

UNIVERSIDAD CATÓLICA DE SANTA MARÍA
FACULTAD DE CIENCIAS E INGENIERÍAS FÍSICAS Y FORMALES
PROGRAMA PROFESIONAL DE INGENIERÍA MECÁNICA, MECÁNICA
ELÉCTRICA Y MECATRÓNICA



**“DISEÑO E IMPLEMENTACIÓN DE UN SISTEMA DE
VIDEO INTELIGENTE PARA EL SEGUIMIENTO DE
PATRONES”**

Tesis para Optar el Título Profesional de

Ingeniero Mecatrónico

Presentado por el Bachiller:

BALUARTE HURTADO, Roberto

Arequipa – Perú

2014

DEDICATORIA

A Roberto Santiago y María Esther, porque su ejemplo y apoyo constante me mantienen convencido de que no podría haberme tocado mejor hogar.

A Katty y Rodri, porque, de manera directa o indirecta, tienen que ver con cada aspecto de mi vida del que podría sentirme orgulloso.

ÍNDICE

CAPITULO I – MARCO METODOLÓGICO	10
1.1. Planteamiento del problema	10
1.2. Justificación	10
1.3. Hipótesis	11
1.4. Objetivos	11
1.4.1. Objetivos generales	11
1.4.2. Objetivos específicos	11
1.5. Alcances y limitaciones	12
1.6. Antecedentes	13
CAPITULO II – MARCO TEÓRICO	14
2.1. Video	14
2.1.1. Partes de la señal de vídeo analógica	15
2.1.2. Características de los flujos de vídeo	17
2.1.3. Estándares de video	20
2.2. Imagen digital	22
2.2.1. Modelos de colores	23
2.2.2. Profundidad de color	27
2.2.3. Tamaños de canal optimizados	27
2.3. Procesamiento de imágenes	28
2.3.1. Procesamiento óptico	29
2.3.2. Procesamiento digital	31
2.3.3. Proceso de filtrado	33
2.4. Lógica Difusa	39
2.4.1. Funcionamiento	40
2.4.2. Teoría de conjuntos difusos	43
2.4.3. El controlador difuso	46
2.4.4. Aplicaciones	51
2.4.5. Ventajas e inconvenientes	53

2.5. Estándar RS-485	54
2.5.1. Especificaciones	55
2.5.2. Aplicaciones	55
2.6. Protocolo Pelco-D	56
2.6.1. Estructura de la trama	56
2.6.2. Comandos especiales	57
CAPITULO III – DISEÑO DEL SISTEMA	59
3.1. Captura de imágenes	65
3.2. Procesamiento de la imagen	66
3.3. Evaluación de datos.....	68
3.3.1. Fusificación:.....	69
3.3.2. Defusificación.....	73
3.3.3. Reglas	75
3.3.4. Aplicación del controlador:.....	77
3.3.5. Preparación de la salida.....	78
3.4. Transmisión de datos.....	88
3.5. Movimiento de la cámara	92
CAPITULO IV – PRUEBAS DE FUNCIONAMIENTO	93
CONCLUSIONES.....	110
RECOMENDACIONES	112
BIBLIOGRAFÍA.....	113
ANEXOS	115

RESUMEN

El presente trabajo muestra un sistema automático de seguimiento que utiliza una cámara con una aplicación de procesamiento de imágenes como sensor, una aplicación de lógica difusa como controlador y un mecanismo de direccionamiento como actuador, enlazándolos entre si sobre una plataforma libre, como lo es el sistema operativo Linux, y estructurado en un lenguaje de programación también libre, como lo es Python.

Propone además promover la investigación en el campo de la visión artificial y la utilización de dispositivos de uso cotidiano como herramientas de desarrollo, ya que su bajo costo y su buena versatilidad los hace idóneos para este propósito.

ABSTRACT

The following shows an automated tracking system that uses a camera with an image processing application as a sensor, a fuzzy logic application as a controller and a directional mechanism as an actuator, linking each other over an open source platform such as Linux operating system and structured over an open source programming language such as Python.

It also promotes the research in the fields of artificial vision and the usage of everyday devices as developing tools, due to their low cost and good versatility makes them ideal for this purpose.

INTRODUCCIÓN

La función principal de un ingeniero es la de realizar diseños o desarrollar soluciones tecnológicas a necesidades sociales, industriales o económicas. En lo general, cuando los usuarios de diferentes sistemas solicitan aplicaciones específicas, no saben todo lo que hay detrás de este requerimiento, pero el impulso de satisfacer dichas necesidades lo pone sobre el camino y las herramientas con que cuentan los ayudan a llegar a estas soluciones.

Los ingenieros utilizan el conocimiento de la ciencia, la matemática y la experiencia como herramientas, pero hay dos particulares que no se desarrollan con el estudio ni con la experiencia y que hacen que el trabajo de un ingeniero deje de ser una buena solución para convertirse en una obra trascendental: “el ingenio y la creatividad”.

El campo de la videovigilancia no es la excepción, ya que se presentan necesidades específicas para los diferentes usuarios y muchas de ellas suelen requerir tecnologías muy complejas y mano de obra permanente, incrementando los costos de los sistemas.

El presente proyecto propone una herramienta, un tanto general, pero que por su versatilidad y bajo costo puede ser pulido para aplicaciones más específicas y así satisfacer diferentes necesidades en dicha área.

ÍNDICE DE FIGURAS

Figura 01 El caballo en movimiento de Eadweard Muybridge	18
Figura 02 Canales del modelo RGB.....	23
Figura 03 Procesamiento óptico de imágenes.	30
Figura 04 División de imagen en pixeles.....	31
Figura 05 Procesamiento digital de imágenes.....	32
Figura 06 Filtros en el dominio de la frecuencia y el espacio.....	35
Figura 07 Representación de un kernel.....	37
Figura 08 Esquema de funcionamiento típico	42
Figura 09 Ejemplo de conjuntos difusos.....	45
Figura 10 Función de pertenencia para conjunto difuso triangular	45
Figura 11 Función de pertenencia para conjunto difuso trapezoidal	46
Figura 12 Estructura de un modelo difuso	47
Figura 13 Funcionamiento de un sistema difuso Mamdani	49
Figura 14 Funcionamiento de un sistema difuso Sugeno.....	50
Figura 15 Tabla de comandos especiales.....	58
Figura 16 Diagrama de bloques del sistema	59
Figura 17 Cámara PTZ Nova NV12X.....	60
Figura 18 Conversor EasyCap	61
Figura 19 Notebook Toshiba NB200	62
Figura 20 Placa para transmisión de datos.....	64
Figura 21 Sets difusos de la entrada “dist”.....	71
Figura 22 Sets difusos de la entrada “movi”	73
Figura 23 Sets difusos de la salida “velo”	74
Figura 24 Ejemplo de entrada “dist”	84
Figura 25 Ejemplo de entrada “movi”	85
Figura 26 Ejemplo Inferencia.....	86
Figura 27 Ejemplo de Salida	87
Figura 28 Diagrama del circuito de la interfaz	91
Figura 29 Circuito impreso para la interfaz	91
Figura 30 Impresión de los parámetros “t”, “dx”, “dy” , “vx” y “vy”	95
Figura 31 Comparación de la imagen procesada con la original.....	97
Figura 32 Comparación de la imagen procesada con la original.....	97
Figura 33 Comparación de la imagen procesada con la original.....	98
Figura 34 Comparación de la imagen procesada con la original.....	99
Figura 35 Comparación de la imagen procesada con la original.....	99
Figura 36 Comparación de la imagen procesada con la original.....	100
Figura 37 Comparación de la imagen procesada con la original.....	101
Figura 38 Comparación de la imagen procesada con la original.....	101
Figura 39 Comparación de la imagen procesada con la original.....	102
Figura 40 Comparación de la imagen procesada con la original.....	102
Figura 41 Comparación de la imagen procesada con la original.....	103
Figura 42 Comparación de la imagen procesada con la original.....	103
Figura 43 Imagen del objetivo en una posición aleatoria (a).....	106
Figura 44 Imagen del objetivo en una posición aleatoria (b).....	106

Figura 45 Imagen del objetivo en una posición aleatoria (c).....	107
Figura 46 Imagen del objetivo en una posición aleatoria (d).....	107
Figura 47 Imagen del objetivo en una posición aleatoria (e).....	108
Figura 48 Imagen del objetivo en una posición aleatoria (f).....	108
Figura 49 Grafica de comparación posición-velocidad en X e Y	109



CAPITULO I – MARCO METODOLÓGICO

1.1. Planteamiento del problema

Se pretende diseñar un sistema capaz de seguir un objetivo en movimiento, el sistema estará dotado de una cámara montada en una plataforma móvil. Se deben capturar imágenes frecuentemente para ser analizadas con técnicas de procesamiento digital de imágenes, de esta etapa se obtendrá la posición del objeto a seguir dentro de la imagen (si está en ella). Haciendo uso de un sistema con inteligencia artificial se procesará esta información para determinar con qué velocidad y en qué dirección se deben mover los motores para que el objeto esté siempre en medio de las imágenes capturadas. Se debe implementar un subsistema de lógica difusa, se realizarán pruebas hasta obtener el mejor modelo, cuya respuesta sea lo más rápida y precisa posible.

1.2. Justificación

Hoy en día contamos con infinidad de recursos tecnológicos a nuestro alrededor, eso permite que estas herramientas sean poco costosas y de fácil acceso, a causa de eso desde hace algunos años se ha generado un fenómeno en el que cada vez son más los sistemas que migran hacia la automatización debido a la importante reducción de los costos de operación en cuanto a mano de obra se refiere. La visión artificial es una de las áreas de mayor interés dentro de

este fenómeno, se utilizan por mencionar solo un ejemplo, en plantas embotelladoras para detectar grietas en botellas y para descartar aquellas que no fueron llenadas al nivel requerido. Desafortunadamente estos sistemas continúan siendo muy costosos y están únicamente al alcance de las más grandes compañías.

Este proyecto pretende plantear un punto de partida en esta área, se implementará un sistema basado en técnicas de procesamiento de imágenes para seguir un patrón, utilizando una cámara móvil.

1.3. Hipótesis

Es posible implementar un sistema que siga automáticamente a un objetivo, haciendo uso de técnicas de visión artificial e inteligencia artificial.

1.4. Objetivos

1.4.1. Objetivos generales

- Diseñar e implementar un sistema de video inteligente para el seguimiento de un patrón específico.

1.4.2. Objetivos específicos

- Capturar imágenes desde una cámara de videovigilancia y enviarlas a un controlador encargado de procesarlas.

- Aplicar técnicas de procesamiento de imágenes para obtener las coordenadas del patrón propuesto dentro de las imágenes previamente capturadas.
- Mover la cámara de modo que el patrón propuesto siempre se encuentre en el centro de la imagen, utilizando métodos de inteligencia artificial.

1.5. Alcances y limitaciones

El sistema importara una señal de video digital, ya sea de una cámara digital o de una cámara analógica previa a un convertidor analógico-digital, analizará esas imágenes y determinará en qué sentido y con qué rapidez se debe mover la cámara para conseguir que el objetivo se encuentre en el centro de la imagen. El sistema será capaz de seguir un patrón específico que será programado.

El procesamiento se realizará en un computador y se implementará una interface para controlar el mecanismo encargado del movimiento de la cámara. La comunicación entre el PC y la interface se realizará por medio de transmisión serial.

La principal limitación del sistema está dada por la velocidad de procesamiento de controlador, de manera que se procurara dejar la menor cantidad de cálculos al sistema.

1.6. Antecedentes

Detección y seguimiento de objetos con cámaras en movimiento

Hector Lopez Paredes

Universidad Autónoma de Madrid, 2011

Diseño de una mini-cámara motorizada para seguimiento de objetos

J.A. Corrales, P. Gil, F. A. Candelas y F. Torres

Universidad de Alicante

Object Tracking and Kalman Filtering

Department of Computer Engineering

University of California and Santa Cruz

CAPITULO II – MARCO TEÓRICO

2.1. Video

El video es la tecnología de la grabación, almacenamiento, transmisión de imágenes y procesamiento para su reconstrucción por medios electrónicos digitales o analógicos de una secuencia de imágenes que representan escenas en movimiento. Este término se aplica generalmente a la señal de video o a la abreviatura del nombre completo de esta.

Esta tecnología se desarrolló, en un principio, orientada a los sistemas de televisión, pero dada su difusión se continuó desarrollando hasta derivar en diferentes formatos que permiten su grabación, procesamiento y hasta su transmisión por medios masivos como Internet.

El nombre aplica, en algunos países, a la grabación de imágenes y sonido sobre una cinta magnética, como VHS o Betamax, que es como antiguamente se guardaba dicha información.

La señal de video puede venir en diferentes estándares, pero en general, está formada por líneas que forman cuadros, y estos cuadros divididos en dos campos que portan la información de color y luz de la imagen. Los estándares definen la cantidad de líneas, cuadros y el formato en que llevan la información del color.

La señal de video tiene una amplitud de 1Vpp con usos diferenciados para la parte positiva y la parte negativa de esta. La información de la imagen va por la parte de la señal con valor positivo que puede estar desde 0V, que corresponde al

negro, hasta 0.7V para el blanco. La parte negativa de la señal corresponde a los sincronismos, pulsos que llegan hasta los -0.3V. Como se indicó anteriormente, existen diferentes estándares que definen los formatos con los que se construirá esta señal, sobretodo en cuestión de informática.

2.1.1. Partes de la señal de vídeo analógica

La señal está formada por lo que se conoce como luminancia, cromancia y por los sincronismos. Como se mencionó, la amplitud va desde los -0.3V hasta los 0.7V. La señal propiamente dicha es la que se refiere a la luminancia con los sincronismos, sobre la cual se monta la señal de cromancia con su propia señal de sincronía, conocida como la salva de color. De esta forma tenemos la conformación de la señal de video a color.

El ancho de banda de la luminancia está, generalmente, en el orden de 5MHz, dependiendo del sistema empleado. La cromancia ha sido modulada en cuadratura (amplitud y fase). La portadora se conoce como “subportadora de color” y su frecuencia está muy cercana a la parte alta de la banda. Esta frecuencia tiene relación con las demás frecuencias fundamentales de la señal de video que tienen como referencia a la frecuencia de campo, que a su vez toma como base la frecuencia de la red de suministro eléctrico, es decir, 50Hz en Europa y 60Hz en diferentes partes de América.

2.1.1.1. Información de la imagen

La información de la imagen está conformada por la combinación de luz y color. La luz puede definir una imagen en blanco y negro. Sobre esta se aplica el color, cuya señal depende el estándar que se maneje. Existen diferentes estándares para la codificación del color, entre los que tenemos el NTSC, SECAM y PAL.

2.1.1.2. Sincronismos

Se manejan tres tipos de sincronismos: los de línea, los de campo y los referentes al color.

Los sincronismos de línea se conocen también como horizontales y se encargan de definir donde comienzan y acaban las líneas que componen la imagen de video. Se dividen en: pórtico trasero, pórtico delantero y pulso de sincronismo.

En cuanto al color, todos los estándares tienen una portadora con la información respectiva, cuya modulación depende de dicho estándar. En NTSC se utiliza una modulación en cuadratura, reservando la amplitud para la saturación y la fase para el tinte. En PAL se hace lo mismo, pero se alterna 180° en cada línea la fase de la portadora para compensar distorsiones de la transmisión. En SECAM se modula cada componente del color en las respectivas líneas.

Los sincronismos de campo se conocen también como verticales y son los que marcan el comienzo y el fin de cada campo. Se dividen también en: pulso de preigualacion, pulso de postigualacion, pulso de sincronismo y líneas de guarda.

La frecuencia de los pulsos de sincronismo depende del sistema de televisión: en América (con ciertas excepciones) se usa frecuencia de línea de 525 líneas por cuadro, mientras que en Europa se utilizan 625 líneas por cuadro, y 50 campos por segundo. Estos valores derivan de la frecuencia de la red eléctrica, que era la frecuencia a la que referenciaban los osciladores de los receptores.

2.1.2. Características de los flujos de vídeo

2.1.2.1. Número de fotogramas por segundo

Es la velocidad con la que los fotogramas se van a mostrar uno tras otro para crear la sugerión de movimiento. Antiguamente se utilizaban cámaras mecánicas que cargaban de 6 a 8 fps. Actualmente los estándares PAL y SECAM tienen una velocidad de 25fps, mientras que NTSC tiene una velocidad de 30fps. Para lograr la ilusión de una imagen en movimiento, se debe mostrar por lo menos 15fps. Existen cámaras de alta velocidad capaces de capturar 700fps en su mejor resolución, alcanzando las 17000fps con resoluciones más bajas.

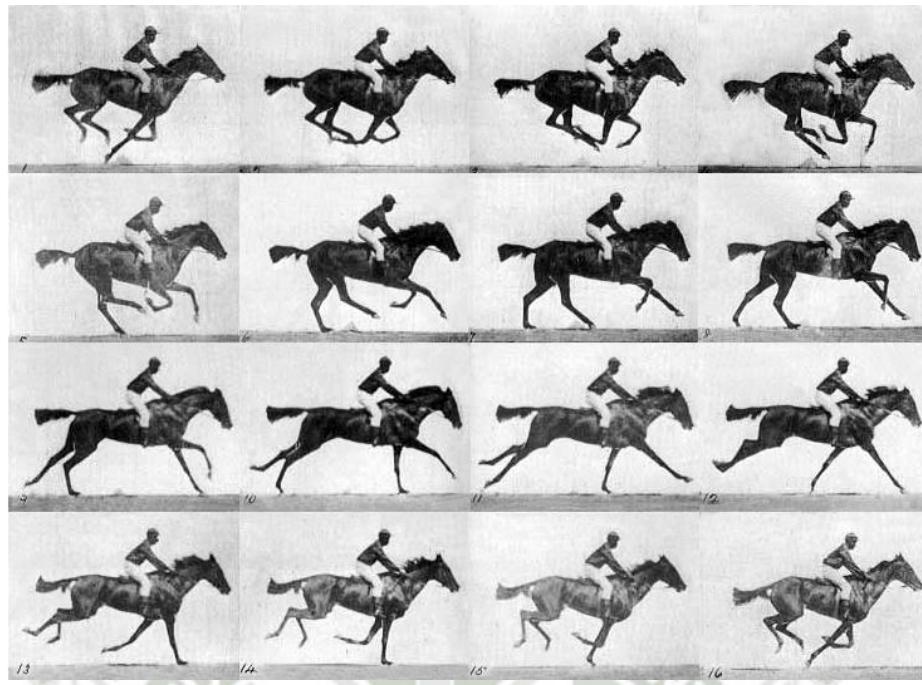


Figura 01 El caballo en movimiento de Eadweard Muybridge

Fuente: wikipedia.com

2.1.2.2. Sistemas de barrido

Entrelazado

La exploración entrelazada se desarrolló con la finalidad de evitar el parpadeo que se presenta cuando se reproduce una imagen de televisión en un tubo de imagen, debido a la persistencia de los luminofosforos que componen la pantalla

La exploración entrelazada 2/1 consiste en dividir cada cuadro en dos semicuadros llamados campos. Las líneas de cada campo están imbricadas alternadamente entre si. Uno de estos campos contiene las líneas pares y se le conoce como “campo par” y el otro contiene las líneas impares y se le conoce

como “campo impar”. Al comienzo de cada uno de estos cuadros va un sincronismo vertical.

Las especificaciones abreviadas de la resolución de vídeo a menudo incluyen una i para indicar entrelazado. Por ejemplo, el formato de vídeo NTSC es a menudo especificado como 480i60, donde 480 indica la línea vertical de resolución, i indica entrelazado, y el 60 indica 650 cuadros (30 imágenes) por segundo.

Progresivo

Este sistema se utiliza generalmente en televisión digital. En el televisor, una memoria digital almacena el campo hasta que esté completo y luego lo proyecta con una velocidad de refresco de 100Hz.

2.1.2.3. Resolución del video

El tamaño de una imagen de video se mide en líneas de barrido horizontal y vertical para video analógico y en píxeles para video digital. En el dominio analógico, se conserva el número de líneas activas de barrido, pero varía el número de líneas horizontal, según la calidad del video. En el dominio digital la televisión de definición estándar se define como 720/704/640 × 480i60 para NTSC y 768/720 × 576i50 para resolución PAL o SECAM.

Los nuevos televisores de alta definición (HDTV) son capaces de resoluciones de hasta 1920 × 1080p60, es decir, 1920 píxeles por línea de barrido por 1080 líneas, a 60 fotogramas por segundo.

2.1.2.4. Relación de aspecto

La relación de aspecto es la relación entre el ancho de la pantalla entre la altura de la misma. El estándar era de 4/3 hasta que se comenzó a estandarización de la HDTV, la cual dejó una relación de 16/9.

Una imagen de 4/3 que se vaya a ver en una pantalla de 16/9 puede presentarse de tres formas diferentes:

- Con barras negras verticales a cada lado
- Agrandando la imagen hasta que ocupe hasta los lados
- Estirando la imagen hasta que ocupe todo el tamaño de la pantalla

Una imagen de 16/9 que se vaya a ver en una pantalla de 4/3, de forma similar, tiene tres formas de verse:

- Con barras negras horizontales arriba y abajo
- Agrandando la imagen hasta que ocupe todo el espacio verticalmente
- Estirando la imagen hasta que ocupe todo el tamaño de la pantalla

2.1.3. Estándares de video

2.1.3.1. Estándares de dispositivos

Analógicos

- MAC (Europa - Obsoleta)

- MUSE (Japón-analog HDTV)
- NTSC (EE. UU., Canadá, Japón, etc.)
- PAL (Europa, Asia, Australia, etc.)
- PALplus (extensión PAL; solo en Europa.)
- PAL-M (variación de PAL; Brasil.)
- SECAM (Francia, la antigua URSS y África central.)

Digitales

- ATSC (EE. UU., Canadá, México, etc.)
- DVB-T (Europa, *Digital Video Broadcasting*)
- ISDB-T (Japón, Argentina, Brasil, Chile, Perú, etc., *Servicios Digitales Integrados de Broadcast*)

2.1.3.2. Estándares de conectores

- Vídeo compuesto (1 RCA o BNC)
- Vídeo componentes (3 RCA o BNC)
- D4 video connector (nuevo para HDTV)
- S-Video (para video separado, 1 mini-DIN)
- SCART Euroconector / Peritel (usado en Europa)
- DVI (sólo video no comprimido). HDCP opcional
- HDMI (video y audio no comprimido). HDCP mandato.
- RFs (para *Radiofrecuencia* conector coaxial)

- BNC (*Bayonet Niell-Concelman*)
- conector C (conector *Concelman*)
- conector GR (conector *General Radio*)
- IEC 169-2 (*IEC connector*, usado habitualmen en Gran Bretaña)
- TNC connector (*Threaded Niell-Concelman*)
- UHF (e.g. PL-259/SO-239)
- SDI y HD-SDI
- VGA (DB-9/15 or *mini sub D15*)
- Mini-VGA (usado por ordenadores portátiles)

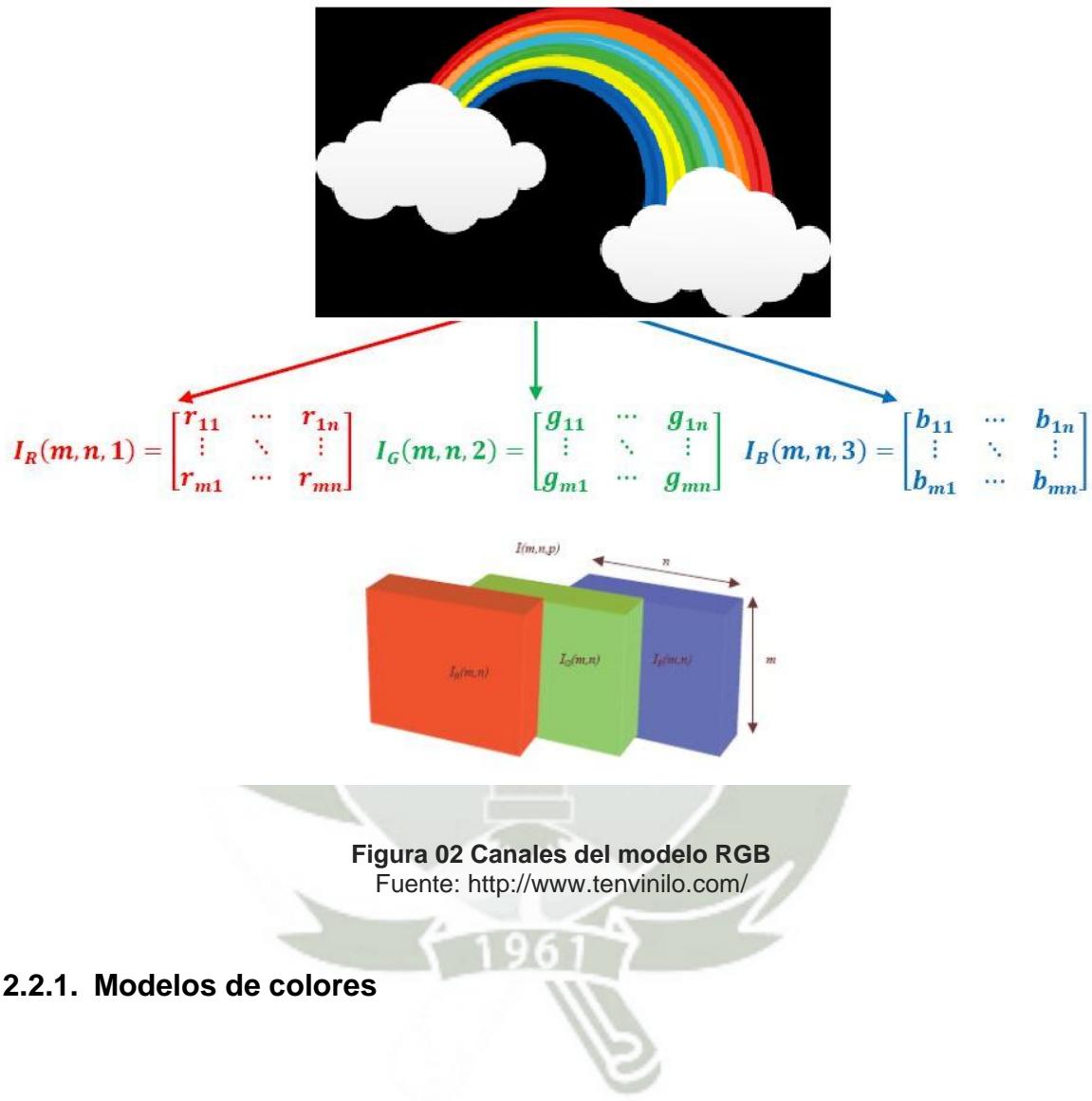
2.2. Imagen digital

Las imágenes digitales están formadas por pixeles, que son la representación más pequeña que existe para el tamaño de la imagen. Estos pixeles están formados por combinaciones de colores primarios a diferentes intensidades.

Un canal es una imagen similar a la imagen en color, del mismo tamaño, pero conformada por una escala del color primario que la gobierna. Independientemente del formato en el que se almacena la imagen, los canales de color pueden determinarse siempre, en la medida en que la imagen final pueda ser renderizada.

El concepto de canales se extiende más allá del espectro visible en imágenes multiespectrales e hyperespectrales. En ese contexto, cada canal

corresponde a un rango de longitudes de onda y contiene información espectroscópica. Los canales pueden tener múltiples anchos y rangos.



2.2.1. Modelos de colores

2.2.1.1. RGB

Las siglas RGB significan “Red, Green, Blue” y tiene tres canales, como su nombre lo indica: Rojo, verde y azul. Esto obedece a los receptores de color del ojo humano y suele ser utilizado en monitores o cámaras digitales.

El estándar de RGB maneja colores de 24 bits, separando 8 bits para cada canal. Esto nos da 256 diferentes tonalidades para cada canal, y en combinación, nos da la conocida paleta de 16 millones de colores (16 777 216 colores). También se conoce un modelo RGB de 48bits, el cual separa 16bits para cada canal.

2.2.1.2. HSV

Define un modelo de color en términos de sus componentes constituyentes en coordenadas cilíndricas:

- Tonalidad:

El tipo de color (como rojo, azul o amarillo). Se representa como un grado de ángulo cuyos valores posibles van de 0 a 360° (aunque para algunas aplicaciones se normalizan del 0 al 100%). Cada valor corresponde a un color.

- Saturación:

Se representa como la distancia al eje de brillo negro-blanco. Los valores posibles van del 0 al 100%. A este parámetro también se le suele llamar "pureza" por la analogía con la pureza de excitación y la pureza colorimétrica de la colorimetría. Cuanto menor sea la saturación de un color, mayor tonalidad grisácea habrá y más decolorado estará. Por eso es útil definir la insaturación como la inversa cualitativa de la saturación.

- Valor del color:

El brillo del color. Representa la altura en el eje blanco-negro. Los valores posibles van del 0 al 100%. 0 siempre es negro. Dependiendo de la saturación, 100 podría ser blanco o un color más o menos saturado

2.2.1.3. CMYK

El modelo CMYK (acrónimo de Cyan, Magenta, Yellow y Key) es un modelo de colores sustractivo que se utiliza en la impresión a colores. Este modelo de 32 bits se basa en la mezcla de pigmentos de los siguientes colores para crear otros más:

C = Cyan (Cian).

M = Magenta (Magenta).

Y = Yellow (Amarillo).

K = Black ó Key (Negro).

La mezcla de colores CMY ideales es sustractiva (pues imprimir cyan, magenta y amarillo en fondo blanco resulta en el color negro). El modelo CMYK se basa en la absorción de la luz. El color que presenta un objeto corresponde a la parte de la luz que incide sobre este y que no es absorbida por el objeto. El cian es el opuesto al rojo, lo que significa que actúa como un filtro que absorbe dicho color (-R +G +B). Magenta es el opuesto al verde (+R -G +B) y amarillo el opuesto al azul (+R +G -B).

2.2.1.4. RYB

El modelo de color RYB (Red, Yellow, Blue = rojo, amarillo, azul) es un modelo de síntesis sustractiva de color al igual que el modelo CMYK. Se basó en los estudios de Goethe en su libro Teoría de los colores de 1810, y adoptado desde aquellos tiempos por las escuelas de pintura y artes gráficas. Hoy en día, gracias a la evolución de la fotografía en color y la mezcla aditiva de color es fácil demostrar que dicho modelo es bastante impreciso (su corrección propiamente dicha es el modelo CMYK), aunque sigue estando presente en la teoría impartida en la educación artística, en especial en las bellas artes y el diseño gráfico. En este modelo, el verde es una mezcla de azul y el amarillo. El amarillo es el complementario del violeta y el naranja el complementario del azul. La imprecisión recae en que el modelo RYB toma a dos colores realmente secundarios - Azul y Rojo- como primarios, debido a lo cual se pierde una importante cantidad de tonos que son imposibles de obtener con este modelo. El modelo CMYK, que usa el cian en lugar del azul y magenta en lugar del rojo corrige en gran parte estas imprecisiones y por eso se utiliza industrial y científicamente.

2.2.1.5. HSL

El modelo HSL (del inglés Hue, Saturation, Lightness – Matiz, Saturación, Luminosidad), que es similar al HSV o HSI (del inglés Hue, Saturation, Intensity – Matiz, Saturación, Intensidad), define un modelo de color en términos de sus componentes constituyentes. El modelo HSL se representa gráficamente como un cono doble o un doble hexágono. Los dos vértices en el modelo HSL se corresponden con el blanco y el negro, el ángulo se corresponde con el matiz, la

distancia al eje con la saturación y la distancia al eje blanco-negro corresponde a la luminancia. Como los modelos HSI y el HSV, es una deformación no lineal del espacio de color RGB.

2.2.2. Profundidad de color

La profundidad de la imagen se refiere a la cantidad de bits necesarios para representar un color. Una vez que la imagen se digitaliza, los valores binarios dependen de la cantidad de bits como la potencia 2^n , donde n es la cantidad de bits que se le dan a dicho color. Los valores típicos son de 8bits o 16bits por canal.

2.2.3. Tamaños de canal optimizados

Como el cerebro no percibe necesariamente diferencias en cada canal para el mismo grado que en otros canales, es posible que cambiar el número de bits para cada canal resulte en un almacenamiento más óptimo; en particular, para imágenes RGB, comprimir más el canal azul y después el rojo puede ser mejor que dar el mismo espacio a cada canal. Este tipo de compresión "preferencial" es el resultado de estudios que muestran que la retina humana en realidad utiliza el canal rojo para distinguir el detalle, junto con el verde en menor medida, y usa el canal azul como información ambiental o para el fondo.

Entre otras técnicas, la compresión de vídeo con pérdida utiliza Chroma subsampling para reducir la profundidad de color de los canales (matiz y saturación), mientras mantiene toda la información sobre el brillo (Value en HSV).

2.3. Procesamiento de imágenes

El procesamiento de imágenes tiene como objetivo mejorar el aspecto de las imágenes y hacer más evidentes en ellas ciertos detalles que se desean hacer notar. La imagen puede haber sido generada de muchas maneras, por ejemplo, fotográficamente, o electrónicamente, por medio de monitores de televisión. El procesamiento de las imágenes se puede, en general, hacer por medio de métodos ópticos, o bien por medio de métodos digitales, en una computadora. Ambos métodos tienen como eje central a la Transformada de Fourier.

El teorema de Fourier afirma que una gráfica o función, cualquiera que sea su forma, se puede representar con alta precisión dentro de un intervalo dado, mediante la suma de una gran cantidad de funciones senoidales, con diferentes frecuencias. Dicho de otro modo, cualquier función, sea o no sea periódica, se puede representar por una superposición de funciones periódicas con diferentes frecuencias. El teorema nos dice de qué manera se puede hacer esta representación, pero hablar de él va más allá del objeto de este trabajo.

La variación de la irradiancia o brillantez de una imagen, medida a lo largo de una dirección cualquiera es entonces una función que se puede representar mediante el teorema de Fourier, con una suma de distribuciones senoidales de varias frecuencias. Sin entrar en detalles técnicos innecesarios, se afirma que

atenuar o reforzar individualmente algunas de estas componentes senoidales puede tener un efecto dramático en la calidad de una imagen, mejorándola o empeorándola, según el caso. Este es el fundamento del procesamiento de imágenes, tanto por medios ópticos como digitales, que ahora describiremos.

2.3.1. Procesamiento óptico

Los principios del procesamiento óptico de imágenes están bien establecidos desde el siglo pasado, cuando se desarrolló la teoría de la difracción de la luz. Sin embargo, su aplicación práctica data apenas del principio de la década de los sesenta, cuando se comenzó a disponer del rayo láser.

El procesamiento óptico se basa en el hecho de que la imagen de difracción de Fraunhofer de una transparencia colocada en el plano focal frontal de una lente es una distribución luminosa que representa la distribución de las frecuencias de Fourier que componen la imagen, a la que se le llama técnicamente transformada de Fourier.

Consideremos el arreglo óptico de la figura 03. En el plano focal frontal de la lente L1 se ha colocado la transparencia T, la cual está siendo iluminada por un haz de rayos paralelos provenientes de un láser de gas. Sobre el plano focal F1 de la lente L1 se forma una distribución luminosa que representa la transformada de Fourier de la transparencia. Si ahora se coloca otra lente L2 como se muestra en la misma figura, se puede formar una imagen de la transparencia en el plano focal

F_2 de esta lente. Si ahora se coloca cualquier objeto o diafragma sobre el plano F_1 , se pueden eliminar las porciones que se deseen de la transformada de Fourier de la transparencia, eliminando así de la imagen las frecuencias de Fourier deseadas.

Cada porción de la transformada de Fourier corresponde a una frecuencia espacial diferente sobre el objeto. Por lo tanto, mediante los diafragmas adecuados se pueden eliminar las frecuencias espaciales, llamadas también de Fourier, que se deseen quitar.

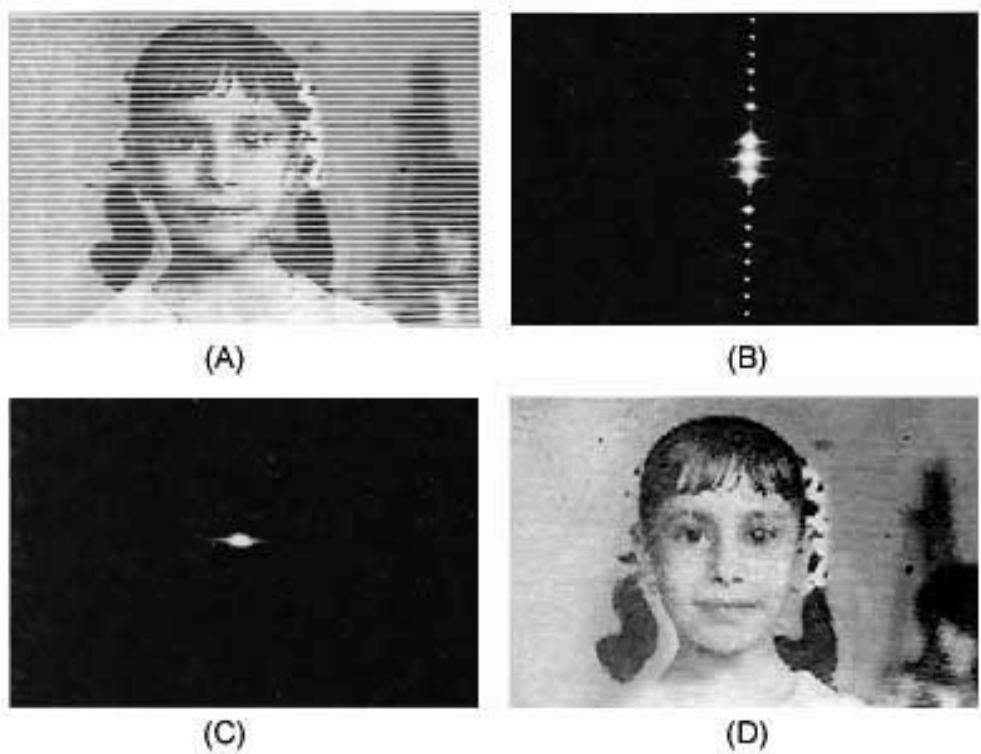


Figura 03 Procesamiento óptico de imágenes. (a) imagen original, con líneas de barrido, tipo imagen de televisión; (b) transformada de Fourier del objeto; (c) transformada de Fourier modificada, después de filtrar y (d) imagen procesada, sin las líneas de barrido.

Fuente: omega.ilce.edu.mx

2.3.2. Procesamiento digital

Al igual que en el caso del procesamiento óptico, los principios fundamentales del procesamiento digital de imágenes están establecidos hace muchos años, pero no se llevaban a cabo debido a la falta de computadoras. Con la aparición de las computadoras de alta capacidad y memoria, era natural que se comenzara a desarrollar este campo.



Figura 04 División de imagen en pixeles
Fuente: omega.ilce.edu.mx

El procesamiento digital de imágenes se efectúa dividiendo la imagen en un arreglo rectangular de elementos, como se muestra en la figura. Cada elemento

de la imagen así dividida se conoce con el nombre de pixel. El siguiente paso es asignar un valor numérico a la luminosidad promedio de cada pixel. Así, los valores de la luminosidad de cada pixel, con sus coordenadas que indican su posición, definen completamente la imagen.

Todos estos números se almacenan en matrices en la memoria de una computadora.

El tercer paso es alterar los valores de la luminosidad de los pixeles mediante las operaciones o transformaciones matemáticas necesarias, a fin de hacer que resalten los detalles de la imagen que sean convenientes. El paso final es pasar la representación de estos pixeles a un monitor de televisión de alta definición, con el fin de mostrar la imagen procesada.

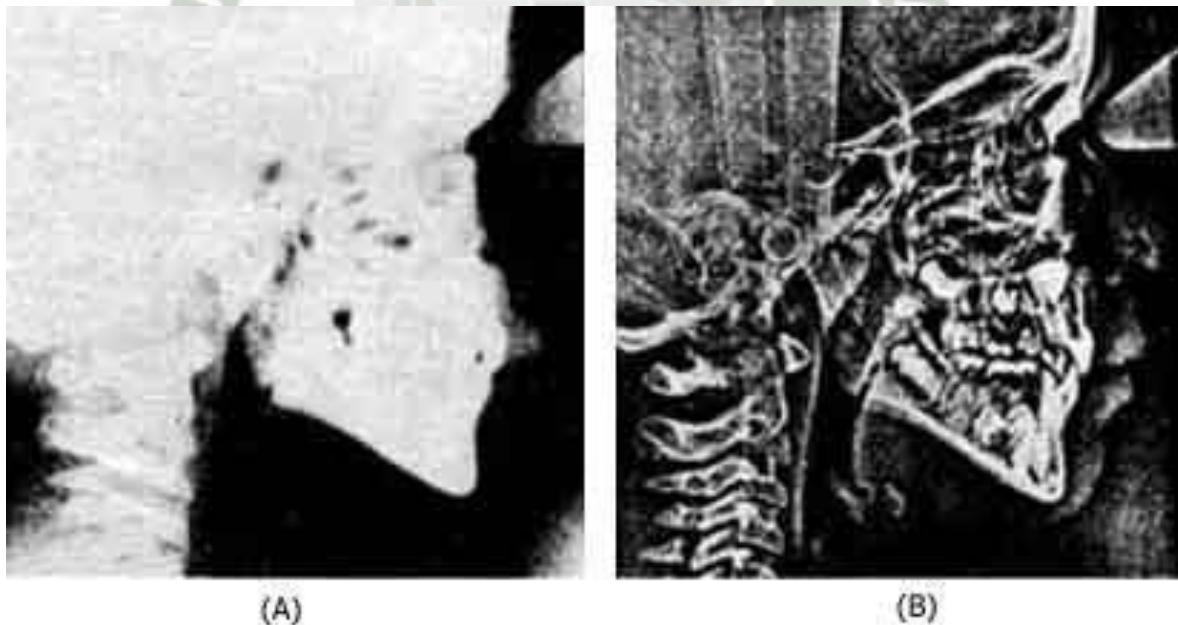


Figura 05 Procesamiento digital de imágenes. Cefalograma en el que se han reforzado las componentes de Fourier de alta frecuencia. (Tomado de S. W. Oka y H. J. Trussell, The Angle Orthodontist, 48, núm. 1, 80, 1978). (a) Imagen original y (b) imagen procesada.

Fuente: omega.ilce.edu.mx

2.3.3. Proceso de filtrado

Es el conjunto de técnicas englobadas dentro del procesamiento de imágenes cuyo objetivo fundamental es obtener, a partir de una imagen origen, otra final cuyo resultado sea más adecuado para una aplicación específica mejorando ciertas características de la misma que posibilite efectuar operaciones del procesado sobre ella.

