S}^+\mathbf{S} + \alpha\mathbf{I})^{-1}\mathbf{S}^+ \tag{4.18}$$

4.4. Algoritmo de Canny

Para obtener las características de bordes de la imagen reconstruida se utiliza el algoritmo Canny el cual se encuentra ya implementado como una función en las bibliotecas de OpenCV (Ver capitulo 3).

El algoritmo de Canny es considerado como uno de los mejores metodos para la detección de bordes hasta la fecha, Canny pretende desarrollar una detección de bordes que satisfaga tres criterios clave [10], [29]:

- Obtención del gradiente (Detección optima): en este paso se calcula la magnitud y orientación del vector gradiente en cada pixel.
- Supresión no máxima (Buena detección): en este paso se logra el adelgazamiento del ancho de los bordes, obtenidos con el gradiente, hasta lograr bordes de un pixel de ancho.
- Histéresis de umbral (Respuesta única): en este paso se aplica una función de histéresis basada en dos umbrales; con este proceso se pretende reducir la posibilidad de aparición de contornos falsos.

El primer requerimiento pretende reducir la respuesta al ruido. Este puede ser afectado por un suavizado óptimo, Canny fue el primero en demostrar que el filtro gaussiano es óptimo para la detección de bordes. El segundo criterio se refiere a la obtención de una buena precisión. Los bordes son detectados en el lugar correcto. Este se puede lograr por un proceso de supresión no máxima (el cual es equivalente a la detección de picos) la supresión no máxima conserva solo aquellos puntos en la cima de una cresta de los datos del borde, mientras que suprime todos los demás. Esto da como resultado un adelgazamiento: la salida de la supresión no máxima son líneas delgadas de los puntos del borde, en el lugar correcto. El tercer criterio se refiere a la localización de un punto de borde único en respuesta a un cambio de brillantez. Esto es por que más de un borde puede ser indicador de que esta presente, consistente con la salida obtenida

por el operador del borde anterior.

Canny muestra que el operador gaussiano fue óptimo para la imagen suavizada. Recordando que el operador gaussiano $g(x, y, \sigma)$ es obtenido por:

$$g(x, y, \sigma) = e^{-\frac{x^2+y^2}{2\sigma^2}} \tag{4.19}$$

por diferenciación, para un vector unitario $U_x = [1, 0]$ y $U_y = [0, 1]$ a lo largo del eje coordenado obtenemos:

$$\nabla g(x, y) = \frac{\partial g(x, y, \sigma)}{\partial x} U_x + \frac{\partial g(x, y, \sigma)}{\partial y} U_y \tag{4.20}$$

$$= -\frac{x}{\sigma^2} e^{-\frac{x^2+y^2}{2\sigma^2}} U_x - \frac{y}{\sigma^2} e^{-\frac{x^2+y^2}{2\sigma^2}} U_y \tag{4.21}$$

La ecuación 4.21 da una forma para calcular los coeficientes de una derivada del modelo gaussiano que combina la diferenciación de primer orden con el suavizado gaussiano. Esto es una imagen suavizada, y así el borde será una cresta de datos. Para marcar un borde en el punto correcto (y para reducir múltiples respuestas) se puede convolucionar una imagen con un operador el cual se obtiene de la primera derivada en dirección normal al borde. El valor máximo de esta función será el dato del pico del borde donde el gradiente en la imagen original es más fuerte, y por lo tanto es la localización del borde. En consecuencia buscamos un operador G_n, el cual es la primera derivada de la función gaussiana g en la dirección normal, $\mathbf{n}_\perp$:

$$G_n = \frac{\partial g}{\partial \mathbf{n}_\perp} \tag{4.22}$$

donde $\mathbf{n}_\perp$ puede ser estimado de la derivada de primer orden de la función gaussiana g convolucionada con la imagen escalada apropiadamente como:

$$\mathbf{n}_\perp = \frac{\nabla(\mathbf{P} * g)}{|\nabla(\mathbf{P} * g)|} \tag{4.23}$$

La localización exacta del punto del borde es entonces el punto máximo de G_n convolucionado con la imagen. Este es máximo cuando el diferencial (a lo largo de $\mathbf{n}_\perp$) es cero:

$$\frac{\partial(G_n * \mathbf{P})}{\partial \mathbf{n}_\perp} = 0 \tag{4.24}$$

Sustituyendo la ecuación 4.22 en la ecuación 4.4 obtenemos:

$$\frac{\partial^2(G * \mathbf{P})}{\partial \mathbf{n}_{\perp}^2} = 0 \tag{4.25}$$

La ecuación 4.4 proporciona la base para un operador, el cual cumple uno de los criterios de Canny, que los bordes deben ser detectados en el lugar correcto. Esta es la supresión no máxima, la cual es equivalente a conservar los picos (y su equivalente a la diferenciación perpendicular al borde), lo que adelgaza la respuesta del operador de detección del borde para obtener el punto del borde que se encuentra en el lugar correcto, sin respuestas múltiples y con una respuesta mínima al ruido. Sin embargo, es imposible conseguir una implementación exacta con el requerimiento dado de Canny para estimar la dirección normal.