Los principales objetivos que se persiguen con la aplicación de filtros son:

- Suavizar la imagen: reducir la cantidad de variaciones de intensidad entre píxeles vecinos.
- Eliminar ruido: eliminar aquellos píxeles cuyo nivel de intensidad es muy diferente al de sus vecinos y cuyo origen puede estar tanto en el proceso de adquisición de la imagen como en el de transmisión.
- Realzar bordes: destacar los bordes que se localizan en una imagen.
- Detectar bordes: detectar los píxeles donde se produce un cambio brusco en la función intensidad.

Por tanto, se consideran los filtros como operaciones que se aplican a los píxeles de una imagen digital para optimizarla, enfatizar cierta información o conseguir un efecto especial en ella.

El proceso de filtrado puede llevarse a cabo sobre los dominios de frecuencia y/o espacio.

2.3.3.1. Filtrado en el dominio de la frecuencia

Los filtros de frecuencia procesan una imagen trabajando sobre el dominio de la frecuencia en la Transformada de Fourier de la imagen. Para ello, ésta se modifica siguiendo el Teorema de la Convolución correspondiente:

- se aplica la Transformada de Fourier,
- se multiplica posteriormente por la función del filtro que ha sido escogido,
- para concluir re-transformándola al dominio espacial empleando la Transformada Inversa de Fourier.

$$\text{Teorema de la Convolución (frecuencia): } G(u, v) = F(u, v) * H(u, v)$$

$F(u, v)$: transformada de Fourier de la imagen original.

$H(u, v)$: filtro atenuador de frecuencias.

Como la multiplicación en el espacio de Fourier es idéntica a la convolución en el dominio espacial, todos los filtros podrían, en teoría, ser implementados como un filtro espacial.

Tipos

Existen básicamente tres tipos distintos de filtros que pueden aplicarse:

- Filtro paso bajo: atenúa las frecuencias altas y mantiene sin variaciones las bajas. El resultado en el dominio espacial es equivalente al de un filtro de suavizado, donde las altas frecuencias que son filtradas se corresponden

con los cambios fuertes de intensidad. Consigue reducir el ruido suavizando las transiciones existentes.

- Filtro paso alto: atenúa las frecuencias bajas manteniendo invariables las frecuencias altas. Puesto que las altas frecuencias corresponden en las imágenes a cambios bruscos de densidad, este tipo de filtros es usado, porque entre otras ventajas, ofrece mejoras en la detección de bordes en el dominio espacial, ya que estos contienen gran cantidad de dichas frecuencias. Refuerza los contrastes que se encuentran en la imagen.
- Filtro paso banda: atenúa frecuencias muy altas o muy bajas manteniendo una banda de rango medio.

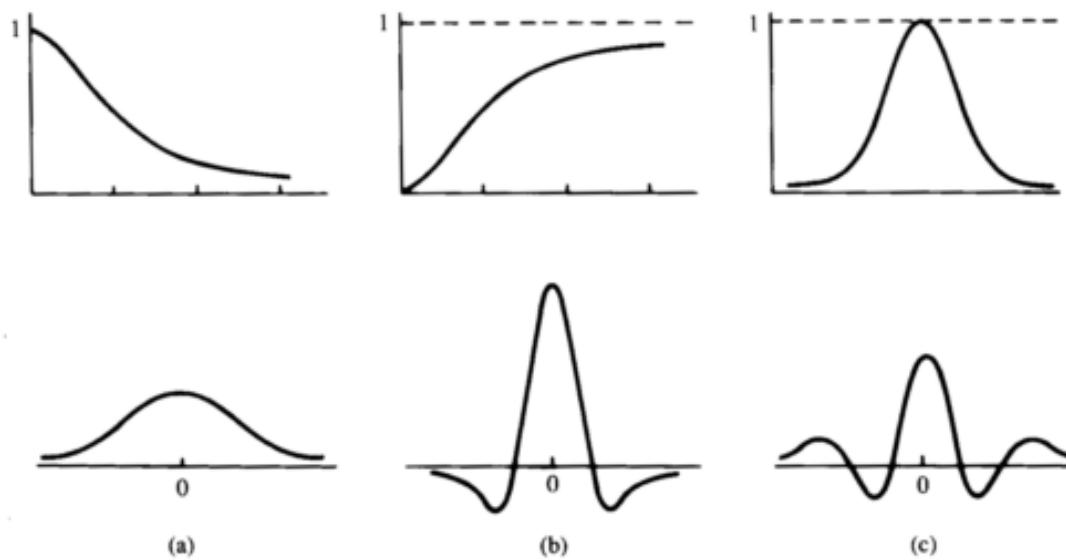


Figura 06 Filtros en el dominio de la frecuencia y el espacio (a) Filtro paso alto (b) Filtro paso bajo (c) Filtro paso banda

Fuente: es.wikipedia.org

Ventajas:

- Método simple y sencillo de implementar.

- Fácil asociación del concepto de frecuencia con ciertas características de la imagen; cambios de tonalidad suaves implican frecuencias bajas y cambios bruscos frecuencias altas.
- Proporciona flexibilidad en el diseño de soluciones de filtrado.
- Rapidez en el filtrado al utilizar el Teorema de la Convolución.

Desventajas:

- Se necesitan conocimientos en varios campos para desarrollar una aplicación para el procesamiento de imágenes.
- El ruido no puede ser eliminado completamente.

2.3.3.2. **Filtrado en el dominio del espacio**

Las operaciones de filtrado se llevan a cabo directamente sobre los píxeles de la imagen. En este proceso se relaciona, para todos y cada uno de los puntos de la imagen, un conjunto de píxeles próximos al píxel objetivo con la finalidad de obtener una información útil, dependiente del tipo de filtro aplicado, que permita actuar sobre el píxel concreto en que se está llevando a cabo el proceso de filtrado para, de este modo, obtener mejoras sobre la imagen y/o datos que podrían ser utilizados en futuras acciones o procesos de trabajo sobre ella.

Los filtros en el dominio del espacio pueden clasificarse en:

- Filtros lineales (filtros basados en kernels o máscaras de convolución).
- Filtros no lineales.

El concepto de kernel se entiende como una matriz de coeficientes donde el entorno del punto (x,y) que se considera en la imagen para obtener $g(x,y)$ está determinado por el tamaño y forma del kernel seleccionado.

Aunque la forma y tamaño de esta matriz es variable y queda a elección de cada usuario, es común el uso de kernels cuadrados $n \times n$. Dependiendo de la implementación, en los límites de la imagen se aplica un tratamiento especial (se asume un marco exterior de ceros o se repiten los valores del borde) o no se aplica ninguno. Es por ello, que el tipo de filtrado queda establecido por el contenido de dicho kernel utilizado.

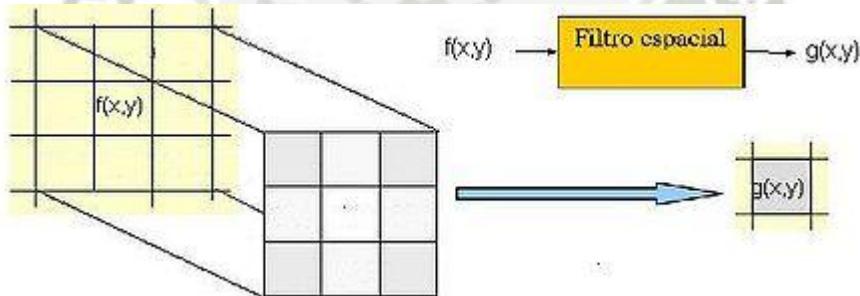


Figura 07 Representación de un kernel
Fuente: es.wikipedia.org

Para realizar un filtrado en el dominio del espacio se realiza una convolución (barrido) del kernel sobre la imagen. Para ello se sigue el Teorema de Convolución en el espacio: $g(x,y) = h(x,y) * f(x,y)$

- 1.- Cada píxel de la nueva imagen se obtiene mediante el sumatorio de la multiplicación del kernel por los píxeles contiguos: $g(x,y) = \sum \sum f(i,j) w(i,j)$

2.- Generalmente se divide sobre cierto valor constante para normalizar que suele obtenerse de la suma de los valores del kernel empleado.

2.3.3.3. Tipos de filtros

- Filtro paso bajo (suavizamiento): utilizados para eliminar ruido o detalles pequeños de poco interés puesto que sólo afecta a zonas con muchos cambios. La frecuencia de corte se determina por el tamaño del kernel y sus coeficientes. Se emplean diversos kernels:
 - Promedio: promedio de píxeles vecinos (kernel de unos).
 - Paso bajo en frecuencia.
 - Media: reemplaza cada píxel por el valor medio de sus contiguos.
 - Mediana: sustituye por el valor de la mediana de los píxeles vecinos (normalmente se comporta mejor que el de promedio).
 - Gaussiano: aproximación a la distribución gaussiana.
- Filtro paso alto (atenuamiento): intensifica los detalles, bordes y cambios de alta frecuencia, mientras que atenúa las zonas de tonalidad uniforme. Esto permite una mejor identificación posterior de los objetos que se encuentren en la imagen, puesto que el brillo se hace mayor en las zonas con frecuencias más altas, al mismo tiempo que se oscurecen las zonas de frecuencias bajas. Es común la aparición de ruido tras el proceso.
- Realce de bordes por desplazamiento y diferencia: sustrae de la imagen original una copia desplazada de la misma. Así, es posible localizar y hacer

resaltar los bordes existentes y que se quieran obtener según el modelo de kernel aplicado:

- Horizontal.
- Vertical.
- Horizontal/Vertical (diagonal).
- Realce de bordes mediante Laplace: este tipo de filtros realza los bordes en todas direcciones (los resultados que se obtienen pueden considerarse como una “suma” de los obtenidos tras aplicar todos los modelos del tipo anterior).
- Resalte de bordes con gradiente direccional: empleado para destacar y resaltar con mayor precisión los bordes que se localizan en una dirección determinada. Trabaja con los cambios de intensidad existentes entre píxeles contiguos.
- Detección de bordes y filtros de contorno (Prewitt y Sobel): al igual que los anteriores, se centra en las diferencias de intensidad que se dan pixel a pixel. Son utilizados para obtener los contornos de objetos y de este modo clasificar las formas existentes dentro de una imagen. Este tipo de filtros requieren un menor coste computacional.

2.4. Lógica Difusa

La lógica difusa (también llamada lógica borrosa) se basa en lo relativo de lo observado como posición diferencial. Este tipo de lógica toma dos valores

aleatorios, pero contextualizados y referidos entre sí. Así, por ejemplo, una persona que mida 2 metros es claramente una persona alta, si previamente se ha tomado el valor de persona baja y se ha establecido en 1 metro. Ambos valores están contextualizados a personas y referidos a una medida métrica lineal.

2.4.1. Funcionamiento

La lógica difusa ("fuzzy logic" en inglés) se adapta mejor al mundo real en el que vivimos, e incluso puede comprender y funcionar con nuestras expresiones, del tipo "hace mucho calor", "no es muy alto", "el ritmo del corazón está un poco acelerado", etc.

La clave de esta adaptación al lenguaje, se basa en comprender los cuantificadores de calidad para nuestras inferencias (en los ejemplos de arriba "mucho", "muy" y "un poco").

En la teoría de conjuntos difusos se definen también las operaciones de unión, intersección, diferencia, negación o complemento, y otras operaciones sobre conjuntos (ver también subconjunto difuso), en los que se basa esta lógica.

Para cada conjunto difuso, existe asociada una función de pertenencia para sus elementos, que indican en qué medida el elemento forma parte de ese conjunto difuso. Las formas de las funciones de pertenencia más típicas son trapezoidal, lineal y curva.

Se basa en reglas heurísticas de la forma SI (antecedente) ENTONCES (consecuente), donde el antecedente y el consecuente son también conjuntos difusos, ya sea puros o resultado de operar con ellos. Sirvan como ejemplos de regla heurística para esta lógica (nótese la importancia de las palabras "muchísimo", "drásticamente", "un poco" y "levemente" para la lógica difusa):

- SI hace muchísimo frío ENTONCES aumento drásticamente la temperatura.
- SI voy a llegar un poco tarde ENTONCES aumento levemente la velocidad.

Los métodos de inferencia para esta base de reglas deben ser sencillos, versátiles y eficientes. Los resultados de dichos métodos son un área final, fruto de un conjunto de áreas solapadas entre sí (cada área es resultado de una regla de inferencia). Para escoger una salida concreta a partir de tanta premisa difusa, el método más usado es el del centroide, en el que la salida final será el centro de gravedad del área total resultante.

Las reglas de las que dispone el motor de inferencia de un sistema difuso pueden ser formuladas por expertos, o bien aprendidas por el propio sistema, haciendo uso en este caso de redes neuronales para fortalecer las futuras tomas de decisiones.

Los datos de entrada suelen ser recogidos por sensores, que miden las variables de entrada de un sistema. El motor de inferencias se basa en chips difusos, que están aumentando exponencialmente su capacidad de procesamiento de reglas año a año.

Un esquema de funcionamiento típico para un sistema difuso podría ser de la siguiente manera:

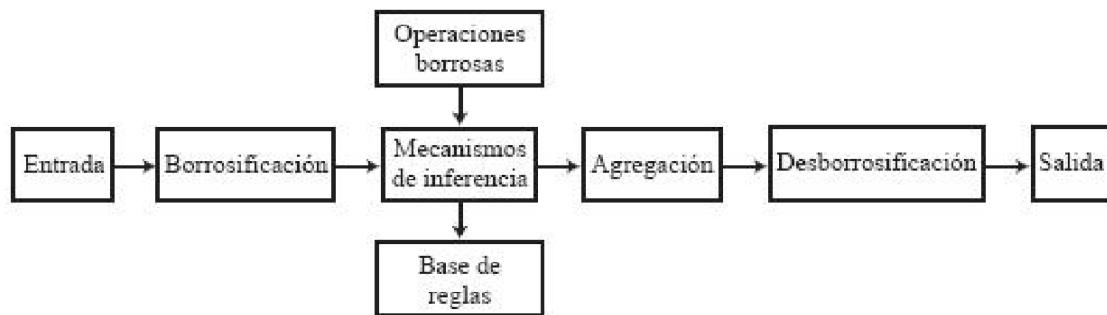


Figura 08 Esquema de funcionamiento típico
Fuente: <http://www.scielo.org.co/>

En la figura, el sistema de control hace los cálculos con base en sus reglas heurísticas, comentadas anteriormente. La salida final actuaría sobre el entorno físico, y los valores sobre el entorno físico de las nuevas entradas (modificadas por la salida del sistema de control) serían tomados por sensores del sistema.

Por ejemplo, imaginando que nuestro sistema difuso fuese el climatizador de un coche que se autorregula según las necesidades: Los chips difusos del climatizador recogen los datos de entrada, que en este caso bien podrían ser la temperatura y humedad simplemente. Estos datos se someten a las reglas del motor de inferencia (como se ha comentado antes, de la forma SI... ENTONCES...), resultando un área de resultados. De esa área se escogerá el centro de gravedad, proporcionándola como salida. Dependiendo del resultado, el

climatizador podría aumentar la temperatura o disminuirla dependiendo del grado de la salida.

2.4.2. Teoría de conjuntos difusos

La lógica difusa permite tratar con información que no es exacta o con un alto grado de imprecisión a diferencia de la lógica convencional la cual trabaja con información precisa. El problema principal surge de la poca capacidad de expresión de la lógica clásica.

2.4.2.1. Conjuntos clásicos

Los conjuntos clásicos surgen por la necesidad del ser humano de clasificar objetos y conceptos. Estos conjuntos pueden definirse como un conjunto bien definido de elementos o mediante una función de pertenencia μ que toma valores de 0 ó 1 de un universo en discurso para todos los elementos que pueden o no pertenecer al conjunto.

Un conjunto clásico se puede definir con la función de pertenencia mostrada en la ecuación:

$$\mu_A(x) = \begin{cases} 0 & \text{si } x \notin A \\ 1 & \text{si } x \in A \end{cases}$$

2.4.2.2. Conjuntos difusos

La necesidad de trabajar con conjuntos difusos surge del hecho que existen conceptos que no tienen límites claros. Un conjunto difuso se encuentra asociado por un valor lingüístico que está definido por una palabra, etiqueta lingüística o adjetivo. En los conjuntos difusos la función de pertenencia puede tomar valores del intervalo entre 0 y 1, y la transición del valor entre cero y uno es gradual y no cambia de manera instantánea como pasa con los conjuntos clásicos. Un conjunto difuso en un universo en discurso puede definirse como lo muestra la ecuación:

$$A = \{(x, \mu_A(x)) | x \in U\}$$

Donde $\mu_A(x)$ es la función de pertenencia de la variable x , y U es el universo en discurso. Cuando más cerca este la pertenencia del conjunto A al valor de 1, mayor será la pertenencia de la variable x al conjunto A .

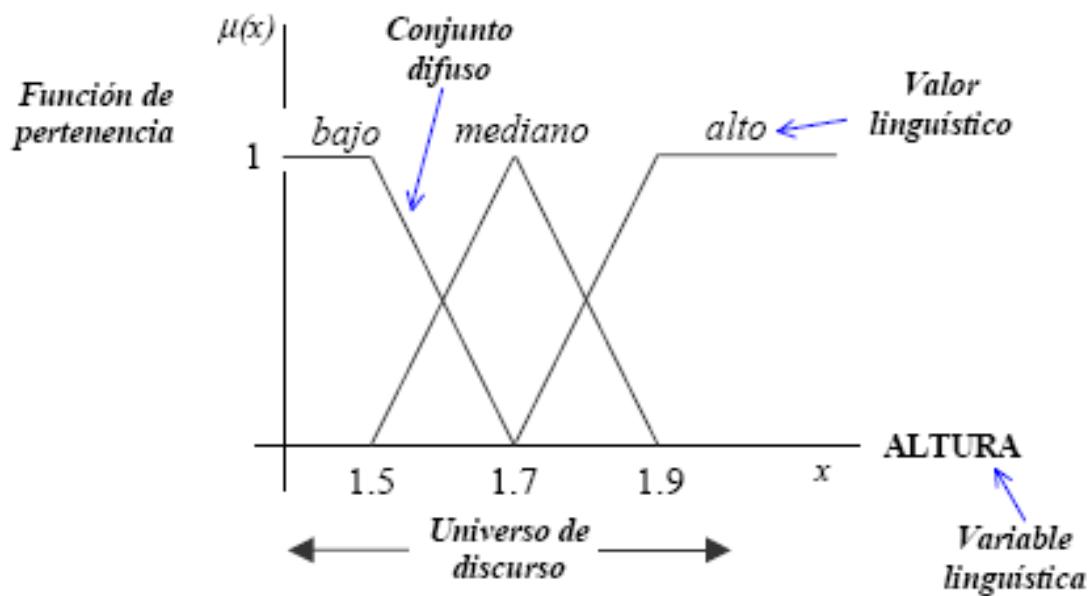


Figura 09 Ejemplo de conjuntos difusos

Fuente: catarina.udlap.mx

2.4.2.3. Funciones de pertenencia

Aun cuando cualquier función puede ser válida para definir un conjunto difuso, existen ciertas funciones que son más comúnmente utilizadas por su simplicidad matemática, entre éstas se encuentran las funciones de tipo triangular, mostrado en la figura 10, trapezoidal mostrado en la figura 11, gaussiana, etc.

$$\mu(x) = \begin{cases} 0 & \text{para } x \leq a \\ \frac{x-a}{m-a} & \text{para } a < x \leq m \\ \frac{b-x}{b-m} & \text{para } m < x \leq b \\ 0 & \text{para } x > b \end{cases}$$

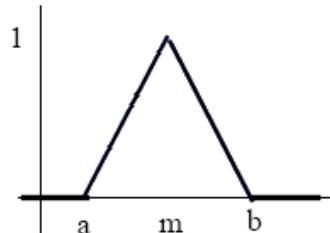


Figura 10 Función de pertenencia para conjunto difuso triangular

Fuente: catarina.udlap.mx

$$\mu(x) = \begin{cases} 0 & \text{para } x \leq a \\ \frac{x-a}{b-a} & \text{para } a < x \leq b \\ 1 & \text{para } b < x \leq c \\ \frac{d-x}{b-c} & \text{para } c < x \leq d \\ 0 & \text{para } x > d \end{cases}$$

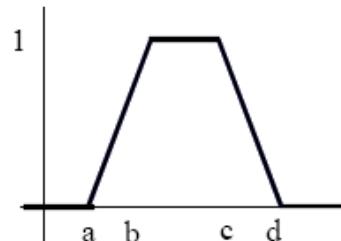


Figura 11 Función de pertenencia para conjunto difuso trapezoidal

Fuente: catarina.udlap.mx

2.4.3. El controlador difuso

La lógica difusa se aplica principalmente en sistemas de control difuso que utilizan expresiones ambiguas para formular reglas que controlen el sistema. Un sistema de control difuso trabaja de manera muy diferente a los sistemas de control convencionales. Estos usan el conocimiento experto para generar una base de conocimientos que dará al sistema la capacidad de tomar decisiones sobre ciertas acciones que se presentan en su funcionamiento. Los sistemas de control difuso permiten describir un conjunto de reglas que utilizaría una persona para controlar un proceso y a partir de estas reglas generar acciones de control. El control difuso puede aplicarse tanto en sistemas muy sencillos como en sistemas cuyos modelos matemáticos sean muy complejos. La estructura de un controlador difuso se muestra en la figura 12.

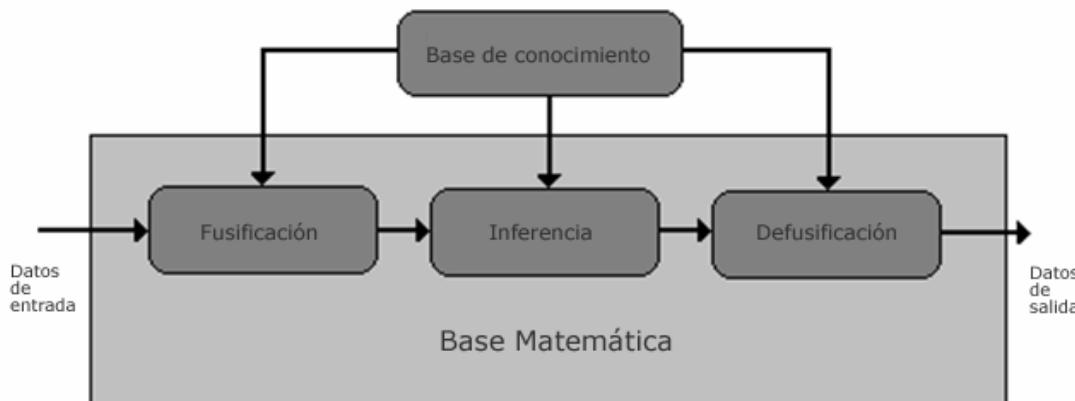


Figura 12 Estructura de un modelo difuso

Fuente: catarina.udlap.mx

2.4.3.1. Fusificación

La fusificación tiene como objetivo convertir valores *crisp* o valores reales en valores difusos. En la fusificación se asignan grados de pertenencia a cada una de las variables de entrada con relación a los conjuntos difusos previamente definidos utilizando las funciones de pertenencia asociadas a los conjuntos difusos.

2.4.3.2. Base de conocimiento

La base de conocimiento contiene el conocimiento asociado con el dominio de la aplicación y los objetivos del control. En esta etapa se deben definir las reglas lingüísticas de control que realizarán la toma de decisiones que decidirán la

forma en la que debe actuar el sistema. Se deben elegir también los métodos de activación y de acumulación.

2.4.3.3. Inferencia

La inferencia relaciona los conjuntos difusos de entrada y salida para representar las reglas que definirán el sistema. En la inferencia se utiliza la información de la base de conocimiento para generar reglas mediante el uso de condiciones, dividiendo dicha regla en antecedente y consecuente.

Método Mamdani

El mecanismo de inferencia desarrollado por Ebrahim Mamdani tiene como antecedente un grupo de variables lingüísticas difusas, y como consecuente, otra variable lingüística que requiere una etapa de defusificación para ser convertida en un valor numérico y entregada como salida hacia una siguiente etapa.

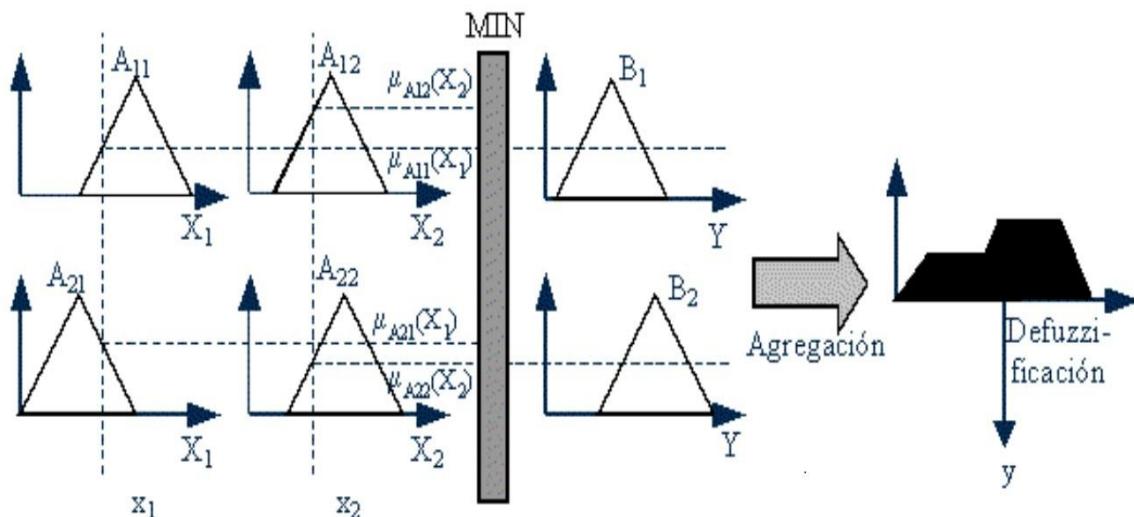


Figura 13 Funcionamiento de un sistema difuso Mamdani

Fuente: Artículo de investigación “Sistema de inferencia difusa basada en relaciones booleanas”

Método Sugeno

Este mecanismo fue propuesto por Takagi, Sugeno y Kang. El antecedente sigue siendo un grupo de variables lingüísticas, pero la consecuencia es una expresión matemática en función a los valores de las entradas que tiene un resultado que no requiere ser defusificado para ser considerado una salida.

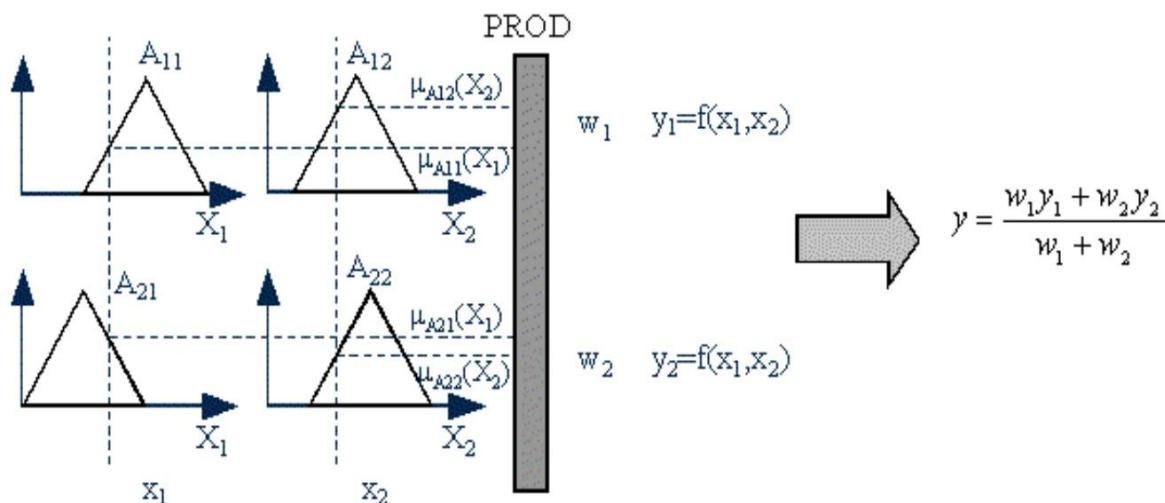


Figura 14 Funcionamiento de un sistema difuso Sugeno

Fuente: Artículo de investigación “Sistema de inferencia difusa basada en relaciones booleanas”

2.4.3.4. Defusificación

La defusificación realiza el proceso de adecuar los valores difusos generados en la inferencia en valores *crisp*, que posteriormente se utilizarán en el proceso de control. En la defusificación se utilizan métodos matemáticos simples como el método del Centroide, Método del Promedio Ponderado, Método de Máximo Derecho, Izquierdo, Medio, Método de Membresía del Medio del Máximo. Esta defusificación puede llevarse a cabo de forma discreta, dividiendo la función en partes iguales y realizando la sumatoria de sus puntos multiplicados por sus ponderaciones para luego dividirlos entre la sumatoria de sus puntos; o de forma continua dividiendo el momento de la función por el área.

2.4.4. Aplicaciones

2.4.4.1. Aplicaciones generales

La lógica difusa se utiliza cuando la complejidad del proceso en cuestión es muy alta y no existen modelos matemáticos precisos, para procesos altamente no lineales y cuando se envuelven definiciones y conocimiento no estrictamente definido (impreciso o subjetivo).

En cambio, no es una buena idea usarla cuando algún modelo matemático ya soluciona eficientemente el problema o cuando no tienen solución.

Esta técnica se ha empleado con bastante éxito en la industria, principalmente en Japón, extendiéndose sus aplicaciones a multitud de campos. La primera vez que se usó de forma importante fue en el metro japonés, con excelentes resultados. Posteriormente se generalizó según la teoría de la incertidumbre desarrollada por el matemático y economista español Jaume Gil Aluja.

A continuación se citan algunos ejemplos de su aplicación:

- Sistemas de control de acondicionadores de aire
- Sistemas de foco automático en cámaras fotográficas
- Electrodomésticos familiares (frigoríficos, lavadoras...)
- Optimización de sistemas de control industriales
- Sistemas de escritura
- Mejora en la eficiencia del uso de combustible en motores

- Sistemas expertos del conocimiento (simular el comportamiento de un experto humano)
- Tecnología informática
- Bases de datos difusas: Almacenar y consultar información imprecisa. Para este punto, por ejemplo, existe el lenguaje FSQL.
- En general, en la gran mayoría de los sistemas de control que no dependen de un Sí/No.

2.4.4.2. Lógica difusa en la inteligencia artificial

En Inteligencia artificial, la lógica difusa, o lógica borrosa se utiliza para la resolución de una variedad de problemas, principalmente los relacionados con control de procesos industriales complejos y sistemas de decisión en general, la resolución y la compresión de datos. Los sistemas de lógica difusa están también muy extendidos en la tecnología cotidiana, por ejemplo en cámaras digitales, sistemas de aire acondicionado, lavarropas, etc. Los sistemas basados en lógica difusa imitan la forma en que toman decisiones los humanos, con la ventaja de ser mucho más rápidos. Estos sistemas son generalmente robustos y tolerantes a imprecisiones y ruidos en los datos de entrada. Algunos lenguajes de programación lógica que han incorporado la lógica difusa serían por ejemplo las diversas implementaciones de Fuzzy PROLOG o el lenguaje Fril.

Consiste en la aplicación de la lógica difusa con la intención de imitar el razonamiento humano en la programación de computadoras. Con

la lógica convencional, las computadoras pueden manipular valores estrictamente duales, como verdadero/falso, sí/no o ligado/desligado. En la lógica difusa, se usan modelos matemáticos para representar nociones subjetivas, como caliente/tibio/frío, para valores concretos que puedan ser manipuladas por los ordenadores.

En este paradigma, también tiene un especial valor la variable del tiempo, ya que los sistemas de control pueden necesitar retroalimentarse en un espacio concreto de tiempo, pueden necesitarse datos anteriores para hacer una evaluación media de la situación en un período anterior.

2.4.5. Ventajas e inconvenientes

Como principal ventaja, cabe destacar los excelentes resultados que brinda un sistema de control basado en lógica difusa: ofrece salidas de una forma veloz y precisa, disminuyendo así las transiciones de estados fundamentales en el entorno físico que controle. Por ejemplo, si el aire acondicionado se encendiese al llegar a la temperatura de 30°, y la temperatura actual oscilase entre los 29°-30°, nuestro sistema de aire acondicionado estaría encendiéndose y apagándose continuamente, con el gasto energético que ello conllevaría. Si estuviese regulado por lógica difusa, esos 30° no serían ningún umbral, y el sistema de control aprendería a mantener una temperatura estable sin continuos apagados y encendidos.

También está la indecisión de decantarse bien por los expertos o bien por la tecnología (principalmente mediante redes neuronales) para reforzar las reglas heurísticas iniciales de cualquier sistema de control basado en este tipo de lógica.

2.5. Estándar RS-485

RS-485 o también conocido como EIA-485, que lleva el nombre del comité que lo convirtió en estándar en 1983. Es un estándar de comunicaciones en bus de la capa física del Modelo OSI.

Está definido como un sistema en bus de transmisión multipunto diferencial, es ideal para transmitir a altas velocidades sobre largas distancias (35 Mbit/s hasta 10 metros y 100 kbit/s en 1200 metros) y a través de canales ruidosos, ya que reduce los ruidos que aparecen en los voltajes producidos en la línea de transmisión. El medio físico de transmisión es un par entrelazado que admite hasta 32 estaciones en 1 solo hilo y la comunicación half-duplex (semiduplex). Soporta 32 transmisiones y 32 receptores. La transmisión diferencial permite múltiples drivers dando la posibilidad de una configuración multipunto. Al tratarse de un estándar bastante abierto permite muchas y muy diferentes configuraciones y utilizaciones.

Desde 2003 está siendo administrado por la Telecommunications Industry Association (TIA) y titulado como TIA-485-A.222.

2.5.1. Especificaciones

- Interfaz diferencial
- Conexión multipunto
- Hasta 32 estaciones (ya existen interfaces que permiten conectar 256 estaciones)
- Velocidad máxima de 10 Mbit/s (a 12 metros)
- Longitud máxima de alcance de 1200 metros (a 100 kbit/s)

2.5.2. Aplicaciones

- SCSI -2 y SCSI-3 usan esta especificación para ejecutar la capa física.
- RS-485 se usa con frecuencia en las UARTs para comunicaciones de datos de poca velocidad en las cabinas de los aviones. Por ejemplo, algunas unidades de control del pasajero lo utilizan, equipos de monitoreo de sistemas fotovoltaicos. Requiere el cableado mínimo, y puede compartir el cableado entre varios asientos. Por lo tanto reduce el peso del sistema.
- RS-485 se utiliza en sistemas grandes de sonido, como los conciertos de música y las producciones de teatro, se usa software especial para controlar remotamente el equipo de sonido de una computadora, es utilizado más generalmente para los micrófonos.

- RS-485 también se utiliza en la automatización de los edificios pues el cableado simple del bus y la longitud de cable es larga por lo que son ideales para ensamblar los dispositivos que se encuentran alejados.
- RS-485 Tiene la mayor parte de su aplicación en las plantas de producción automatizadas.

2.6. Protocolo Pelco-D

Es un protocolo de control de cámaras PTZ utilizado en la industria de la videovigilancia y los circuitos cerrados de televisión.

2.6.1. Estructura de la trama

Una trama de información en protocolo Pelco-D está conformada por siete bytes hexadecimales, como se muestra en el cuadro a continuación:

Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6	Byte 7
Sync	Adress	Command 1	Command 2	Data 1	Data 2	Checksum

Byte 1: para sincronizar el comienzo de la trama, se fija en FF

Byte 2: con el cual se selecciona a que cámara se envía la trama

Byte 3 y 4: diferentes comandos construidos por el movimiento requerido

Byte 5: controla la velocidad del movimiento horizontal (pan)

Byte 6: controla la velocidad del movimiento vertical (tilt)

Byte 7: suma del byte 2 al byte 7

	Bit 1	Bit 2	Bit 3	Bit 4	Bit 5	Bit 6	Bit 7	Bit 8
Com 1	Sense	Reserved	Reserved	Auto/ Manual Scan	Camera On/Off	Iris close	Iris open	Focus near
Com 2	Focus far	Zoom wide	Zoom tele	Tilt down	Tilt up	Pan left	Pan right	Fixed to 0

2.6.2. Comandos especiales

Existen comandos que gobiernan a la cámara para realizar acciones específicas, tales como establecer ciertas posiciones, llamar dichas posiciones o establecer rondas de patrullaje basadas en dichas posiciones.

A nivel de usuario, estos comandos son introducidos de diferentes formas, ya sea a través de un teclado que va conectado a la cámara o a través del software que se instala para controlar el equipo. Pero a nivel de programador, esos comandos son valores hexadecimales que reemplazan cualquier valor que se le dé a los bytes 3,4, 5 y 6.

Command	Word 3	Word 4	Word 5	Word 6
Set Preset	00	03	00	01 to 20
Clear Preset	00	05	00	01 to 20
Go To Preset	00	07	00	01 to 20
Flip (180° about)	00	07	00	21
Go To Zero Pan	00	07	00	22
Set Auxiliary	00	09	00	01 to 08
Clear Auxiliary	00	0B	00	01 to 08
Remote Reset	00	0F	00	00
Set Zone Start	00	11	00	01 to 08
Set Zone End	00	13	00	01 to 08
Write Char. To Screen	00	15	X Position 00 to 28	ASCII Value
Clear Screen	00	17	00	00
Alarm Acknowledge	00	19	00	Alarm No.
Zone Scan On	00	1B	00	00
Zone Scan Off	00	1D	00	00
Set Pattern Start	00	1F	00	00
Set Pattern Stop	00	21	00	00
Run Pattern	00	23	00	00
Set Zoom Speed	00	25	00	00 to 03
Set Focus Speed	00	27	00	00 to 03
Reset Camera to defaults	00	29	00	00
Auto-focus auto/on/off	00	2B	00	00-02
Auto Iris auto/on/off	00	2D	00	00-02
AGC auto/on/off	00	2F	00	00-02
Backlight compensation on/off	00	31	00	01-02
Auto white balance on/off	00	33	00	01-02
Enable device phase delay mode	00	35	00	00
Set shutter speed	00	37	Any	Any
Adjust line lock phase delay	00-01	39	Any	Any
Adjust white balance (R-B)	00-01	3B	Any	Any
Adjust white balance (M-G)	00-01	3D	Any	Any
Adjust gain	00-01	3F	Any	Any
Adjust auto-iris level	00-01	41	Any	Any
Adjust auto-iris peak value	00-01	43	Any	Any
Query ¹	00	45	Any	Any

Figura 15 Tabla de comandos especiales

Fuente: Manual del protocolo Pelco-D

CAPITULO III – DISEÑO DEL SISTEMA

El sistema funciona a partir de la captura de una imagen digital, la cual se procesa con la finalidad de obtener de ella la posición del objetivo, cuyo patrón en este caso y para efectos de las pruebas será un objeto circular de color verde. Una vez obtenidas las coordenadas del objetivo, estas son entregadas a un controlador difuso, que las evalúa junto con la velocidad del estado anterior del sistema y entrega un nuevo vector de velocidad. Ese nuevo vector de velocidad es la salida del sistema y debe ser comunicado al mecanismo encargado de mover la cámara para que esta pueda ubicarse en una posición más cercana al objetivo.

De esta manera se tienen cinco etapas diferenciadas y esenciales que parten de cada estado del entorno y terminan en cada orden que se le da al actuador, y se repiten de manera indefinida mientras el programa se esté corriendo.

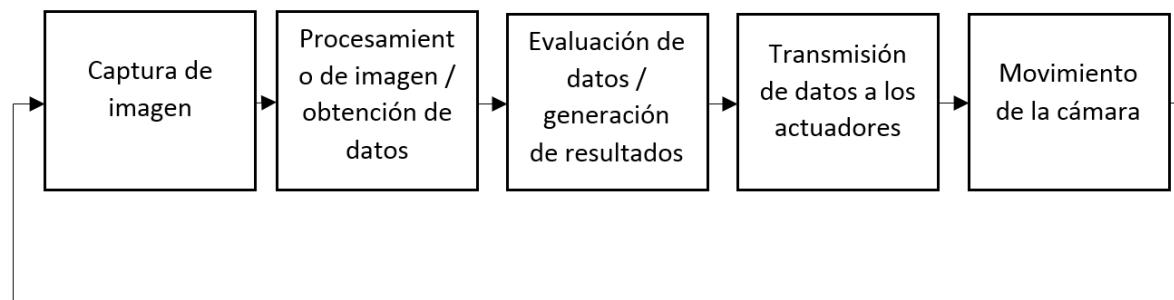


Figura 16 Diagrama de bloques del sistema
Fuente: Elaboración propia

Debido a que se requiere el uso de una cámara y de un mecanismo que pueda moverla, es que se decide utilizar una cámara PTZ (Pan/Tilt/Zoom), es decir, una cámara analógica que viene de fábrica anexa a un mecanismo encargado de su movimiento. De esta manera quedan enlazadas en un solo equipo las etapas de Captura de Imagen y Movimiento de la cámara Para las pruebas se utilizara el modelo NV12X de la marca NOVA, la cual sirve en diversos proyectos de videovigilancia y cuyas especificaciones se encuentran en el anexo D.



Figura 17 Cámara PTZ Nova NV12X
Fuente: <http://www.scimic.net/>

La señal de salida de esta cámara es, como se mencionó anteriormente, analógica, de manera que para poder ingresar al controlador a ser procesada debe ser convertida en digital a través de un dispositivo convertidor. En este caso

se utiliza un dispositivo de fabricación china conocido como EasyCap, el cual recibe la señal analógica de la cámara a través de un conector RCA y la entrega en formato digital al controlador a través de su conector USB. Los datos técnicos del EasyCap se encuentran en el Anexo F.



Figura 18 Conversor EasyCap
Fuente: <http://www.taringa.net/>

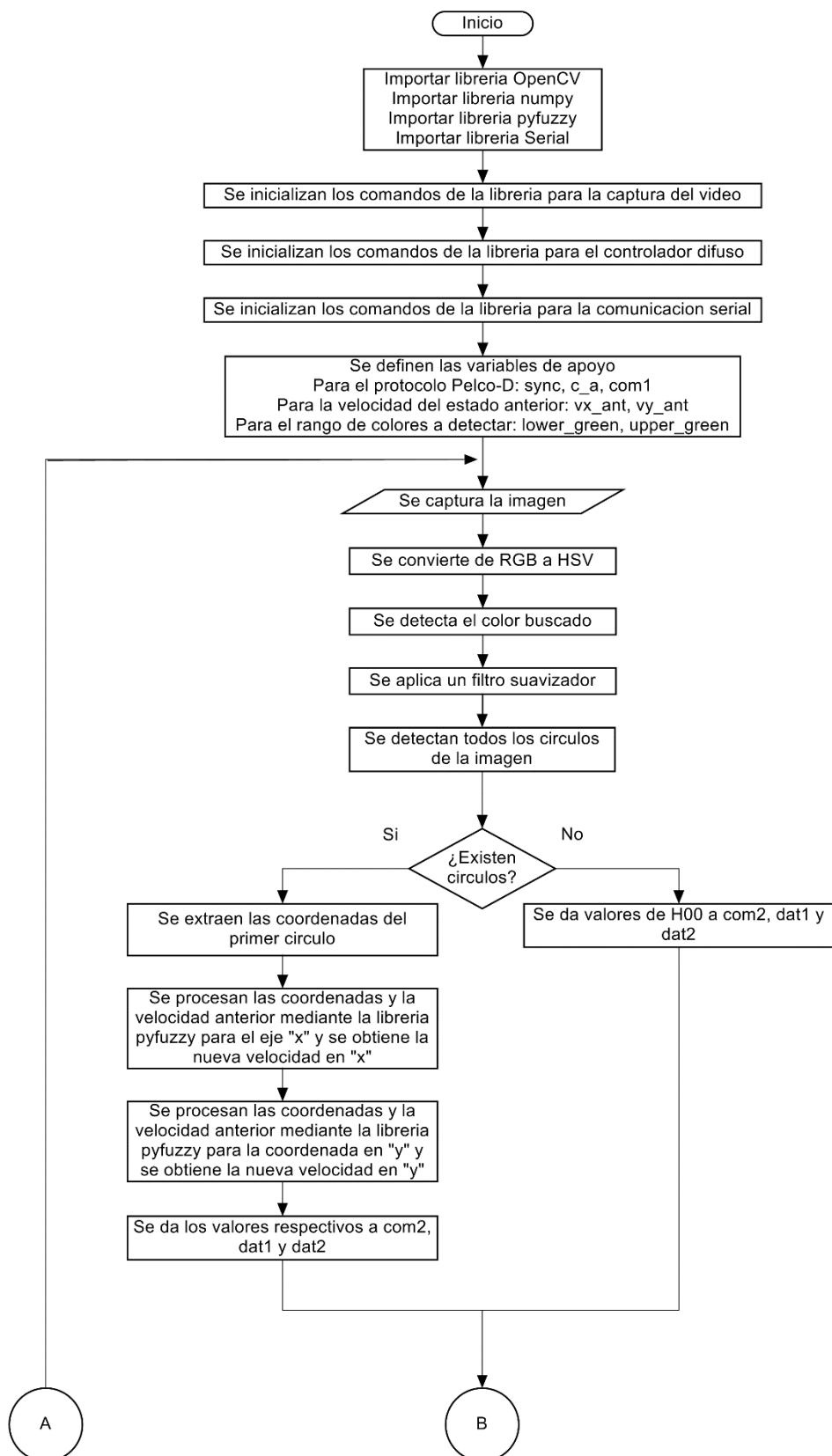
Las etapas de Procesamiento de la Imagen y de Evaluación de datos se llevan a cabo juntas dentro del controlador. Este controlador está programado en lenguaje Python dentro de un archivo con nombre “seguimiento.py”, el cual se

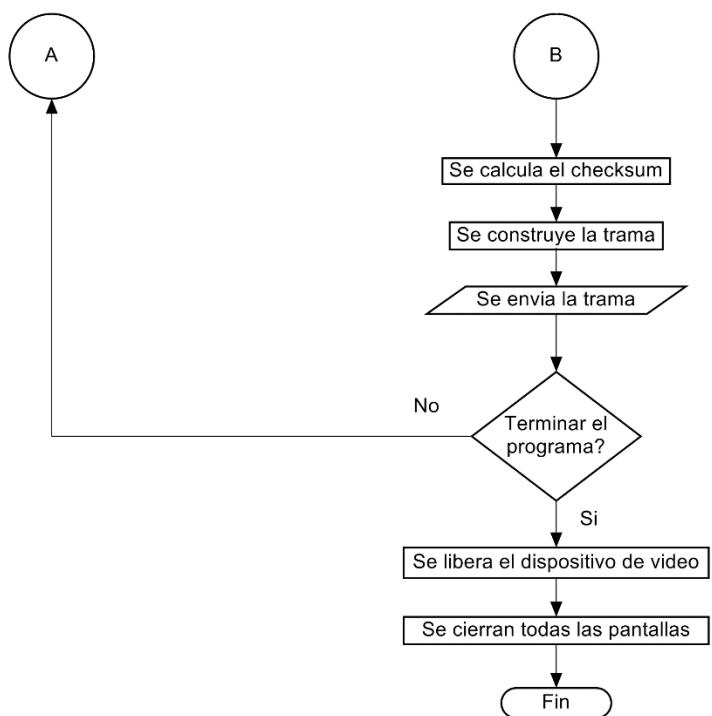
corre directamente desde el terminal de Linux. Para esta labor se utiliza una netbook de marca TOSHIBA y modelo NB200, cuyas características técnicas se describen en el Anexo E.



Figura 19 Notebook Toshiba NB200
Fuente: <http://www.notebookcheck.org/>

A continuación se muestra el diagrama de flujo del sistema.





La etapa de Transmisión de Datos se apoya una placa que va conectada a la netbook mediante su puerto USB y a la cámara mediante una bornera de dos polos.



Figura 20 Placa para transmisión de datos
Fuente: Elaboración propia

En esta placa tenemos, además, tres puertos RCA para la distribución del video. Uno de ellos sirve como entrada y los otros dos como salidas, conectados en paralelo y de forma pasiva, sin cumplir mayor función que la de un splitter.

3.1. Captura de imágenes

En esta etapa se captura una imagen digital a través de una entrada digital de video y se le entrega al siguiente bloque.

Para todo el trabajo con imágenes, en este proyecto, se utilizará la librería “OpenCV”, la cual deberá ser previamente importada. Además deben ser preparados los datos que serán procesados mediante comandos de esta librería. A continuación se muestra la porción del código que cumple dicha función:

```
import cv2
import cv2.cv as cv

cap = cv2.VideoCapture(1)
ret=cap.set(3, 320)
ret=cap.set(4, 240)
```

Se establece un tamaño de imagen de 320x240 pixeles y se establece que se conseguirá dicha imagen del dispositivo (1). Habiendo preparado todo, se captura la imagen y se almacena en la matriz “frame”, mediante el siguiente código:

```
ret, frame = cap.read()
```

Esta matriz contiene la imagen digital en un modelo RGB, la cual es la salida de este bloque.

3.2. Procesamiento de la imagen

En esta etapa se recibe la matriz “frame” y se procesa (por color y forma) para encontrar la posición del objetivo dentro del campo de visión de la cámara.

Dado que se utilizaran más comandos de la librería “OpenCV” y estos comandos trabajan con datos de imagen en modelo HSV, es que se requiere convertir la imagen de la matriz “frame” a este modelo. Para este fin, utilizamos la función “cvtColor” de la librería “OpenCV”.

```
hsv = cv2.cvtColor(frame, cv2.COLOR_BGR2HSV)
```

El siguiente paso dentro de esta etapa consiste en la selección de las partes de la imagen que tienen el color que nos interesa. Ese rango debe ser especificado mediante dos límites de colores, expresados en modelo HSV y se construyen utilizando el comando “array” de la librería “NumPy”, la cual debe ser

previamente importada. Una vez establecido el rango de colores a reconocer, se utiliza la función “InRange” para separar los que está dentro del rango y lo que no, y en seguida se suaviza la imagen para eliminar cualquier ruido en la imagen utilizando la función “medianBlur”, el cual utiliza la mediana como método de suavizamiento con un tamaño de kernel de 9 pixeles; ambas funciones propias de la librería “OpenCV”. Esta imagen, ya separada y suavizada, se guarda en la matriz “gray_ff”.

```
lower_green = np.array([30,50,50])
upper_green = np.array([90,255,255])

mask = cv2.inRange(hsv, lower_green, upper_green)

gray_ff = cv2.medianBlur(mask,9)
```

Ahora que ya se ha detectado las partes de la imagen que tienen el color que nos interesa, el siguiente paso consiste en detectar la forma que nos interesa. Como se especificó anteriormente, se requiere detectar los objetos circulares. Para esto se utiliza la función “HoughCircle” de la librería “OpenCV”. El uso de este comando requiere que se especifique, principalmente, la distancia mínima entre los centros (40) de los círculos encontrados y un rango de diámetro mínimo (10) y máximo (50). Este comando utiliza la transformada de Hough para encontrar las posiciones de los círculos. Dado que ya conocemos el rango de diámetros, solo quedaría utilizar un acumulador de dos dimensiones.

```
circles = cv2.HoughCircles(gray_ff, cv.CV_HOUGH_GRADIENT, 2, 40,
param1=50, param2=30, minRadius=10, maxRadius=50)
```

Como resultado de la función anterior, la información de todos los círculos se almacena en la matriz “circles”. Luego de eso, si la matriz no se encuentra vacía, se procede a obtener las coordenadas en X e Y del centro del primer círculo en la lista y se les resta la mitad del tamaño de la imagen para asegurarnos que se establezca el centro de la pantalla como el centro de un plano cartesiano.

```
cx = circles[0,0,0]
cy = circles[0,0,1]

dx = cx-160
dy = cy-120
```

Los datos “dx” y “dy” indican la posición del objeto dentro de un plano cartesiano que tiene su origen en el centro de la imagen, y serán utilizados como datos para el controlador difuso, en la siguiente etapa.

3.3. Evaluación de datos

La etapa de la evaluación de datos se da a través de un controlador difuso, que tiene como entradas la posición actual del objetivo dentro del rango de visión de la cámara y la velocidad del estado anterior de esta; y tiene como salida la velocidad que tendrá que tomar el mecanismo encargado de mover la cámara para alcanzar el objetivo.

Este controlador difuso se apoya en la librería “pyfuzzy”, la cual, como en los casos anteriores, deberá ser importada previamente. Además deberá ser preparado un archivo donde se determinen los set difusos, reglas y algoritmo de defusificación. El archivo va en el formato establecido por dicha librería y se guarda con el nombre “camara_fuzzy.fcl”.

Los sets difusos, operadores y reglas fueron asignados con la consigna de hacerlos lo más proporcionados posibles, pero conforme se fueron realizando pruebas sobre el sistema, se notó que debían variarse, de manera que después de varias pruebas se fue afinando el sistema y estos parámetros fueron variando hasta tomar los valores que se explicaran a continuación.

3.3.1. Fusificación:

La entrada “dist” indica la posición del objetivo, en cuanto a distancia y dirección, sobre un eje que tiene su origen en el centro de la imagen.

Para comenzar, se definen tres estados de posición de la bola hacia cada lado: cerca, medio y lejos; lo que da lugar a seis subsets a lo largo del set. Además de eso se agrega un subset al centro para la posición central. De esta forma tenemos 7 subsets que se dividirán los 320 pixeles que tomamos como referencia para el eje x.

Se decide utilizar conjuntos triangulares para todos los subsets, salvo ciertas excepciones que serán explicadas. La decisión se utilizar conjuntos

triangulares se debe a que las bases de conocimiento dictan que el cambio entre un estado y otro debe ser lineal e inversamente proporcional.

Se tiene una escala de -160 a 160 pixeles y los subsets son simétricos entre sí para la derecha y para la izquierda, de manera que se especificara solo un lado para la asignación de los valores de estos subsets, considerando que el otro lado se repite.

Es un hecho que ningún objeto podría posicionarse sobre la distancia -160, ya que para que su centro esté ahí, la mitad de su área estaría fuera de la pantalla y no estaría reconocido como un círculo. Por este motivo es que el subset “i_lejos” debe tener un valor de 1 desde una distancia más cercana a la distancia 0.

Considerando que el parámetro de tamaño máximo de círculo, especificado en la etapa de procesamiento de la imagen, contempla un círculo con un tamaño máximo de 50 pixeles, podemos deducir que la distancia máxima al borde a la que podría encontrarse el centro de este círculo es de 25 pixeles. Esto multiplicado por el factor de compensación de pantalla, explicado líneas más adelante en esta misma sección, nos da un resultado entero de 34 pixeles. Esto quiere decir que el decrecimiento de valores del subset “i_lejos” debe comenzar 34 pixeles más cerca al centro.

Por otro lado, se requiere dejar una brecha sin reacción en la parte central de la escala de distancia, con el objetivo de que se quede quieto en un espacio cercano al centro, y no solo en el centro exacto. Esto reducirá la exactitud del

sistema, pero es necesario ya que, como se pudo observar en las pruebas realizadas, el sistema no llegaba a frenarse por completo.

Para esto se toma en consideración el mismo principio de la definición del subset “i_lejos” y se da una distancia total de la columna de “centro” de 34 píxeles, 17 para el lado positivo y 17 para el lado negativo.

Dejando de lado los 34 de la columna de “i_lejos” y los 17 de la columna de “centro”, nos quedan 109 píxeles a repartir entre los demás subsets. Estos se reparten de manera equitativa entre ellos, de manera que se obtiene el siguiente esquema:

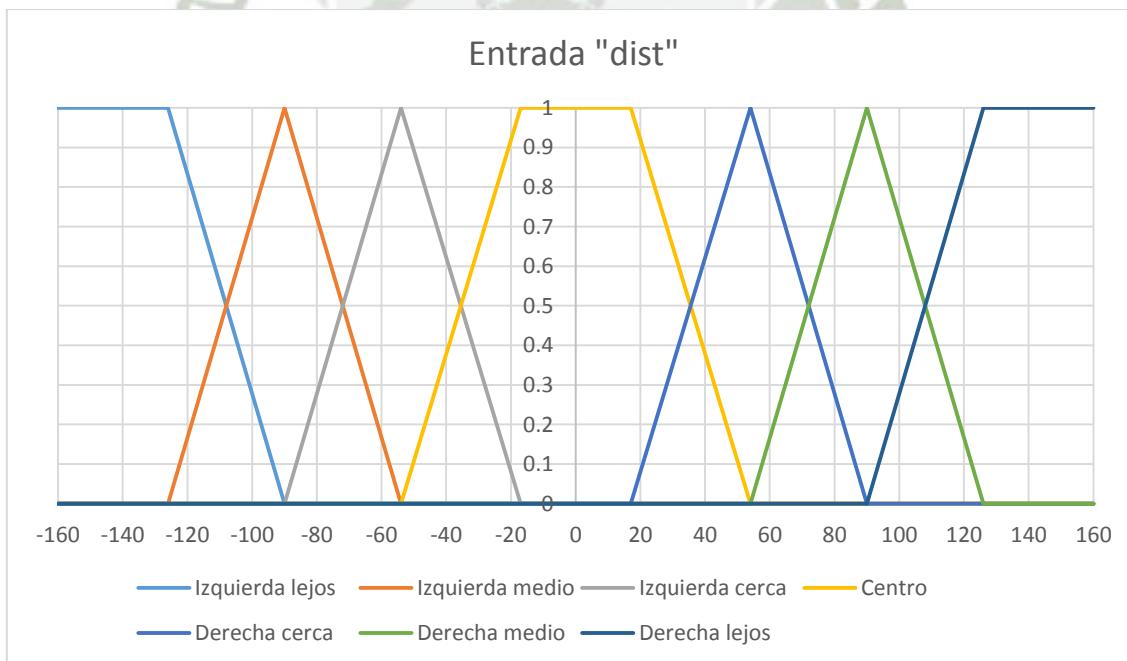


Figura 21 Sets difusos de la entrada “dist”
Fuente: Elaboración propia

Los sets de la entrada “dist” se definen dentro del archivo “camara_fuzzy” de la siguiente manera:

```
FUZZIFY dist
TERM i_lejos := (-161,0) (-160,1) (-126,1) (-90,0);
TERM i_medio := (-126,0) (-90,1) (-54,0);
TERM i_cerca := (-90,0) (-54,1) (-17,0);
TERM centro := (-54,0) (-17,1) (17,1) (42,0);
TERM d_cerca := (17,0) (54,1) (90,0);
TERM d_medio := (54,0) (90,1) (126,0);
TERM d_lejos := (84,0) (126,1) (160,1) (161,0);
END_FUZZIFY
```

La entrada “movi” indica la velocidad y sentido del estado anterior del controlador, es decir el último valor de velocidad que se envió al mecanismo encargado del movimiento de la cámara y que en este momento es la velocidad a la que se está moviendo dicho mecanismo. En este caso se consideran dos estados para cada lado: rápido y lento, y uno de velocidad cero. Esto nos da un total de 5 subsets, entre los que se dividen los 64 puntos del universo de discurso del set.

En este caso se considera una distribución equitativa de los puntos de esta escala entre los subsets de esta entrada. Por lo tanto se divide de forma equitativa procurando valores enteros, y los decimales que sobran de este redondeo se asignan a una brecha al centro. Los subsets de esta entrada quedan de la siguiente manera:



Figura 22 Sets difusos de la entrada “movi”

Fuente: Elaboración propia

Los sets de la entrada “movi” se definen dentro del archivo “camara_fuzzy” de la siguiente manera:

```
FUZZIFY movi
  TERM i_rapido := (-64,0) (-63,1) (-33,0);
  TERM i_lento := (-63,0) (-33,1) (-3,0);
  TERM cero := (-33,0) (-3,1) (3,1) (33,0);
  TERM d_lento := (33,0) (63,1) (63,0);
  TERM d_rapido := (33,0) (63,1) (64,1);
END_FUZZIFY
```

3.3.2. Defusificación

La salida “velo” indica la velocidad y sentido que deberá tomar el mecanismo encargado del movimiento de la cámara para acercarse hacia el objetivo, con la finalidad de que el centro de la imagen coincida con el centro de

dicho objetivo. Para la salida se toman en cuenta cinco velocidades en cada sentido y una central, en la que se le ordena al dispositivo quedarse quieto. Esto nos da un total de 11 subsets, entre los cuales se dividen de manera equitativa todos los puntos del set. El set consta, al igual que la entrada “movi” de 64 puntos de universo de discurso, los cuales van desde el 0 para la velocidad mínima y 63 para la máxima. Esto se debe a que el valor de velocidad es un número binario de 6 bits. Se procura que todos los valores sean enteros, redondeándolos al inmediato inferior y todos los decimales que sobran se suman para la columna del subset “cero”.



Figura 23 Sets difusos de la salida “velo”
Fuente: Elaboración propia

Se establece además que el operador de agregación será el máximo, el método de defusificación será el de Centros de Gravedad, utilizando la forma discreta, y que para cuando el sistema no tenga ninguna entrada, el valor por defecto será 0, de manera que el mecanismo se quedara quieto. Los sets de la salida “velo” y el método de defusificación se definen en el archivo “camara_fuzzy” de la siguiente manera:

DEFUZZIFY velo

```
TERM i_muy_rapido := (-64,0) (-63,1) (-51,0);
TERM i_rapido := (-63,0) (-51,1) (-39,0);
TERM i_medio := (-51,0) (-39,1) (-27,0);
TERM i_lento := (-39,0) (-27,1) (-15,0);
TERM i_muy_lento := (-27,0) (-15,1) (-3,0);
TERM cero := (-15,0) (-3,1) (3,1) (15,0);
TERM d_muy_lento := (3,0) (15,1) (27,0);
TERM d_lento := (15,0) (27,1) (39,0);
TERM d_medio := (15,0) (39,1) (51,0);
TERM d_rapido := (39,0) (51,1) (63,0);
TERM d_muy_rapido := (51,0) (63,1) (64,0);
ACCU: MAX;
METHOD : COG;
DEFAULT := 0;
END_DEFUZZIFY
```

3.3.3. Reglas

Tenemos una entrada con 7 y otra con 5 sets difusos, lo cual nos da la suma de 35 reglas, repartidas entre los 11 conjuntos de la salida “velo”. Se debe definir además el método de activación. Debido a que se utilizan 64 puntos de salida, no existe mucha diferencia entre utilizar el mínimo y el producto, y como la idea principal de este proyecto es que el sistema analice el entorno de la forma

más parecida a la que utilizaría un humano, se define utilizar el mínimo como método de activación.

Las reglas quedan definidas de la siguiente manera:

RULEBLOCK No1

AND : MIN;

ACT : MIN;

RULE 1: IF dist IS i_lejos AND movi IS i_rapido THEN velo IS i_rapido;

RULE 2: IF dist IS i_lejos AND movi IS i_lento THEN velo IS i_muy_rapido;

RULE 3: IF dist IS i_lejos AND movi IS cero THEN velo IS i_muy_rapido;

RULE 4: IF dist IS i_lejos AND movi IS d_lento THEN velo IS i_muy_rapido;

RULE 5: IF dist IS i_lejos AND movi IS d_rapido THEN velo IS i_muy_rapido;

RULE 6: IF dist IS i_medio AND movi IS i_rapido THEN velo IS i_medio;

RULE 7: IF dist IS i_medio AND movi IS i_lento THEN velo IS i_medio;

RULE 8: IF dist IS i_medio AND movi IS cero THEN velo IS i_rapido;

RULE 9: IF dist IS i_medio AND movi IS d_lento THEN velo IS i_muy_rapido;

RULE 10: IF dist IS i_medio AND movi IS d_rapido THEN velo IS i_muy_rapido;

RULE 11: IF dist IS i_cerca AND movi IS i_rapido THEN velo IS i_muy_lento;

RULE 12: IF dist IS i_cerca AND movi IS i_lento THEN velo IS i_lento;

RULE 13: IF dist IS i_cerca AND movi IS cero THEN velo IS i_lento;

RULE 14: IF dist IS i_cerca AND movi IS d_lento THEN velo IS i_medio;

RULE 15: IF dist IS i_cerca AND movi IS d_rapido THEN velo IS i_rapido;

RULE 16: IF dist IS centro AND movi IS i_rapido THEN velo IS i_muy_lento;

RULE 17: IF dist IS centro AND movi IS i_lento THEN velo IS cero;

RULE 18: IF dist IS centro AND movi IS cero THEN velo IS cero;

RULE 19: IF dist IS centro AND movi IS d_lento THEN velo IS cero;

RULE 20: IF dist IS centro AND movi IS d_rapido THEN velo IS d_muy_lento;

RULE 21: IF dist IS d_cerca AND movi IS i_rapido THEN velo IS d_rapido;

RULE 22: IF dist IS d_cerca AND movi IS i_lento THEN velo IS d_medio;

RULE 23: IF dist IS d_cerca AND movi IS cero THEN velo IS d_lento;

RULE 24: IF dist IS d_cerca AND movi IS d_lento THEN velo IS d_lento;

RULE 25: IF dist IS d_cerca AND movi IS d_rapido THEN velo IS d_muy_lento;

RULE 26: IF dist IS d_medio AND movi IS i_rapido THEN velo IS d_muy_rapido;

RULE 27: IF dist IS d_medio AND movi IS i_lento THEN velo IS d_muy_rapido;

RULE 28: IF dist IS d_medio AND movi IS cero THEN velo IS d_rapido;

RULE 29: IF dist IS d_medio AND movi IS d_lento THEN velo IS d_medio;

RULE 30: IF dist IS d_medio AND movi IS d_rapido THEN velo IS d_medio;

```
RULE 31: IF dist IS d_lejos AND movi IS i_rapido THEN velo IS d_muy_rapido;  
RULE 32: IF dist IS d_lejos AND movi IS i_lento THEN velo IS d_muy_rapido;  
RULE 33: IF dist IS d_lejos AND movi IS cero THEN velo IS d_muy_rapido;  
RULE 34: IF dist IS d_lejos AND movi IS d_lento THEN velo IS d_muy_rapido;  
RULE 35: IF dist IS d_lejos AND movi IS d_rapido THEN velo IS d_rapido;  
  
END_RULEBLOCK
```

Una vez establecidos todos los parámetros y reglas necesarias para la evaluación, se aplica la herramienta “system_velo”, el cual ha sido cargado con la función “fuzzy.storage.fcl.Reader” de la librería “pyfuzzy”.

```
system_velo=fuzzy.storage.fcl.Reader().load_from_file("cama  
ra_fuzzy.fcl")
```

3.3.4. Aplicación del controlador:

Esta herramienta se aplica dos veces en cada ciclo del controlador: Una para los datos que dependen del eje X y otra para los que dependen del eje Y; con una diferencia en el procesamiento de los datos del eje Y, debido a que la longitud del eje es de 240 pixeles, frente a los 320 pixeles del eje X. Debido a esta diferencia es que antes del procesamiento de los datos del eje Y, tenemos que multiplicar la distancia por 4/3. Además se le da el valor de la velocidad a una variable que deberá guardarlo hasta el siguiente estado para que sirva como la entrada de velocidad anterior al controlador.

```
my_input["dist"] = dx  
my_input["movi"] = vx_ant
```

```
system_velo.calculate(my_input, my_output)
vx = int(my_output["velo"])
vx_ant = vx

my_input["dist"] = dy*4/3
my_input["movi"] = vy_ant
system_velo.calculate(my_input, my_output)
vy= int(my_output["velo"])
vy_ant = vy
```

3.3.5. Preparación de la salida.

Una vez obtenidas las salidas “vx” y “vy”, se necesita prepararlas para ser transmitidas al mecanismo.

Dado que para las pruebas se utiliza una cámara PTZ, la cual cuenta con un mecanismo para su movimiento, y que existe un protocolo diseñado específicamente para la comunicación con estas cámaras, se programó el sistema para entregar los datos bajo este protocolo. Debido a eso es que, al final de la etapa de la evaluación de datos, se requiere preparar la trama que será enviada a este mecanismo.

Los datos de la trama se transmiten en serie, bajo una estructura la descrita anteriormente, y nuestro trabajo se da únicamente sobre los bytes 4, 5 y 6, los cuales se arman a continuación:

```
if vx == 0: #no mueve
    vvx = vx
    com_2 = 0x00      #I:0x04 R:0x02 U:0x08 D:0x10

if vx > 0: #mueve derecha
```

```
vvx = vx
com_2 = 0x02      #L:0x04 R:0x02 U:0x08 D:0x10

if vx < 0: #mueve izquierda
    vvx = -vx
    com_2 = 0x04      #L:0x04 R:0x02 U:0x08 D:0x10

if vy == 0: #no mueve
    vvy = vy
    com_2 = com_2 + 0x00      #L:0x04 R:0x02 U:0x08 D:0x10

if vy > 0: #mueve arriba
    vvy = vy
    com_2 = com_2 + 0x10      #L:0x04 R:0x02 U:0x08 D:0x10

if vy < 0: #mueve abajo
    vvy = -vy
    com_2 = com_2 + 0x08      #L:0x04 R:0x02 U:0x08 D:0x10

dat_1 = vvx  #pan Speed: 0x00 -> 0x3f  Turbo: 0xff
dat_2 = vvy  #tilt Speed: 0x00 -> 0x3f  Turbo: 0xff

c_s = c_a + com_1 + com_2 + dat_1 + dat_2

envio=chr(sync)+chr(c_a)+chr(com_1)+chr(com_2)+chr(dat_1)+chr(dat_2)+chr(c_s)
```

Todo esto funciona si hay algún dato dentro de la matriz “circles” descrita en la etapa anterior. Si no hubiera ningún dato, simplemente se igualan a cero los bytes 4, 5 y 6 de la trama y el mecanismo se quedara quieto.

```
com_2 = 0
dat_1 = 0      #pan Speed: 0x00 -> 0x3f
dat_2 = 0      #tilt Speed: 0x00 -> 0x3f

c_s = c_a + com_1 + com_2 + dat_1 + dat_2

envio=chr(sync)+chr(c_a)+chr(com_1)+chr(com_2)+chr(dat_1)+chr(dat_2)+chr(c_s)
```

Hasta el momento se tiene preparada la trama que contiene las órdenes para el mecanismo. Esta trama será enviada en la siguiente etapa.

Para detener el programa en cualquier momento se utiliza el comando “waitKey” de la librería “OpenCV”. De esta manera, presionando la letra “q” dentro de cualquiera de las pantallas de dicha librería el programa se detiene, se cierran todas las ventanas con la función “destroyAllWindows” y se libera el enlace con el dispositivo de video con el comando “reléase”, ambos de la misma librería.

```
cap.release()  
cv2.destroyAllWindows()
```

Finalmente se tiene el código completo de la aplicación, puesto ya en el orden correcto y especificando la importación de cada librería:

```
#import video  
import numpy as np  
import cv2  
import cv2.cv as cv  
  
#import fuzzy  
import fuzzy.storage.fcl.Reader  
  
#import Serial COM  
import serial  
  
#init video  
cap = cv2.VideoCapture(1)  
  
ret=cap.set(3, 320)  
ret=cap.set(4, 240)
```

```
#inicia fuzzy
system_velo = fuzzy.storage.fcl.Reader.Reader().load_from_file("camara_fuzzy.fcl")
my_input = {
    "dist" : 0.0,
    "movi" : 0.0
}
my_output = {
    "velo" : 0.0
}

#init serial COM
ser=serial.Serial('/dev/ttyACM0',9600,timeout=0.5)
#ser=serial.Serial('/dev/ttyUSB1',9600,timeout=0.5)

#PELCO-D
sync= 0xff
c_a= 0x01
com_1= 0x00

#variables velocidad inicial
vx_ant = 0
vy_ant = 0

while(True):
    ret, frame = cap.read()

    #Covierte de RGB a HSV
    hsv = cv2.cvtColor(frame, cv2.COLOR_BGR2HSV)

    #verde RGB 0 255 0 -> 60 255 255
    lower_green = np.array([30,50,50])
    upper_green = np.array([90,255,255])

    #Detecta objetos verdes
    mask = cv2.inRange(hsv, lower_green, upper_green)

    #filtro
    gray_ff = cv2.medianBlur(mask,9)

    #detecta circulos
    circles =
cv2.HoughCircles(gray_ff,cv.CV_HOUGH_GRADIENT,2,40,param1=50,param2=30,minRadius=10,maxRadius=50)
```

```
#print circles
```

```
if circles is not None:
```

```
    #convierte formato de datos
    circles = np.uint16(np.around(circles))
```

```
    #obtiene centro del circulo
    cx = circles[0,0,0]
    cy = circles[0,0,1]
    dx = cx-160
    dy = cy-120
```

```
    #Evalua fuzzy
    my_input["dist"] = dx
    my_input["movi"] = vx_ant
    system_velo.calculate(my_input, my_output)
    vx = int(my_output["velo"])
    vx_ant = vx
```

```
    my_input["dist"] = dy*4/3
    my_input["movi"] = vy_ant
    system_velo.calculate(my_input, my_output)
    vy= int(my_output["velo"])
    vy_ant = vy

    if vx == 0: #no mueve
        vvx = vx
        com_2 = 0x00 #L:0x04 R:0x02 U:0x08 D:0x10
    if vx > 0: #mueve derecha
        vvx = vx
        com_2 = 0x02 #L:0x04 R:0x02 U:0x08 D:0x10
    if vx < 0: #mueve izquierda
        vvx = -vx
        com_2 = 0x04 #L:0x04 R:0x02 U:0x08 D:0x10
    if vy == 0: #no mueve
        vvy = vy
        com_2 = com_2 + 0x00 #L:0x04 R:0x02 U:0x08 D:0x10
    if vy > 0: #mueve arriba
        vvy = vy
        com_2 = com_2 + 0x10 #L:0x04 R:0x02 U:0x08 D:0x10
    if vy < 0: #mueve abajo
        vvy = -vy
        com_2 = com_2 + 0x08 #L:0x04 R:0x02 U:0x08 D:0x10
```

```
print "dx: ", dx
print "dy: ", dy
print "vx: ", vx
print "vy: ", vy

dat_1 = vvx    #pan Speed: 0x00 -> 0x3f  Turbo: 0xff
dat_2 = vvy    #tilt Speed: 0x00 -> 0x3f  Turbo: 0xff
c_s = c_a + com_1 + com_2 + dat_1 + dat_2

else:
    com_2 = 0
    dat_1 = 0    #pan Speed: 0x00 -> 0x3f
    dat_2 = 0    #tilt Speed: 0x00 -> 0x3f
    c_s = c_a + com_1 + com_2 + dat_1 + dat_2
envio = chr(sync)+chr(c_a)+chr(com_1)+chr(com_2)+chr(dat_1)+chr(dat_2)+chr(c_s)
ser.write(envio)

if cv2.waitKey(1) & 0xff == ord('q'):
    break

cap.release()
cv2.destroyAllWindows()
```

Para concluir con la explicación del controlador, se procede a evaluar manualmente un estado del sistema. Se toma un valor aleatorio de posición y otro de velocidad sobre uno de los ejes: dist = 50, movi = 40.

En este momento se puede analizar los valores aleatorios que tenemos como entradas y notar que el objetivo se encuentra más o menos en la cuarta parte de camino desde el centro hacia el lado derecho, y la velocidad que tiene la cámara es 2/3 del total hacia la derecha, si estuviéramos hablando del eje X. Esto quiere decir que la cámara va acercándose al objetivo con una velocidad relativamente alta para la posición que tiene el objetivo, lo cual permite anticipar que el próximo

estado será una velocidad menor y en el mismo sentido, es decir, la cámara va a disminuir su velocidad.

Ahora se procede a ubicar los conjuntos sobre los que estos valores actúan: Ya que la distancia es 50, el punto tiene cierto grado de pertenencia al conjunto “centro” y cierto grado de pertenencia al conjunto “d_cerca”.

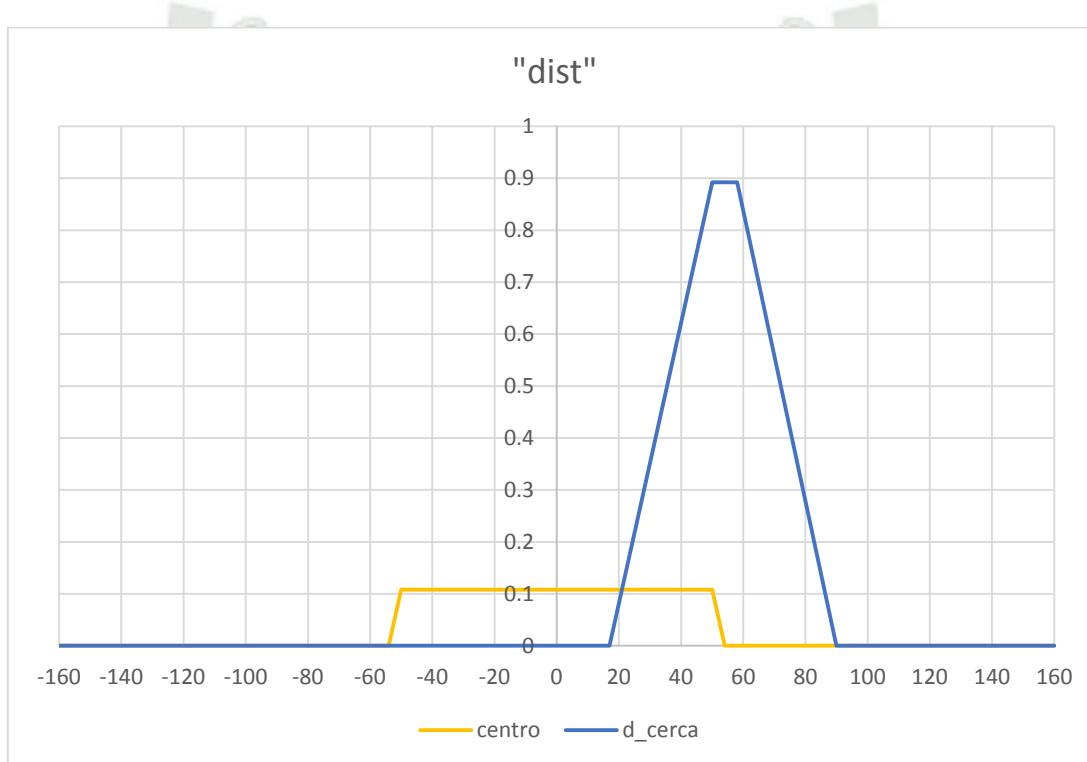


Figura 24 Ejemplo de entrada “dist”
Fuente: Elaboración propia

De la misma forma, ya que la velocidad del estado anterior tiene un valor de 40, tiene cierto grado de pertenencia al conjunto “d_lento” y cierto grado de pertenencia al conjunto “d_rapido”.

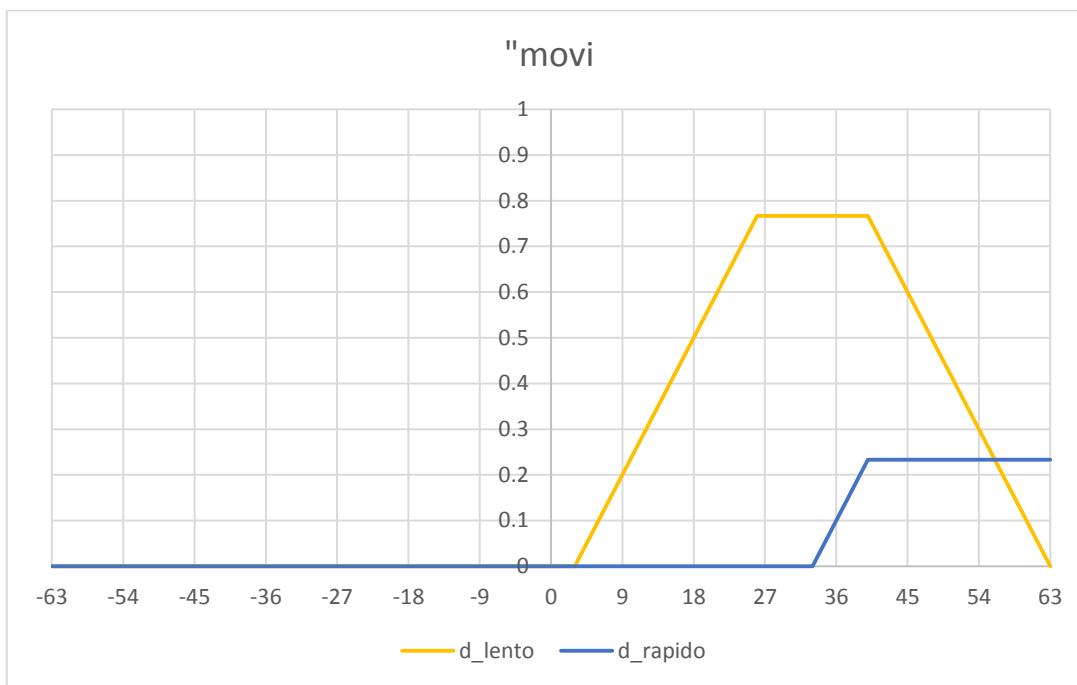


Figura 25 Ejemplo de entrada “movi”
Fuente: Elaboración propia

En este momento se procede a aplicar las reglas difusas que se han estipulado. Podemos notar que cuatro de las reglas definidas anteriormente tienen inferencia sobre estos conjuntos:

RULE 19: IF dist IS centro AND movi IS d_lento THEN velo IS cero;

RULE 20: IF dist IS centro AND movi IS d_rapido THEN velo IS d_muy_lento;

RULE 24: IF dist IS d_cerca AND movi IS d_lento THEN velo IS d_lento;

RULE 25: IF dist IS d_cerca AND movi IS d_rapido THEN velo IS d_muy_lento;

El método de activación que hemos definido es el del mínimo, de manera que se lleva a cabo la inferencia por Mamdani de los conjuntos sobre las salidas correspondientes a las respectivas reglas, y estas quedan de la siguiente manera:

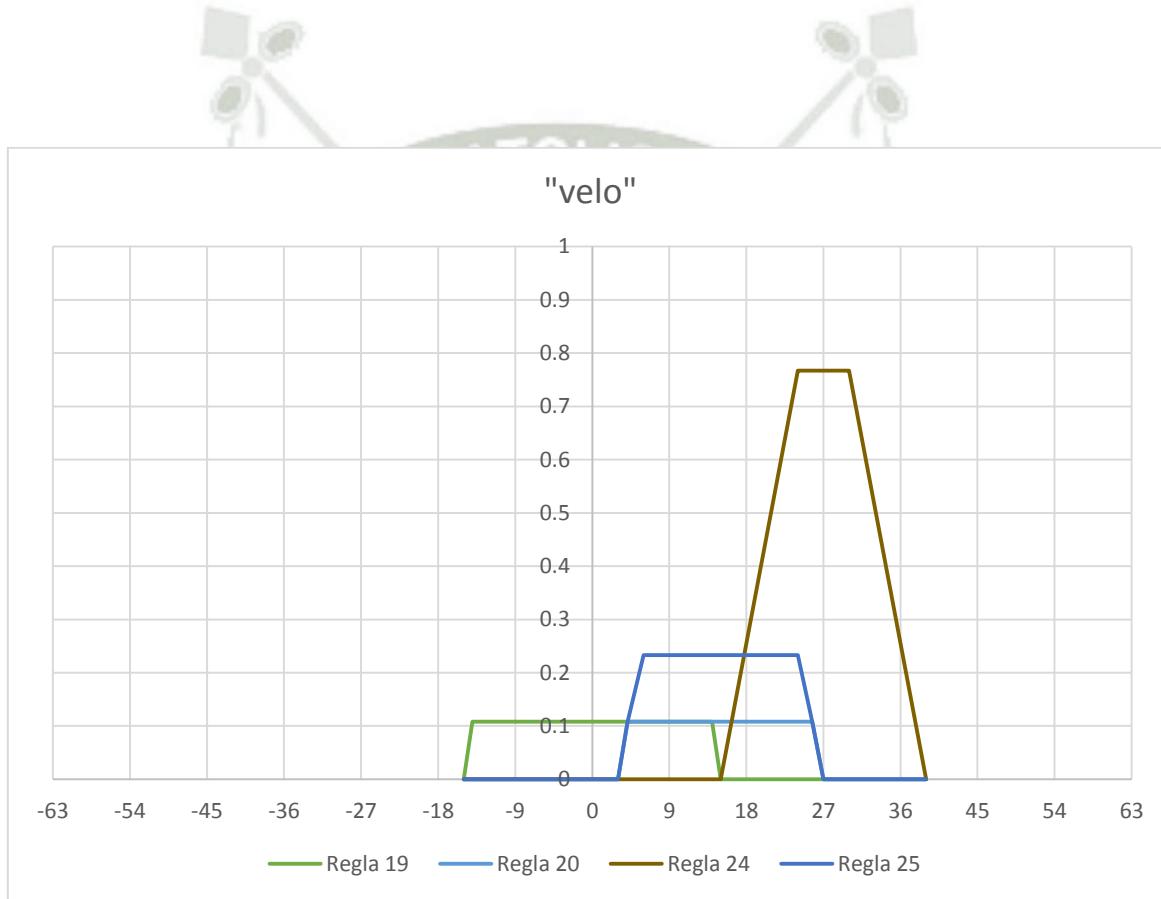


Figura 26 Ejemplo Inferencia
Fuente: Elaboración propia

El método de agregación que se ha definido es el del máximo, de manera que se procede a aplicar dicho operador a los cuatro conjuntos resultantes, quedando de la siguiente manera:

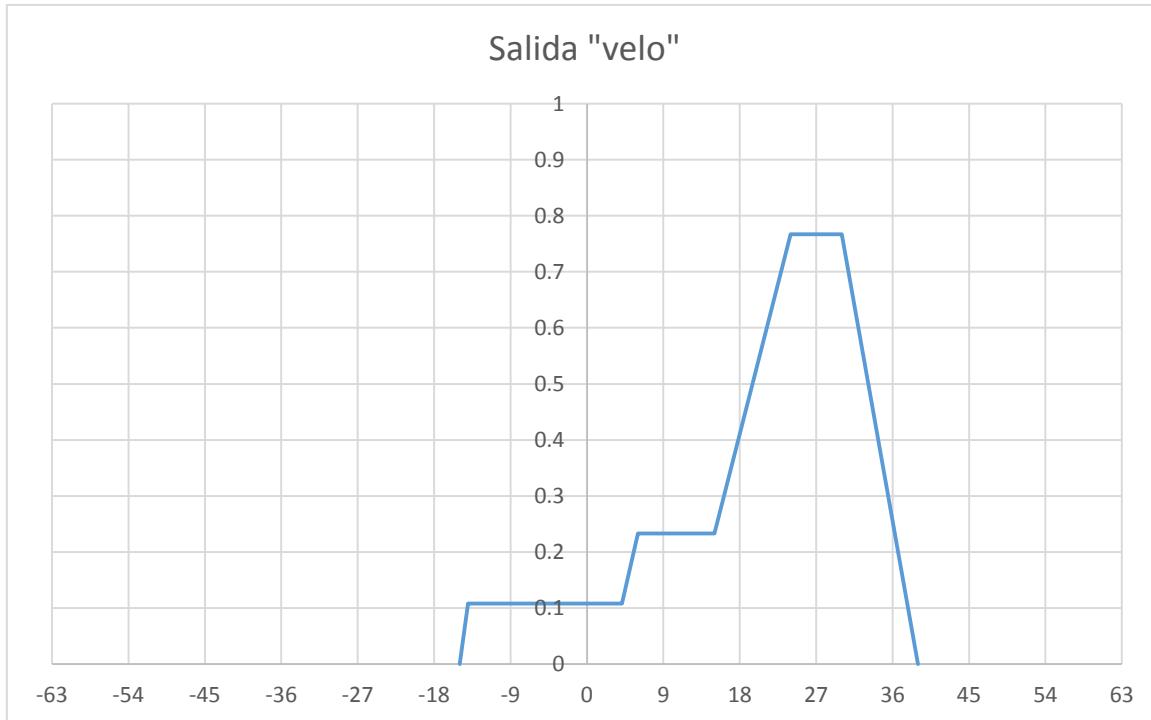


Figura 27 Ejemplo de Salida
Fuente: Elaboración propia

La salida, hasta este punto, sigue siendo un conjunto difuso, que no puede ser tomado como dato para la siguiente etapa, de manera que se requiere convertirla en un valor crisp. Para eso utilizamos el método de defusificación de centros de gravedad. Por tratarse de un cálculo manual se utiliza la forma

continua, la cual va a tener una pequeña diferencia con la que realmente entregara el sistema, pero mantendrá la tendencia. Se encuentra el centro de gravedad del conjunto mostrado en la figura 27:

$$\text{COG} = 22$$

Nuestra respuesta es, finalmente, 22. Esto concuerda con el comportamiento descrito al comienzo del ejemplo.