Una aproximación común es:

1. Uso de la degradación gaussiana.
2. Uso del operador de Sobel.
3. Uso de la supresión no máxima.
4. Umbral con histéresis para conectar los puntos del borde.

La supresión no máxima esencialmente localiza los puntos mas altos en la magnitud de los datos del borde. Esto se realiza por el uso de la información de dirección del borde, para verificar que el punto es el pico de una cresta. Dada una región de 3x3, un punto que se encuentra en un máximo si el gradiente a ambos lados de este es menor que el gradiente en dicho punto.

Usando la interpolación de primer orden M_x y M_y en $\mathbf{P}_{x,y}$ y los valores de M_x y M_y para los vecinos representados como:

$$M_1 = \frac{M_y}{M_x} M(x+1, y-1) + \frac{M_x - M_y}{M_x} M(x, y-1) \tag{4.26}$$

y

$$M_2 = \frac{M_y}{M_x} M(x-1, y+1) + \frac{M_x - M_y}{M_x} M(x, y+1) \tag{4.27}$$

El punto $\mathbf{P}_{x,y}$ es entonces, marcado como un máximo si $M_{x,y}$ excede entre M_1 y M_2 de otra manera se coloca en cero. De esta forma el pico de la cresta de la magnitud del dato del borde es conservado mientras este no sea el pico colocado en cero. La implementación de la supresión no máxima requiere de una función que genere la coordenada del punto entre los cuales la magnitud del borde está interpolado.

El ciclo de histéresis requiere dos umbrales, un umbral superior y otro inferior. El proceso inicia cuando un punto del borde de la supresión no máxima es superior al valor de umbral alto. Este es etiquetado como un punto de borde (usualmente blanco, con un valor de 255) y forma el primer punto de una linea de puntos del borde. Los vecinos del punto son entonces buscados para determinar si exceden o no el umbral inferior. Cualquier vecino que exceda el umbral inferior es etiquetado como un punto del borde y sus vecinos entonces son buscados para determinar si exceden o no el umbral inferior. De esta forma el primer punto encontrado del borde (el único que excedió el umbral superior), se convierte en el punto "semilla" para una búsqueda. Sus vecinos en turno, se convierten en puntos semilla si estos exceden el umbral inferior, y así la búsqueda se extiende a lo largo de las ramas de los vecinos que exceden el umbral inferior. Para cada rama, la búsqueda termina en los puntos que no tienen vecinos arriba del umbral inferior.

4.5. Transformada de Hough

Para la detección de líneas se utiliza la función `HoughLinesP` [30] la cual encuentra segmentos de línea en una imagen binaria utilizando la transformación probabilística de Hough.

La transformada Hough (TH) [29] es una técnica que localiza formas en una imagen. En particular ha sido usada para extraer líneas, círculos y elipses (o secciones cónicas). La TH fue introducida por Hough en 1962 y en 1969 Rosenfeld reconoció su ventaja potencial como un algoritmo de procesamiento de imágenes. La TH fue implementada para encontrar líneas en una imagen y ha sido ampliada extremadamente, ya que tiene muchas ventajas y muchas rutas potenciales para mejorar.

Su principal ventaja es que puede entregar el mismo resultado que la metodología de coincidencia de plantilla, pero mas rápido. Esto se logra por una reformulación del proceso de coincidencia del modelo, basado en un enfoque de recopilación de pruebas, donde las evidencias son los votos emitidos en un arreglo acumulador.

La implementación de la TH define un mapeo del punto de la imagen dentro de un espacio acumulador (espacio de Hough). El mapeo se logra de una manera computacionalmente eficiente, basado en la función que describe la forma de destino. Este mapeo requiere de muchos menos recursos computacionales que la coincidencia del modelo.

4.5.0.1. Líneas

Se considera encontrar líneas en una imagen. En una parametrización cartesiana, puntos colindantes en una imagen con coordenadas (x, y) que están relacionados por su pendiente m y una intersección c acorde a:

$$y = mx + c \tag{4.28}$$

Esta ecuación puede ser escrita en forma homogénea como:

$$Ay + Bx + 1 = 0 \tag{4.29}$$

donde $A = \frac{-1}{c}$ y $B = \frac{m}{c}$. Así una línea esta definida al dar un par de valores (A, B). Sin embargo, podemos observar una simetría en la definición de la ecuación 4.29.

La TH recopila evidencias del punto (A, B) considerando que todos los puntos (x, y) definen la misma línea en el espacio (A, B). Es decir, si el conjunto de puntos colineales $\{x_i, y_i\}$ define la línea (A, B), entonces:

$$Ay_i + Bx_i + 1 = 0 \tag{4.30}$$

Esta ecuación puede ser vista como un sistema de ecuaciones y puede simplemente ser escrita en terminos de la parametrización cartesiana como:

$$c = -x_i m + y_i \tag{4.31}$$

Así, para determinar la recta debemos encontrar los valores de los parámetros (m, c) (o (A, B) en forma homogénea) que satisfacen la ecuación 4.31 (o 4.30, respectivamente). Sin embargo, se debe notar que el sistema está generalmente sobredeterminado. Es decir, se tiene más ecuaciones que incógnitas.

Por lo tanto, se debe encontrar la solución que se acerca para satisfacer todas las ecuaciones simultáneamente. Este tipo de problema puede resolverse, por ejemplo, utilizando técnicas de

mínimos cuadrados lineales. La TH utiliza un enfoque de recopilación de pruebas para proporcionar la solución.

La TH proporciona una respuesta correcta; es decir, estimaciones correctas de los parámetros utilizados para especificar la línea, siempre y cuando el número de puntos colineales a lo largo de esa línea exceda el número de puntos colineales en cualquier otra línea de la imagen. Como tal, la TH tiene las mismas propiedades con respecto al ruido y la *oclusión* como coincidencia del modelo. Sin embargo, la no linealidad del parámetros y la discretización producen acumuladores ruidosos. Un problema importante en la implementación de la TH básico para líneas es la definición de un espacio de acumulador apropiado. En aplicación, el algoritmo de dibujo lineal de Bresenham puede usarse para trazar las líneas de los votos en el espacio acumulador. Esto asegura que las líneas de votos conectados se dibujan, a diferencia del uso de la ecuación 4.31, que puede conducir a lagunas en la línea dibujada. El mapeo (Backmapping) puede utilizarse para determinar exactamente qué puntos de borde contribuyen a un pico. Backmapping es una asignación inversa desde el espacio del acumulador hasta los datos del borde y puede permitir el análisis de forma de la imagen mediante la eliminación de los puntos de borde que contribuyeron a picos particulares, y luego por reacumulación utilizando la TH. Teniendo en cuenta que el coste computacional de la TH depende del número de puntos de borde n_e y de la longitud de las líneas formadas en el parámetro l, dando un coste computacional de $O(n_e l)$. Esto es considerablemente menor para la coincidencia del modelo, dado como $O(N^2 m^2)$.

Una forma de evitar los problemas de la parametrización cartesiana en la TH es basar el mapeo en una parametrización alternativa. Una de las técnicas más probadas es la llamada parametrización pie de normal. Esto parametriza una línea considerando un punto (x, y) como una función de un ángulo normal a la línea, pasando por el origen de la imagen. Esta da una forma de la TH para las líneas conocidas como TH polar para las líneas. El punto donde esta línea intersecta la línea en la imagen está dada por:

$$\rho = xcos(\theta) + ysin(\theta) \tag{4.32}$$

donde θ es el angulo de la línea normal a la línea en una imagen y ρ es la longitud entre el origen y el punto donde las líneas se cruzan. Recordando que dos líneas son perpendiculares si el producto de sus pendientes es -1, y considerando la geometría de la disposición de la figura 4.1 obtenemos:

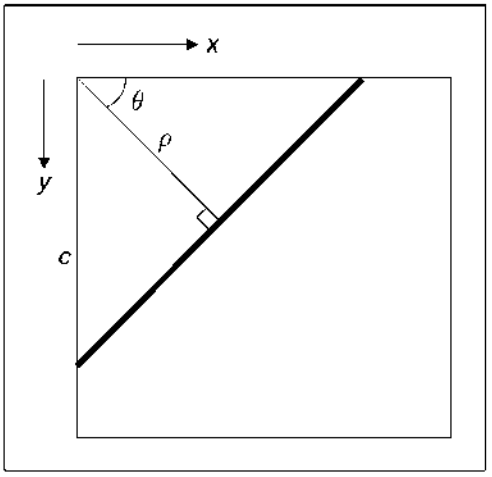

FIGURA 4.1: Consideración polar de una línea.

$$c = \frac{\rho}{sin(\theta)} m = -\frac{1}{tan(\theta)} \tag{4.33}$$

Por sustitución en la ecuación 4.28 obtenemos la forma polar (ecuación 4.32). Esto proporciona una función de asignación diferente: los votos son ahora emitidos de manera sinusoidal, en una matriz de acumuladores 2D en términos de los parámetros de interés. La ventaja de esta cartografía alternativa es que los valores de los parámetros ahora están limitados a estar dentro de un rango específico. El rango para θ es entre $0°$ y $180°$; los valores posibles de ρ son dados por el tamaño de la imagen, ya que la longitud máxima de la línea es $\sqrt{2} \times N$, donde N es el tamaño (cuadrado) de la imagen. El rango de los valores posibles ahora está fijo, por lo que la técnica es practica.