3.4. Transmisión de datos

Esta etapa se encarga de transmitir la trama, generada en la etapa anterior, al mecanismo de la cámara.

Para que el controlador pueda enviar los datos, de forma serial y a través de los puertos, se recurre al uso de la librería “serial” la cual debe ser previamente importada y se hace uso del comando “Serial” de dicha librería, el cual se carga en nuestra herramienta “ser”, en la que se especifica también el puerto por el que va a transmitir y la tasa de bits.

```
ser=serial.Serial ('/dev/ttyACM0', 9600,timeout=0.5)
```

En ese momento el controlador está listo para transmitir, y al final de cada etapa de Evaluación de Datos envía los datos por el puerto asignado.

```
ser.write(envio)
```

En este momento la trama que contiene los datos ha sido enviada al mecanismo encargado de mover la cámara. De esta manera concluye el trabajo del controlador en el ciclo presente, y se da paso al comienzo del siguiente ciclo con una nueva captura de imagen.

El mecanismo recibe los datos en transmisión serial y a través de un par de cables que cumplen con el estándar RS-485. Para poder entregar los datos a dicho par, se utiliza un trnceptor de baja potencia de código MAX485, el cual convierte una señal RS-232 a una señal RS-485 para transmisión y recepción de datos. En nuestro caso no hay ningún dato que se pueda recibir del mecanismo de movimiento de la cámara, de manera que este trnceptor se comportara únicamente como un transmisor. La conexión y forma de trabajo de este dispositivo se muestra en la hoja de datos, en el Anexo G.

Debido a que la netbook que utilizamos como controlador no cuenta con un puerto serial propiamente dicho, necesitamos de una interfaz que sirva para conectar su puerto USB con nuestro trnceptor. Para este fin, se utilizará un chip 18F4550 de MICROCHIP, cuya única función será recibir los datos del puerto USB y entregarlos en estándar RS-232, para que así pueda leerlos nuestro MAX485.

Dado que el chip no va a cumplir mayor función que la de recibir datos y luego transmitirlos, es decir, no va a funcionar como un controlador, se le cargo un programa libre publicado en la página de MICROCHIP, el cual tiene entre sus funciones precisamente la de recibir los datos del puerto USB y transmitirlos en estándar RS-232. A continuación se muestra la porción del código que se encarga de dicha función:

```
// ****
void ProcessIO(void)
{
    BlinkUSBStatus();

    if((USBDeviceState < CONFIGURED_STATE) || (USBSuspendControl==1)) return;

    LastRS232Out = getsUSBUSART(RS232_Out_Data,1000); //until the buffer is free.

    if(LastRS232Out > 0)
    {
        for(j=0;j<LastRS232Out;j++)
        {
            while(TXSTAbits.TRMT==0)
            {}
            TXREG=RS232_Out_Data[j];
        }
    }
}

//end ProcessIO
```

A continuación se muestra el circuito que se utiliza como interfaz entre la netbook y la cámara.

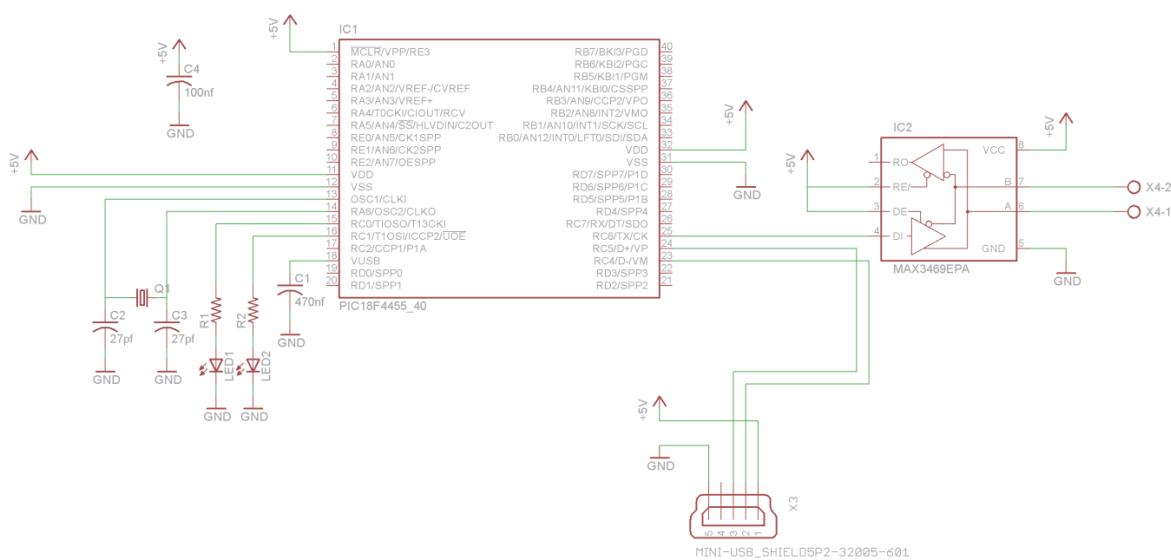


Figura 28 Diagrama del circuito de la interfaz
Fuente: Elaboración propia

Final Dicho circuito va montado sobre una placa, la cual debe ser quemada con el circuito impreso mostrado a continuación en una medida de 8cm x 8cm.

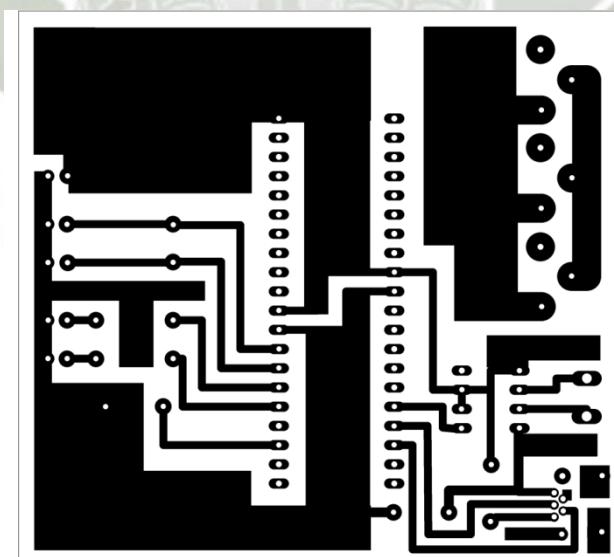


Figura 29 Circuito impreso para la interfaz
Fuente: Elaboración propia

Las especificaciones técnicas del chip se encuentran en el Anexo H.

3.5. Movimiento de la cámara

La cámara está dotada, de fábrica, de un mecanismo que se encarga de rotarla en dos ejes: el horizontal (o panning) y el vertical (o tilt). Esto depende de la posición de montaje, pero para usos de video vigilancia se monta, de manera estándar, sobre el campo que se requiere vigilar. Respetando este hecho, lo idóneo será montar la cámara en una posición más alta que el objetivo, para que todo quede bajo el rango de visión de media esfera que tiene la cámara y para que el “pan” siga siendo el eje horizontal y el “tilt” siga siendo el vertical.

CAPITULO IV – PRUEBAS DE FUNCIONAMIENTO

En la placa se ha instalado, como se mencionó anteriormente, tres puertos RCA, conectados en paralelo para utilizarlos para espejar la señal de la cámara y enviar una a la computadora para el procesamiento, y otra al usuario, quien suele no tener interés en ver el código. Esto nos permite tener una imagen limpia para el usuario, mientras que en la pantalla de la computadora se pueden mostrar ciertos parámetros cuya medición resultaría útil para las diferentes pruebas de funcionamiento.

El objetivo para las pruebas será una pequeña esfera de color verde. Se utilizara una esfera debido a que este objeto se ve circular desde cualquier dirección. El tono de verde de este objeto está dentro del rango de colores para el que fue programada la etapa de procesamiento de imágenes.

Como en toda aplicación de visión artificial, se requiere, en lo posible, controlar el entorno. Por tanto se requiere de un ambiente en el que no haya más objetos de ese color y de esa forma.

A continuación se mostrarán algunas pruebas sobre los parámetros que se consideran importantes de medir. Para tener un mejor control sobre el dispositivo, se realizaran dichas pruebas sin conectar el par que le da las órdenes al mecanismo encargado de mover la cámara.

Es necesario conocer los valores de posición y velocidad de cada estado, así como el tiempo en que se presentan. Los valores de velocidad y posición se imprimen directamente sobre la pantalla del terminal.

```
print "dx: ", dx
print "dy: ", dy
print "vx: ", vx
print "vy: ", vy
```

Para el caso del tiempo es necesario cargar la librería “time”, la cual debe ser importada al inicio del programa, inicializada antes del bucle infinito y de la cual se debe tomar el valor en cada estado.

```
import time
start_time = time.time()
t = time.time() - start_time
print "t", t
```

A continuación se muestra una toma de pantalla de la netbook, en la que se puede ver la ventana del terminal donde se imprimen los valores de la posición, la velocidad y el tiempo. Esto sirvió para analizar el tiempo de procesamiento de cada ciclo. Después de varias pruebas se encontró que el periodo estimado por ciclo varía entre los 80ms y los 210ms. Esto nos da un mínimo de 4 órdenes por

segundo en el peor de los casos. El hecho de visualizar estos datos nos ayuda, también, a darnos cuenta de la posición en que se está detectando al objetivo y de las órdenes que se envían al actuador, de manera que fue lo primero que se utilizó para comprobar el funcionamiento del sistema.

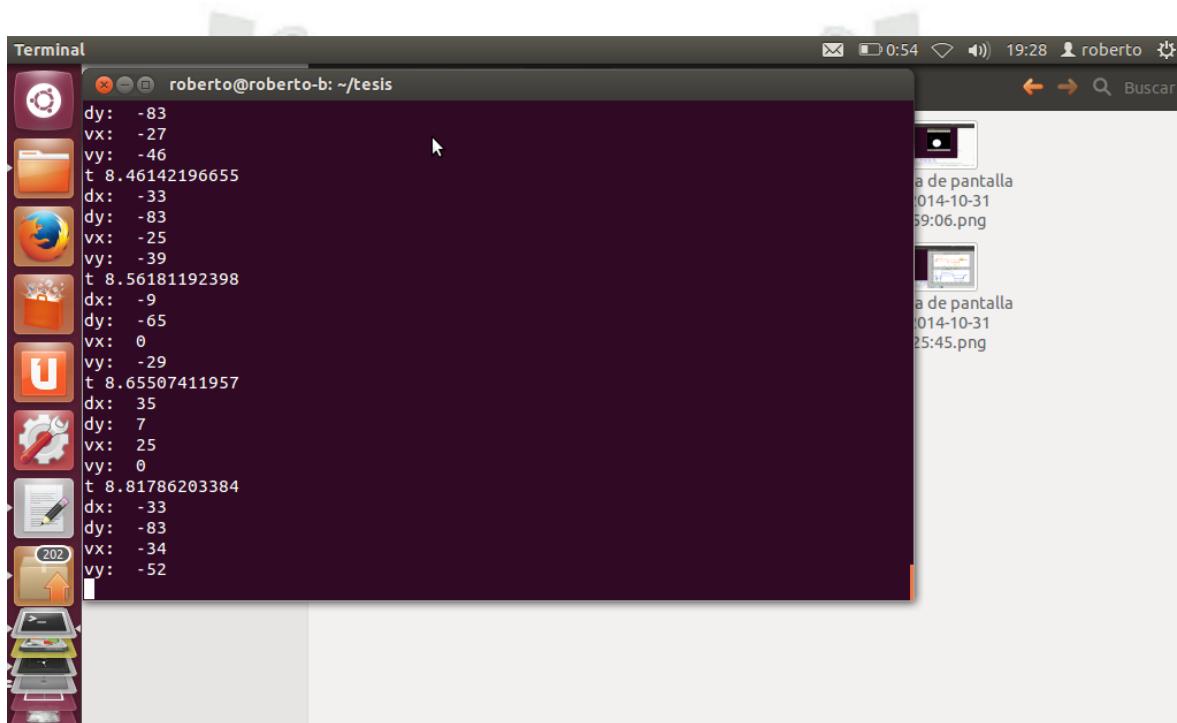


Figura 30 Impresión de los parámetros “t”, “dx”, “dy”, “vx” y “vy”
Fuente: Elaboración propia

Luego de varias pruebas, se notó que los valores del tiempo entre cada estado varían entre 0.08 y 0.21 segundos. Esto nos da un máximo de 12.5 imágenes por segundo y un mínimo de 4.8. Esto nos indica que en el peor de los casos se estarían procesando 4.8 imágenes por segundo, lo cual resulta relativamente lento, y que probablemente se vean algunos cambios bruscos de

velocidad. Pese a esto, en la pantalla de usuario se ve un movimiento bastante suave y una precisión aceptable.

Resulta útil mostrar una ventana donde se vea la imagen de la cámara y se marque sobre ella los objetos que presentan un color que se encuentra dentro del rango buscado. Para esto se hace uso de la función “imshow” de la librería “OpenCV” y se muestra la imagen que se obtiene después del filtro suavizador.

```
cv2.imshow('mask', gray_ff)
```

A continuación se muestran tres pares de imágenes, solidarias entre cada par. Las de la columna de la izquierda muestra las tomas de pantalla de la netbook en un estado aleatorio en el que se puso el objetivo en la pantalla y las de la izquierda muestran una fotografía tomada a un televisor en el que se espeja la señal de la cámara. Si vemos la ventana que se abre en la pantalla de la netbook y la comparamos con la fotografía tomada al televisor, podemos darnos cuenta de que la ventana “mask” muestra en color blanco la forma del objetivo y en color negro todo el resto de la imagen. Esto nos indica que está discriminando todos los pixeles de la imagen que no están dentro del rango de colores a detectar y nos está mostrando en blanco los pixeles que si se encuentran dentro de este rango.

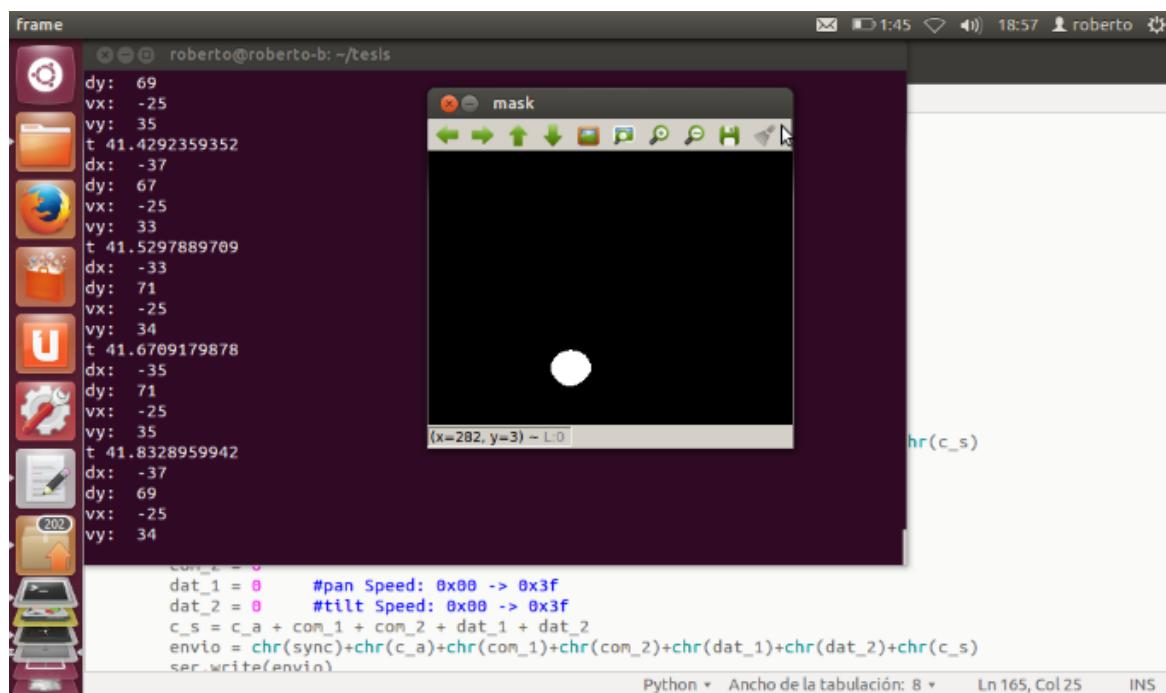


Figura 31 Comparación de la imagen procesada contra la original, para una posición inferior izquierda

Fuente: Elaboración propia

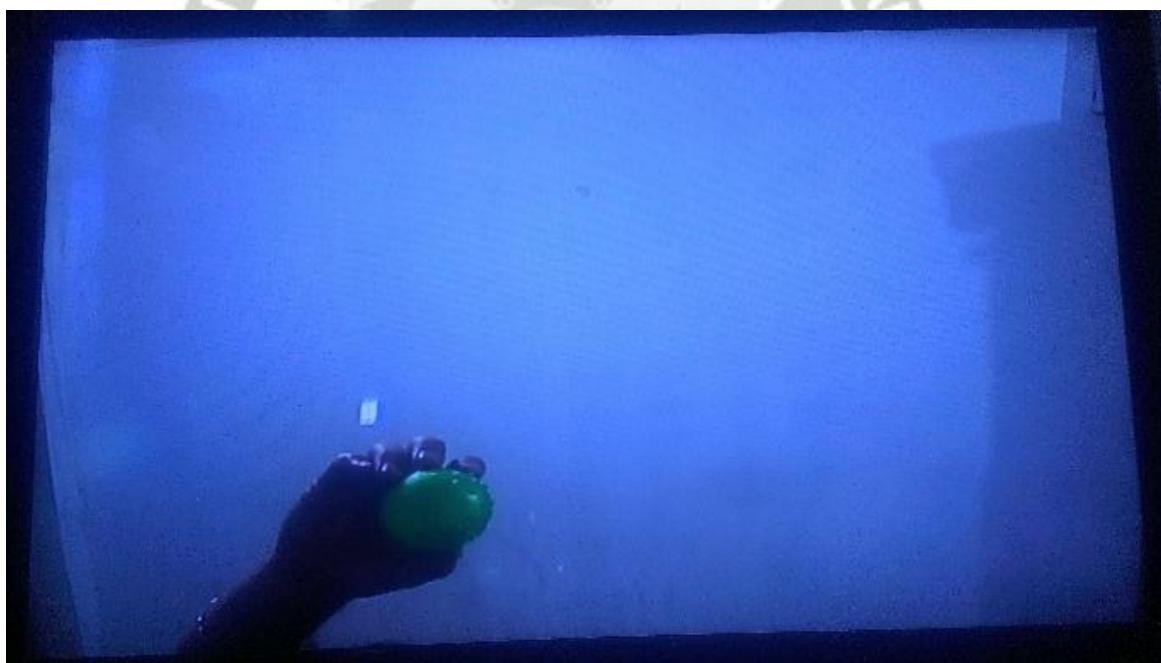


Figura 32 Comparación de la imagen procesada contra la original, para una posición inferior izquierda

Fuente: Elaboración propia

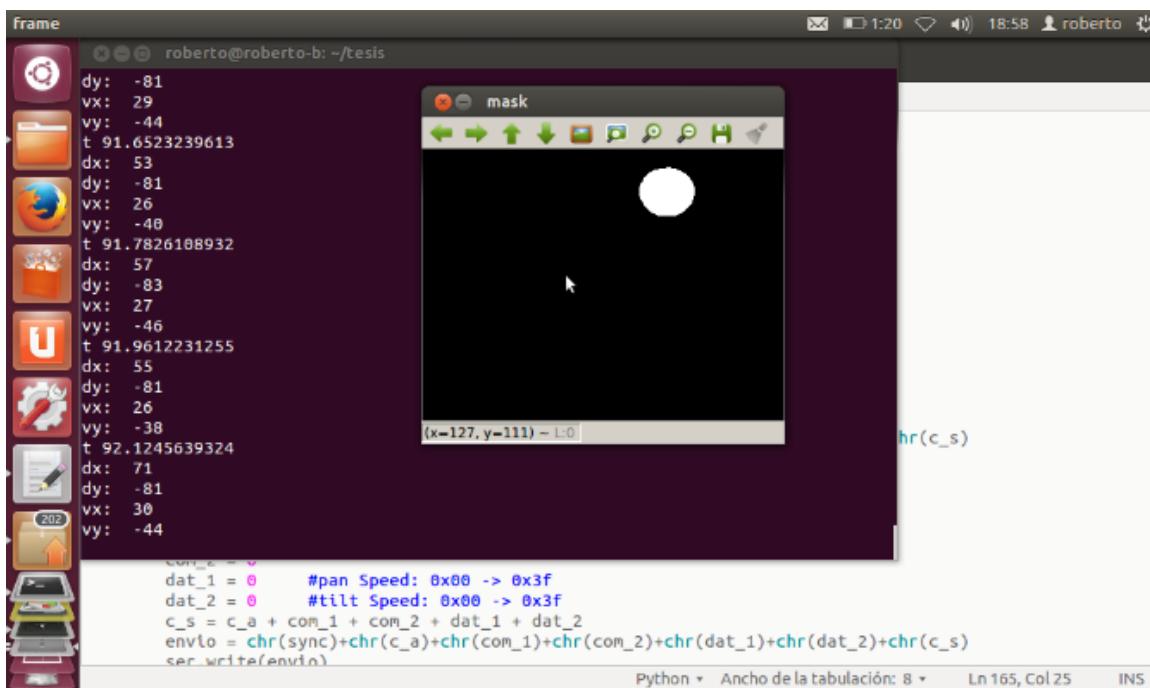


Figura 33 Comparación de la imagen procesada contra la original, para una posición superior central

Fuente: Elaboración propia



Figura 34 Comparación de la imagen procesada contra la original, para una posición superior central
Fuente: Elaboración propia

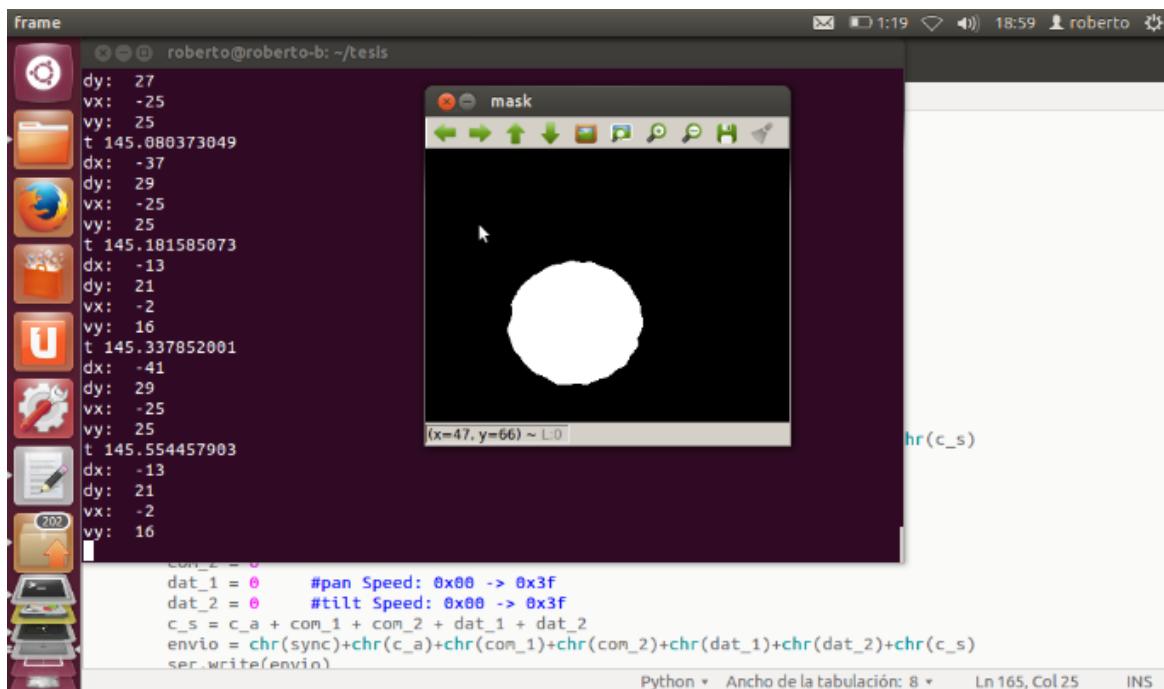


Figura 35 Comparación de la imagen procesada contra la original, para una posición ligeramente inferior y ligeramente izquierda
Fuente: Elaboración propia



Figura 36 Comparación de la imagen procesada contra la original, para una posición ligeramente inferior y ligeramente izquierda

Fuente: Elaboración propia

Por otro lado, también resulta útil señalar los objetos que, luego de encontrarse incluidos dentro del rango de colores, también se han detectado como círculos. Para se dibuja un punto rojo en su centro y una circunferencia verde alrededor del objeto, y se muestra dicha imagen, apoyados en la librería “circle” de OpenCV, y se muestra la imagen con el mismo método que se usó para la prueba anterior.

```
cv2.circle(frame,(i[0],i[1]),i[2],(0,255,0),2)
cv2.circle(frame,(i[0],i[1]),2,(0,0,255),3)

cv2.imshow('frame', frame)
```

Tal como se hizo en la prueba anterior, a continuación se muestran tres pares de imágenes. La prueba funciona de la misma forma que la anterior, pero además se muestra una ventana en la que se dibuja un punto rojo sobre el centro del objetivo y una línea verde sobre su circunferencia. Esto nos indica que se ha detectado el centro del objetivo y que además se tiene su dimensión.

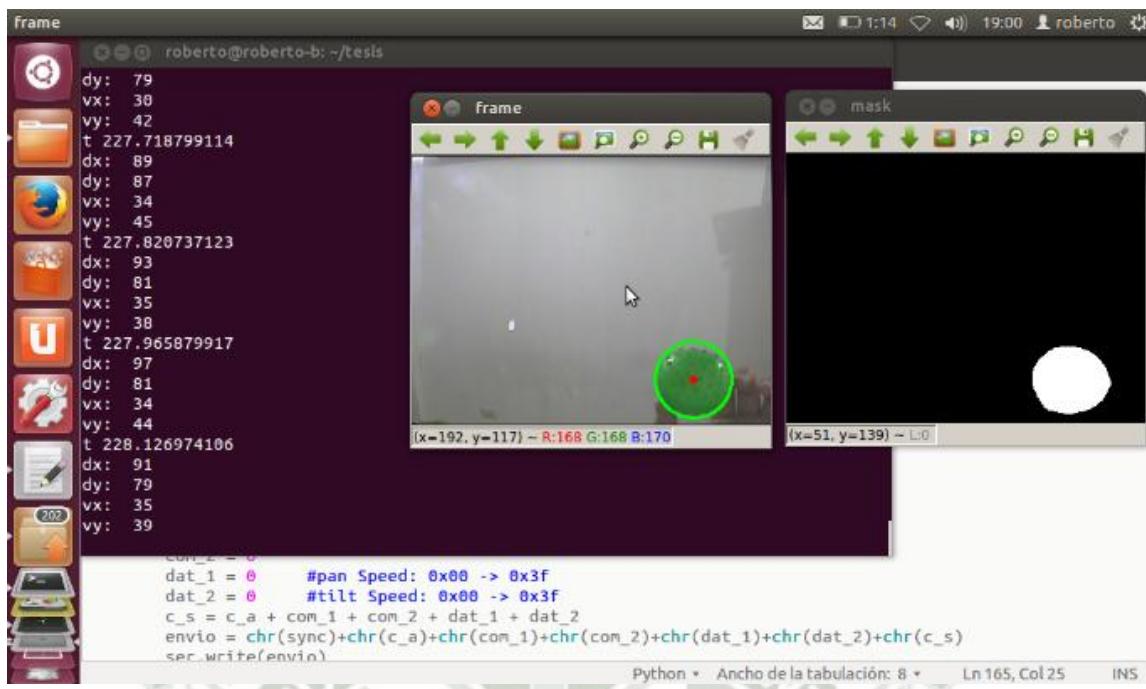


Figura 37 Comparación de la imagen procesada con la original, para una posición inferior derecha

Fuente: Elaboración propia



Figura 38 Comparación de la imagen procesada con la original, para una posición inferior derecha

Fuente: Elaboración propia

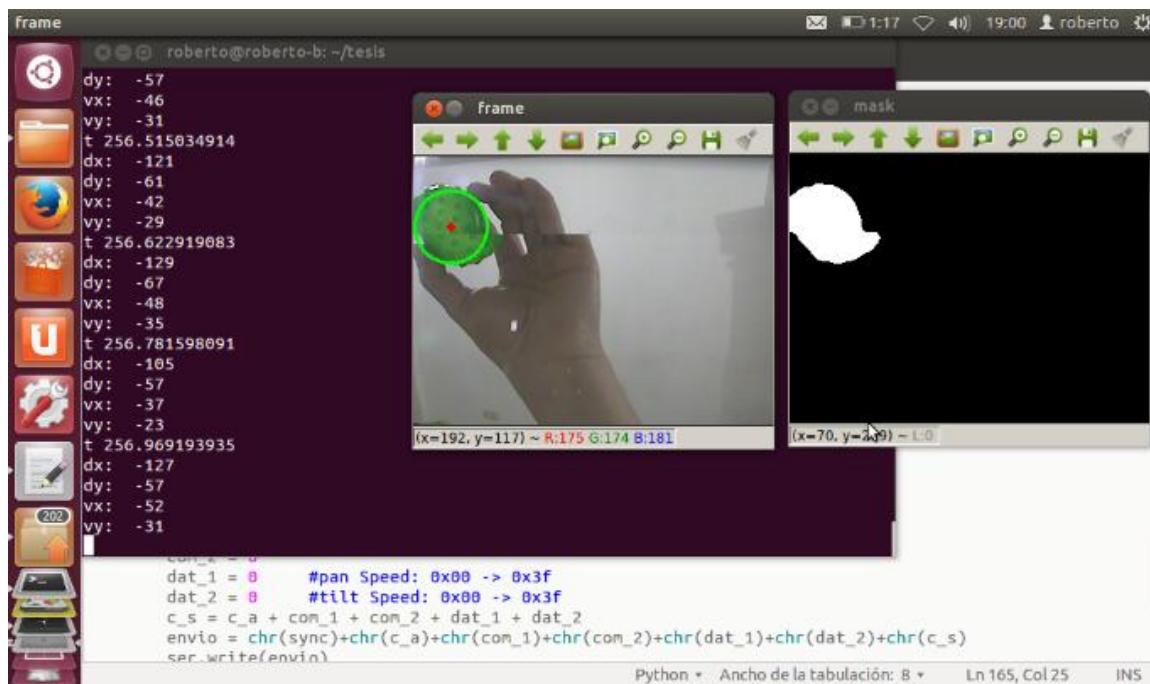


Figura 39 Comparación de la imagen procesada con la original, para una posición superior izquierda

Fuente: Elaboración propia



Figura 40 Comparación de la imagen procesada con la original, para una posición superior izquierda

Fuente: Elaboración propia

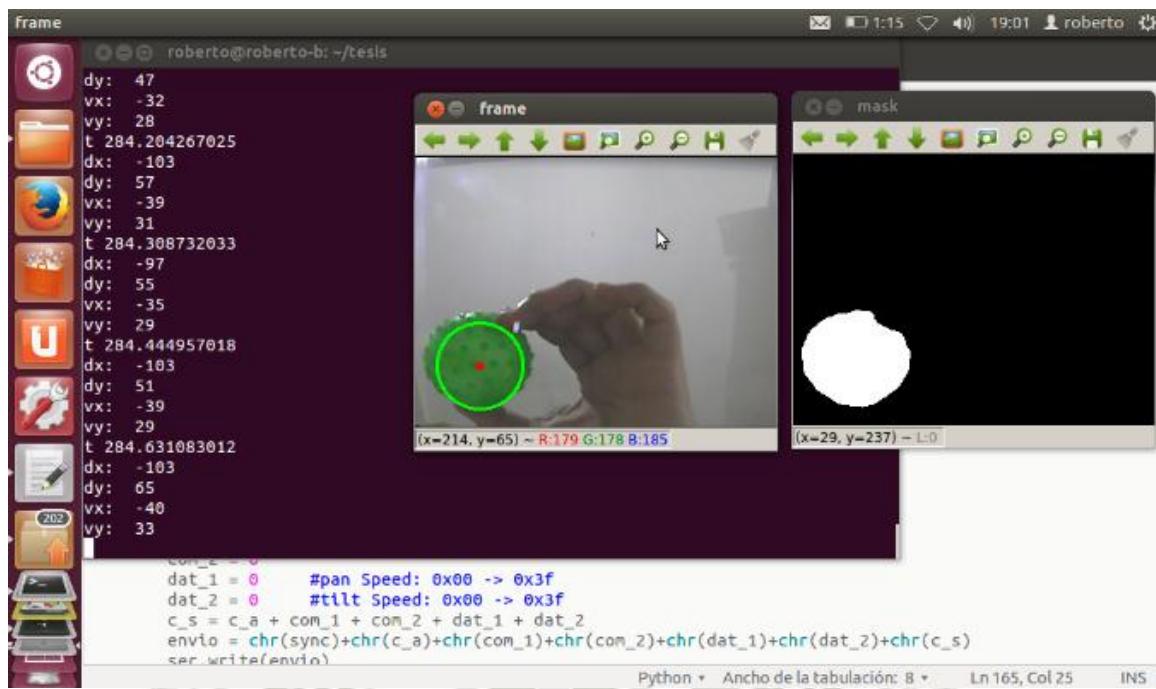


Figura 41 Comparación de la imagen procesada con la original, para una posición inferior izquierda

Fuente: Elaboración propia



Figura 42 Comparación de la imagen procesada con la original, para una posición inferior izquierda

Fuente: Elaboración propia

Finalmente, es importante conocer el comportamiento que tuvo el sistema durante su jornada de trabajo, de manera que resulta muy útil mostrar una gráfica en la que se represente la posición del objeto y la velocidad de la cámara comparados con el tiempo.

Para esto fue necesario el uso del comando “pyplot” de la librería “Matplotlib”, la cual, como en todos los casos anteriores, debe ser previamente importada. Se debe también inicializar las variables que serán utilizadas en esta tarea. Dentro del bucle se debe ir guardando los valores de cada estado. Como resultado, mediante esta función, obtenemos una gráfica que se presentara al interrumpir el proceso y que nos mostrara en dos subgraficas el comportamiento del objetivo a seguir y el de la cámara, en cada uno de los ejes.

```
import matplotlib.pyplot as plt

pt = [0]
pdx= [0]
pdy= [0]
pxx= [0]
pvy= [0]
s = 0

    pt.insert(s,t)
    pdx.insert(s,dx)
    pdy.insert(s,dy)
    pxx.insert(s,vx)
    pvy.insert(s,vy)
    s = s + 1

plt.subplot(211)
plt.title('X (Amarillo: DX      Rojo: VX) ')
plt.plot(pt,pdx,'y')
plt.plot(pt,pxx,'r')
```

```
plt.subplot(212)
plt.title('Y (Verde: DY      Azul: VY)')
plt.plot(pt,pdy,'g')
plt.plot(pt,pvy,'b')

plt.show()
```

A continuación se muestran siete imágenes; seis de ellas son fotografías tomadas a la pantalla de un televisor, el cual proyecta las imágenes emitidas por la cámara y la última muestra una ventana con dos gráficas. Estas seis fotografías muestran los seis estados que han sido tomados en cuenta para esta prueba en el orden correspondiente. Las dos gráficas, una para el eje X y la otra para el eje Y, nos muestran una comparación entre la posición del objetivo y la velocidad que se envía al actuador. Lo primero que podemos notar es que en las gráficas describen las posiciones que se muestran en las seis fotografías, en el orden correcto. También podemos observar que el comportamiento de la línea de velocidad es muy parecido al de la línea de posición, pese a la diferencia que hay en cuanto a la magnitud y a la vibración en la línea. Esto nos indica, ahora, que el controlador está funcionando de manera correcta y en su momento fue crucial para las pruebas sobre dicho controlador, en cuanto a la configuración de los sets difusos, determinación de las reglas y afinamiento del sistema.



Figura 43 Imagen del objetivo en una posición aleatoria (a)
Fuente: Elaboración propia



Figura 44 Imagen del objetivo en una posición aleatoria (b)
Fuente: Elaboración propia



Figura 45 Imagen del objetivo en una posición aleatoria (c)

Fuente: Elaboración propia



Figura 46 Imagen del objetivo en una posición aleatoria (d)

Fuente: Elaboración propia



Figura 47 Imagen del objetivo en una posición aleatoria (e)
Fuente: Elaboración propia



Figura 48 Imagen del objetivo en una posición aleatoria (f)
Fuente: Elaboración propia

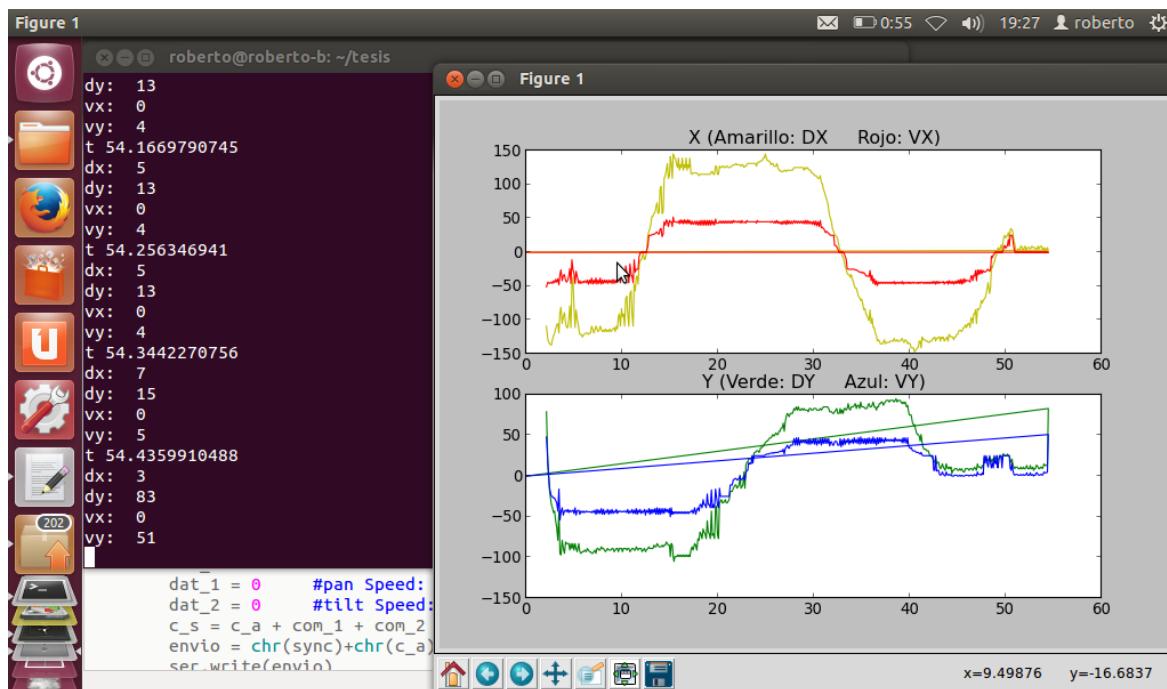


Figura 49 Grafica de comparación posición-velocidad en X e Y de los estados anteriores (a), (b), (c), (d), (e) y (f)

Fuente: Elaboración propia

Como se indicó anteriormente, todas las pruebas descritas en esta sección se hicieron sin conectar el actuador al controlador, con la intención de controlar mejor las posiciones que se mostraron. Esto debido a que dichas pruebas mostraban la función del sensor y la del actuador. La función del actuador consiste únicamente moverse, de manera que resulta muy complicado de demostrar en este documento. Es por eso que el sistema estará disponible, con todos los códigos mencionados en este capítulo, para las diferentes pruebas que se requieran.

CONCLUSIONES

- Se diseñó e implementó un sistema de video que permite seguir a un patrón objetivo previamente especificado. El sistema utiliza una técnica de inteligencia artificial, como lo es la Lógica Difusa.
- Se logró capturar las imágenes desde una cámara analógica y convertirlas a digital para ser ingresadas a una etapa de procesamiento capaz de encontrar las coordenadas de un patrón objetivo y entregársela a un controlador.
- Se diseñó un controlador difuso con dos entradas, una salida y 35 reglas apoyados en la librería “pyfuzzy”, el cual se encarga determinar la velocidad que deberá tomar el mecanismo encargado de mover la cámara y posteriormente convertirla en instrucciones.
- Se programó el sistema para entregar los datos en un estándar y bajo un protocolo existente y funcional para el movimiento de la cámara, de manera que pueda ser utilizado con la mayoría de equipos fabricados para la industria de la videovigilancia.
- Se utilizó la última versión de Ubuntu de Linux como sistema operativo; se utilizaron librerías de difusión gratuita; se utilizó Python como lenguaje de programación y se corre el programa directamente desde el terminal. Estas condiciones le dan un muy buen grado de versatilidad y un costo nulo en

cuanto a licencias, lo cual lo convierte en un buen punto de partida para la búsqueda de soluciones más específicas y aplicables en la industria.



RECOMENDACIONES

- El procesamiento de imágenes como método de reconocimiento de patrones resulta funcional cuando estos son tan simples como colores y figuras geométricas. Mientras los patrones se hacen más complejos, este procesamiento de imágenes se hace más lento y el método pierde su efectividad. Se recomienda promover la investigación de métodos avanzados al tratamiento de imágenes, como por ejemplo las redes neuronales.
- Se recomienda promover la utilización de dispositivos de uso diario (cámaras web, computadoras personales, etc.) para proyectos de investigación debido a su bajo costo y prestaciones bastante aceptables. Al poder controlar estos dispositivos en la etapa de pregrado, mejoran las probabilidades de poder controlar de forma más eficiente otros dispositivos más especializados.
- Como en todo proyecto de visión artificial, la probabilidad de éxito se incrementa considerablemente al controlar el entorno, en lo concerniente a iluminación e interferencias. El control sobre la iluminación asegura que la tonalidad del color sea la real, y el control sobre las interferencias asegura que el sistema no se desvíe por detectar otros elementos con patrones parecidos al objetivo.
- Una vez programado el controlador difuso, bajo los parámetros estipulados inicialmente, se requiere afinarlo mediante un procedimiento repetitivo de pruebas y ajustes sobre dichos parámetros y sobre las reglas.

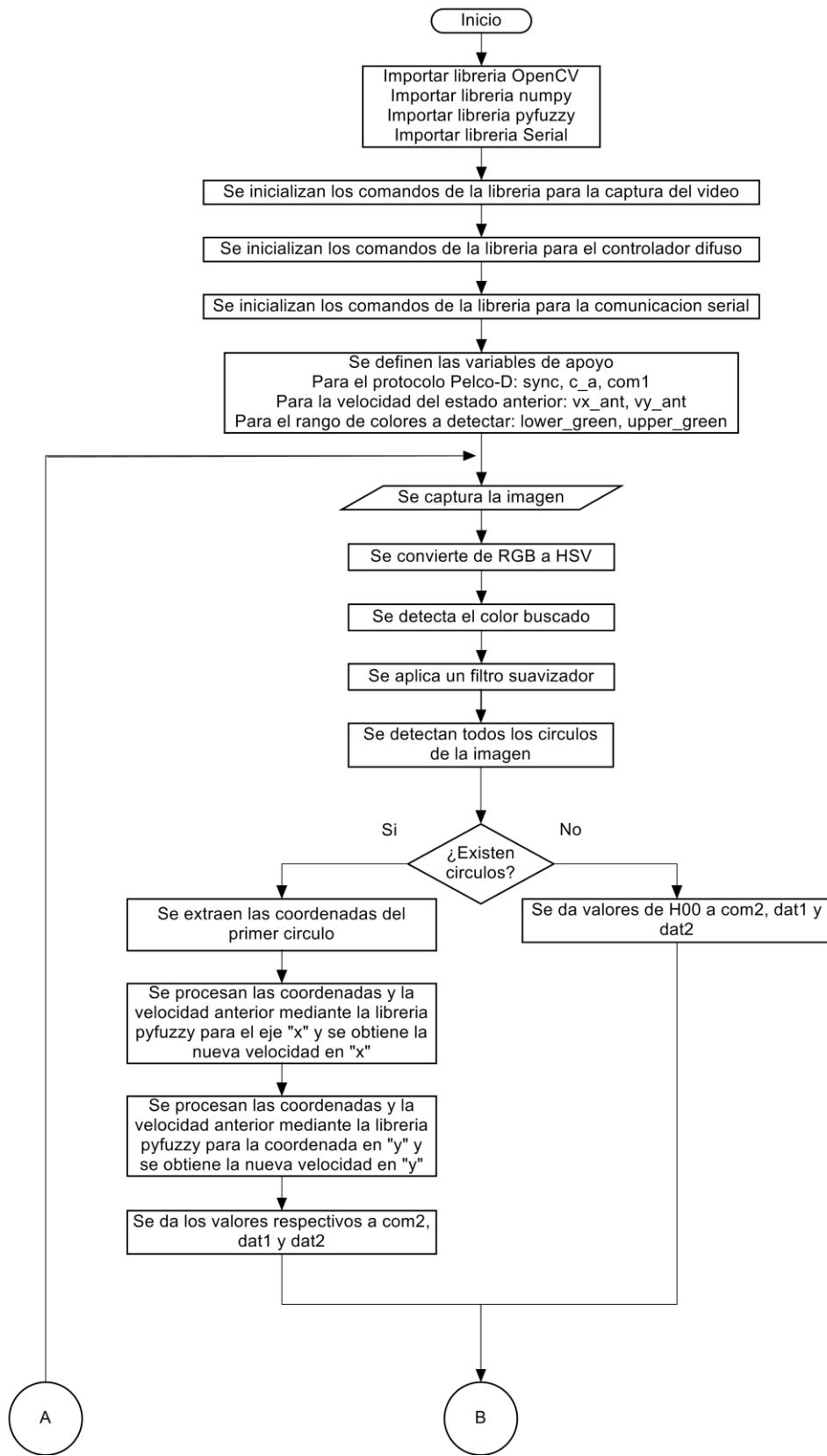
BIBLIOGRAFÍA

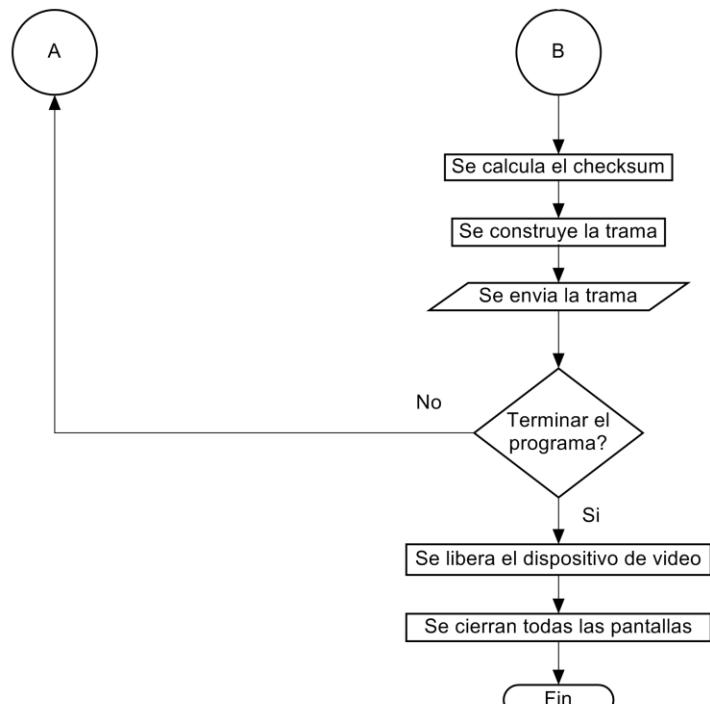
- Cuevas, E. "Procesamiento Digital de Imágenes usando Matlab & Simulink", Editorial Alfaomega Grupo Editor S.A., México 2010
- Garcia-Calderon, E. "Television 1, Fundamentos, Dispositivos, TV monocroma"
- Escuela Técnica Superior de Ingenieros en Telecomunicaciones, 1986
- George Klir J. "Fuzzy Sets and Fuzzy Logic and Applications", Editorial Prentice Hall, 1995
- Allison, H.C. "The three-dimensional graphical input method for architecture", IEEE Press Piscataway, 1978
- Dr. Vega Falcón, V. "Herramientas gerenciales para toma de decisiones", 2008
- Hernandez I. "Procesamiento digital de señales", Editorial RHC, 2004
- Lopez, H. "Detección y seguimiento de objetos con cámaras en movimiento", Universidad Autónoma de Madrid, 2011
- Corrales, J.A. "Diseño de una mini-cámara motorizada para seguimiento de objetos" Universidad de Alicante, 2012
- "Object Tracking and Kalman Filtering", Department of Computer Engineering, University of California and Santa Cruz

- Arduino, Foro de proyectos de Arduino
Pelco-D Camera Control with Dual Axis Hall Effect Joystick, UncleBone
<http://forum.arduino.cc/index.php?topic=67590.0>
- Departamento de Matematica Aplicada a las Tecnologias de la Informacion
Tutorial de Introduccion a la lógica difusa
<http://www.dma.fi.upm.es/java/fuzzy/tutfuzzy/indice.html>
- COMMFRONT, Communications made easy
Pelco-D Control Tutorials
http://www.commfront.com/RS232_Examples/CCTV/Pelco_D_Pelco_P_Examples_Tutorial.HTM
- FONOSTRA
Fundamentos de diseño
<http://www.fotonostra.com/grafico/rgb.htm>
- Colección de tesis digitales
Simulación en simmechanics de un sistema de control difuso para el robot UDLAP
http://catarina.udlap.mx/u_dl_a/tales/documentos

ANEXOS

ANEXO A: Diagrama de flujo del programa





ANEXO B: Código de la aplicación seguimiento.py

```
#importa video
import numpy as np
import cv2
import cv2.cv as cv
```



```
#importa fuzzy
import fuzzy.storage.fcl.Reader
```

```
#importa Serial COM
import serial
```

```
#inicia video
cap = cv2.VideoCapture(1)
```

```
ret=cap.set(3, 320)
ret=cap.set(4, 240)
```

```
#inicia fuzzy
system_velo = fuzzy.storage.fcl.Reader.Reader().load_from_file("camara_fuzzy.fcl")
```

```
my_input = {
    "dist" : 0.0,
    "movi" : 0.0
}
```

```
my_output = {
    "velo" : 0.0
}
```

```
#inicia serial COM
ser=serial.Serial('/dev/ttyACM0',9600,timeout=0.5)
```

```
#PELCO-D
sync= 0xff
c_a= 0x01
com_1= 0x00
```

```
#variables velocidad inicial
vx_ant = 0
vy_ant = 0
```

```
#verde RGB 0 255 0 -> 60 255 255
lower_green = np.array([30,50,50])
upper_green = np.array([90,255,255])
```

```
while(True):
    ret, frame = cap.read()
```

```
#Covierte de RGB a HSV
```

```
hsv = cv2.cvtColor(frame, cv2.COLOR_BGR2HSV)
```

```
#Detecta objetos verdes
```

```
mask = cv2.inRange(hsv, lower_green, upper_green)
```

```
#filtro
```

```
gray_ff = cv2.medianBlur(mask,9)
```

```
#detecta circulos
```

```
circles =
```

```
cv2.HoughCircles(gray_ff, cv.CV_HOUGH_GRADIENT, 2, 40, param1=50, param2=30, minRadius=10, maxRadius=50)
```

```
if circles is not None:
```

```
    #convierte formato de datos
```

```
circles = np.uint16(np.around(circles))
```

```
    #obtiene centro del circulo
```

```
cx = circles[0,0,0]
```

```
cy = circles[0,0,1]
```

```
dx = cx-160
```

```
dy = cy-120
```

```
#Evalua fuzzy
```

```
my_input["dist"] = dx
```

```
my_input["movi"] = vx_ant
```

```
system_velo.calculate(my_input, my_output)
```

```
vx = int(my_output["velo"])
```

```
vx_ant = vx
```

```
my_input["dist"] = dy*4/3
```

```
my_input["movi"] = vy_ant
```

```
system_velo.calculate(my_input, my_output)
```

```
vy = int(my_output["velo"])
```

```
vy_ant = vy
```

```
if vx == 0: #no mueve
```

```
vvx = vx
```

```
com_2 = 0x00 #L:0x04 R:0x02 U:0x08 D:0x10
```

```
if vx > 0: #mueve derecha
```

```
vvx = vx
```

```
com_2 = 0x02 #L:0x04 R:0x02 U:0x08 D:0x10
```

```
if vx < 0: #mueve izquierda
```

```
vvx = -vx
```

```
if vy == 0: #no mueve
    vvy = vy
    com_2 = com_2 + 0x00 #L:0x04 R:0x02 U:0x08 D:0x10

if vy > 0: #mueve arriba
    vvy = vy
    com_2 = com_2 + 0x10 #L:0x04 R:0x02 U:0x08 D:0x10

if vy < 0: #mueve abajo
    vvy = -vy
    com_2 = com_2 + 0x08 #L:0x04 R:0x02 U:0x08 D:0x10

dat_1 = vvx #pan Speed: 0x00 -> 0x3f Turbo: 0xff
dat_2 = vvy #tilt Speed: 0x00 -> 0x3f Turbo: 0xff

c_s = c_a + com_1 + com_2 + dat_1 + dat_2

else:
    com_2 = 0
    dat_1 = 0 #pan Speed: 0x00 -> 0x3f
    dat_2 = 0 #tilt Speed: 0x00 -> 0x3f
    c_s = c_a + com_1 + com_2 + dat_1 + dat_2

envio = chr(sync)+chr(c_a)+chr(com_1)+chr(com_2)+chr(dat_1)+chr(dat_2)+chr(c_s)
ser.write(envio)

if cv2.waitKey(1) & 0xff == ord('q'):
    break

cap.release()
cv2.destroyAllWindows()
```

ANEXO C: Código del FCL cámara_fuzzy.fcl

```
VAR_INPUT
    dist: REAL;
    movi: REAL;
END_VAR
```

```
VAR_OUTPUT
    velo: REAL;
END_VAR
```

FUZZIFY dist

```
TERM i_lejos := (-161,0) (-160,1) (-126,1) (-90,0);
TERM i_medio := (-126,0) (-90,1) (-54,0);
TERM i_cerca := (-90,0) (-54,1) (-17,0);
TERM centro := (-54,0) (-17,1) (17,1) (42,0);
TERM d_cerca := (17,0) (54,1) (90,0);
TERM d_medio := (54,0) (90,1) (126,0);
TERM d_lejos := (84,0) (126,1) (160,1) (161,0);
```

END_FUZZIFY

FUZZIFY movi

```
TERM i_rapido := (-64,0) (-63,1) (-33,0);
TERM i_lento := (-63,0) (-33,1) (-3,0);
TERM cero := (-33,0) (-3,1) (3,1) (33,0);
TERM d_lento := (33,0) (63,1) (63,0);
TERM d_rapido := (33,0) (63,1) (64,1);
```

END_FUZZIFY

DEFUZZIFY velo

```
TERM i_muy_rapido := (-64,0) (-63,1) (-51,0);
TERM i_rapido := (-63,0) (-51,1) (-39,0);
TERM i_medio := (-51,0) (-39,1) (-27,0);
TERM i_lento := (-39,0) (-27,1) (-15,0);
TERM i_muy_lento := (-27,0) (-15,1) (-3,0);
TERM cero := (-15,0) (-3,1) (3,1) (15,0);
TERM d_muy_lento := (3,0) (15,1) (27,0);
TERM d_lento := (15,0) (27,1) (39,0);
TERM d_medio := (15,0) (39,1) (51,0);
TERM d_rapido := (39,0) (51,1) (63,0);
TERM d_muy_rapido := (51,0) (63,1) (64,0);
ACCU: MAX;
METHOD : COG;
DEFAULT := 0;
END_DEFUZZIFY
```

RULEBLOCK No1

```
AND : MIN;
ACT : MIN;
```

```
RULE 1: IF dist IS i_lejos AND movi IS i_rapido THEN velo IS i_rapido;
RULE 2: IF dist IS i_lejos AND movi IS i_lento THEN velo IS i_muy_rapido;
RULE 3: IF dist IS i_lejos AND movi IS cero THEN velo IS i_muy_rapido;
RULE 4: IF dist IS i_lejos AND movi IS d_lento THEN velo IS i_muy_rapido;
```



RULE 6: IF dist IS i_medio AND movi IS i_rapido THEN velo IS i_medio;

RULE 7: IF dist IS i_medio AND movi IS i_lento THEN velo IS i_medio;

RULE 8: IF dist IS i_medio AND movi IS cero THEN velo IS i_rapido;

RULE 9: IF dist IS i_medio AND movi IS d_lento THEN velo IS i_muy_rapido;

RULE 10: IF dist IS i_medio AND movi IS d_rapido THEN velo IS i_muy_rapido;

RULE 11: IF dist IS i_cerca AND movi IS i_rapido THEN velo IS i_muy_lento;

RULE 12: IF dist IS i_cerca AND movi IS i_lento THEN velo IS i_lento;

RULE 13: IF dist IS i_cerca AND movi IS cero THEN velo IS i_lento;

RULE 14: IF dist IS i_cerca AND movi IS d_lento THEN velo IS i_medio;

RULE 15: IF dist IS i_cerca AND movi IS d_rapido THEN velo IS i_rapido;

RULE 16: IF dist IS centro AND movi IS i_rapido THEN velo IS i_muy_lento;

RULE 17: IF dist IS centro AND movi IS i_lento THEN velo IS cero;

RULE 18: IF dist IS centro AND movi IS cero THEN velo IS cero;

RULE 19: IF dist IS centro AND movi IS d_lento THEN velo IS cero;

RULE 20: IF dist IS centro AND movi IS d_rapido THEN velo IS d_muy_lento;

RULE 21: IF dist IS d_cerca AND movi IS i_rapido THEN velo IS d_rapido;

RULE 22: IF dist IS d_cerca AND movi IS i_lento THEN velo IS d_medio;

RULE 23: IF dist IS d_cerca AND movi IS cero THEN velo IS d_lento;

RULE 24: IF dist IS d_cerca AND movi IS d_lento THEN velo IS d_lento;

RULE 25: IF dist IS d_cerca AND movi IS d_rapido THEN velo IS d_muy_lento;

RULE 26: IF dist IS d_medio AND movi IS i_rapido THEN velo IS d_muy_rapido;

RULE 27: IF dist IS d_medio AND movi IS i_lento THEN velo IS d_muy_rapido;

RULE 28: IF dist IS d_medio AND movi IS cero THEN velo IS d_rapido;

RULE 29: IF dist IS d_medio AND movi IS d_lento THEN velo IS d_medio;

RULE 30: IF dist IS d_medio AND movi IS d_rapido THEN velo IS d_medio;

RULE 31: IF dist IS d_lejos AND movi IS i_rapido THEN velo IS d_muy_rapido;

RULE 32: IF dist IS d_lejos AND movi IS i_lento THEN velo IS d_muy_rapido;

RULE 33: IF dist IS d_lejos AND movi IS cero THEN velo IS d_muy_rapido;

RULE 34: IF dist IS d_lejos AND movi IS d_lento THEN velo IS d_muy_rapido;

RULE 35: IF dist IS d_lejos AND movi IS d_rapido THEN velo IS d_rapido;

END_RULEBLOCK

END_FUNCTION_BLOCK

ANEXO D: Ficha técnica NOVA NV12X

■ Specification

Model	NV12X
Pick up Element	1/3" SUPER HAD II Color Sony CCD
Effective Picture Elements (HxV)	NTSC: 768(H) x 494(V) PAL: 752(H) x 582(V)
Horizontal Resolution	700TV Line
Minimum Illumination	0.001 Lux
S/N Ratio	50dB
Scanning System	2:1 Interface
Synchronous System	Internal Negative sync.
Auto Electronic Shutter	NTSC: 1/60s~1/100,000s, PAL: 1/50s~1/100,000s
Gama Characteristic	0.45
Video Output	1Vpp, 75Ω
Auto Gain Control	Auto
Power/Current	DC12V(+/-10%), 1A
Lens	5-60 mm Dynamic Varifocal Lens
Dimension (mm)	φ121 x 177(H)
Weight (g)	2800
Storage Temperature	-30~+60°C RH95% MAX
Operating Temperature	-10~+50°C RH95% MAX

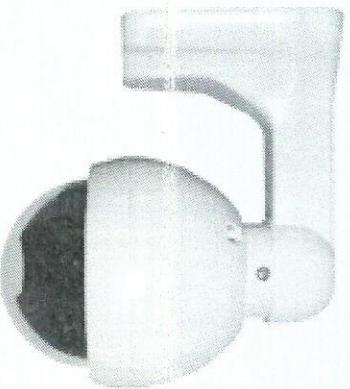
■ Packing List

Franchiser:

Notes: Before providing power for the camera, please read this User Guide in detail!

Do not attempt to disassemble the camera. If the camera can not work, please contact local franchiser or our company.

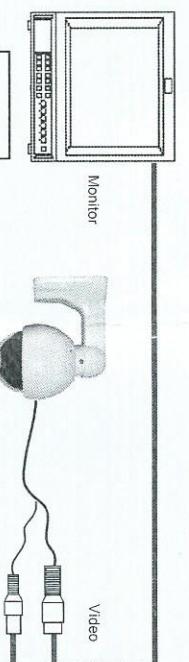
No	Name	Model	Number
1.	Weatherproof & Vandalproof Dome Color Camera	NV12X	1
2.	User Guide	NV12X	1



NV12X Color Camera

Weatherproof & Vandalproof Dome Color Camera

- * **Sensor (CCD)**
Adopt SONY 1/4" CCD Sensor.
- * **Auto Gain Control (AGC)**
Built-in auto gain control (AGC) circuit. The color camera can get high definition picture in low Lux condition.
- * **Scanning Mode**
NTSC or PAL mode.
- * **Water Resistance**
IP66.



TROUBLE AND SOLUTION

1. No Picture after providing power

- . May be the power supply voltage abnormality, please check the power supply voltage and pole whether exactitude.
- . Please check all the connecting cable and monitors whether be connected correctly or not.

2. The picture level direction have flowing interference ripples

- . May be caused by the power supply AC ripples, it need filter the wave of the power supply.
- . Check the monitor and peripheral equipments used.

3. The picture background color changes continuously

- . The fluorescent lamp's electromagnetic field cause color roll. This is proper phenomenon of the cameras.
- . Reduce the fluorescent lamp numbers or increase the distance between the camera and the fluorescent lamps can improve it.
- . Use power supply external sync. camera can solve it.

4. The picture smear too mass

- . The power supply's voltage unstable.
- . In order to capture high quality pictures, the power supply's cable and video output cable should not be too long.



Notes:

- The power supply must through safe attestation. Its output voltage, current, voltage polarity and operating temperature must match the camera's requirement.
- When using the camera in the thunderbolt condition, please note to mount Anti-thunder device or put off the power supply plug and cable.
- In order to capture high quality pictures, the power supply's cable and video output cable should not be too long.

The symbol is intended to alert the user to the presence of uninsulated 'dangerous voltage' within the product's enclosure that may be of sufficient magnitude to constitute a risk of electric shock to persons.

The symbol is intended to alert the user to the presence of uninsulated 'dangerous voltage' within the product's enclosure that may be of sufficient magnitude to constitute a risk of electric shock to persons.



DO NOT REMOVE COVER OR BACK NO USER SERVICEABLE PARTS INSIDE. REFER SERVICING

CAUTION: TO PREVENT ELECTRIC SHOCKS AND RISK OF FIRE HAZARDS, DO NOT USE OTHER THAN SPECIFIED POWER SOURCE.



Note:

- Please note the camera's operating temperature and its using environment requirement. Avoid using the camera at too high or too low temperatures. The operating temperature is -30~+60°C. (Recommenatory operating temperature is 10~+50°C.)
- Never make the camera face the sun or bright object. Otherwise, it will damage the CCD.
- Do not mount the camera near by the radiator or heater.

ANEXO E: Ficha técnica TOSHIBA NB200

TOSHIBA MINI NB200

The 10.1" mini notebook designed with your fingers in mind



Touch me, feel me, see me, hold me! Created to deliver a whole new sensory experience, the Toshiba mini NB200 comes with a unique feel-good factor built-in. It's fully kitted-out to take your digital lifestyle anywhere, in compelling textured covers and cool colours that appeal to the tactile and visual senses. Share music, photos or movies with friends in a café. Write emails, post a blog or browse the web on the train. Chat in real time via the built-in webcam and mic. Work on docs, play casual online games, communicate wherever life takes you. This stylish 24/7 companion combines the hottest design with the latest mini notebook technology: powerful processor, 10.1" Toshiba TruBrite® display, capacious HDD, latest wireless technology and lots more. The Toshiba mini NB200 – style you can feel, performance you can rely on.

► MASSIVE STAYING POWER

Enjoy the freedom of up to 3½ hours' mobile computing with the standard 3-cell battery,

► KEEPS GOING ON THE MOVE

Charge your phone, PDA, MP3 player, gaming console or other device even when the mini NB200 is turned off.

► PERFECT PORTABLE PROTECTION

High-capacity HDD of up to 160 GB* includes 3D accelerometer monitoring to shield precious data from bumps and knocks.

Publicación autorizada con fines académicos e investigativos

En su investigación no olvide referenciar esta tesis

* depending on models and local availability



Full-sized tile keyboard



Compelling 3D textured tactile design



Also available with glossy finish in Cosmic Black*

Toshiba recommends Windows® for everyday computing

REPOSITORIO DE TOSHIBA MINI NB200 TESIS UCM



UNIVERSIDAD
CATÓLICA
DE SANTA MARÍA

Model	NB200-10F	NB200-10J
Processor/technology	Intel® Atom™ Processor N280 (1.66 GHz, 512 KB L2 Cache, 667 MHz FSB)	Intel® Atom™ Processor N270 (1.60 GHz, 512 KB L2 Cache, 533 MHz FSB)
Operating System	Genuine Microsoft® Windows® XP Home Edition	
Colour	Satin Brown	Cosmic Black
Display	10.1" Toshiba TruBrite® WSVGA LED Backlight display, resolution: 1,024 x 600	
Hard disk	160 GB (5,400 rpm) Serial ATA	120 GB (5,400 rpm) Serial ATA
System memory	1,024 MB, technology DDR2 RAM 800 MHz (running at 533 MHz)	
Graphics adapter	Intel GMA 950, up to 224 MB total available graphics memory with 512 MB system memory	
Pointing device	Touch Pad	
Wired communication	Fast Ethernet LAN (10/100 Mbps)	
Wireless communication	Wireless LAN (802.11b/g), Bluetooth® 2.1 with EDR (Enhanced Data Rate)	
Sound system	Built-in mono speakers, built-in microphone, external microphone port	
Interfaces	DC-in, external monitor, 3 USB 2.0 including 1x Sleep-and-Charge, 2-in-1 Bridge Media slot (supports SD™ & mini SD™, MultiMedia Card™), external microphone	
Expansions	1 memory slot accessible	
Battery	Lithium-ion technology, battery life: up to 9 hours (Mobile Mark™ 2007)	Lithium-ion technology, battery life: up to 3.5 hours (Mobile Mark™ 2007)
Bundled software	Toshiba ConfigFree™ incl. Bluetooth®/Wireless LAN Radar, Toshiba Bluetooth® Stack, Toshiba Bluetooth® Monitor, Toshiba Utilities and Driver, Toshiba password utilities, 60-day Microsoft Works & Office Home Student Trial Version, McAfee® Internet Security Suite – Toshiba Edition (includes free Internet updates for 30 days), Toshiba User's Manual, Toshiba Recovery Disc Creator, Online User's Manual, Adobe® PDF Reader, Google Toolbar, Google Picasa, iGoogle, Wildtangent ORB Game Console, MyPhotobook, Microsoft SyncToy (PC Synchronisation Software)	
Bundled hardware	AC adapter	
Special features	3D Accelerometer for HDD Protection, Kensington Lock, Password utilities, integrated VGA Web Camera for Video over IP, integrated microphone for Voice over IP, ENERGY STAR 4.0 qualified computer, Computrace enabled BIOS, Microsoft SyncToy (PC Synchronisation Software)	
Physical dimensions	W x L x H: 263 x 211.5 x 25.4/32.25 mm, weight: 1.33 kg	W x L x H: 263x192.3x25.4/32.25 mm, weight: 1.18 kg
Warranty	One year international warranty. Upgrade your standard warranty with Toshiba warranty extension and uplift packs. Contact your nearest Toshiba Authorised Reseller for details.	

> TOSHIBA EASYMEDIA

Easy connectivity	Toshiba ConfigFree™, integrated Bridge Media slot, Diversity Antenna, built-in Web Camera & Microphone, Bluetooth® 2.1 with EDR
Easy usability	USB Sleep-and-Charge, McAfee® Internet Security Suite

> COMPATIBLE TOSHIBA ACCESSORIES

- Compact Optical Mouse (silver/black) – PA3678E-1ETS
- External Super MultiDrive, 8x DVD+/-R multi drive, USB2.0 I/F, Bus power – PA3761E-1DV2
- Leather Case (up to 12.1" notebooks) – PX1448E-1NCA
- Universal AC Adaptor 30 W/19 V 3 pin AC Adaptor (RoHS) – PA3743C-1AC3
- Battery Pack 6-cell/10.8 V/5800mAh (Min:5400mAh)/63 Wh (black) – PA3733U-1BRS
- Battery Pack 6-cell/10.8 V/5800mAh (Min:5400mAh)/63 Wh (silver) – PA3734U-1BRS
- USB Flash Drive 8 GB, High-Speed USB 2.0 – PA1521E-1M8G



Compact Optical Mouse
(silver/black)



External Super MultiDrive



Leather Case
(up to 12.1" notebooks)

Dealer stamp:



CPU performance may vary from specifications under certain conditions such as the use of battery instead of AC power, certain external peripherals, certain multimedia applications or network connections, complex modeling software and in areas with low air pressure at high altitude (1,000 metres above sea level) and/or certain temperatures. CPU performance may also vary from specifications due to design configuration. Under some conditions, your computer may automatically shut down as a normal protective measure. To avoid risk of lost data, please make periodic back-up copies. For optimum performance, use your computer only under recommended conditions. Please read detailed restrictions in the product resource guide, visit the Toshiba web site: www.computers.toshiba-europe.com and/or contact Toshiba Technical Support. • Graphics processor unit (GPU) performance may vary depending on product model, design configuration, applications, power management settings and features utilized. • Part of the main system memory may be used by the graphics system and therefore reduce the amount of main system memory available. • For PCs configured with 4 GB of system memory, the full system memory space for computing activities will be considerably less and will vary by model and system configuration. • For information about upgrading your system memory please refer to the user manual or contact your local Toshiba service provider. • One GB means one billion bytes, accessible capacity may be less. • Wired and wireless communication capabilities to be applied in countries where approved. • Battery life may vary depending on applications, power management settings and features utilized. Recharge time varies depending on usage. Battery may not charge while computer is consuming full power. After a period of time, the battery will lose its ability to perform at maximum capacity and will need to be replaced. This is normal for all batteries. To purchase a new battery pack, see your accessories information that shipped with your computer or visit the Toshiba web site at www.computers.toshiba-europe.com. • Small bright dots may appear on your TFT display when you turn on your PC. Your display contains an extremely large number of thin-film transistors (TFT) and is manufactured using high-precision technology. Any small bright dots that may appear on your display are an intrinsic characteristic of the TFT manufacturing technology. • Weight may vary depending on product configuration, vendor components, manufacturing variability and options selected. • Please check your system specifications to ensure optimal results. • Certain notebook chassis are designed to accommodate all possible configurations for an entire product series. Your selected model may not have all the features and specifications corresponding to all of the icons / switches shown on the notebook chassis. • Performance may vary depending on product model, configuration, video content/format/settings as well as the performance variations of individual hardware components. Results were achieved on select models and configurations tested by Toshiba at the time of publication.

TISB April 2009, Art. No. EN_MNINB200_Datas_Apr09. Microsoft, Windows and Vista are either registered trademarks or trademarks of Microsoft Corporation in the United States and/or other countries. All trademarks are acknowledged. Product specifications, configurations, prices and system component/options availability are all subject to change without notice. Product design specifications and colours are subject to change without notice and may vary from those shown. Errors and omissions excepted.

ANEXO F: Ficha técnica EasyCap

USB Video Capture use manual

Package Contents

- New Pro series USB2.0 Video Grabber
- USB Cable
- CD-ROM/included driver and the professional video editor software

Key Features

- Include Professional and easy to learn & used video editor Software: honestech HD DVR 2.5
- Popular USB 2.0 interface and not need external power
- Capture Video & Audio though USB 2.0 interface
- Control the Brightness, Contrast, Hue, and Saturation of color
- Portable and easy to store.
- Can capture audio without the need of a sound card
- Plug & play
- Support most of the formats: record in DVD+/-R/RW, DV+/-VR, and DVD-Video.
- Can be used for internet conference / net meeting grabber.

Specifications

- USB 2.0

System Requirements

- USB: Compliant USB 2.0 free port
- OS: Windows 2000/XP, Vista, WIN7
- CPU: Intel Pentium 4 or higher
- HD: 1GB of available hard drive space for program installation, 4GB+ hard drive space for video capture and editing
- Memory: 256MB of RAM (512MB or above for editing)
- Display: Windows-compatible display with at least 1024X768
- Sound card: compatible Windows-sound card

Hardware Installation

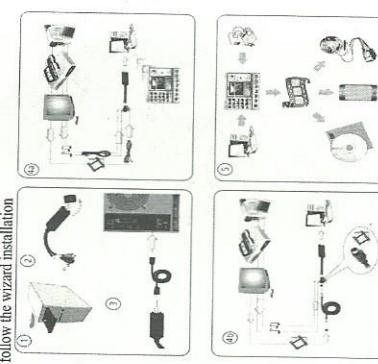
Please follow instructions below for the set up of your video grabber.

Software Installation

Please follow instructions below for the set up of your video grabber.

Please insert the "Software CD-ROM" into your PC

follow the wizard installation



ANEXO G: Ficha técnica MAX- 485



Low-Power, Slew-Rate-Limited RS-485/RS-422 Transceivers

General Description

The MAX481, MAX483, MAX485, MAX487–MAX491, and MAX1487 are low-power transceivers for RS-485 and RS-422 communication. Each part contains one driver and one receiver. The MAX483, MAX487, MAX488, and MAX489 feature reduced slew-rate drivers that minimize EMI and reduce reflections caused by improperly terminated cables, thus allowing error-free data transmission up to 250kbps. The driver slew rates of the MAX481, MAX485, MAX490, MAX491, and MAX1487 are not limited, allowing them to transmit up to 2.5Mbps.

These transceivers draw between 120 μ A and 500 μ A of supply current when unloaded or fully loaded with disabled drivers. Additionally, the MAX481, MAX483, and MAX487 have a low-current shutdown mode in which they consume only 0.1 μ A. All parts operate from a single 5V supply.

Drivers are short-circuit current limited and are protected against excessive power dissipation by thermal shutdown circuitry that places the driver outputs into a high-impedance state. The receiver input has a fail-safe feature that guarantees a logic-high output if the input is open circuit.

The MAX487 and MAX1487 feature quarter-unit-load receiver input impedance, allowing up to 128 MAX487/MAX1487 transceivers on the bus. Full-duplex communications are obtained using the MAX488–MAX491, while the MAX481, MAX483, MAX485, MAX487, and MAX1487 are designed for half-duplex applications.

Applications

- Low-Power RS-485 Transceivers
- Low-Power RS-422 Transceivers
- Level Translators
- Transceivers for EMI-Sensitive Applications
- Industrial-Control Local Area Networks

Next Generation Device Features

♦ For Fault-Tolerant Applications

MAX3430: \pm 80V Fault-Protected, Fail-Safe, 1/4 Unit Load, +3.3V, RS-485 Transceiver
MAX3440E–MAX3444E: \pm 15kV ESD-Protected, \pm 60V Fault-Protected, 10Mbps, Fail-Safe, RS-485/J1708 Transceivers

♦ For Space-Constrained Applications

MAX3460–MAX3464: +5V, Fail-Safe, 20Mbps, Profibus RS-485/RS-422 Transceivers
MAX3362: +3.3V, High-Speed, RS-485/RS-422 Transceiver in a SOT23 Package
MAX3280E–MAX3284E: \pm 15kV ESD-Protected, 52Mbps, +3V to +5.5V, SOT23, RS-485/RS-422, True Fail-Safe Receivers
MAX3293/MAX3294/MAX3295: 20Mbps, +3.3V, SOT23, RS-855/RS-422 Transmitters

♦ For Multiple Transceiver Applications

MAX3030E–MAX3033E: \pm 15kV ESD-Protected, +3.3V, Quad RS-422 Transmitters

♦ For Fail-Safe Applications

MAX3080–MAX3089: Fail-Safe, High-Speed (10Mbps), Slew-Rate-Limited RS-485/RS-422 Transceivers

♦ For Low-Voltage Applications

MAX3483E/MAX3485E/MAX3486E/MAX3488E/MAX3490E/MAX3491E: +3.3V Powered, \pm 15kV ESD-Protected, 12Mbps, Slew-Rate-Limited, True RS-485/RS-422 Transceivers

Ordering Information appears at end of data sheet.

Selection Table

PART NUMBER	HALF/FULL DUPLEX	DATA RATE (Mbps)	SLEW-RATE LIMITED	LOW-POWER SHUTDOWN	RECEIVER/DRIVER ENABLE	QUIESCENT CURRENT (μ A)	NUMBER OF TRANSMITTERS ON BUS	PIN COUNT
MAX481	Half	2.5	No	Yes	Yes	300	32	8
MAX483	Half	0.25	Yes	Yes	Yes	120	32	8
MAX485	Half	2.5	No	No	Yes	300	32	8
MAX487	Half	0.25	Yes	Yes	Yes	120	128	8
MAX488	Full	0.25	Yes	No	No	120	32	8
MAX489	Full	0.25	Yes	No	Yes	120	32	14
MAX490	Full	2.5	No	No	No	300	32	8
MAX491	Full	2.5	No	No	Yes	300	32	14
MAX1487	Half	2.5	No	No	Yes	230	128	8

MAX481/MAX483/MAX485/MAX487-MAX491/MAX1487

Low-Power, Slew-Rate-Limited RS-485/RS-422 Transceivers

ABSOLUTE MAXIMUM RATINGS

Supply Voltage (VCC).....	12V
Control Input Voltage (RE, DE).....	-0.5V to (VCC + 0.5V)
Driver Input Voltage (DI).....	-0.5V to (VCC + 0.5V)
Driver Output Voltage (A, B).....	-8V to +12.5V
Receiver Input Voltage (A, B).....	-8V to +12.5V
Receiver Output Voltage (RO).....	-0.5V to (VCC + 0.5V)
Continuous Power Dissipation (TA = +70°C)	
8-Pin Plastic DIP (derate 9.09mW/°C above +70°C)	727mW
14-Pin Plastic DIP (derate 10.00mW/°C above +70°C)	800mW
8-Pin SO (derate 5.88mW/°C above +70°C).....	471mW

14-Pin SO (derate 8.33mW/°C above +70°C).....	667mW
8-Pin µMAX (derate 4.1mW/°C above +70°C)	830mW
8-Pin CERDIP (derate 8.00mW/°C above +70°C).....	640mW
14-Pin CERDIP (derate 9.09mW/°C above +70°C).....	727mW

Operating Temperature Ranges

MAX4_ _C_ _MAX1487C_ A	0°C to +70°C
MAX4_ _E_ _MAX1487E_ A	-40°C to +85°C
MAX4_ _MJ_ /MAX1487MJA	-55°C to +125°C

Storage Temperature Range	-65°C to +160°C
Lead Temperature (soldering, 10sec)	+300°C

Stresses beyond those listed under "Absolute Maximum Ratings" may cause permanent damage to the device. These are stress ratings only, and functional operation of the device at these or any other conditions beyond those indicated in the operational sections of the specifications is not implied. Exposure to absolute maximum rating conditions for extended periods may affect device reliability.