4.6. Desarrollo de las pruebas realizadas.

A continuación se presentan los resultados obtenidos al implementar los algoritmos descritos en la sección anterior.

La figura 4.2 representa la escena original con la cual se realizaron las pruebas necesarias para comprobar la investigación del trabajo.

FIGURA 4.2: Escena original.

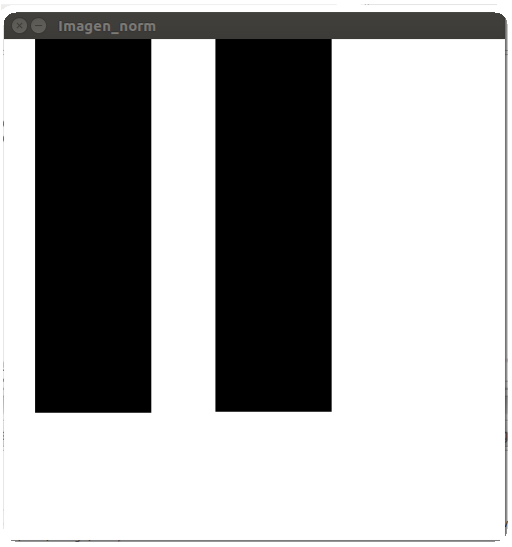

FIGURA 4.3: Escena original normalizada **v**.

Los valores de la imagen de la escena real son de $0-255$ por lo que es necesario normalizar dicha imagen en valores entre $0-1$ (Figura 4.3), debido a que al realizar las operaciones necesarias, las matrices resultan con valores muy grandes, esto impide que el ruido gaussiano tenga algún efecto en la imagen.

En la siguiente figura 4.4 se muestra la forma del filtro lineal (matriz Toeplitz) utilizado para operar la escena real.

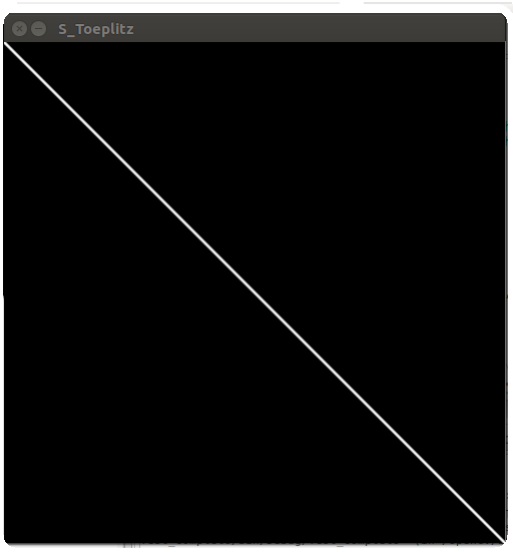

FIGURA 4.4: Representación de la matriz Toeplitz **S**.

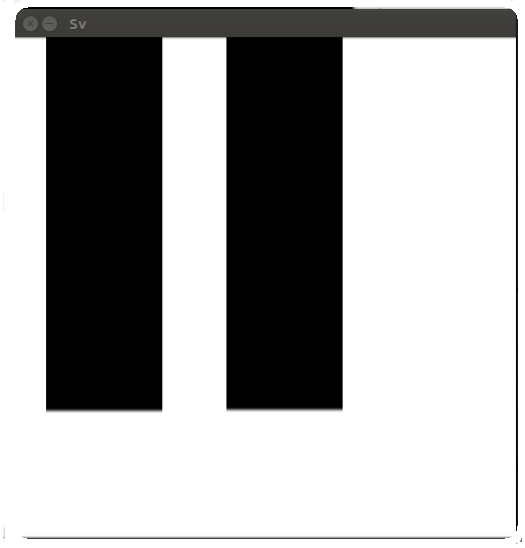

FIGURA 4.5: Representación de la matriz **Sv** (operador de convolución).

La figura 4.5 es la representación del sensor o del sistema de adquisición de datos en el que puede observarse que el borde inferior de las franjas negras a sido suavizado debido el proceso de filtrado.

La representación de la matriz de ruido gaussiano, que afectará a la representación de la imagen adquirida por el sensor, se muestra en la figura 4.6.

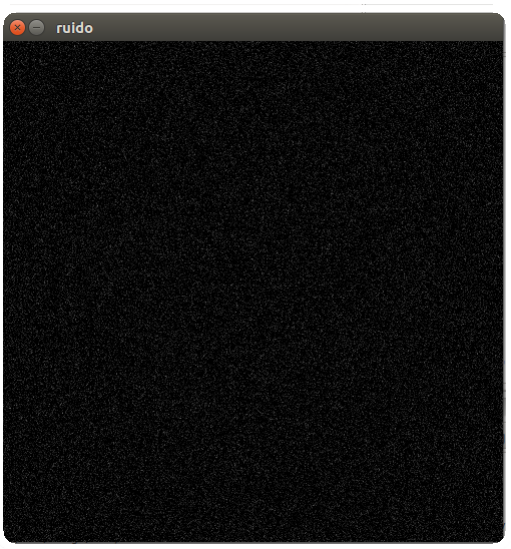

FIGURA 4.6: Representación de la matriz de ruido gaussiano.