DC ELECTRICAL CHARACTERISTICS

(VCC = 5V ±5%, TA = TMIN to TMAX, unless otherwise noted.) (Notes 1, 2)

PARAMETER	SYMBOL	CONDITIONS	MIN	TYP	MAX	UNITS
Differential Driver Output (no load)	VOD1				5	V
Differential Driver Output (with load)	VOD2	R = 50Ω (RS-422)	2			V
		R = 27Ω (RS-485), Figure 4	1.5	5		
Change in Magnitude of Driver Differential Output Voltage for Complementary Output States	ΔVOD	R = 27Ω or 50Ω, Figure 4			0.2	V
Driver Common-Mode Output Voltage	VOC	R = 27Ω or 50Ω, Figure 4			3	V
Change in Magnitude of Driver Common-Mode Output Voltage for Complementary Output States	ΔVOD	R = 27Ω or 50Ω, Figure 4			0.2	V
Input High Voltage	VIH	DE, DI, RE	2.0			V
Input Low Voltage	VIL	DE, DI, RE			0.8	V
Input Current	IIN1	DE, DI, RE			±2	µA
Input Current (A, B)	IIN2	DE = 0V; VCC = 0V or 5.25V, all devices except MAX487/MAX1487	VIN = 12V		1.0	mA
			VIN = -7V		-0.8	
		MAX487/MAX1487, DE = 0V, VCC = 0V or 5.25V	VIN = 12V		0.25	mA
			VIN = -7V		-0.2	
Receiver Differential Threshold Voltage	VTH	-7V ≤ VCM ≤ 12V	-0.2	0.2		V
Receiver Input Hysteresis	ΔVTH	VCM = 0V	70			mV
Receiver Output High Voltage	VOH	IO = -4mA, VID = 200mV	3.5			V
Receiver Output Low Voltage	VOL	IO = 4mA, VID = -200mV			0.4	V
Three-State (high impedance) Output Current at Receiver	IOZR	0.4V ≤ VO ≤ 2.4V			±1	µA
Receiver Input Resistance	RIN	-7V ≤ VCM ≤ 12V, all devices except MAX487/MAX1487	12			kΩ
		-7V ≤ VCM ≤ 12V, MAX487/MAX1487	48			kΩ

Low-Power, Slew-Rate-Limited RS-485/RS-422 Transceivers

DC ELECTRICAL CHARACTERISTICS (continued)

($V_{CC} = 5V \pm 5\%$, $T_A = T_{MIN}$ to T_{MAX} , unless otherwise noted.) (Notes 1, 2)

PARAMETER	SYMBOL	CONDITIONS	MIN	TYP	MAX	UNITS
No-Load Supply Current (Note 3)	I _{CC}	MAX488/MAX489, DE, DI, RE = 0V or V _{CC}		120	250	μA
		MAX490/MAX491, DE, DI, RE = 0V or V _{CC}		300	500	
		MAX481/MAX485, RE = 0V or V _{CC}	DE = V _{CC}	500	900	
			DE = 0V	300	500	
		MAX1487, RE = 0V or V _{CC}	DE = V _{CC}	300	500	
			DE = 0V	230	400	
		MAX483/MAX487, RE = 0V or V _{CC}	DE = 5V	MAX483	350	650
				MAX487	250	400
			DE = 0V		120	250
Supply Current in Shutdown	I _{SHDN}	MAX481/483/487, DE = 0V, RE = V _{CC}		0.1	10	μA
Driver Short-Circuit Current, V _O = High	I _{OSD1}	-7V ≤ V _O ≤ 12V (Note 4)		35	250	mA
Driver Short-Circuit Current, V _O = Low	I _{OSD2}	-7V ≤ V _O ≤ 12V (Note 4)		35	250	mA
Receiver Short-Circuit Current	I _{OSR}	0V ≤ V _O ≤ V _{CC}		7	95	mA

SWITCHING CHARACTERISTICS—MAX481/MAX485, MAX490/MAX491, MAX1487

($V_{CC} = 5V \pm 5\%$, $T_A = T_{MIN}$ to T_{MAX} , unless otherwise noted.) (Notes 1, 2)

PARAMETER	SYMBOL	CONDITIONS	MIN	TYP	MAX	UNITS
Driver Input to Output	t _{PLH}	Figures 6 and 8, RD _{IFF} = 54Ω, C _{L1} = C _{L2} = 100pF	10	30	60	ns
	t _{PHL}		10	30	60	
Driver Output Skew to Output	t _{SKEW}	Figures 6 and 8, RD _{IFF} = 54Ω, C _{L1} = C _{L2} = 100pF		5	10	ns
Driver Rise or Fall Time	t _R , t _F	Figures 6 and 8, RD _{IFF} = 54Ω, C _{L1} = C _{L2} = 100pF	MAX481, MAX485, MAX1487	3	15	40
			MAX490C/E, MAX491C/E	5	15	25
			MAX490M, MAX491M	3	15	40
Driver Enable to Output High	t _{ZH}	Figures 7 and 9, C _L = 100pF, S ₂ closed		40	70	ns
Driver Enable to Output Low	t _{ZL}	Figures 7 and 9, C _L = 100pF, S ₁ closed		40	70	ns
Driver Disable Time from Low	t _{LZ}	Figures 7 and 9, C _L = 15pF, S ₁ closed		40	70	ns
Driver Disable Time from High	t _{HZ}	Figures 7 and 9, C _L = 15pF, S ₂ closed		40	70	ns
Receiver Input to Output	t _{PLH} , t _{PHL}	Figures 6 and 10, RD _{IFF} = 54Ω, C _{L1} = C _{L2} = 100pF	MAX481, MAX485, MAX1487	20	90	200
			MAX490C/E, MAX491C/E	20	90	150
			MAX490M, MAX491M	20	90	200
t _{PLH} - t _{PHL} Differential Receiver Skew	t _{SKD}	Figures 6 and 10, RD _{IFF} = 54Ω, C _{L1} = C _{L2} = 100pF		13		ns
Receiver Enable to Output Low	t _{ZL}	Figures 5 and 11, C _{RL} = 15pF, S ₁ closed		20	50	ns
Receiver Enable to Output High	t _{ZH}	Figures 5 and 11, C _{RL} = 15pF, S ₂ closed		20	50	ns
Receiver Disable Time from Low	t _{LZ}	Figures 5 and 11, C _{RL} = 15pF, S ₁ closed		20	50	ns
Receiver Disable Time from High	t _{HZ}	Figures 5 and 11, C _{RL} = 15pF, S ₂ closed		20	50	ns
Maximum Data Rate	f _{MAX}			2.5		Mbps
Time to Shutdown	t _{SHDN}	MAX481 (Note 5)	50	200	600	ns

Low-Power, Slew-Rate-Limited RS-485/RS-422 Transceivers

SWITCHING CHARACTERISTICS—MAX481/MAX485, MAX490/MAX491, MAX1487 (continued)

(VCC = 5V ±5%, TA = TMIN to TMAX, unless otherwise noted.) (Notes 1, 2)

PARAMETER	SYMBOL	CONDITIONS	MIN	TYP	MAX	UNITS
Driver Enable from Shutdown to Output High (MAX481)	tZH(SHDN)	Figures 7 and 9, CL = 100pF, S2 closed		40	100	ns
Driver Enable from Shutdown to Output Low (MAX481)	tZL(SHDN)	Figures 7 and 9, CL = 100pF, S1 closed		40	100	ns
Receiver Enable from Shutdown to Output High (MAX481)	tZH(SHDN)	Figures 5 and 11, CL = 15pF, S2 closed, A - B = 2V		300	1000	ns
Receiver Enable from Shutdown to Output Low (MAX481)	tZL(SHDN)	Figures 5 and 11, CL = 15pF, S1 closed, B - A = 2V		300	1000	ns

SWITCHING CHARACTERISTICS—MAX483, MAX487/MAX488/MAX489

(VCC = 5V ±5%, TA = TMIN to TMAX, unless otherwise noted.) (Notes 1, 2)

PARAMETER	SYMBOL	CONDITIONS	MIN	TYP	MAX	UNITS
Driver Input to Output	tPLH	Figures 6 and 8, RDIFF = 54Ω, CL1 = CL2 = 100pF	250	800	2000	ns
	tPHL		250	800	2000	
Driver Output Skew to Output	tSKEW	Figures 6 and 8, RDIFF = 54Ω, CL1 = CL2 = 100pF		100	800	ns
Driver Rise or Fall Time	tR, tF	Figures 6 and 8, RDIFF = 54Ω, CL1 = CL2 = 100pF	250		2000	ns
Driver Enable to Output High	tZH	Figures 7 and 9, CL = 100pF, S2 closed	250		2000	ns
Driver Enable to Output Low	tZL	Figures 7 and 9, CL = 100pF, S1 closed	250		2000	ns
Driver Disable Time from Low	tLZ	Figures 7 and 9, CL = 15pF, S1 closed	300		3000	ns
Driver Disable Time from High	tHZ	Figures 7 and 9, CL = 15pF, S2 closed	300		3000	ns
Receiver Input to Output	tPLH	Figures 6 and 10, RDIFF = 54Ω, CL1 = CL2 = 100pF	250		2000	ns
	tPHL		250		2000	
tPLH - tPHL Differential Receiver Skew	tSKD	Figures 6 and 10, RDIFF = 54Ω, CL1 = CL2 = 100pF		100		ns
Receiver Enable to Output Low	tZL	Figures 5 and 11, CRL = 15pF, S1 closed		20	50	ns
Receiver Enable to Output High	tZH	Figures 5 and 11, CRL = 15pF, S2 closed		20	50	ns
Receiver Disable Time from Low	tLZ	Figures 5 and 11, CRL = 15pF, S1 closed		20	50	ns
Receiver Disable Time from High	tHZ	Figures 5 and 11, CRL = 15pF, S2 closed		20	50	ns
Maximum Data Rate	fMAX	tPLH, tPHL < 50% of data period	250			kbps
Time to Shutdown	tSHDN	MAX483/MAX487 (Note 5)	50	200	600	ns
Driver Enable from Shutdown to Output High	tZH(SHDN)	MAX483/MAX487, Figures 7 and 9, CL = 100pF, S2 closed			2000	ns
Driver Enable from Shutdown to Output Low	tZL(SHDN)	MAX483/MAX487, Figures 7 and 9, CL = 100pF, S1 closed			2000	ns
Receiver Enable from Shutdown to Output High	tZH(SHDN)	MAX483/MAX487, Figures 5 and 11, CL = 15pF, S2 closed			2500	ns
Receiver Enable from Shutdown to Output Low	tZL(SHDN)	MAX483/MAX487, Figures 5 and 11, CL = 15pF, S1 closed			2500	ns

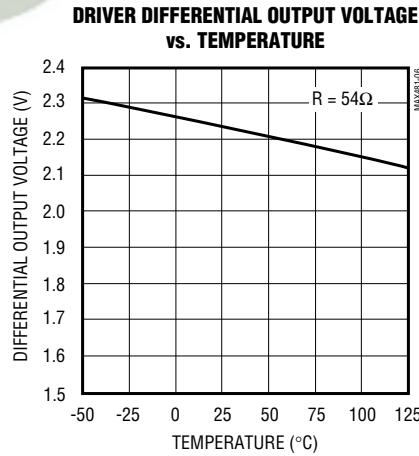
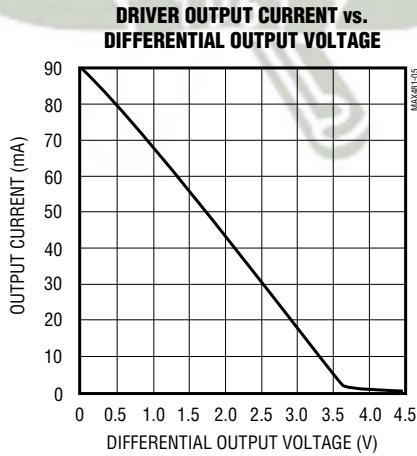
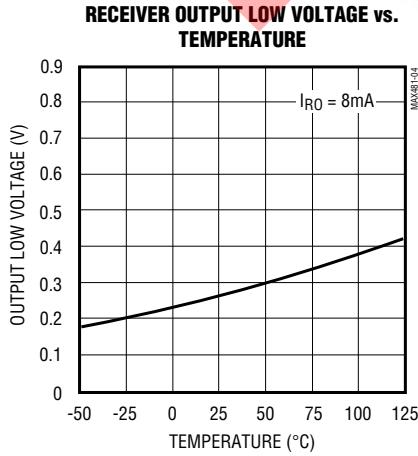
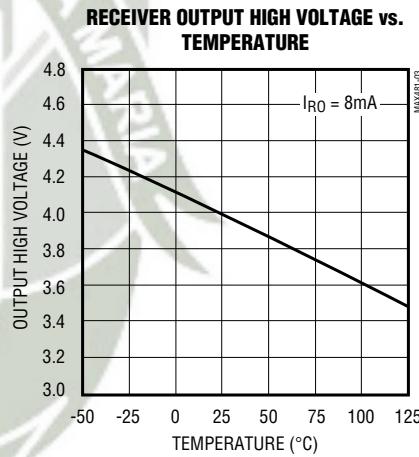
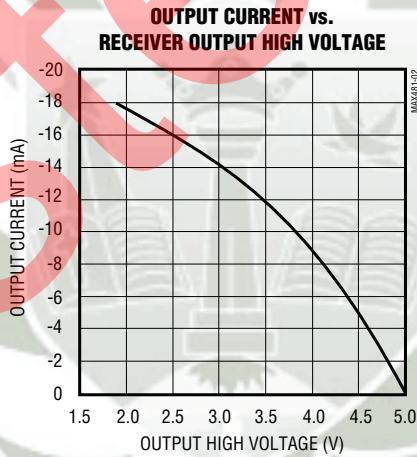
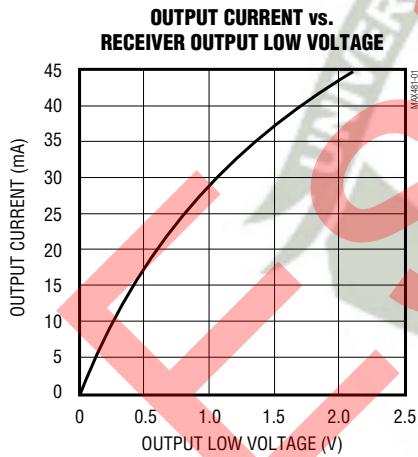
Low-Power, Slew-Rate-Limited RS-485/RS-422 Transceivers

NOTES FOR ELECTRICAL/SWITCHING CHARACTERISTICS

- Note 1:** All currents into device pins are positive; all currents out of device pins are negative. All voltages are referenced to device ground unless otherwise specified.
- Note 2:** All typical specifications are given for $V_{CC} = 5V$ and $T_A = +25^\circ C$.
- Note 3:** Supply current specification is valid for loaded transmitters when $DE = 0V$.
- Note 4:** Applies to peak current. See *Typical Operating Characteristics*.
- Note 5:** The MAX481/MAX483/MAX487 are put into shutdown by bringing \overline{RE} high and DE low. If the inputs are in this state for less than 50ns, the parts are guaranteed not to enter shutdown. If the inputs are in this state for at least 600ns, the parts are guaranteed to have entered shutdown. See *Low-Power Shutdown Mode* section.

Typical Operating Characteristics

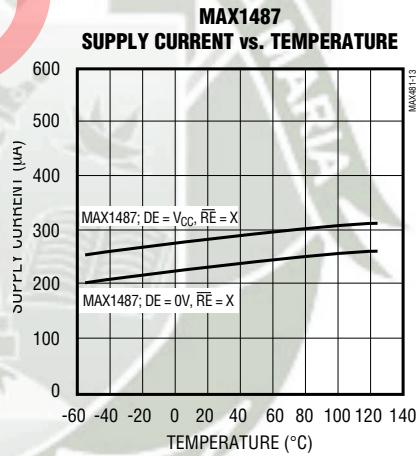
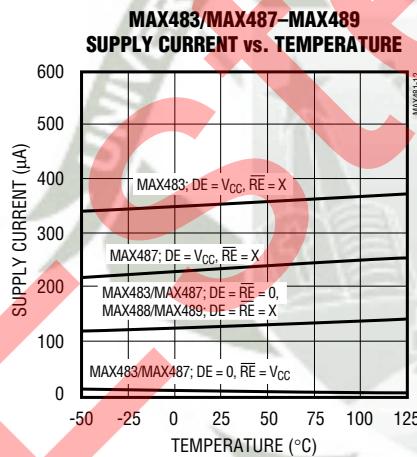
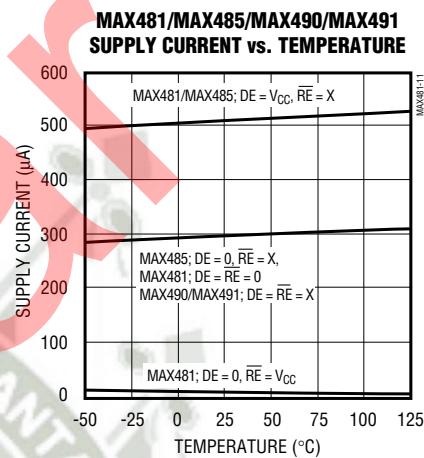
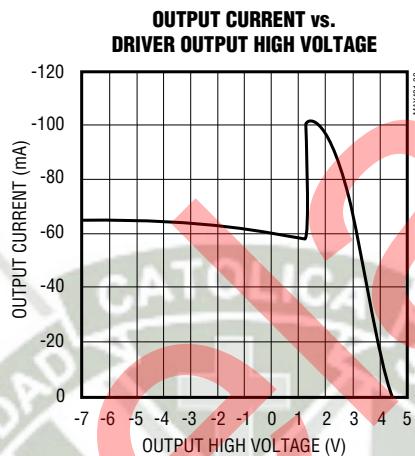
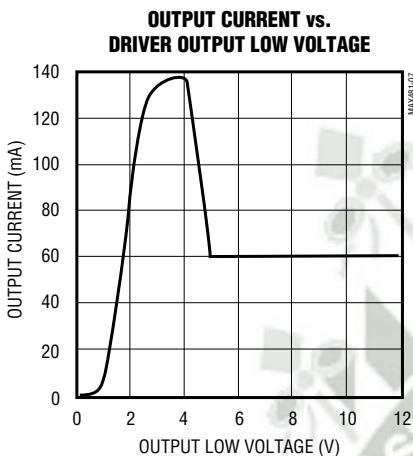
($V_{CC} = 5V$, $T_A = +25^\circ C$, unless otherwise noted.)



Low-Power, Slew-Rate-Limited RS-485/RS-422 Transceivers

Typical Operating Characteristics (continued)

($V_{CC} = 5V$, $T_A = +25^\circ C$, unless otherwise noted.)



Low-Power, Slew-Rate-Limited RS-485/RS-422 Transceivers

Pin Description

PIN					NAME	FUNCTION
MAX481/MAX483/ MAX485/MAX487/ MAX1487		MAX488/ MAX490		MAX489/ MAX491		
DIP/SO	μ MAX	DIP/SO	μ MAX	DIP/SO		
1	3	2	4	2	RO	Receiver Output: If A > B by 200mV, RO will be high; If A < B by 200mV, RO will be low.
2	4	—	—	3	\bar{RE}	Receiver Output Enable. RO is enabled when \bar{RE} is low; RO is high impedance when \bar{RE} is high.
3	5	—	—	4	DE	Driver Output Enable. The driver outputs, Y and Z, are enabled by bringing DE high. They are high impedance when DE is low. If the driver outputs are enabled, the parts function as line drivers. While they are high impedance, they function as line receivers if \bar{RE} is low.
4	6	3	5	5	DI	Driver Input. A low on DI forces output Y low and output Z high. Similarly, a high on DI forces output Y high and output Z low.
5	7	4	6	6, 7	GND	Ground
—	—	5	7	9	Y	Noninverting Driver Output
—	—	6	8	10	Z	Inverting Driver Output
6	8	—	—	—	A	Noninverting Receiver Input and Noninverting Driver Output
—	—	8	2	12	A	Noninverting Receiver Input
7	1	—	—	—	B	Inverting Receiver Input and Inverting Driver Output
—	—	7	1	11	B	Inverting Receiver Input
8	2	1	3	14	VCC	Positive Supply: $4.75V \leq VCC \leq 5.25V$
—	—	—	—	1, 8, 13	N.C.	No Connect—not internally connected

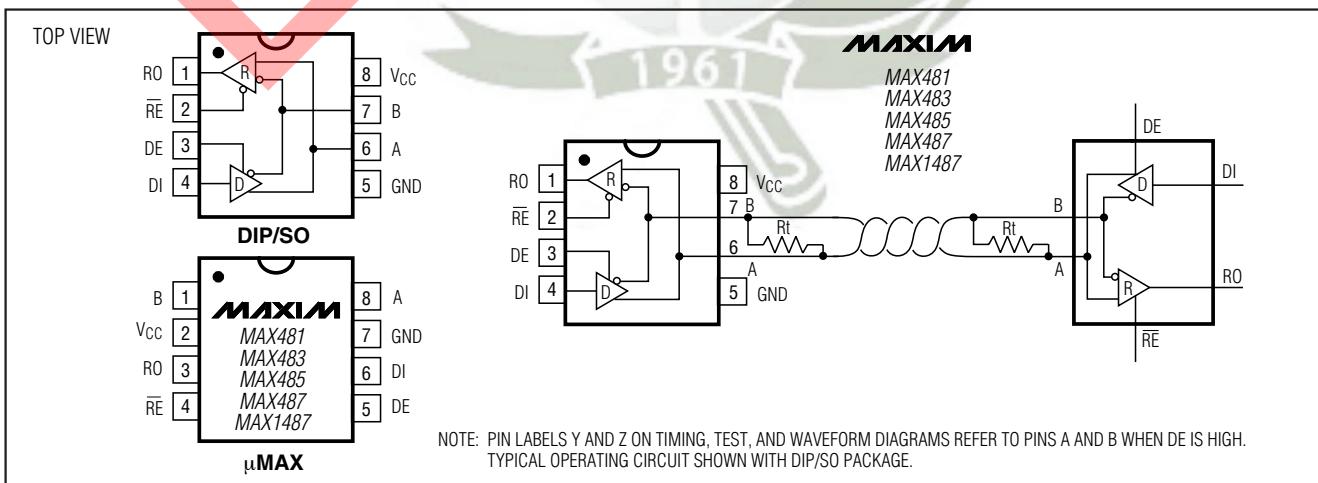


Figure 1. MAX481/MAX483/MAX485/MAX487/MAX1487 Pin Configuration and Typical Operating Circuit

Low-Power, Slew-Rate-Limited RS-485/RS-422 Transceivers

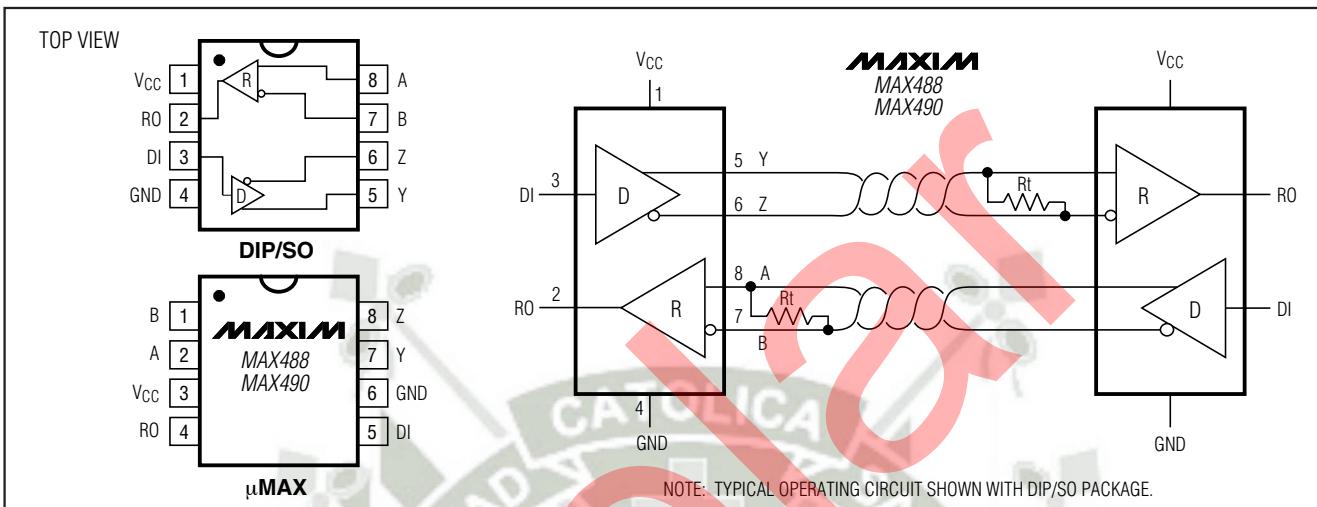


Figure 2. MAX488/MAX490 Pin Configuration and Typical Operating Circuit

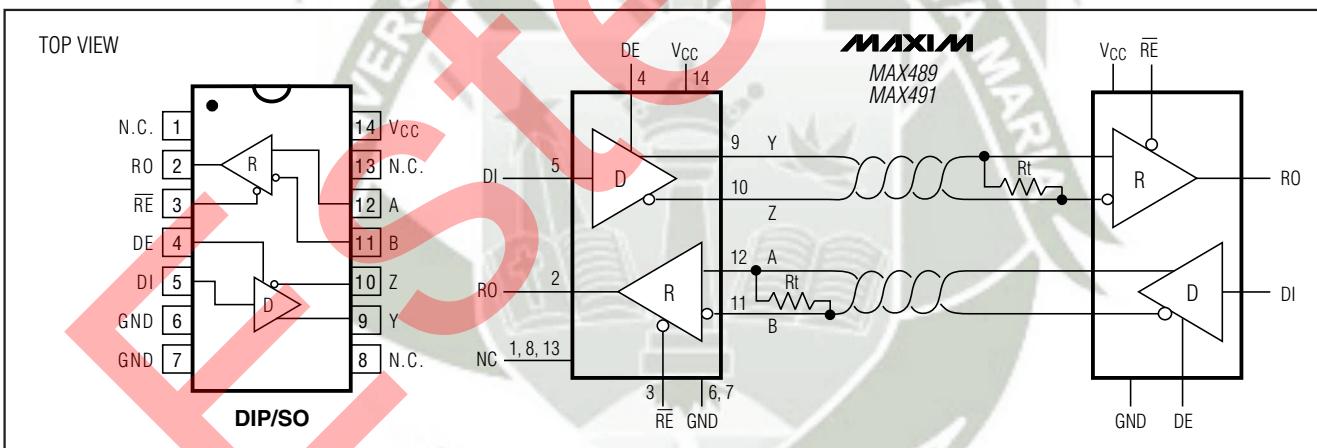


Figure 3. MAX489/MAX491 Pin Configuration and Typical Operating Circuit

Applications Information

The MAX481/MAX483/MAX485/MAX487-MAX491 and MAX1487 are low-power transceivers for RS-485 and RS-422 communications. The MAX481, MAX485, MAX490, MAX491, and MAX1487 can transmit and receive at data rates up to 2.5Mbps, while the MAX483, MAX487, MAX488, and MAX489 are specified for data rates up to 250kbps. The MAX488-MAX491 are full-duplex transceivers while the MAX481, MAX483, MAX485, MAX487, and MAX1487 are half-duplex. In addition, Driver Enable (DE) and Receiver Enable (\bar{RE}) pins are included on the MAX481, MAX483, MAX485, MAX487, MAX489, MAX491, and MAX1487. When disabled, the driver and receiver outputs are high impedance.

MAX487/MAX1487: 128 Transceivers on the Bus

The $48k\Omega$, $1/4$ -unit-load receiver input impedance of the MAX487 and MAX1487 allows up to 128 transceivers on a bus, compared to the 1-unit load ($12k\Omega$ input impedance) of standard RS-485 drivers (32 transceivers maximum). Any combination of MAX487/MAX1487 and other RS-485 transceivers with a total of 32 unit loads or less can be put on the bus. The MAX481/MAX483/MAX485 and MAX488-MAX491 have standard $12k\Omega$ Receiver Input impedance.

ANEXO H: Ficha técnica

PIC18F4550



MICROCHIP

PIC18F2455/2550/4455/4550
Data Sheet

28/40/44-Pin, High-Performance,
Enhanced Flash, USB Microcontrollers
with nanoWatt Technology

ESTE

Note the following details of the code protection feature on Microchip devices:

- Microchip products meet the specification contained in their particular Microchip Data Sheet.
- Microchip believes that its family of products is one of the most secure families of its kind on the market today, when used in the intended manner and under normal conditions.
- There are dishonest and possibly illegal methods used to breach the code protection feature. All of these methods, to our knowledge, require using the Microchip products in a manner outside the operating specifications contained in Microchip's Data Sheets. Most likely, the person doing so is engaged in theft of intellectual property.
- Microchip is willing to work with the customer who is concerned about the integrity of their code.
- Neither Microchip nor any other semiconductor manufacturer can guarantee the security of their code. Code protection does not mean that we are guaranteeing the product as "unbreakable."

Code protection is constantly evolving. We at Microchip are committed to continuously improving the code protection features of our products. Attempts to break Microchip's code protection feature may be a violation of the Digital Millennium Copyright Act. If such acts allow unauthorized access to your software or other copyrighted work, you may have a right to sue for relief under that Act.

Information contained in this publication regarding device applications and the like is provided only for your convenience and may be superseded by updates. It is your responsibility to ensure that your application meets with your specifications. MICROCHIP MAKES NO REPRESENTATIONS OR WARRANTIES OF ANY KIND WHETHER EXPRESS OR IMPLIED, WRITTEN OR ORAL, STATUTORY OR OTHERWISE, RELATED TO THE INFORMATION, INCLUDING BUT NOT LIMITED TO ITS CONDITION, QUALITY, PERFORMANCE, MERCHANTABILITY OR FITNESS FOR PURPOSE. Microchip disclaims all liability arising from this information and its use. Use of Microchip devices in life support and/or safety applications is entirely at the buyer's risk, and the buyer agrees to defend, indemnify and hold harmless Microchip from any and all damages, claims, suits, or expenses resulting from such use. No licenses are conveyed, implicitly or otherwise, under any Microchip intellectual property rights.

Trademarks

The Microchip name and logo, the Microchip logo, Accuron, dsPIC, KEELOQ, microID, MPLAB, PIC, PICmicro, PICSTART, PRO MATE, PowerSmart, rfPIC and SmartShunt are registered trademarks of Microchip Technology Incorporated in the U.S.A. and other countries.

AmpLab, FilterLab, Migratable Memory, MXDEV, MXLAB, PICMASTER, SEEVAL, SmartSensor and The Embedded Control Solutions Company are registered trademarks of Microchip Technology Incorporated in the U.S.A.

Analog-for-the-Digital Age, Application Maestro, dsPICDEM, dsPICDEM.net, dsPICworks, ECAN, ECONOMONITOR, FanSense, FlexROM, fuzzyLAB, In-Circuit Serial Programming, ICSP, ICEPIC, Linear Active Thermistor, MPASM, MPLIB, MPLINK, MPSIM, PICkit, PICDEM, PICDEM.net, PICLAB, PICtail, PowerCal, PowerInfo, PowerMate, PowerTool, Real ICE, rfLAB, rfPICDEM, Select Mode, Smart Serial, SmartTel, Total Endurance, UNI/O, WiperLock and Zena are trademarks of Microchip Technology Incorporated in the U.S.A. and other countries.

SQTP is a service mark of Microchip Technology Incorporated in the U.S.A.

All other trademarks mentioned herein are property of their respective companies.

© 2006, Microchip Technology Incorporated, Printed in the U.S.A., All Rights Reserved.

Printed on recycled paper.

QUALITY MANAGEMENT SYSTEM

CERTIFIED BY DNV

== ISO/TS 16949:2002 ==

Microchip received ISO/TS-16949:2002 quality system certification for its worldwide headquarters, design and wafer fabrication facilities in Chandler and Tempe, Arizona and Mountain View, California in October 2003. The Company's quality system processes and procedures are for its PICmicro® 8-bit MCUs, KEELOQ® code hopping devices, Serial EEPROMs, microperipherals, nonvolatile memory and analog products. In addition, Microchip's quality system for the design and manufacture of development systems is ISO 9001:2000 certified.



MICROCHIP PIC18F2455/2550/4455/4550

28/40/44-Pin, High-Performance, Enhanced Flash, USB Microcontrollers with nanoWatt Technology

Universal Serial Bus Features:

- USB V2.0 Compliant
- Low Speed (1.5 Mb/s) and Full Speed (12 Mb/s)
- Supports Control, Interrupt, Isochronous and Bulk Transfers
- Supports up to 32 Endpoints (16 bidirectional)
- 1-Kbyte Dual Access RAM for USB
- On-Chip USB Transceiver with On-Chip Voltage Regulator
- Interface for Off-Chip USB Transceiver
- Streaming Parallel Port (SPP) for USB streaming transfers (40/44-pin devices only)

Power-Managed Modes:

- Run: CPU on, peripherals on
- Idle: CPU off, peripherals on
- Sleep: CPU off, peripherals off
- Idle mode currents down to 5.8 μ A typical
- Sleep mode currents down to 0.1 μ A typical
- Timer1 Oscillator: 1.1 μ A typical, 32 kHz, 2V
- Watchdog Timer: 2.1 μ A typical
- Two-Speed Oscillator Start-up

Flexible Oscillator Structure:

- Four Crystal modes, including High Precision PLL for USB
- Two External Clock modes, up to 48 MHz
- Internal Oscillator Block:
 - 8 user-selectable frequencies, from 31 kHz to 8 MHz
 - User-tunable to compensate for frequency drift
- Secondary Oscillator using Timer1 @ 32 kHz
- Dual Oscillator options allow microcontroller and USB module to run at different clock speeds
- Fail-Safe Clock Monitor:
 - Allows for safe shutdown if any clock stops

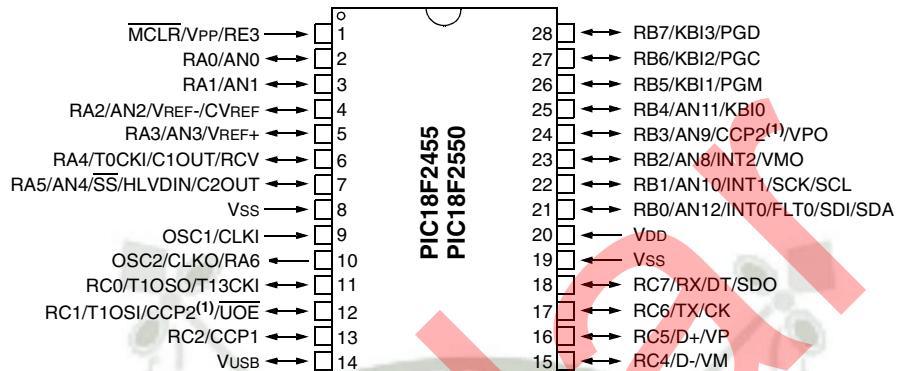
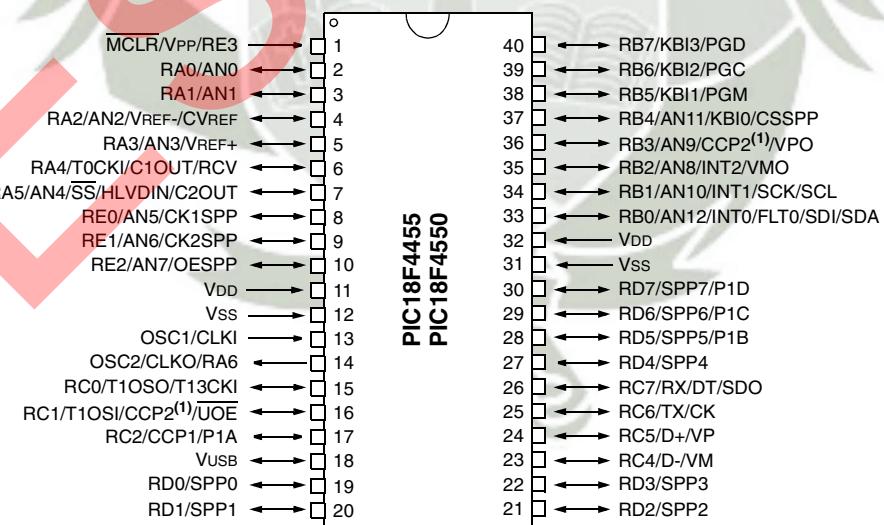
Peripheral Highlights:

- High-Current Sink/Source: 25 mA/25 mA
- Three External Interrupts
- Four Timer modules (Timer0 to Timer3)
- Up to 2 Capture/Compare/PWM (CCP) modules:
 - Capture is 16-bit, max. resolution 5.2 ns (Tcy/16)
 - Compare is 16-bit, max. resolution 83.3 ns (Tcy)
 - PWM output: PWM resolution is 1 to 10-bit
- Enhanced Capture/Compare/PWM (ECCP) module:
 - Multiple output modes
 - Selectable polarity
 - Programmable dead time
 - Auto-shutdown and auto-restart
- Enhanced USART module:
 - LIN bus support
- Master Synchronous Serial Port (MSSP) module supporting 3-wire SPI (all 4 modes) and I²C™ Master and Slave modes
- 10-bit, up to 13-channel Analog-to-Digital Converter module (A/D) with Programmable Acquisition Time
- Dual Analog Comparators with Input Multiplexing

Special Microcontroller Features:

- C Compiler Optimized Architecture with optional Extended Instruction Set
- 100,000 Erase/Write Cycle Enhanced Flash Program Memory typical
- 1,000,000 Erase/Write Cycle Data EEPROM Memory typical
- Flash/Data EEPROM Retention: > 40 years
- Self-Programmable under Software Control
- Priority Levels for Interrupts
- 8 x 8 Single-Cycle Hardware Multiplier
- Extended Watchdog Timer (WDT):
 - Programmable period from 41 ms to 131s
- Programmable Code Protection
- Single-Supply 5V In-Circuit Serial Programming™ (ICSP™) via two pins
- In-Circuit Debug (ICD) via two pins
- Optional dedicated ICD/ICSP port (44-pin devices only)
- Wide Operating Voltage Range (2.0V to 5.5V)

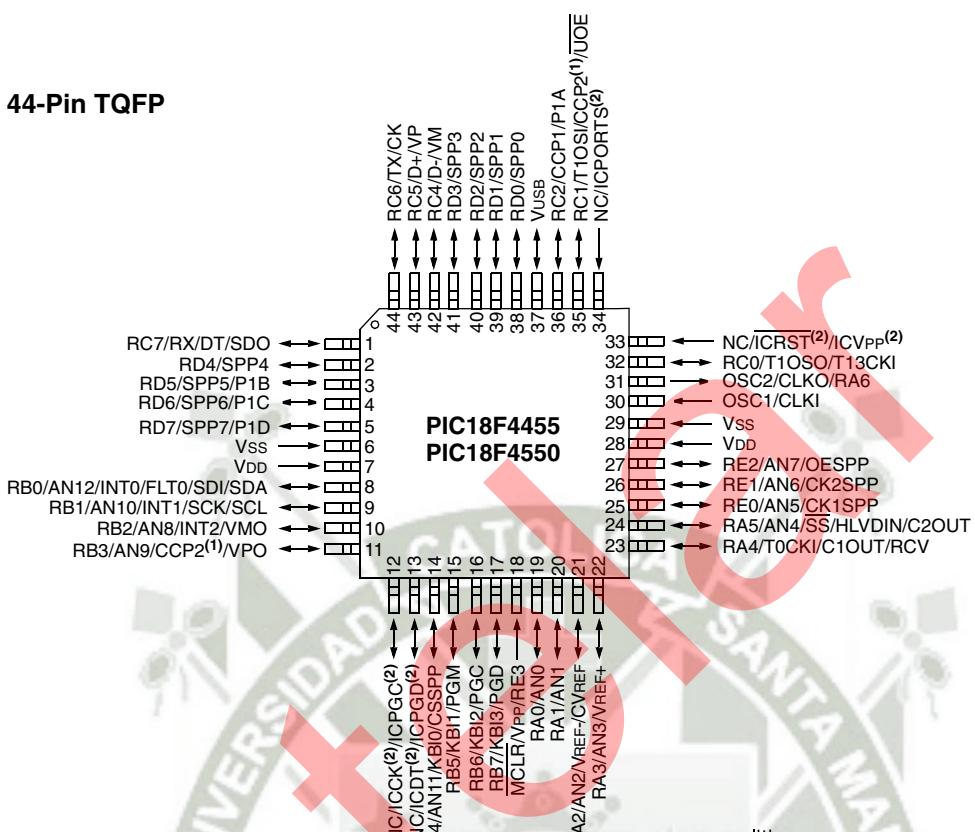
Device	Program Memory		Data Memory		I/O	10-Bit A/D (ch)	CCP/ECCP (PWM)	SPP	MSSP		USART	Comparators	Timers 8/16-Bit
	Flash (bytes)	# Single-Word Instructions	SRAM (bytes)	EEPROM (bytes)					SPI	Master I ² C™			
PIC18F2455	24K	12288	2048	256	24	10	2/0	No	Y	Y	1	2	1/3
PIC18F2550	32K	16384	2048	256	24	10	2/0	No	Y	Y	1	2	1/3
PIC18F4455	24K	12288	2048	256	35	13	1/1	Yes	Y	Y	1	2	1/3
PIC18F4550	32K	16384	2048	256	35	13	1/1	Yes	Y	Y	1	2	1/3

Pin Diagrams**28-Pin PDIP, SOIC****40-Pin PDIP**

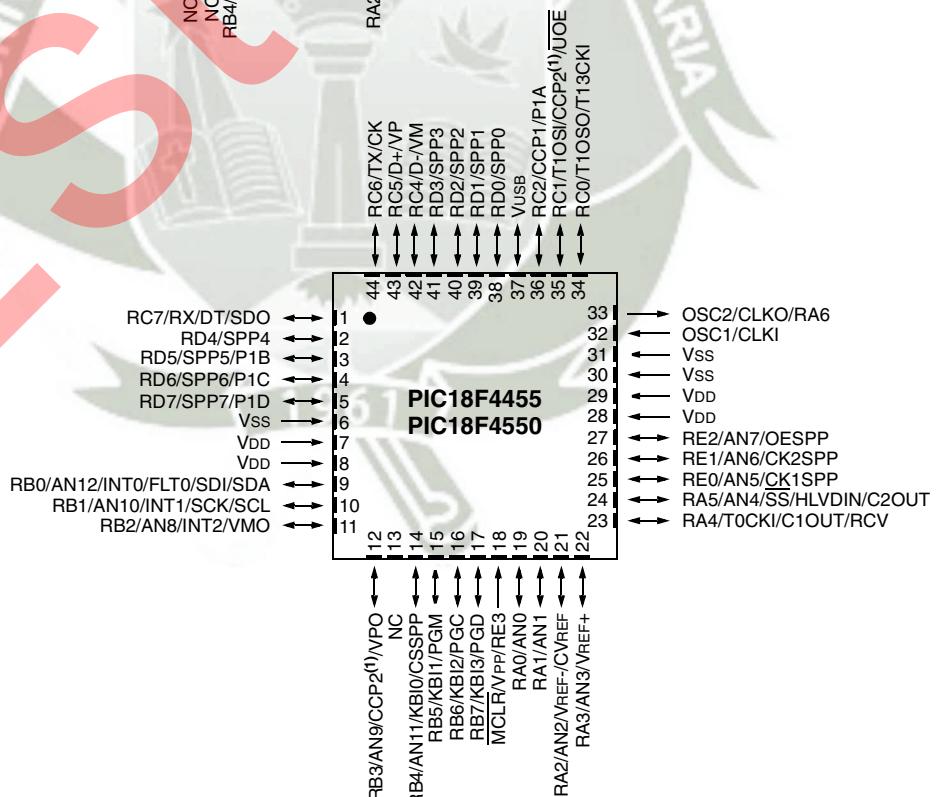
Note 1: RB3 is the alternate pin for CCP2 multiplexing.

Pin Diagrams (Continued)

44-Pin TQFP



44-Pin QFN



Note 1: RB3 is the alternate pin for CCP2 multiplexing.

2: Special ICPORTS features available in select circumstances. See **Section 25.9 "Special ICPORT Features (Designated Packages Only)"** for more information.