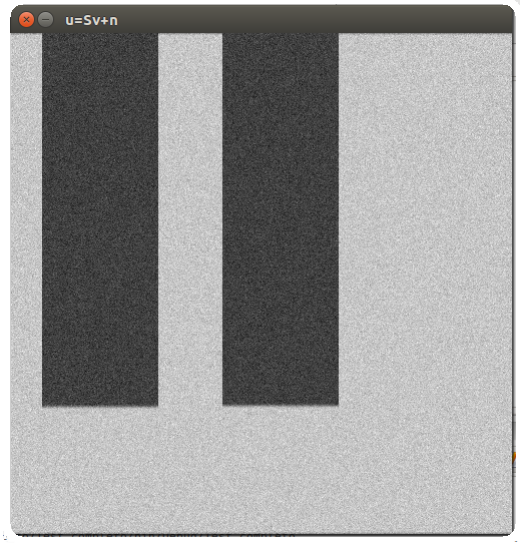

FIGURA 4.7: Representación de los datos capturados.

En la figura 4.7 se muestra la representación de la matriz de la ecuación de observación **u** la cual muestra los datos capturados.

Al realizar el proceso de estimación usando el método de mínimos cuadrados restringidos, se obtiene la imagen mostrada en la Figura 4.8, que demuestra una mejora reduciendo la cantidad de ruido presente en la imagen capturada .

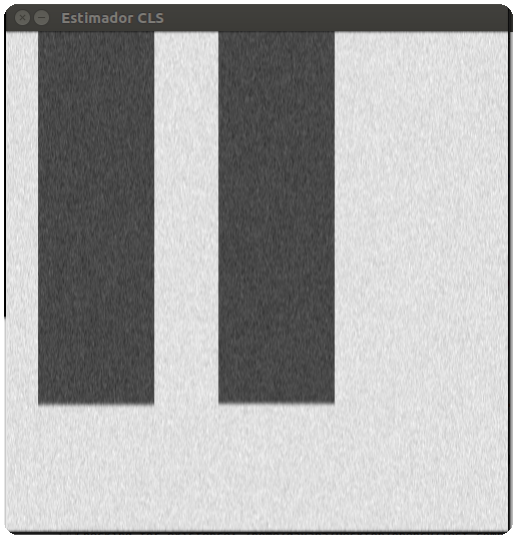

FIGURA 4.8: Imagen recuperada usando el método CLS.

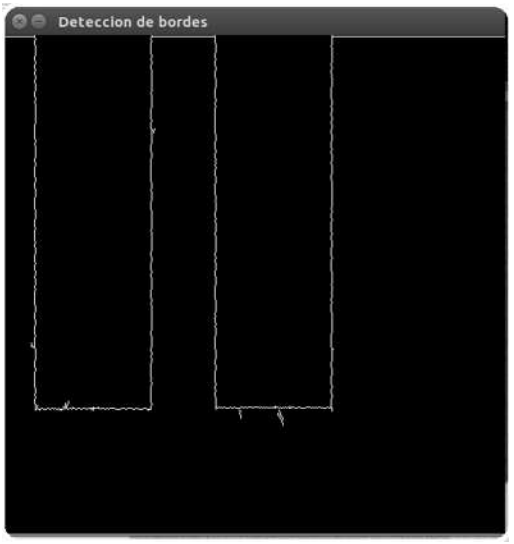

FIGURA 4.9: Detección de bordes por el algoritmo Canny.

Una vez realizado el mejoramiento de la imagen capturada, se hizo uso de la función del algoritmo Canny para la detección de bordes,visto anteriormente, la imagen obtenida por dicha función se muestra en la Figura 4.9.

Al obtener los bordes de la imagen podemos implementar la función del algoritmo de Líneas de Hough para resaltar en la imagen los bordes detectados por la función Canny. En la Figura 4.10 se muestran los bordes detectados resaltados en color burdeos.

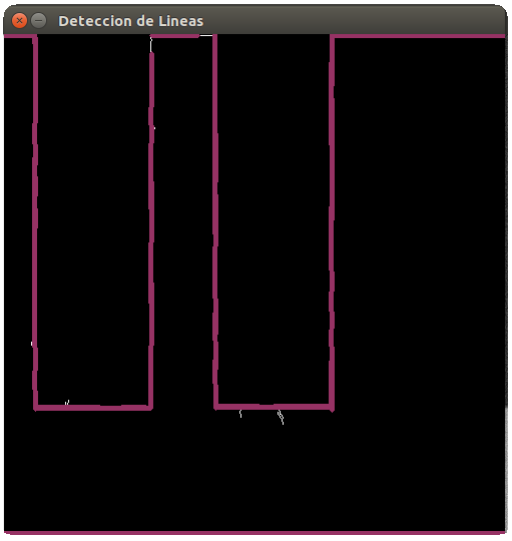

FIGURA 4.10: Detección de líneas de la matriz estimada $\hat{v}$ mediante el algoritmo Hough.