ANEXO I: Código de la aplicación seguimiento.py con las pruebas

```
#time
import time

#plot
import matplotlib.pyplot as plt

#import video
import numpy as np
import cv2
import cv2.cv as cv

#import os

#import fuzzy
import fuzzy.storage.fcl.Reader

#import Serial COM
import serial

#init video
#os.system("v4l2-ctl -d /dev/video1 --set-fmt-video=width=360,height=240")
cap = cv2.VideoCapture(1)

ret=cap.set(3, 320)
ret=cap.set(4, 240)
#ret=cap.set(5, 1) #FPS

#inicia fuzzy
system_velo = fuzzy.storage.fcl.Reader().load_from_file("camara_fuzzy.fcl")

my_input = {
    "dist" : 0.0,
    "movi" : 0.0
}

my_output = {
    "velo" : 0.0
}

#init serial COM
ser=serial.Serial('/dev/ttyACM0',9600,timeout=0.5)
#ser=serial.Serial('/dev/ttyUSB1',9600,timeout=0.5)

#PELCO-D
sync= 0xff
c_a= 0x01
com_1= 0x00

#variables velocidad inicial
vx_ant = 0
```

```

vy_ant = 0

#tiempo inicial
start_time = time.time()

#Variables plot
pt = [0]
pdx= [0]
pdy= [0]
pxv= [0]
pyv= [0]
s = 0

while(True):
    ret, frame = cap.read()

    #Covierte de RGB a HSV
    hsv = cv2.cvtColor(frame, cv2.COLOR_BGR2HSV)

    #verde RGB 0 255 0 -> 60 255 255
    lower_green = np.array([30,50,50])
    upper_green = np.array([90,255,255])

    #Detecta objetos verdes
    mask = cv2.inRange(hsv, lower_green, upper_green)

    #filtro
    gray_ff = cv2.medianBlur(mask,9)

    #detecta circulos
    circles =
    cv2.HoughCircles(gray_ff,cv.CV_HOUGH_GRADIENT,2,40,param1=50,param2=30,minRadius=
    10,maxRadius=50)

    #print circles

    if circles is not None:

        #convierte formato de datos
        circles = np.uint16(np.around(circles))

        #Dibuja circulos
        for i in circles[0,:]:
            cv2.circle(frame,(i[0],i[1]),i[2],(0,255,0),2)
            cv2.circle(frame,(i[0],i[1]),2,(0,0,255),3)

        cv2.imshow('frame', frame)
        cv2.imshow('mask', gray_ff)

        #obtiene centro del circulo
        cx = circles[0,0,0]
        cy = circles[0,0,1]

```

dx = cx-160

dy = cy-120

```
#Evalua fuzzy
my_input["dist"] = dx
my_input["movi"] = vx_ant
system_velo.calculate(my_input, my_output)
vx = int(my_output["velo"])
vx_ant = vx

my_input["dist"] = dy*4/3
my_input["movi"] = vy_ant
system_velo.calculate(my_input, my_output)
vy= int(my_output["velo"])
vy_ant = vy
```

#PLOT

```
t = time.time() - start_time
print "t", t
```

```
#plt.plot([1, 2, 3, 4])
pt.insert(s, t)
pdx.insert(s, dx)
pdy.insert(s, dy)
pxv.insert(s, vx)
pyv.insert(s, vy)
s = s + 1
```

```
#fig = plt.figure()
#plt.show()
```

if vx == 0: #no mueve

vvx = vx

com_2 = 0x00 #L:0x04 R:0x02 U:0x08 D:0x10

if vx > 0: #mueve derecha

vvx = vx

com_2 = 0x02 #L:0x04 R:0x02 U:0x08 D:0x10

if vx < 0: #mueve izquierda

vvx = -vx

com_2 = 0x04 #L:0x04 R:0x02 U:0x08 D:0x10

if vy == 0: #no mueve

vvy = vy

com_2 = com_2 + 0x00 #L:0x04 R:0x02 U:0x08 D:0x10

```

if vy > 0: #mueve arriba
    vvy = vy
    com_2 = com_2 + 0x10    #L:0x04 R:0x02 U:0x08 D:0x10

if vy < 0: #mueve abajo
    vvy = -vy
    com_2 = com_2 + 0x08    #L:0x04 R:0x02 U:0x08 D:0x10

print "dx: ", dx
print "dy: ", dy
print "vx: ", vx
print "vy: ", vy

dat_1 = vvx    #pan Speed: 0x00 -> 0x3f  Turbo: 0xff
dat_2 = vvy    #tilt Speed: 0x00 -> 0x3f  Turbo: 0xff

c_s = c_a + com_1 + com_2 + dat_1 + dat_2

envio = chr(sync)+chr(c_a)+chr(com_1)+chr(com_2)+chr(dat_1)+chr(dat_2)+chr(c_s)
ser.write(envio)

else:
    #print circles
    cv2.imshow('frame', frame)
    com_2 = 0
    dat_1 = 0    #pan Speed: 0x00 -> 0x3f
    dat_2 = 0    #tilt Speed: 0x00 -> 0x3f
    c_s = c_a + com_1 + com_2 + dat_1 + dat_2
    envio = chr(sync)+chr(c_a)+chr(com_1)+chr(com_2)+chr(dat_1)+chr(dat_2)+chr(c_s)
    ser.write(envio)

if cv2.waitKey(1) & 0xff == ord('q'):
    break

cap.release()
cv2.destroyAllWindows()

plt.subplot(211)
plt.title('X (Amarillo: DX    Rojo: VX)')
plt.plot(pt,pdx,'y')
plt.plot(pt,pxv,'r')

plt.subplot(212)
plt.title('Y (Verde: DY    Azul: VY)')
plt.plot(pt,pdy,'g')
plt.plot(pt,pvy,'b')

plt.show()

```

ANEXO J: Manual de VideoCapture de OpenCV

```
#include <vector>
#include <stdio.h>
#include <opencv2/opencv.hpp>

using namespace cv;
using namespace std;

void createAlphaMat(Mat &mat)
{
    for (int i = 0; i < mat.rows; ++i) {
        for (int j = 0; j < mat.cols; ++j) {
            Vec4b& rgba = mat.at<Vec4b>(i, j);
            rgba[0] = UCHAR_MAX;
            rgba[1] = saturate_cast<uchar>((float (mat.cols - j)) / ((float)mat.cols) * UCHAR_MAX);
            rgba[2] = saturate_cast<uchar>((float (mat.rows - i)) / ((float)mat.rows) * UCHAR_MAX);
            rgba[3] = saturate_cast<uchar>(0.5 * (rgba[1] + rgba[2]));
        }
    }
}

int main(int argc, char **argv)
{
    // Create mat with alpha channel
    Mat mat(480, 640, CV_8UC4);
    createAlphaMat(mat);

    vector<int> compression_params;
    compression_params.push_back(CV_IMWRITER_PNG_COMPRESSION);
    compression_params.push_back(9);

    try {
        imwrite("alpha.png", mat, compression_params);
    }
    catch (runtime_error& ex) {
        fprintf(stderr, "Exception converting image to PNG format: %s\n", ex.what());
        return 1;
    }

    fprintf(stdout, "Saved PNG file with alpha data.\n");
    return 0;
}
```

VideoCapture

class VideoCapture

Class for video capturing from video files or cameras. The class provides C++ API for capturing video from cameras or for reading video files. Here is how the class can be used:

```
#include "opencv2/opencv.hpp"

using namespace cv;

int main(int, char**)
{
    VideoCapture cap(0); // open the default camera
    if(!cap.isOpened()) // check if we succeeded
```



```
return -1;

Mat edges;
namedWindow("edges",1);
for(;;)
{
    Mat frame;
    cap >> frame; // get a new frame from camera
    cvtColor(frame, edges, CV_BGR2GRAY);
    GaussianBlur(edges, edges, Size(7,7), 1.5, 1.5);
    Canny(edges, edges, 0, 30, 3);
    imshow("edges", edges);
    if(waitKey(30) >= 0) break;
}
// the camera will be deinitialized automatically in VideoCapture destructor
return 0;
}
```

Note: In C API the black-box structure CvCapture is used instead of VideoCapture.

Note:

- A basic sample on using the VideoCapture interface can be found at [opencv_source_code/samples/cpp/starter_video.cpp](#)
- Another basic video processing sample can be found at [opencv_source_code/samples/cpp/video_dmtx.cpp](#)
- (Python) A basic sample on using the VideoCapture interface can be found at [opencv_source_code/samples/python2/video.py](#)
- (Python) Another basic video processing sample can be found at [opencv_source_code/samples/python2/video_dmtx.py](#)
- (Python) A multi threaded video processing sample can be found at [opencv_source_code/samples/python2/video_threaded.py](#)

VideoCapture::VideoCapture

VideoCapture constructors.

C++: `VideoCapture::VideoCapture()`

C++: `VideoCapture::VideoCapture(const string& filename)`

C++: `VideoCapture::VideoCapture(int device)`

Python: `cv2.VideoCapture() → <VideoCapture object>`

Python: `cv2.VideoCapture(filename) → <VideoCapture object>`

Python: `cv2.VideoCapture(device) → <VideoCapture object>`

C: `CvCapture* cvCaptureFromCAM(int device)`

Python: `cv.CaptureFromCAM(index) → CvCapture`

C: `CvCapture* cvCaptureFromFile(const char* filename)`

Python: `cv.CaptureFromFile(filename) → CvCapture`

Parameters

filename – name of the opened video file (eg. video.avi) or image sequence (eg. img_%02d.jpg, which will read samples like img_00.jpg, img_01.jpg, img_02.jpg, ...)

device – id of the opened video capturing device (i.e. a camera index). If there is a single camera connected, just pass 0.

Note: In C API, when you finished working with video, release CvCapture structure with cvReleaseCapture(), or use Ptr<CvCapture> that calls cvReleaseCapture() automatically in the destructor.

VideoCapture::open

Open video file or a capturing device for video capturing

C++: bool VideoCapture::**open**(const string& **filename**)

C++: bool VideoCapture::**open**(int **device**)

Python: cv2.VideoCapture.**open**(filename) → retval

Python: cv2.VideoCapture.**open**(device) → retval

Parameters

filename – name of the opened video file (eg. video.avi) or image sequence (eg. img_%02d.jpg, which will read samples like img_00.jpg, img_01.jpg, img_02.jpg, ...)

device – id of the opened video capturing device (i.e. a camera index).

The methods first call [VideoCapture::release\(\)](#) to close the already opened file or camera.

VideoCapture::isOpened

Returns true if video capturing has been initialized already.

C++: bool VideoCapture::**isOpened**()

Python: cv2.VideoCapture.**isOpened**() → retval

If the previous call to VideoCapture constructor or VideoCapture::open succeeded, the method returns true.

VideoCapture::release

Closes video file or capturing device.

C++: void VideoCapture::**release**()

Python: cv2.VideoCapture.**release**() → None

C: void **cvReleaseCapture**(CvCapture** **capture**)

The methods are automatically called by subsequent [VideoCapture::open\(\)](#) and by VideoCapture destructor.

The C function also deallocates memory and clears *capture pointer.



VideoCapture::grab

Grabs the next frame from video file or capturing device.

C++: bool `VideoCapture::grab()`

Python: `cv2.VideoCapture.grab() → retval`

C: int `cvGrabFrame(CvCapture* capture)`

Python: `cv.GrabFrame(capture) → int`

The methods/functions grab the next frame from video file or camera and return true (non-zero) in the case of success.

The primary use of the function is in multi-camera environments, especially when the cameras do not have hardware synchronization. That is, you call `VideoCapture::grab()` for each camera and after that call the slower method `VideoCapture::retrieve()` to decode and get frame from each camera. This way the overhead on demosaicing or motion jpeg decompression etc. is eliminated and the retrieved frames from different cameras will be closer in time.

Also, when a connected camera is multi-head (for example, a stereo camera or a Kinect device), the correct way of retrieving data from it is to call `VideoCapture::grab` first and then call `VideoCapture::retrieve()` one or more times with different values of the channel parameter. See https://github.com/Itseez/opencv/tree/master/samples/cpp/openni_capture.cpp

VideoCapture::retrieve

Decodes and returns the grabbed video frame.

C++: bool `VideoCapture::retrieve(Mat& image, int channel=0)`

Python: `cv2.VideoCapture.retrieve([image[, channel]]) → retval, image`

C: IplImage* `cvRetrieveFrame(CvCapture* capture, int streamIdx=0)`

Python: `cv.RetrieveFrame(capture) → image`

The methods/functions decode and return the just grabbed frame. If no frames has been grabbed (camera has been disconnected, or there are no more frames in video file), the methods return false and the functions return NULL pointer.

Note: OpenCV 1.x functions `cvRetrieveFrame` and `cv.RetrieveFrame` return image stored inside the video capturing structure. It is not allowed to modify or release the image! You can copy the frame using `cvCloneImage()` and then do whatever you want with the copy.

VideoCapture::read

Grabs, decodes and returns the next video frame.

C++: `VideoCapture& VideoCapture::operator>>(Mat& image)`

C++: bool `VideoCapture::read(Mat& image)`

Python: `cv2.VideoCapture.read([image]) → retval, image`

C: IplImage* `cvQueryFrame(CvCapture* capture)`

Python: `cv.QueryFrame(capture) → image`

The methods/functions combine `VideoCapture::grab()` and `VideoCapture::retrieve()` in one call. This is the most convenient method for reading video files or capturing data from decode and return the just grabbed frame. If no frames has been grabbed (camera has been disconnected, or there are no more frames in video file), the methods return false and the functions return NULL pointer.

Note: OpenCV 1.x functions `cvRetrieveFrame` and `cv.RetrieveFrame` return image stored inside the video capturing structure. It is not allowed to modify or release the image! You can copy the frame using `cvCloneImage()` and then do whatever you want with the copy.

VideoCapture::get

Returns the specified VideoCapture property

C++: double `VideoCapture::get(int propId)`

Python: `cv2.VideoCapture.get(propId) → retval`

C: double `cvGetCaptureProperty(CvCapture* capture, int property_id)`

Python: `cv.GetCaptureProperty(capture, property_id) → float`

Parameters

propId – Property identifier. It can be one of the following:

- **CV_CAP_PROP_POS_MSEC** Current position of the video file in milliseconds or video capture timestamp.
- **CV_CAP_PROP_POS_FRAMES** 0-based index of the frame to be decoded/captured next.
- **CV_CAP_PROP_POS_AVI_RATIO** Relative position of the video file: 0 - start of the film, 1 - end of the film.
- **CV_CAP_PROP_FRAME_WIDTH** Width of the frames in the video stream.
- **CV_CAP_PROP_FRAME_HEIGHT** Height of the frames in the video stream.
- **CV_CAP_PROP_FPS** Frame rate.
- **CV_CAP_PROP_FOURCC** 4-character code of codec.
- **CV_CAP_PROP_FRAME_COUNT** Number of frames in the video file.
- **CV_CAP_PROP_FORMAT** Format of the Mat objects returned by `retrieve()`.
- **CV_CAP_PROP_MODE** Backend-specific value indicating the current capture mode.
- **CV_CAP_PROP_BRIGHTNESS** Brightness of the image (only for cameras).
- **CV_CAP_PROP_CONTRAST** Contrast of the image (only for cameras).
- **CV_CAP_PROP_SATURATION** Saturation of the image (only for cameras).
- **CV_CAP_PROP_HUE** Hue of the image (only for cameras).
- **CV_CAP_PROP_GAIN** Gain of the image (only for cameras).
- **CV_CAP_PROP_EXPOSURE** Exposure (only for cameras).
- **CV_CAP_PROP_CONVERT_RGB** Boolean flags indicating whether images should be converted to RGB.
- **CV_CAP_PROP_WHITE_BALANCE** Currently not supported



- **CV_CAP_PROP_RECTIFICATION** Rectification flag for stereo cameras (note: only supported by DC1394 v 2.x backend currently)

Note: When querying a property that is not supported by the backend used by the VideoCapture class, value 0 is returned.

VideoCapture::set

Sets a property in the VideoCapture.

C++: bool VideoCapture::set(int propId, double value)

Python: cv2.VideoCapture.set(propId, value) → retval

C: int cvSetCaptureProperty(CvCapture* capture, int property_id, double value)

Python: cv.SetCaptureProperty(capture, property_id, value) → retval

Parameters

propId – Property identifier. It can be one of the following:

- **CV_CAP_PROP_POS_MSEC** Current position of the video file in milliseconds.
- **CV_CAP_PROP_POS_FRAMES** 0-based index of the frame to be decoded/captured next.
- **CV_CAP_PROP_POS_AVI_RATIO** Relative position of the video file: 0 - start of the film, 1 - end of the film.
- **CV_CAP_PROP_FRAME_WIDTH** Width of the frames in the video stream.
- **CV_CAP_PROP_FRAME_HEIGHT** Height of the frames in the video stream.
- **CV_CAP_PROP_FPS** Frame rate.
- **CV_CAP_PROP_FOURCC** 4-character code of codec.
- **CV_CAP_PROP_FRAME_COUNT** Number of frames in the video file.
- **CV_CAP_PROP_FORMAT** Format of the Mat objects returned by `retrieve()`.
- **CV_CAP_PROP_MODE** Backend-specific value indicating the current capture mode.
- **CV_CAP_PROP_BRIGHTNESS** Brightness of the image (only for cameras).
- **CV_CAP_PROP_CONTRAST** Contrast of the image (only for cameras).
- **CV_CAP_PROP_SATURATION** Saturation of the image (only for cameras).
- **CV_CAP_PROP_HUE** Hue of the image (only for cameras).
- **CV_CAP_PROP_GAIN** Gain of the image (only for cameras).
- **CV_CAP_PROP_EXPOSURE** Exposure (only for cameras).
- **CV_CAP_PROP_CONVERT_RGB** Boolean flags indicating whether images should be converted to RGB.
- **CV_CAP_PROP_WHITE_BALANCE** Currently unsupported
- **CV_CAP_PROP_RECTIFICATION** Rectification flag for stereo cameras (note: only supported by DC1394 v 2.x backend currently)

value – Value of the property.

ANEXO K: Manual de cvtColor de OpenCV



- **THRESH_BINARY_INV**

$$dst(x, y) = \begin{cases} 0 & \text{if } src(x, y) > T(x, y) \\ maxValue & \text{otherwise} \end{cases}$$

where $T(x, y)$ is a threshold calculated individually for each pixel.

- For the method `ADAPTIVE_THRESH_MEAN_C`, the threshold value $T(x, y)$ is a mean of the `blockSize` \times `blockSize` neighborhood of (x, y) minus C .
- For the method `ADAPTIVE_THRESH_GAUSSIAN_C`, the threshold value $T(x, y)$ is a weighted sum (cross-correlation with a Gaussian window) of the `blockSize` \times `blockSize` neighborhood of (x, y) minus C . The default sigma (standard deviation) is used for the specified `blockSize`. See `getGaussianKernel()`.

The function can process the image in-place.

See Also:

`threshold()`, `blur()`, `GaussianBlur()`

cvtColor

Converts an image from one color space to another.

C++: `void cvtColor(InputArray src, OutputArray dst, int code, int dstCn=0)`

Python: `cv2.cvtColor(src, code[, dst[, dstCn]]) → dst`

C: `void cvCvtColor(const CvArr* src, CvArr* dst, int code)`

Python: `cv.CvtColor(src, dst, code) → None`

Parameters

src – input image: 8-bit unsigned, 16-bit unsigned (`CV_16UC...`), or single-precision floating-point.

dst – output image of the same size and depth as **src**.

code – color space conversion code (see the description below).

dstCn – number of channels in the destination image; if the parameter is 0, the number of the channels is derived automatically from **src** and **code**.

The function converts an input image from one color space to another. In case of a transformation to-from RGB color space, the order of the channels should be specified explicitly (RGB or BGR). Note that the default color format in OpenCV is often referred to as RGB but it is actually BGR (the bytes are reversed). So the first byte in a standard (24-bit) color image will be an 8-bit Blue component, the second byte will be Green, and the third byte will be Red. The fourth, fifth, and sixth bytes would then be the second pixel (Blue, then Green, then Red), and so on.

The conventional ranges for R, G, and B channel values are:

- 0 to 255 for `CV_8U` images
- 0 to 65535 for `CV_16U` images
- 0 to 1 for `CV_32F` images

In case of linear transformations, the range does not matter. But in case of a non-linear transformation, an input RGB image should be normalized to the proper value range to get the correct results, for example, for $RGB \rightarrow L^*u^*v^*$ transformation. For example, if you have a 32-bit floating-point image directly converted from an 8-bit image without

any scaling, then it will have the 0..255 value range instead of 0..1 assumed by the function. So, before calling `cvtColor`, you need first to scale the image down:

```
img *= 1./255;
cvtColor(img, img, CV_BGR2Luv);
```

If you use `cvtColor` with 8-bit images, the conversion will have some information lost. For many applications, this will not be noticeable but it is recommended to use 32-bit images in applications that need the full range of colors or that convert an image before an operation and then convert back.

If conversion adds the alpha channel, its value will set to the maximum of corresponding channel range: 255 for `CV_8U`, 65535 for `CV_16U`, 1 for `CV_32F`.

The function can do the following transformations:

- $\text{RGB} \leftrightarrow \text{GRAY}$ (`CV_BGR2GRAY`, `CV_RGB2GRAY`, `CV_GRAY2BGR`, `CV_GRAY2RGB`) Transformations within RGB space like adding/removing the alpha channel, reversing the channel order, conversion to/from 16-bit RGB color (R5:G6:B5 or R5:G5:B5), as well as conversion to/from grayscale using:

$$\text{RGB}[A] \text{ to Gray: } Y \leftarrow 0.299 \cdot R + 0.587 \cdot G + 0.114 \cdot B$$

and

$$\text{Gray to RGB}[A]: R \leftarrow Y, G \leftarrow Y, B \leftarrow Y, A \leftarrow \max(\text{ChannelRange})$$

The conversion from a RGB image to gray is done with:

```
cvtColor(src, bwsrc, CV_RGB2GRAY);
```

More advanced channel reordering can also be done with `mixChannels()`.

- $\text{RGB} \leftrightarrow \text{CIE XYZ Rec 709}$ with D65 white point (`CV_BGR2XYZ`, `CV_RGB2XYZ`, `CV_XYZ2BGR`, `CV_XYZ2RGB`):

$$\begin{bmatrix} X \\ Y \\ Z \end{bmatrix} \leftarrow \begin{bmatrix} 0.412453 & 0.357580 & 0.180423 \\ 0.212671 & 0.715160 & 0.072169 \\ 0.019334 & 0.119193 & 0.950227 \end{bmatrix} \cdot \begin{bmatrix} R \\ G \\ B \end{bmatrix}$$

$$\begin{bmatrix} R \\ G \\ B \end{bmatrix} \leftarrow \begin{bmatrix} 3.240479 & -1.53715 & -0.498535 \\ -0.969256 & 1.875991 & 0.041556 \\ 0.055648 & -0.204043 & 1.057311 \end{bmatrix} \cdot \begin{bmatrix} X \\ Y \\ Z \end{bmatrix}$$

X, Y and Z cover the whole value range (in case of floating-point images, Z may exceed 1).

- $\text{RGB} \leftrightarrow \text{YCrCb JPEG}$ (or `YCC`) (`CV_BGR2YCrCb`, `CV_RGB2YCrCb`, `CV_YCrCb2BGR`, `CV_YCrCb2RGB`)

$$Y \leftarrow 0.299 \cdot R + 0.587 \cdot G + 0.114 \cdot B$$

$$Cr \leftarrow (R - Y) \cdot 0.713 + \text{delta}$$

$$Cb \leftarrow (B - Y) \cdot 0.564 + \text{delta}$$

$$R \leftarrow Y + 1.403 \cdot (Cr - \text{delta})$$

$$G \leftarrow Y - 0.714 \cdot (Cr - \text{delta}) - 0.344 \cdot (Cb - \text{delta})$$



$$B \leftarrow Y + 1.773 \cdot (Cb - \text{delta})$$

where

$$\text{delta} = \begin{cases} 128 & \text{for 8-bit images} \\ 32768 & \text{for 16-bit images} \\ 0.5 & \text{for floating-point images} \end{cases}$$

Y , Cr , and Cb cover the whole value range.

- **RGB \leftrightarrow HSV (`CV_BGR2HSV`, `CV_RGB2HSV`, `CV_HSV2BGR`, `CV_HSV2RGB`)** In case of 8-bit and 16-bit images, R , G , and B are converted to the floating-point format and scaled to fit the 0 to 1 range.

$$V \leftarrow \max(R, G, B)$$

$$S \leftarrow \begin{cases} \frac{V - \min(R, G, B)}{V} & \text{if } V \neq 0 \\ 0 & \text{otherwise} \end{cases}$$

$$H \leftarrow \begin{cases} 60(G - B) / (V - \min(R, G, B)) & \text{if } V = R \\ 120 + 60(B - R) / (V - \min(R, G, B)) & \text{if } V = G \\ 240 + 60(R - G) / (V - \min(R, G, B)) & \text{if } V = B \end{cases}$$

If $H < 0$ then $H \leftarrow H + 360$. On output $0 \leq V \leq 1$, $0 \leq S \leq 1$, $0 \leq H \leq 360$.

The values are then converted to the destination data type:

- 8-bit images

$$V \leftarrow 255V, S \leftarrow 255S, H \leftarrow H/2(\text{to fit to 0 to 255})$$

- 16-bit images (currently not supported)

$$V < -65535V, S < -65535S, H < -H$$

- 32-bit images H , S , and V are left as is

- **RGB \leftrightarrow HLS (`CV_BGR2HLS`, `CV_RGB2HLS`, `CV_HLS2BGR`, `CV_HLS2RGB`)**. In case of 8-bit and 16-bit images, R , G , and B are converted to the floating-point format and scaled to fit the 0 to 1 range.

$$V_{\max} \leftarrow \max(R, G, B)$$

$$V_{\min} \leftarrow \min(R, G, B)$$

$$L \leftarrow \frac{V_{\max} + V_{\min}}{2}$$

$$S \leftarrow \begin{cases} \frac{V_{\max} - V_{\min}}{V_{\max} + V_{\min}} & \text{if } L < 0.5 \\ \frac{V_{\max} - V_{\min}}{2 - (V_{\max} + V_{\min})} & \text{if } L \geq 0.5 \end{cases}$$

3.3. Miscellaneous Image Transformations

Publicación autorizada con fines académicos e investigativos

En su investigación no olvide referenciar esta tesis

$$H \leftarrow \begin{cases} 60(G - B)/S & \text{if } V_{\max} = R \\ 120 + 60(B - R)/S & \text{if } V_{\max} = G \\ 240 + 60(R - G)/S & \text{if } V_{\max} = B \end{cases}$$

If $H < 0$ then $H \leftarrow H + 360$. On output $0 \leq L \leq 1, 0 \leq S \leq 1, 0 \leq H \leq 360$.

The values are then converted to the destination data type:

- 8-bit images

$$V \leftarrow 255 \cdot V, S \leftarrow 255 \cdot S, H \leftarrow H/2 \text{ (to fit to 0 to 255)}$$

- 16-bit images (currently not supported)

$$V < -65535 \cdot V, S < -65535 \cdot S, H < -H$$

- 32-bit images H, S, V are left as is

- **RGB \leftrightarrow CIE L*a*b*** (**CV_BGR2Lab**, **CV_RGB2Lab**, **CV_Lab2BGR**, **CV_Lab2RGB**). In case of 8-bit and 16-bit images, R, G, and B are converted to the floating-point format and scaled to fit the 0 to 1 range.

$$\begin{bmatrix} X \\ Y \\ Z \end{bmatrix} \leftarrow \begin{bmatrix} 0.412453 & 0.357580 & 0.180423 \\ 0.212671 & 0.715160 & 0.072169 \\ 0.019334 & 0.119193 & 0.950227 \end{bmatrix} \cdot \begin{bmatrix} R \\ G \\ B \end{bmatrix}$$

$$X \leftarrow X/X_n, \text{ where } X_n = 0.950456$$

$$Z \leftarrow Z/Z_n, \text{ where } Z_n = 1.088754$$

$$L \leftarrow \begin{cases} 116 * Y^{1/3} - 16 & \text{for } Y > 0.008856 \\ 903.3 * Y & \text{for } Y \leq 0.008856 \end{cases}$$

$$a \leftarrow 500(f(X) - f(Y)) + \text{delta}$$

$$b \leftarrow 200(f(Y) - f(Z)) + \text{delta}$$

where

$$f(t) = \begin{cases} t^{1/3} & \text{for } t > 0.008856 \\ 7.787t + 16/116 & \text{for } t \leq 0.008856 \end{cases}$$

and

$$\text{delta} = \begin{cases} 128 & \text{for 8-bit images} \\ 0 & \text{for floating-point images} \end{cases}$$

This outputs $0 \leq L \leq 100, -127 \leq a \leq 127, -127 \leq b \leq 127$. The values are then converted to the destination data type:

- 8-bit images



$$L \leftarrow L * 255/100, a \leftarrow a + 128, b \leftarrow b + 128$$

- **16-bit images** (currently not supported)
- **32-bit images** L, a, and b are left as is
- **RGB \leftrightarrow CIE L*u*v*** (**CV_BGR2Luv**, **CV_RGB2Luv**, **CV_Luv2BGR**, **CV_Luv2RGB**). In case of 8-bit and 16-bit images, R, G, and B are converted to the floating-point format and scaled to fit 0 to 1 range.

$$\begin{bmatrix} X \\ Y \\ Z \end{bmatrix} \leftarrow \begin{bmatrix} 0.412453 & 0.357580 & 0.180423 \\ 0.212671 & 0.715160 & 0.072169 \\ 0.019334 & 0.119193 & 0.950227 \end{bmatrix} \cdot \begin{bmatrix} R \\ G \\ B \end{bmatrix}$$

$$L \leftarrow \begin{cases} 116Y^{1/3} & \text{for } Y > 0.008856 \\ 903.3Y & \text{for } Y \leq 0.008856 \end{cases}$$

$$u' \leftarrow 4 * X / (X + 15 * Y + 3Z)$$

$$v' \leftarrow 9 * Y / (X + 15 * Y + 3Z)$$

$$u \leftarrow 13 * L * (u' - u_n) \quad \text{where } u_n = 0.19793943$$

$$v \leftarrow 13 * L * (v' - v_n) \quad \text{where } v_n = 0.46831096$$

This outputs $0 \leq L \leq 100, -134 \leq u \leq 220, -140 \leq v \leq 122$.

The values are then converted to the destination data type:

- 8-bit images

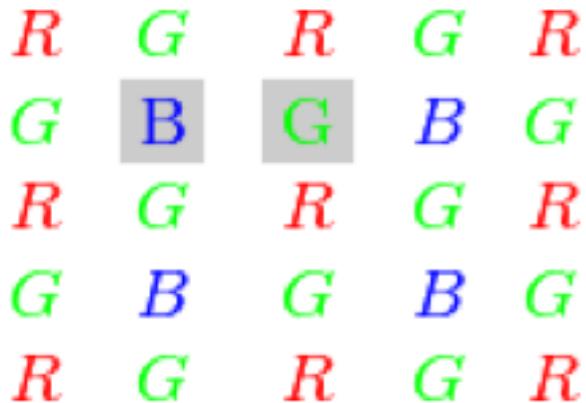
$$L \leftarrow 255/100L, u \leftarrow 255/354(u + 134), v \leftarrow 255/256(v + 140)$$

- **16-bit images** (currently not supported)

- **32-bit images** L, u, and v are left as is

The above formulae for converting RGB to/from various color spaces have been taken from multiple sources on the web, primarily from the Charles Poynton site <http://www.poynton.com/ColorFAQ.html>

- **Bayer \rightarrow RGB** (**CV_BayerBG2BGR**, **CV_BayerGB2BGR**, **CV_BayerRG2BGR**, **CV_BayerGR2BGR**, **CV_BayerBG2RGB**, **CV_BayerGB2RGB**, **CV_BayerRG2RGB**, **CV_BayerGR2RGB**). The Bayer pattern is widely used in CCD and CMOS cameras. It enables you to get color pictures from a single plane where R,G, and B pixels (sensors of a particular component) are interleaved as follows:



The output RGB components of a pixel are interpolated from 1, 2, or 4 neighbors of the pixel having the same color. There are several modifications of the above pattern that can be achieved by shifting the pattern one pixel left and/or one pixel up. The two letters C_1 and C_2 in the conversion constants CV_Bayer C_1C_2 2BGR and CV_Bayer C_1C_2 2RGB indicate the particular pattern type. These are components from the second row, second and third columns, respectively. For example, the above pattern has a very popular “BG” type.

distanceTransform

Calculates the distance to the closest zero pixel for each pixel of the source image.

C++: void **distanceTransform**(InputArray src, OutputArray dst, int **distanceType**, int **maskSize**)

C++: void **distanceTransform**(InputArray src, OutputArray dst, OutputArray labels, int **distanceType**, int **maskSize**, int **labelType**=DIST_LABEL_CCOMP)

Python: cv2.**distanceTransform**(src, distanceType, maskSize[, dst]) → dst

C: void **cvDistTransform**(const CvArr* src, CvArr* dst, int **distance_type**=CV_DIST_L2, int **mask_size**=3, const float* **mask**=NULL, CvArr* labels=NULL, int **label_Type**=CV_DIST_LABEL_CCOMP)

Python: cv.**DistTransform**(src, dst, distance_type=CV_DIST_L2, mask_size=3, mask=None, labels=None) → None

Parameters

src – 8-bit, single-channel (binary) source image.

dst – Output image with calculated distances. It is a 32-bit floating-point, single-channel image of the same size as **src**.

distanceType – Type of distance. It can be CV_DIST_L1, CV_DIST_L2, or CV_DIST_C.

maskSize – Size of the distance transform mask. It can be 3, 5, or CV_DIST_MASK_PRECISE (the latter option is only supported by the first function). In case of the CV_DIST_L1 or CV_DIST_C distance type, the parameter is forced to 3 because a 3×3 mask gives the same result as 5×5 or any larger aperture.

labels – Optional output 2D array of labels (the discrete Voronoi diagram). It has the type CV_32SC1 and the same size as **src**. See the details below.

labelType – Type of the label array to build. If **labelType**==DIST_LABEL_CCOMP then each connected component of zeros in **src** (as well as all the non-zero pixels closest to the connected component) will be assigned the same label. If **labelType**==DIST_LABEL_PIXEL then each zero pixel (and all the non-zero pixels closest to it) gets its own label.

ANEXO L: Manual de array de numPy

This statement will allow us to access NumPy objects using `np.X` instead of `numpy.X`. It is also possible to import NumPy directly into the current namespace so that we don't have to use dot notation at all, but rather simply call the functions as if they were built-in:

```
>>> from numpy import *
```

However, this strategy is usually frowned upon in Python programming because it starts to remove some of the nice organization that modules provide. For the remainder of this tutorial, we will assume that the `import numpy as np` has been used.

Arrays

The central feature of NumPy is the `array` object class. Arrays are similar to lists in Python, except that every element of an array must be of the same type, typically a numeric type like `float` or `int`. Arrays make operations with large amounts of numeric data very fast and are generally much more efficient than lists.

An array can be created from a list:

```
>>> a = np.array([1, 4, 5, 8], float)
>>> a
array([ 1.,  4.,  5.,  8.])
>>> type(a)
<type 'numpy.ndarray'>
```

Here, the function `array` takes two arguments: the list to be converted into the array and the type of each member of the list. Array elements are accessed, sliced, and manipulated just like lists:

```
>>> a[:2]
array([ 1.,  4.])
>>> a[3]
8.0
>>> a[0] = 5.
>>> a
array([ 5.,  4.,  5.,  8.])
```

Arrays can be multidimensional. Unlike lists, different axes are accessed using commas inside bracket notation. Here is an example with a two-dimensional array (e.g., a matrix):

```
>>> a = np.array([[1, 2, 3], [4, 5, 6]], float)
>>> a
array([[ 1.,  2.,  3.],
       [ 4.,  5.,  6.]])
>>> a[0,0]
1.0
>>> a[0,1]
2.0
```

Array slicing works with multiple dimensions in the same way as usual, applying each slice specification as a filter to a specified dimension. Use of a single ":" in a dimension indicates the use of everything along that dimension:

```
>>> a = np.array([[1, 2, 3], [4, 5, 6]], float)
>>> a[1,:]
array([ 4.,  5.,  6.])
>>> a[:,2]
array([ 3.,  6.])
>>> a[-1:,-2:]
array([[ 5.,  6.]])
```

The `shape` property of an array returns a tuple with the size of each array dimension:

```
>>> a.shape
(2, 3)
```

The `dtype` property tells you what type of values are stored by the array:

```
>>> a.dtype
dtype('float64')
```

Here, `float64` is a numeric type that NumPy uses to store double-precision (8-byte) real numbers, similar to the `float` type in Python.

When used with an array, the `len` function returns the length of the first axis:

```
>>> a = np.array([[1, 2, 3], [4, 5, 6]], float)
>>> len(a)
2
```

The `in` statement can be used to test if values are present in an array:

```
>>> a = np.array([[1, 2, 3], [4, 5, 6]], float)
>>> 2 in a
True
>>> 0 in a
False
```

Arrays can be reshaped using tuples that specify new dimensions. In the following example, we turn a ten-element one-dimensional array into a two-dimensional one whose first axis has five elements and whose second axis has two elements:

```
>>> a = np.array(range(10), float)
>>> a
array([ 0.,  1.,  2.,  3.,  4.,  5.,  6.,  7.,  8.,  9.])
>>> a = a.reshape((5, 2))
>>> a
array([[ 0.,  1.],
       [ 2.,  3.],
       [ 4.,  5.],
```

```
[ 6.,  7.],  
[ 8.,  9.])  
>>> a.shape  
(5, 2)
```

Notice that the `reshape` function creates a new array and does not itself modify the original array.

Keep in mind that Python's name-binding approach still applies to arrays. The `copy` function can be used to create a new, separate copy of an array in memory if needed:

```
>>> a = np.array([1, 2, 3], float)  
>>> b = a  
>>> c = a.copy()  
>>> a[0] = 0  
>>> a  
array([0.,  2.,  3.])  
>>> b  
array([0.,  2.,  3.])  
>>> c  
array([1.,  2.,  3.])
```

Lists can also be created from arrays:

```
>>> a = np.array([1, 2, 3], float)  
>>> a.tolist()  
[1.0, 2.0, 3.0]  
>>> list(a)  
[1.0, 2.0, 3.0]
```

One can convert the raw data in an array to a binary string (i.e., not in human-readable form) using the `tostring` function. The `fromstring` function then allows an array to be created from this data later on. These routines are sometimes convenient for saving large amount of array data in files that can be read later on:

```
>>> a = array([1, 2, 3], float)  
>>> s = a.tostring()  
>>> s  
'\x00\x00\x00\x00\x00\x00\xf0?\x00\x00\x00\x00\x00\x00@\x00\x00\x00\x00  
\x00\x00\x08@'  

```

One can fill an array with a single value:

```
>>> a = array([1, 2, 3], float)  
>>> a  
array([ 1.,  2.,  3.])  
>>> a.fill(0)  
>>> a  
array([ 0.,  0.,  0.])
```

Transposed versions of arrays can also be generated, which will create a new array with the final two axes switched:

```
>>> a = np.array(range(6), float).reshape((2, 3))
>>> a
array([[ 0.,  1.,  2.],
       [ 3.,  4.,  5.]])
>>> a.transpose()
array([[ 0.,  3.],
       [ 1.,  4.],
       [ 2.,  5.]])
```

One-dimensional versions of multi-dimensional arrays can be generated with flatten:

```
>>> a = np.array([[1, 2, 3], [4, 5, 6]], float)
>>> a
array([[ 1.,  2.,  3.],
       [ 4.,  5.,  6.]])
>>> a.flatten()
array([ 1.,  2.,  3.,  4.,  5.,  6.])
```

Two or more arrays can be concatenated together using the concatenate function with a tuple of the arrays to be joined:

```
>>> a = np.array([1,2], float)
>>> b = np.array([3,4,5,6], float)
>>> c = np.array([7,8,9], float)
>>> np.concatenate((a, b, c))
array([1.,  2.,  3.,  4.,  5.,  6.,  7.,  8.,  9.])
```

If an array has more than one dimension, it is possible to specify the axis along which multiple arrays are concatenated. By default (without specifying the axis), NumPy concatenates along the first dimension:

```
>>> a = np.array([[1, 2], [3, 4]], float)
>>> b = np.array([[5, 6], [7,8]], float)
>>> np.concatenate((a,b))
array([[ 1.,  2.],
       [ 3.,  4.],
       [ 5.,  6.],
       [ 7.,  8.]])
>>> np.concatenate((a,b), axis=0)
array([[ 1.,  2.],
       [ 3.,  4.],
       [ 5.,  6.],
       [ 7.,  8.]])
>>> np.concatenate((a,b), axis=1)
array([[ 1.,  2.,  5.,  6.],
       [ 3.,  4.,  7.,  8.]])
```

Finally, the dimensionality of an array can be increased using the newaxis constant in bracket notation:

```
>>> a = np.array([1, 2, 3], float)
>>> a
array([1., 2., 3.])
>>> a[:,np.newaxis]
array([[ 1.],
       [ 2.],
       [ 3.]])
>>> a[:,np.newaxis].shape
(3,1)
>>> b[np.newaxis,:,:]
array([[ 1., 2., 3.]])
>>> b[np.newaxis,:,:].shape
(1,3)
```

Notice here that in each case the new array has two dimensions; the one created by `newaxis` has a length of one. The `newaxis` approach is convenient for generating the proper-dimensioned arrays for vector and matrix mathematics.

Other ways to create arrays

The `arange` function is similar to the `range` function but returns an array:

```
>>> np.arange(5, dtype=float)
array([ 0., 1., 2., 3., 4.])
>>> np.arange(1, 6, 2, dtype=int)
array([1, 3, 5])
```

The functions `zeros` and `ones` create new arrays of specified dimensions filled with these values. These are perhaps the most commonly used functions to create new arrays:

```
>>> np.ones((2,3), dtype=float)
array([[ 1., 1., 1.],
       [ 1., 1., 1.]])
>>> np.zeros(7, dtype=int)
array([0, 0, 0, 0, 0, 0, 0])
```

The `zeros_like` and `ones_like` functions create a new array with the same dimensions and type of an existing one:

```
>>> a = np.array([[1, 2, 3], [4, 5, 6]], float)
>>> np.zeros_like(a)
array([[ 0., 0., 0.],
       [ 0., 0., 0.]])
>>> np.ones_like(a)
array([[ 1., 1., 1.],
       [ 1., 1., 1.]])
```

There are also a number of functions for creating special matrices (2D arrays). To create an identity matrix of a given size,

```
>>> np.identity(4, dtype=float)
```

ANEXO M: Manual de medianBlur de OpenCV



getStructuringElement

Returns a structuring element of the specified size and shape for morphological operations.

C++: Mat **getStructuringElement** (int **shape**, Size **ksize**, Point **anchor**=Point(-1,-1))

Python: cv2.**getStructuringElement**(shape, ksize[, anchor]) → retval

C: IplConvKernel* **cvCreateStructuringElementEx**(int **cols**, int **rows**, int **anchor_x**, int **anchor_y**, int **shape**, int* **values**=NULL)

Python: cv.**CreateStructuringElementEx**(cols, rows, anchorX, anchorY, shape, values=None) → kernel

Parameters

shape – Element shape that could be one of the following:

- **MORPH_RECT** - a rectangular structuring element:

$$E_{ij} = 1$$

- **MORPH_ELLIPSE** - an elliptic structuring element, that is, a filled ellipse inscribed into the rectangle `Rect(0, 0, esize.width, 0.esize.height)`

- **MORPH_CROSS** - a cross-shaped structuring element:

$$E_{ij} = \begin{cases} 1 & \text{if } i=\text{anchor.y} \text{ or } j=\text{anchor.x} \\ 0 & \text{otherwise} \end{cases}$$

- **CV_SHAPE_CUSTOM** - custom structuring element (OpenCV 1.x API)

ksize – Size of the structuring element.

cols – Width of the structuring element

rows – Height of the structuring element

anchor – Anchor position within the element. The default value $(-1, -1)$ means that the anchor is at the center. Note that only the shape of a cross-shaped element depends on the anchor position. In other cases the anchor just regulates how much the result of the morphological operation is shifted.

anchor_x – x-coordinate of the anchor

anchor_y – y-coordinate of the anchor

values – integer array of `cols '*' 'rows` elements that specifies the custom shape of the structuring element, when `shape=CV_SHAPE_CUSTOM`.