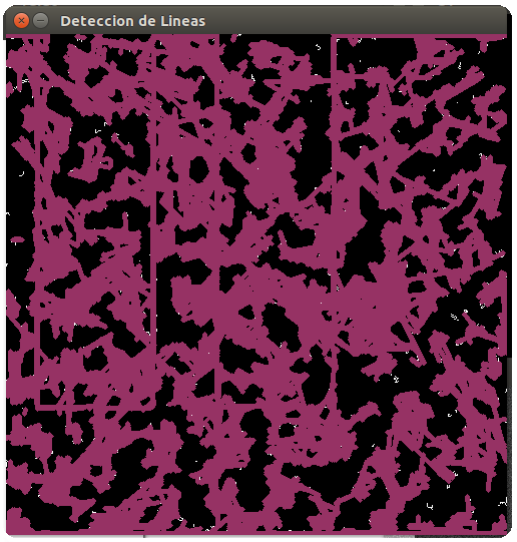

FIGURA 4.11: Detección de líneas de la image observada **u** mediante el algoritmo Hough.

Para comprobar que es conveniente realizar un preprocesamiento de imágenes, y lograr desarrollar operaciones de visión computacional, se utiliza la función del algoritmo de Líneas de Hough para detectar las líneas de los bordes sin aplicar el mejoramiento a la imagen capturada. Figura 4.11.

CAPÍTULO 5

Conclusiones y Trabajos a futuro.

5.0.1. Conclusiones.

- El objetivo del trabajo fue cumplido con éxito ya que fue posible comprobar el funcionamiento del algoritmo de estimación CLS para mejorar una imagen y las ventajas que ofrece este como etapa previa al uso de los algoritmos que permiten la recuperación de información de interés como los algoritmos de Canny y Hough, que también fueron probados.

- Los algoritmos mencionados previamente, fueron probados en un entorno de desarrollo consistente en un dispositivo embebido y el conjunto de bibliotecas OpenCV, lo cuál era el segundo objetivo del presente trabajo.

5.0.2. Trabajos futuros.

Como trabajo futuro se pretende colocar una cámara digital TRDB_D5M de 5 megapíxeles en la tarjeta DE1_SoC así como realizar una mejora al código implementado para analizar las imágenes capturadas por dicho dispositivo y montarlo sobre un bastón para realizar el procesamiento y detectar las esquinas y/o alertar de un obstáculo próximo a personas invidentes.

El trabajo realizado se considera como una base para la implementación de varios proyectos tales como la detección de baches en la ciudad, actualmente también en la industria agrícola para la selección de los alimentos, entre otros muchos trabajos de visión computacional que permiten la optimización y mejora de procesos.

5.0.3. Comentarios.

Se comprobó que es una opción muy factible el procesamiento previo de imágenes con el uso del estimador CLS para poder obtener los resultados de la detección de líneas, ya que sin la estimación se obtiene información errónea que no es de utilidad.

El uso del dispositivo DE1_SoC fue una herramienta muy útil ya que no se había trabajado antes con este tipo de dispositivos en la Facultad de Ingeniería Eléctrica y ofrece una gran ventaja sobre los dispositivos como son los microcontroladores y/o un CPU, ya que el SoC puede procesar una gran cantidad de información, y al mismo tiempo mantener una gran flexibilidad. Al utilizar el entorno de escritorio Linux Ubuntu se permite trabajar de manera más sencilla.

CAPÍTULO 6

Apéndice: Código desarrollado.

```
#include <iostream>
#include <iomanip>
#include <stdio.h>
#include <opencv2/core/core.hpp>
#include <opencv2/imgproc/imgproc.hpp>
#include <opencv2/highgui/highgui.hpp>


using namespace cv;
using namespace std;

int main(){
int i,j,m,k,H,W;

Mat src = cv::imread("./test5.png", 0);
double minVal, maxVal;
H=src.rows;
W=src.cols;

namedWindow( "Imagen", 1 );
imshow( "Imagen", src );
waitKey(0);


/**************Normaliza escena real ************************/
cv::minMaxLoc(src, &minVal, &maxVal);
Mat src_n;
src.convertTo(src_n, CV_32FC1, 1/(maxVal - minVal),
               -minVal * 1/(maxVal - minVal));
```

```
namedWindow("Imagen_norm", CV_WINDOW_AUTOSIZE);
imshow("Imagen_norm", src_n);
waitKey();

///***Vector Toeplitz con arreglo ***/
int N=7; //Tamaño del kernel: vector sinc
float sinc[7],toeplitz[500][500];

for (i=0;i<N;i++){
 sinc[i]=0;
}
//
for(i=0;i<H; i++){
   for(j=0; j<W ;j++){
       toeplitz[i][j]=0;
        }
}
///*** Creación del vector con sinc **/
for (i=-N/2,j=0;j<N;i++,j++){
    if(i==0)
        sinc[j]=1;
    else   sinc[j] = sin(i)/i;
}
/** Creacion de la matriz Toeplitz **/
for(i=0;i<H;i++){
        m=i+(N/2);
            for(k=N-1;(m>=0);k--,m--){
                if(k<0) toeplitz[i][m]=0;
                else  if(m<H) toeplitz[i][m]=sinc[k];
        }
}

/****** Convierte la matriz Toeplitz a una clase Mat******/
Mat Toeplitz(H,H, CV_32FC1, toeplitz);
namedWindow( "S_Toeplitz", 1 );
imshow( "S_Toeplitz", Toeplitz );
waitKey(0);

/********** Matriz de ruido gausiano *************/
Mat noise = Mat(src.size(),CV_32FC1);
randn(noise, 0, 0.1);
```

```
namedWindow( "ruido", 1 );
imshow( "ruido", noise );
waitKey(0);

/******** Realiza La multiplicacion de S*v ********/
Mat Sv(Toeplitz.rows, src.cols, CV_32FC1);

/*La multiplicación debe realizarse con el mismo tipo de datos
por lo que se convierte a flotantes de 32 bits y a un solo canal*/
Toeplitz.convertTo(Toeplitz, CV_32FC1);
src_n.convertTo(src_n, CV_32FC1);
Sv=Toeplitz*src_n;

cv::minMaxLoc(Sv, &minVal, &maxVal);
Mat Sv_n;
Sv.convertTo(Sv_n, CV_32FC1, 1/(maxVal - minVal),
              -minVal * 1/(maxVal - minVal));

namedWindow( "Sv", 1 );
imshow( "Sv", Sv_n );
waitKey(0);

/*****Matriz de datos u=S*v+noise ecuación de observación lineal****/

Mat u=Sv_n+noise;

minMaxLoc(u, &minVal, &maxVal); ////Encuentra la intensidad mínima y máxima
u.convertTo(u, CV_32FC1, 1/(maxVal - minVal), -minVal * 1/(maxVal - minVal));

namedWindow( "u=Sv+n", 1 );
imshow( "u=Sv+n", u);
waitKey(0);

/*** Estimación de la señal con Mínimos cuadrados restringidos (CLS)***/
int alpha=0.85;
Mat St = Toeplitz.t();

Mat W_inv;
Mat W_est=(invert((St*Toeplitz)+(alpha*(Mat::eye(src.rows,src.rows,
             CV_32FC1))), W_inv)*St);

Mat CLS = W_est*u;
```

```
minMaxLoc(CLS, &minVal, &maxVal); //Encuentra la intensidad mínima y máxima
CLS.convertTo(CLS, CV_32FC1, 1/(maxVal), 0);

namedWindow( "Estimador CLS", 1 );
imshow( "Estimador CLS", CLS);
waitKey(0);

Mat dst,color_dst;

//u.convertTo(u, CV_8UC1,255,0);
//Canny( u, dst , 50 , 200 , 3 );

    CLS.convertTo(CLS, CV_8UC1,255,0);
    Canny( CLS, dst , 50 , 200 , 3 );


    namedWindow( "Deteccion de bordes ", 1 );
    imshow( "Deteccion de bordes", dst );
    waitKey(0);


    cvtColor( dst, color_dst, CV_GRAY2BGR );

#if 0
    vector<Vec2f> lines;
    HoughLines( dst, lines, 1, CV_PI/180, 100 );

    for( size_t i = 0; i < lines.size(); i++ ){
        float rho = lines[i][0];hh
        float theta = lines[i][1];
        double a = cos(theta), b = sin(theta);
        double x0 = a*rho, y0 = b*rho;
        Point pt1(cvRound(x0 + 1000*(-b)),
                  cvRound(y0 + 1000*(a)));
        Point pt2(cvRound(x0 - 1000*(-b)),
                  cvRound(y0 - 1000*(a)));
        line( color_dst, pt1, pt2, Scalar(0,0,255), 3, 8 );
    }
#else
    vector<Vec4i> lines;
    HoughLinesP( dst, lines, 1,0.01 , 19, 5, 10 );//CV_PI/180
```

```
    for( size_t i = 0; i < lines.size(); i++ )
    {
        line( color_dst, Point(lines[i][0], lines[i][1]),
              Point(lines[i][2], lines[i][3]), Scalar(100,50,150), 3, 8 );
    }
#endif

    namedWindow( "Deteccion de Lineas ", 1 );
    imshow( "Deteccion de Lineas ", color_dst );
    waitKey(0);

    return 0;
}
```

Bibliografía

[1] Alberto Carlos Salas Mier. *Método de mejoramiento de imágenes de radar de apertura sintética fraccional basado en el framework de regularización estructurada l2-l1.* Tesis maestria CINVESTAV del IPN Unidad Guadalajara, 2012.

[2] Enrique Herrera Pérez. *Comunicaciones I Señales, Modulación y Transmisión.* Limusa, 2013.

[3] Yuriy Shkvarko. Probability and statistic. course notes. •, .

[4] OpenCV team. Basic structures. `http://docs.opencv.org/2.4/modules/core/doc/basic_structures.html?highlight=mat#vec`, . [09/02/17].

[5] Doxygen. Operations on arrays. `http://docs.opencv.org/3.1.0/d2/de8/group__core__array.html#gaeff1f61e972d133a04ce3a5f81cf6808`, . [15/06/2017].

[6] OpenCV team. Load and display an image. `http://docs.opencv.org/2.4/doc/tutorials/introduction/display_image/display_image.html`, . [03/02/2017].

[7] The Linux Foundation. What is linux? `http://www.linux.com/what-is-linux`. [28/07/17].

[8] Yuriy Shkvarko. Digital signal processing. course notes. •, .

[9] Kingston Technology Corporation. Tarjeta sd. `https://www.kingston.com/latam/flash/sd_cards`. [11/08/17].

[10] Anil K. Jain. *Fundamentals of Digital Image Processing.* Prentice Hall, 1988.

[11] Vicente Blas Cabello Hernández. *Tesis Licenciatura Reconocimiento inteligente de flujo vehicular basado en el procesamiento de imágenes.* Universidad Michoacana de San Nicolas de Hidalgo, 2010.

[12] José Martín Ruiz Pérez. *Tesis Licenciatura Índices de proximidad en el reconocimiento de voz.* Universidad Michoacana de San Nicolas de Hidalgo, 2013.

[13] José Francisco Mercado Miramontes. *Tesis Licenciatura Diseño e implementación de un dispositivo dtector de profundidad de apoyo para personas invidentes.* Universidad Michoacana de San Nicolas de Hidalgo, 2013.

[14] Carlos Alberto Trejo Seráfico. *Tesis Licenciatura Aplicación de visión con LabView en el seguimiento de línea para un móvil.* Universidad Michoacana de San Nicolas de Hidalgo, 2013.

[15] Hector Ortega Reyes. *Tesis Licenciatura Robot movil seguidor de linea con Raspberry Pi y biblioteca de vision computacional (OpenCV).* Universidad Michoacana de San Nicolas de Hidalgo, 2015.

[16] Misael Armenta Nieto. *Tesis Licenciatura Sistema de visión computacional para el conteo de personas, heuristica para el seguimiento de objetos.* Universidad Michoacana de San Nicolas de Hidalgo, 2015.

[17] L.E. Franks. *Signal Theory.* Prentice Hall, 1969.

[18] Gene H. Golub and Charles F. Van Loan. *Matrix Computations.* No, 1996.

[19] Irwin M. Jacobs John M. Wozencraft. *Principles of Communication Engineering.* Waveland Press, 1965.

[20] John G. Proakis and Dimitris G. Manolakis. *Tratamiento Digital de Señales.* Person Educación, 2007.

[21] Hwei P. Hsu. *Análisis de Fourier.* Prentice Hall, 1987.

[22] Hwei P. Hsu. *Sistemas y Señales.* Mc Graw Hill, 2011.

[23] OpenCV team. About|opencv. `http://opencv.org/about.html.`, . [03/07/17].

[24] Altera Corporation. *Cyclone V Hard Processor System Technical Reference Manual.* Altera Corporation, 2012.

[25] Terasic Technologies. *DE1-SoC Getting Started Guide.* Terasic Technologies, 2014.

[26] OpenCV team. Cross compilation for arm based linux systems. `http://docs.opencv.org/2.4/doc/tutorials/introduction/crosscompilationarm_crosscompile_with_cmake.html.`, . [03/07/2017].

[27] MortenMacFly Code::Blocks. The open source, cross platform, free c, c++ and fortran ide. `http://www.codeblocks.org`. [20/07/2017].

[28] Doxygen. cv::mat class reference. `http://docs.opencv.org/3.1.0/d3/d63/classcv_1_1Mat.html`, . [10/06/2017].

[29] Mark Nixon and ALberto Aguado. *Feature Extraction and Imagen Processing*. Elsevier Ltd, 2008.

[30] Doxygen. Feature detection. `http://docs.opencv.org/3.1.0/dd/d1a/group__imgproc__feature.html#ga8618180a5948286384e3b7ca02f6feeb`, . [20/06/2017].