The function constructs and returns the structuring element that can be further passed to `createMorphologyFilter()`, `erode()`, `dilate()` or `morphologyEx()`. But you can also construct an arbitrary binary mask yourself and use it as the structuring element.

Note: When using OpenCV 1.x C API, the created structuring element `IplConvKernel*` element must be released in the end using `cvReleaseStructuringElement(&element)`.

medianBlur

Blurs an image using the median filter.

C++: void **medianBlur** (InputArray **src**, OutputArray **dst**, int **ksize**)

3.1. Image Filtering



Python: `cv2.medianBlur(src, ksize[, dst]) → dst`

Parameters

src – input 1-, 3-, or 4-channel image; when **ksize** is 3 or 5, the image depth should be `CV_8U`, `CV_16U`, or `CV_32F`, for larger aperture sizes, it can only be `CV_8U`.

dst – destination array of the same size and type as **src**.

ksize – aperture linear size; it must be odd and greater than 1, for example: 3, 5, 7 ...

The function smoothes an image using the median filter with the $ksize \times ksize$ aperture. Each channel of a multi-channel image is processed independently. In-place operation is supported.

See Also:

`bilateralFilter()`, `blur()`, `boxFilter()`, `GaussianBlur()`

morphologyEx

Performs advanced morphological transformations.

C++: `void morphologyEx(InputArray src, OutputArray dst, int op, InputArray kernel, Point anchor=Point(-1,-1), int iterations=1, int borderType=BORDER_CONSTANT, const Scalar& borderValue=morphologyDefaultBorderValue())`

Python: `cv2.morphologyEx(src, op, kernel[, dst[, anchor[, iterations[, borderType[, borderValue]]]]]) → dst`

C: `void cvMorphologyEx(const CvArr* src, CvArr* dst, CvArr* temp, IplConvKernel* element, int operation, int iterations=1)`

Python: `cv.MorphologyEx(src, dst, temp, element, operation, iterations=1) → None`

Parameters

src – Source image. The number of channels can be arbitrary. The depth should be one of `CV_8U`, `CV_16U`, `CV_16S`, `CV_32F` or `'CV_64F'`.

dst – Destination image of the same size and type as **src**.

element – Structuring element.

op – Type of a morphological operation that can be one of the following:

- **MORPH_OPEN** - an opening operation
- **MORPH_CLOSE** - a closing operation
- **MORPH_GRADIENT** - a morphological gradient
- **MORPH_TOPHAT** - “top hat”
- **MORPH_BLACKHAT** - “black hat”

iterations – Number of times erosion and dilation are applied.

borderType – Pixel extrapolation method. See `borderInterpolate()` for details.

borderValue – Border value in case of a constant border. The default value has a special meaning. See `createMorphologyFilter()` for details.

The function can perform advanced morphological transformations using an erosion and dilation as basic operations.

Opening operation:

```
dst = open(src, element) = dilate(erode(src, element))
```

ANEXO N: Manual de inRange de OpenCV



dst – output array of the same size and type as **src**.

flags – operation flags.

`idct(src, dst, flags)` is equivalent to `dct(src, dst, flags | DCT_INVERSE)`.

See Also:

`dct()`, `dft()`, `idft()`, `getOptimalDFTSize()`

idft

Calculates the inverse Discrete Fourier Transform of a 1D or 2D array.

C++: `void idft(InputArray src, OutputArray dst, int flags=0, int nonzeroRows=0)`

Python: `cv2.idft(src[, dst[, flags[, nonzeroRows]]]) → dst`

Parameters

src – input floating-point real or complex array.

dst – output array whose size and type depend on the **flags**.

flags – operation flags (see `dft()`).

nonzeroRows – number of **dst** rows to process; the rest of the rows have undefined content (see the convolution sample in `dft()` description).

`idft(src, dst, flags)` is equivalent to `dft(src, dst, flags | DFT_INVERSE)`.

See `dft()` for details.

Note: None of `dft` and `idft` scales the result by default. So, you should pass `DFT_SCALE` to one of `dft` or `idft` explicitly to make these transforms mutually inverse.

See Also:

`dft()`, `dct()`, `idct()`, `mulSpectrums()`, `getOptimalDFTSize()`

inRange

Checks if array elements lie between the elements of two other arrays.

C++: `void inRange(InputArray src, InputArray lowerb, InputArray upperb, OutputArray dst)`

Python: `cv2.inRange(src, lowerb, upperb[, dst]) → dst`

C: `void cvInRange(const CvArr* src, const CvArr* lower, const CvArr* upper, CvArr* dst)`

C: `void cvInRangeS(const CvArr* src, CvScalar lower, CvScalar upper, CvArr* dst)`

Python: `cv.InRange(src, lower, upper, dst) → None`

Python: `cv.InRangeS(src, lower, upper, dst) → None`

Parameters

src – first input array.

lowerb – inclusive lower boundary array or a scalar.

upperb – inclusive upper boundary array or a scalar.

dst – output array of the same size as **src** and CV_8U type.

The function checks the range as follows:

- For every element of a single-channel input array:

$$\text{dst}(I) = \text{lowerb}(I)_0 \leq \text{src}(I)_0 \leq \text{upperb}(I)_0$$

- For two-channel arrays:

$$\text{dst}(I) = \text{lowerb}(I)_0 \leq \text{src}(I)_0 \leq \text{upperb}(I)_0 \wedge \text{lowerb}(I)_1 \leq \text{src}(I)_1 \leq \text{upperb}(I)_1$$

- and so forth.

That is, **dst** (**I**) is set to 255 (all 1 -bits) if **src** (**I**) is within the specified 1D, 2D, 3D, ... box and 0 otherwise.

When the lower and/or upper boundary parameters are scalars, the indexes (**I**) at **lowerb** and **upperb** in the above formulas should be omitted.

invert

Finds the inverse or pseudo-inverse of a matrix.

C++: double **invert**(InputArray **src**, OutputArray **dst**, int **flags**=DECOMP_LU)

Python: cv2.**invert**(src[, dst[, flags]]) → retval, dst

C: double **cvInvert**(const CvArr* **src**, CvArr* **dst**, int **method**=CV_LU)

Python: cv.**Invert**(src, dst, method=CV_LU) → float

Parameters

src – input floating-point M × N matrix.

dst – output matrix of N × M size and the same type as **src**.

flags – inversion method :

- **DECOMP_LU** Gaussian elimination with the optimal pivot element chosen.
- **DECOMP_SVD** singular value decomposition (SVD) method.
- **DECOMP_CHOLESKY** Cholesky decomposition; the matrix must be symmetrical and positively defined.

The function **invert** inverts the matrix **src** and stores the result in **dst**. When the matrix **src** is singular or non-square, the function calculates the pseudo-inverse matrix (the **dst** matrix) so that **norm(src*dst - I)** is minimal, where **I** is an identity matrix.

In case of the **DECOMP_LU** method, the function returns non-zero value if the inverse has been successfully calculated and 0 if **src** is singular.

In case of the **DECOMP_SVD** method, the function returns the inverse condition number of **src** (the ratio of the smallest singular value to the largest singular value) and 0 if **src** is singular. The SVD method calculates a pseudo-inverse matrix if **src** is singular.

Similarly to **DECOMP_LU**, the method **DECOMP_CHOLESKY** works only with non-singular square matrices that should also be symmetrical and positively defined. In this case, the function stores the inverted matrix in **dst** and returns non-zero. Otherwise, it returns 0.

See Also:

solve(), **SVD**

ANEXO O: Manual de HoughCircles de OpenCV

image – Input 8-bit or floating-point 32-bit, single-channel image.

eig_image – The parameter is ignored.

temp_image – The parameter is ignored.

corners – Output vector of detected corners.

maxCorners – Maximum number of corners to return. If there are more corners than are found, the strongest of them is returned.

qualityLevel – Parameter characterizing the minimal accepted quality of image corners. The parameter value is multiplied by the best corner quality measure, which is the minimal eigenvalue (see [cornerMinEigenVal\(\)](#)) or the Harris function response (see [cornerHarris\(\)](#)). The corners with the quality measure less than the product are rejected. For example, if the best corner has the quality measure = 1500, and the **qualityLevel**=0.01, then all the corners with the quality measure less than 15 are rejected.

minDistance – Minimum possible Euclidean distance between the returned corners.

mask – Optional region of interest. If the image is not empty (it needs to have the type CV_8UC1 and the same size as **image**), it specifies the region in which the corners are detected.

blockSize – Size of an average block for computing a derivative covariation matrix over each pixel neighborhood. See [cornerEigenValsAndVecs\(\)](#).

useHarrisDetector – Parameter indicating whether to use a Harris detector (see [cornerHarris\(\)](#) or [cornerMinEigenVal\(\)](#)).

k – Free parameter of the Harris detector.

The function finds the most prominent corners in the image or in the specified image region, as described in [Shi94]:

1. Function calculates the corner quality measure at every source image pixel using the [cornerMinEigenVal\(\)](#) or [cornerHarris\(\)](#).
2. Function performs a non-maximum suppression (the local maximums in 3×3 neighborhood are retained).
3. The corners with the minimal eigenvalue less than $\text{qualityLevel} \cdot \max_{x,y} \text{qualityMeasureMap}(x, y)$ are rejected.
4. The remaining corners are sorted by the quality measure in the descending order.
5. Function throws away each corner for which there is a stronger corner at a distance less than **maxDistance**.

The function can be used to initialize a point-based tracker of an object.

Note: If the function is called with different values A and B of the parameter **qualityLevel**, and A > {B}, the vector of returned corners with **qualityLevel**=A will be the prefix of the output vector with **qualityLevel**=B.

See Also:

[cornerMinEigenVal\(\)](#), [cornerHarris\(\)](#), [calcOpticalFlowPyrLK\(\)](#), [estimateRigidTransform\(\)](#),

HoughCircles

Finds circles in a grayscale image using the Hough transform.

C++: void **HoughCircles**(InputArray **image**, OutputArray **circles**, int **method**, double **dp**, double **minDist**, double **param1**=100, double **param2**=100, int **minRadius**=0, int **maxRadius**=0)



C: CvSeq* **cvHoughCircles**(CvArr* **image**, void* **circle_storage**, int **method**, double **dp**, double **min_dist**, double **param1**=100, double **param2**=100, int **min_radius**=0, int **max_radius**=0)

Python: cv2.HoughCircles(**image**, **method**, **dp**, **minDist**[, **circles**[, **param1**[, **param2**[, **minRadius**[, **maxRadius**]]]]) → **circles**

Parameters

image – 8-bit, single-channel, grayscale input image.

circles – Output vector of found circles. Each vector is encoded as a 3-element floating-point vector ($x, y, radius$).

circle_storage – In C function this is a memory storage that will contain the output sequence of found circles.

method – Detection method to use. Currently, the only implemented method is CV_HOUGH_GRADIENT , which is basically 2HT , described in [Yuen90].

dp – Inverse ratio of the accumulator resolution to the image resolution. For example, if **dp**=1 , the accumulator has the same resolution as the input image. If **dp**=2 , the accumulator has half as big width and height.

minDist – Minimum distance between the centers of the detected circles. If the parameter is too small, multiple neighbor circles may be falsely detected in addition to a true one. If it is too large, some circles may be missed.

param1 – First method-specific parameter. In case of CV_HOUGH_GRADIENT , it is the higher threshold of the two passed to the [Canny\(\)](#) edge detector (the lower one is twice smaller).

param2 – Second method-specific parameter. In case of CV_HOUGH_GRADIENT , it is the accumulator threshold for the circle centers at the detection stage. The smaller it is, the more false circles may be detected. Circles, corresponding to the larger accumulator values, will be returned first.

minRadius – Minimum circle radius.

maxRadius – Maximum circle radius.

The function finds circles in a grayscale image using a modification of the Hough transform.

Example:

```
#include <cv.h>
#include <highgui.h>
#include <math.h>

using namespace cv;

int main(int argc, char** argv)
{
    Mat img, gray;
    if( argc != 2 && !(img=imread(argv[1], 1)).data)
        return -1;
    cvtColor(img, gray, CV_BGR2GRAY);
    // smooth it, otherwise a lot of false circles may be detected
    GaussianBlur( gray, gray, Size(9, 9), 2, 2 );
    vector<Vec3f> circles;
    HoughCircles(gray, circles, CV_HOUGH_GRADIENT,
                 2, gray->rows/4, 200, 100 );
    for( size_t i = 0; i < circles.size(); i++ )
    {
```

```
Point center(cvRound(circles[i][0]), cvRound(circles[i][1]));
int radius = cvRound(circles[i][2]);
// draw the circle center
circle( img, center, 3, Scalar(0,255,0), -1, 8, 0 );
// draw the circle outline
circle( img, center, radius, Scalar(0,0,255), 3, 8, 0 );
}
namedWindow( "circles", 1 );
imshow( "circles", img );
return 0;
}
```

Note: Usually the function detects the centers of circles well. However, it may fail to find correct radii. You can assist to the function by specifying the radius range (`minRadius` and `maxRadius`) if you know it. Or, you may ignore the returned radius, use only the center, and find the correct radius using an additional procedure.

See Also:

[fitEllipse\(\)](#), [minEnclosingCircle\(\)](#)

Note:

- An example using the Hough circle detector can be found at [opencv_source_code/samples/cpp/houghcircles.cpp](#)

HoughLines

Finds lines in a binary image using the standard Hough transform.

C++: void **HoughLines**(InputArray **image**, OutputArray **lines**, double **rho**, double **theta**, int **threshold**, double **srn**=0, double **stn**=0)

Python: `cv2.HoughLines(image, rho, theta, threshold[, lines[, srn[, stn]]])` → lines

C: CvSeq* **cvHoughLines2**(CvArr* **image**, void* **line_storage**, int **method**, double **rho**, double **theta**, int **threshold**, double **param1**=0, double **param2**=0)

Python: `cv.HoughLines2(image, storage, method, rho, theta, threshold, param1=0, param2=0)` → lines

Parameters

image – 8-bit, single-channel binary source image. The image may be modified by the function.

lines – Output vector of lines. Each line is represented by a two-element vector (ρ, θ) . ρ is the distance from the coordinate origin $(0,0)$ (top-left corner of the image). θ is the line rotation angle in radians ($0 \sim$ vertical line, $\pi/2 \sim$ horizontal line).

rho – Distance resolution of the accumulator in pixels.

theta – Angle resolution of the accumulator in radians.

threshold – Accumulator threshold parameter. Only those lines are returned that get enough votes ($> \text{threshold}$).

srn – For the multi-scale Hough transform, it is a divisor for the distance resolution **rho**. The coarse accumulator distance resolution is **rho** and the accurate accumulator resolution is **rho/srn**. If both **srn**=0 and **stn**=0, the classical Hough transform is used. Otherwise, both these parameters should be positive.

ANEXO P: Manual de fuzzy.storage de Pyfuzzy

99 Module fuzzy.storage.fcl.Reader

Load a fuzzy system from FCL file, stream or string.

99.1 Variables

Name	Description
revision	Value: '\$Id: Reader.py,v 1.6 2013-01-09 20:10:19 rliebscher Exp \$'
package	Value: 'fuzzy.storage.fcl'

99.2 Class Reader

object —

fuzzy.storage.fcl.Reader.Reader

Parses a FCL file to a fuzzy.System.System instance

99.2.1 Methods

load_from_file(self, filename)

Load a fuzzy system from FCL file.

load_from_stream(self, stream)

Load a fuzzy system from FCL stream.

load_from_string(self, str)

Load a fuzzy system from FCL string.

Inherited from object

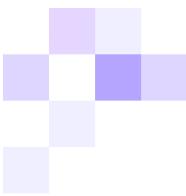
__delattr__(self), __format__(self), __getattribute__(self), __hash__(self), __init__(self),
__new__(self), __reduce__(self), __reduce_ex__(self), __repr__(self), __setattr__(self),
__sizeof__(self), __str__(self), __subclasshook__(self)

99.2.2 Properties

Name	Description
<i>Inherited from object</i> <u>class</u>	



ANEXO Q: Estructura de un archivo FCL



Free Fuzzy Logic Library

[Home](#) | [License](#) | [History](#) | [Class Hierarchy](#) | [API](#) | [Code Details](#) | [FCL](#) | [Developers](#) | [Downloads](#) |

SOURCEFORGE.NET

FCL

FCL stands for Fuzzy Control Language, which is a standard for Fuzzy Control Programming published by the [International Electrotechnical Commission](#) (IEC). The specifications for the FCL syntax can be found in IEC document 61131-7. Unfortunately, this standard is not freely available and must be purchased (see [www.iec.ch](#) or [www.ansi.org](#)). **However**, the [Draft 1.0 version from 1997](#) is available and I have not noticed any significant differences between the draft and the final version.

FFLL is able to load files that adhere to the IEC 61131-7 standard.

Here's a generic FCL outline:

General FCL Notes:

- (* and *) are used as comment delimiters
- The **RANGE** comment after variable declarations is NOT part of the standard FCL. It was added so FFLL will know the variable's range
- FCL refers to the individual sets in a variable as **TERMs**. In other fuzzy logic literature you will see these referred to as sets or fuzzy subset
- The points that define a **TERM** are declared in (x, y) pairs. The x value is within the **RANGE** of the variable and the y value is between 0 and 1
- The **RULEBLOCK** name is not used in FFLL, but it's included to comply with FCL standards
- Rule conditions are variable term's ANDed together
- The rule conclusion is a term from the output variable
- You can find more details on FFLL and FCLs compliance by checking out the [FCL production rules](#).

```
FUNCTION_BLOCK

VAR_INPUT
    <variable name> REAL; (* RANGE(<variable minimum value> .. <variable maximum value>) *)
END_VAR

VAR_OUTPUT
    <variable name> REAL; (* RANGE(<variable minimum value> .. <variable maximum value>) *)
END_VAR

FUZZIFY <variable name>
    TERM <term (or set) name> := <points that make up the term> ;
END_FUZZIFY

DEFUZZIFY valve
    METHOD: <defuzzification method>;
END_DEFUZZIFY

RULEBLOCK <ruleblock name>
    <operator>:<algorithm>;
    ACCUM:<accumulation method>;
    RULE <rule number>: IF <condition> THEN <conclusion>;
END_RULEBLOCK

END_FUNCTION_BLOCK
```

Here's an example of a real FCL file that's used to calculate the aggressiveness of an AI controlled character based on its health and its enemy's health. The model has two input variables: *Our_Health* and *Enemy_Health* and one output variable: *Aggressiveness*. Note that the sets that comprise the conditional part of the rules (specified in the RULEBLOCK section) are ANDed together in the order that the variables they belong to are declared in the FCL file.

```
FUNCTION_BLOCK

VAR_INPUT
    Our_Health      REAL; (* RANGE(0 .. 100) *)
    Enemy_Health    REAL; (* RANGE(0 .. 100) *)
END_VAR

VAR_OUTPUT
    Aggressiveness REAL; (* RANGE(0 .. 4) *)
END_VAR
```

```

FUZZIFY Our_Health
  TERM Near_Death := (0, 0) (0, 1) (50, 0) ;
  TERM Good := (14, 0) (50, 1) (83, 0) ;
  TERM Excellent := (50, 0) (100, 1) (100, 0) ;
END_FUZZIFY

FUZZIFY Enemy_Health
  TERM Near_Death := (0, 0) (0, 1) (50, 0) ;
  TERM Good := (14, 0) (50, 1) (83, 0) ;
  TERM Excellent := (50, 0) (100, 1) (100, 0) ;
END_FUZZIFY

FUZZIFY Aggressiveness
  TERM Run_Away := 1 ;
  TERM Fight_Defensively := 2 ;
  TERM All_Out_Attack := 3 ;
END_FUZZIFY

DEFUZZIFY valve
  METHOD: MoM;
END_DEFUZZIFY

RULEBLOCK first
AND:MIN;
ACCU:MAX;
RULE 0: IF (Our_Health IS Near_Death) AND (Enemy_Health IS Near_Death) THEN (Aggressiveness IS Fight_Defensively);
RULE 1: IF (Our_Health IS Near_Death) AND (Enemy_Health IS Good) THEN (Aggressiveness IS Run_Away);
RULE 2: IF (Our_Health IS Near_Death) AND (Enemy_Health IS Excellent) THEN (Aggressiveness IS Run_Away);
RULE 3: IF (Our_Health IS Good) AND (Enemy_Health IS Near_Death) THEN (Aggressiveness IS All_Out_Attack);
RULE 4: IF (Our_Health IS Good) AND (Enemy_Health IS Good) THEN (Aggressiveness IS Fight_Defensively);
RULE 5: IF (Our_Health IS Good) AND (Enemy_Health IS Excellent) THEN (Aggressiveness IS Fight_Defensively);
RULE 6: IF (Our_Health IS Excellent) AND (Enemy_Health IS Near_Death) THEN (Aggressiveness IS All_Out_Attack);
RULE 7: IF (Our_Health IS Excellent) AND (Enemy_Health IS Good) THEN (Aggressiveness IS All_Out_Attack);
RULE 8: IF (Our_Health IS Excellent) AND (Enemy_Health IS Excellent) THEN (Aggressiveness IS Fight_Defensively);

END_RULEBLOCK
END_FUNCTION_BLOCK

```

ANEXO R: Manual de waitKey de OpenCV



waitKey

Waits for a pressed key.

C++: int **waitKey**(int **delay**=0)

Python: cv2.**waitKey**([delay]) → retval

C: int **cvWaitKey**(int **delay**=0)

Python: cv.**WaitKey**(delay=0) → int

Parameters

delay – Delay in milliseconds. 0 is the special value that means “forever”.

The function `waitKey` waits for a key event infinitely (when `delay ≤ 0`) or for `delay` milliseconds, when it is positive. Since the OS has a minimum time between switching threads, the function will not wait exactly `delay` ms, it will wait at least `delay` ms, depending on what else is running on your computer at that time. It returns the code of the pressed key or -1 if no key was pressed before the specified time had elapsed.

Note: This function is the only method in HighGUI that can fetch and handle events, so it needs to be called periodically for normal event processing unless HighGUI is used within an environment that takes care of event processing.

Note: The function only works if there is at least one HighGUI window created and the window is active. If there are several HighGUI windows, any of them can be active.

setOpenGLDrawCallback

Set OpenGL render handler for the specified window.

C++: void **setOpenGLDrawCallback**(const string& **winname**, OpenGLDrawCallback **onOpenGLDraw**,
void* **userdata**=0)

Parameters

winname – Window name

onOpenGLDraw – Draw callback.

userdata – The optional parameter passed to the callback.

setOpenGLContext

Sets the specified window as current OpenGL context.

C++: void **setOpenGLContext**(const string& **winname**)

Parameters

winname – Window name

ANEXO S: Manual de Serial de serial

SHORT INTRODUCTION

2.1 Opening serial ports

Open port 0 at “9600,8,N,1”, no timeout:

```
>>> import serial
>>> ser = serial.Serial(0)      # open first serial port
>>> print ser.portstr        # check which port was really used
>>> ser.write("hello")        # write a string
>>> ser.close()               # close port
```

Open named port at “19200,8,N,1”, 1s timeout:

```
>>> ser = serial.Serial('/dev/ttyS1', 19200, timeout=1)
>>> x = ser.read()            # read one byte
>>> s = ser.read(10)          # read up to ten bytes (timeout)
>>> line = ser.readline()     # read a '\n' terminated line
>>> ser.close()
```

Open second port at “38400,8,E,1”, non blocking HW handshaking:

```
>>> ser = serial.Serial(1, 38400, timeout=0,
...                      parity=serial.PARITY_EVEN, rtscts=1)
>>> s = ser.read(100)         # read up to one hundred bytes
...                      # or as much is in the buffer
```

2.2 Configuring ports later

Get a Serial instance and configure/open it later:

```
>>> ser = serial.Serial()
>>> ser.baudrate = 19200
>>> ser.port = 0
>>> ser
Serial<id=0xa81c10, open=False>(port='COM1', baudrate=19200, bytesize=8, parity='N', stopbits=1, time...
>>> ser.open()
>>> ser.isOpen()
True
>>> ser.close()
>>> ser.isOpen()
False
```

ANEXO T: Manual de write de serial

PYSERIAL API

4.1 Classes

4.1.1 Native ports

```
class serial.Serial
```

```
    __init__(port=None, baudrate=9600, bytesize=EIGHTBITS, parity=PARITY_NONE, stopbits=STOPBITS_ONE, timeout=None, xonxoff=False, rtscts=False, writeTimeout=None, dsrdtr=False, interCharTimeout=None)
```

Parameters

- **port** – Device name or port number number or None.
- **baudrate** – Baud rate such as 9600 or 115200 etc.
- **bytesize** – Number of data bits. Possible values: FIVEBITS, SIXBITS, SEVENBITS, EIGHTBITS
- **parity** – Enable parity checking. Possible values: PARITY_NONE, PARITY_EVEN, PARITY_ODD, PARITY_MARK, PARITY_SPACE
- **stopbits** – Number of stop bits. Possible values: STOPBITS_ONE, STOPBITS_ONE_POINT_FIVE, STOPBITS_TWO
- **timeout** – Set a read timeout value.
- **xonxoff** – Enable software flow control.
- **rtscts** – Enable hardware (RTS/CTS) flow control.
- **dsrdtr** – Enable hardware (DSR/DTR) flow control.
- **writeTimeout** – Set a write timeout value.
- **interCharTimeout** – Inter-character timeout, None to disable (default).

Raises

- **ValueError** – Will be raised when parameter are out of range, e.g. baud rate, data bits.
- **SerialException** – In case the device can not be found or can not be configured.

The port is immediately opened on object creation, when a *port* is given. It is not opened when *port* is None and a successive call to `open()` will be needed.

Possible values for the parameter *port*:



- Number: number of device, numbering starts at zero.
- Device name: depending on operating system. e.g. /dev/ttyUSB0 on GNU/Linux or COM3 on Windows.

The parameter *baudrate* can be one of the standard values: 50, 75, 110, 134, 150, 200, 300, 600, 1200, 1800, 2400, 4800, 9600, 19200, 38400, 57600, 115200. These are well supported on all platforms. Standard values above 115200 such as: 230400, 460800, 500000, 576000, 921600, 1000000, 1152000, 1500000, 2000000, 2500000, 3000000, 3500000, 4000000 also work on many platforms.

Non-standard values are also supported on some platforms (GNU/Linux, MAC OSX >= Tiger, Windows). Though, even on these platforms some serial ports may reject non-standard values.

Possible values for the parameter *timeout*:

- *timeout* = `None`: wait forever
- *timeout* = `0`: non-blocking mode (return immediately on read)
- *timeout* = `x`: set timeout to *x* seconds (float allowed)

Writes are blocking by default, unless *writeTimeout* is set. For possible values refer to the list for *timeout* above.

Note that enabling both flow control methods (*xonxoff* and *rtscts*) together may not be supported. It is common to use one of the methods at once, not both.

dsrdtr is not supported by all platforms (silently ignored). Setting it to `None` has the effect that its state follows *rtscts*.

Also consider using the function `serial_for_url()` instead of creating Serial instances directly. Changed in version 2.5: *dsrdtr* now defaults to `False` (instead of `None`)

open()
Open port.

close()
Close port immediately.

__del__()
Destructor, close port when serial port instance is freed.

The following methods may raise `ValueError` when applied to a closed port.

read(size=1)

Parameters `size` – Number of bytes to read.

Returns Bytes read from the port.

Read `size` bytes from the serial port. If a timeout is set it may return less characters as requested. With no timeout it will block until the requested number of bytes is read. Changed in version 2.5: Returns an instance of `bytes` when available (Python 2.6 and newer) and `str` otherwise.

write(data)

Parameters `data` – Data to send.

Returns Number of bytes written.

Raises `SerialTimeoutException` In case a write timeout is configured for the port and the time is exceeded.

Write the string `data` to the port. Changed in version 2.5: Accepts instances of `bytes` and `bytearray` when available (Python 2.6 and newer) and `str` otherwise. Changed in version 2.5: Write returned `None` in previous versions.

ANEXO U: Manual de destroyAllWindows de OpenCV

destroyWindow

Destroys a window.

C++: void **destroyWindow**(const string& **winname**)

Python: cv2.**destroyWindow**(winname) → None

C: void **cvDestroyWindow**(const char* **name**)

Python: cv.**DestroyWindow**(name) → None

Parameters

winname – Name of the window to be destroyed.

The function `destroyWindow` destroys the window with the given name.

destroyAllWindows

Destroys all of the HighGUI windows.

C++: void **destroyAllWindows**()

Python: cv2.**destroyAllWindows**() → None

C: void **cvDestroyAllWindows**()

Python: cv.**DestroyAllWindows**() → None

The function `destroyAllWindows` destroys all of the opened HighGUI windows.

MoveWindow

Moves window to the specified position

C++: void **moveWindow**(const string& **winname**, int **x**, int **y**)

Python: cv2.**moveWindow**(winname, x, y) → None

C: void **cvMoveWindow**(const char* **name**, int **x**, int **y**)

Python: cv.**MoveWindow**(name, x, y) → None

Parameters

winname – Window name

x – The new x-coordinate of the window

y – The new y-coordinate of the window

ResizeWindow

Resizes window to the specified size

C++: void **resizeWindow**(const string& **winname**, int **width**, int **height**)

Python: cv2.**resizeWindow**(winname, width, height) → None

C: void **cvResizeWindow**(const char* **name**, int **width**, int **height**)

Python: cv.**ResizeWindow**(name, width, height) → None

ANEXO V: Manual de time

15.3. time — Time access and conversions

This module provides various time-related functions. For related functionality, see also the [datetime](#) and [calendar](#) modules.

Although this module is always available, not all functions are available on all platforms. Most of the functions defined in this module call platform C library functions with the same name. It may sometimes be helpful to consult the platform documentation, because the semantics of these functions varies among platforms.

An explanation of some terminology and conventions is in order.

- The *epoch* is the point where the time starts. On January 1st of that year, at 0 hours, the “time since the epoch” is zero. For Unix, the epoch is 1970. To find out what the epoch is, look at `gmtime(0)`.
- The functions in this module do not handle dates and times before the epoch or far in the future. The cut-off point in the future is determined by the C library; for Unix, it is typically in 2038.
- **Year 2000 (Y2K) issues:** Python depends on the platform’s C library, which generally doesn’t have year 2000 issues, since all dates and times are represented internally as seconds since the epoch. Functions accepting a `struct_time` (see below) generally require a 4-digit year. For backward compatibility, 2-digit years are supported if the module variable `accept2dyear` is a non-zero integer; this variable is initialized to `1` unless the environment variable `PYTHONY2K` is set to a non-empty string, in which case it is initialized to `0`. Thus, you can set `PYTHONY2K` to a non-empty string in the environment to require 4-digit years for all year input. When 2-digit years are accepted, they are converted according to the POSIX or X/Open standard: values 69–99 are mapped to 1969–1999, and values 0–68 are mapped to 2000–2068. Values 100–1899 are always illegal. Note that this is new as of Python 1.5.2(a2); earlier versions, up to Python 1.5.1 and 1.5.2a1, would add 1900 to year values below 1900.
- UTC is Coordinated Universal Time (formerly known as Greenwich Mean Time, or GMT). The acronym UTC is not a mistake but a compromise between English and French.
- DST is Daylight Saving Time, an adjustment of the timezone by (usually) one hour during part of the year. DST rules are magic (determined by local law) and can change from year to year. The C library has a table containing the local rules (often it is read from a system file for flexibility) and is the only source of True Wisdom in this respect.
- The precision of the various real-time functions may be less than suggested by the units in which their value or argument is expressed. E.g. on most Unix systems, the clock “ticks” only 50 or 100 times a second.
- On the other hand, the precision of `time()` and `sleep()` is better than their Unix

equivalents: times are expressed as floating point numbers, `time()` returns the most accurate time available (using Unix `gettimeofday()` where available), and `sleep()` will accept a time with a nonzero fraction (Unix `select()` is used to implement this, where available).

- The time value as returned by `gmtime()`, `localtime()`, and `strptime()`, and accepted by `asctime()`, `mktime()` and `strftime()`, may be considered as a sequence of 9 integers. The return values of `gmtime()`, `localtime()`, and `strptime()` also offer attribute names for individual fields.

See `struct_time` for a description of these objects.

Changed in version 2.2: The time value sequence was changed from a tuple to a `struct_time`, with the addition of attribute names for the fields.

- Use the following functions to convert between time representations:

From	To	Use
seconds since the epoch	<code>struct_time</code> in UTC	<code>gmtime()</code>
seconds since the epoch	<code>struct_time</code> in local time	<code>localtime()</code>
<code>struct_time</code> in UTC	seconds since the epoch	<code>calendar.timegm()</code>
<code>struct_time</code> in local time	seconds since the epoch	<code>mktime()</code>

The module defines the following functions and data items:

`time.accept2dyear`

Boolean value indicating whether two-digit year values will be accepted. This is true by default, but will be set to false if the environment variable `PYTHON2K` has been set to a non-empty string. It may also be modified at run time.

`time.altzone`

The offset of the local DST timezone, in seconds west of UTC, if one is defined. This is negative if the local DST timezone is east of UTC (as in Western Europe, including the UK). Only use this if `daylight` is nonzero.

`time.asctime([t])`

Convert a tuple or `struct_time` representing a time as returned by `gmtime()` or `localtime()` to a 24-character string of the following form: 'Sun Jun 20 23:21:05 1993'. If `t` is not provided, the current time as returned by `localtime()` is used. Locale information is not used by `asctime()`.

Note: Unlike the C function of the same name, there is no trailing newline.

Changed in version 2.1: Allowed `t` to be omitted.

`time.clock()`

On Unix, return the current processor time as a floating point number expressed in seconds. The precision, and in fact the very definition of the meaning of "processor

time”, depends on that of the C function of the same name, but in any case, this is the function to use for benchmarking Python or timing algorithms.

On Windows, this function returns wall-clock seconds elapsed since the first call to this function, as a floating point number, based on the Win32 function `QueryPerformanceCounter()`. The resolution is typically better than one microsecond.

`time.ctime([secs])`

Convert a time expressed in seconds since the epoch to a string representing local time. If `secs` is not provided or `None`, the current time as returned by `time()` is used. `ctime(secs)` is equivalent to `asctime(localtime(secs))`. Locale information is not used by `ctime()`.

Changed in version 2.1: Allowed `secs` to be omitted.

Changed in version 2.4: If `secs` is `None`, the current time is used.

`time.daylight`

Nonzero if a DST timezone is defined.

`time.gmtime([secs])`

Convert a time expressed in seconds since the epoch to a `struct_time` in UTC in which the `dst` flag is always zero. If `secs` is not provided or `None`, the current time as returned by `time()` is used. Fractions of a second are ignored. See above for a description of the `struct_time` object. See `calendar.timegm()` for the inverse of this function.

Changed in version 2.1: Allowed `secs` to be omitted.

Changed in version 2.4: If `secs` is `None`, the current time is used.

`time.localtime([secs])`

Like `gmtime()` but converts to local time. If `secs` is not provided or `None`, the current time as returned by `time()` is used. The `dst` flag is set to `1` when DST applies to the given time.

Changed in version 2.1: Allowed `secs` to be omitted.

Changed in version 2.4: If `secs` is `None`, the current time is used.

`time.mktime(t)`

This is the inverse function of `localtime()`. Its argument is the `struct_time` or full 9-tuple (since the `dst` flag is needed; use `-1` as the `dst` flag if it is unknown) which expresses the time in *local* time, not UTC. It returns a floating point number, for compatibility with `time()`. If the input value cannot be represented as a valid time, either `OverflowError` or `ValueError` will be raised (which depends on whether the invalid value is caught by Python or the underlying C libraries). The earliest date for which it can generate a time is platform-dependent.

time.sleep(secs)

Suspend execution for the given number of seconds. The argument may be a floating point number to indicate a more precise sleep time. The actual suspension time may be less than that requested because any caught signal will terminate the `sleep()` following execution of that signal's catching routine. Also, the suspension time may be longer than requested by an arbitrary amount because of the scheduling of other activity in the system.

time.strftime(format[, t])

Convert a tuple or `struct_time` representing a time as returned by `gmtime()` or `localtime()` to a string as specified by the `format` argument. If `t` is not provided, the current time as returned by `localtime()` is used. `format` must be a string. `ValueError` is raised if any field in `t` is outside of the allowed range. `strftime()` returns a locale dependent byte string; the result may be converted to unicode by doing `strftime(<myformat>).decode(locale.getlocale()[1])`.

Changed in version 2.1: Allowed `t` to be omitted.

Changed in version 2.4: `ValueError` raised if a field in `t` is out of range.

Changed in version 2.5: 0 is now a legal argument for any position in the time tuple; if it is normally illegal the value is forced to a correct one.

The following directives can be embedded in the `format` string. They are shown without the optional field width and precision specification, and are replaced by the indicated characters in the `strftime()` result:

Directive	Meaning	Notes
%a	Locale's abbreviated weekday name.	
%A	Locale's full weekday name.	
%b	Locale's abbreviated month name.	
%B	Locale's full month name.	
%c	Locale's appropriate date and time representation.	
%d	Day of the month as a decimal number [01,31].	
%H	Hour (24-hour clock) as a decimal number [00,23].	
%I	Hour (12-hour clock) as a decimal number [01,12].	
%j	Day of the year as a decimal number [001,366].	
%m	Month as a decimal number [01,12].	
%M	Minute as a decimal number [00,59].	
%p	Locale's equivalent of either AM or PM.	(1)
%S	Second as a decimal number [00,61].	(2)
%U	Week number of the year (Sunday as the first day of the week) as a decimal number [00,53]. All days in a new year preceding the first Sunday are considered to be in week 0.	(3)

%w	Weekday as a decimal number [0(Sunday),6].	
%W	Week number of the year (Monday as the first day of the week) as a decimal number [00,53]. All days in a new year preceding the first Monday are considered to be in week 0.	(3)
%x	Locale's appropriate date representation.	
%X	Locale's appropriate time representation.	
%y	Year without century as a decimal number [00,99].	
%Y	Year with century as a decimal number.	
%z	Time zone name (no characters if no time zone exists).	
%%	A literal '%' character.	

Notes:

1. When used with the `strptime()` function, the `%p` directive only affects the output hour field if the `%I` directive is used to parse the hour.
2. The range really is 0 to 61; this accounts for leap seconds and the (very rare) double leap seconds.
3. When used with the `strptime()` function, `%u` and `%w` are only used in calculations when the day of the week and the year are specified.

Here is an example, a format for dates compatible with that specified in the [RFC 2822](#) Internet email standard. [\[1\]](#)

```
>>> from time import gmtime, strftime
>>> strftime("%a, %d %b %Y %H:%M:%S +0000", gmtime())
'Thu, 28 Jun 2001 14:17:15 +0000'
```

>>>

Additional directives may be supported on certain platforms, but only the ones listed here have a meaning standardized by ANSI C. To see the full set of format codes supported on your platform, consult the `strftime(3)` documentation.

On some platforms, an optional field width and precision specification can immediately follow the initial '%' of a directive in the following order; this is also not portable. The field width is normally 2 except for `%j` where it is 3.

time.strptime(string[, format])

Parse a string representing a time according to a format. The return value is a `struct_time` as returned by `gmtime()` or `localtime()`.

The `format` parameter uses the same directives as those used by `strftime()`; it defaults to "%a %b %d %H:%M:%S %Y" which matches the formatting returned by `ctime()`. If `string` cannot be parsed according to `format`, or if it has excess data after parsing, `ValueError` is raised. The default values used to fill in any missing data when more accurate values cannot be inferred are (1900, 1, 1, 0, 0, 0, 0, 1, -1).

For example:

```
>>> import time
>>> time.strptime("30 Nov 00", "%d %b %y")
time.struct_time(tm_year=2000, tm_mon=11, tm_mday=30, tm_hour=0, tm_min=0,
                 tm_sec=0, tm_wday=3, tm_yday=335, tm_isdst=-1)
```

>>>

Support for the `%z` directive is based on the values contained in `tzname` and whether `daylight` is true. Because of this, it is platform-specific except for recognizing UTC and GMT which are always known (and are considered to be non-daylight savings timezones).

Only the directives specified in the documentation are supported. Because `strftime()` is implemented per platform it can sometimes offer more directives than those listed. But `strptime()` is independent of any platform and thus does not necessarily support all directives available that are not documented as supported.

`class time.struct_time`

The type of the time value sequence returned by `gmtime()`, `localtime()`, and `strptime()`. It is an object with a *named tuple* interface: values can be accessed by index and by attribute name. The following values are present:

Index	Attribute	Values
0	<code>tm_year</code>	(for example, 1993)
1	<code>tm_mon</code>	range [1, 12]
2	<code>tm_mday</code>	range [1, 31]
3	<code>tm_hour</code>	range [0, 23]
4	<code>tm_min</code>	range [0, 59]
5	<code>tm_sec</code>	range [0, 61]; see (2) in <code>strftime()</code> description
6	<code>tm_wday</code>	range [0, 6], Monday is 0
7	<code>tm_yday</code>	range [1, 366]
8	<code>tm_isdst</code>	0, 1 or -1; see below

New in version 2.2.

Note that unlike the C structure, the month value is a range of [1, 12], not [0, 11]. A year value will be handled as described under [Year 2000 \(Y2K\) issues](#) above. A `-1` argument as the daylight savings flag, passed to `mktime()` will usually result in the correct daylight savings state to be filled in.

When a tuple with an incorrect length is passed to a function expecting a `struct_time`, or having elements of the wrong type, a `TypeError` is raised.

`time.time()`

Return the time in seconds since the epoch as a floating point number. Note that even though the time is always returned as a floating point number, not all systems provide time with a better precision than 1 second. While this function normally returns non-decreasing values, it can return a lower value than a previous call if the system clock has been set back between the two calls.

time.timezone

The offset of the local (non-DST) timezone, in seconds west of UTC (negative in most of Western Europe, positive in the US, zero in the UK).

time.tzname

A tuple of two strings: the first is the name of the local non-DST timezone, the second is the name of the local DST timezone. If no DST timezone is defined, the second string should not be used.

time.tzset()

Resets the time conversion rules used by the library routines. The environment variable `tz` specifies how this is done.

New in version 2.3.

Availability: Unix.

Note: Although in many cases, changing the `tz` environment variable may affect the output of functions like `localtime()` without calling `tzset()`, this behavior should not be relied on.

The `tz` environment variable should contain no whitespace.

The standard format of the `tz` environment variable is (whitespace added for clarity):

`std offset [dst [offset [,start[/time], end[/time]]]]`

Where the components are:

`std` and `dst`

Three or more alphanumerics giving the timezone abbreviations. These will be propagated into `time.tzname`

`offset`

The offset has the form: `± hh[:mm[:ss]]`. This indicates the value added to the local time to arrive at UTC. If preceded by a '`-`', the timezone is east of the Prime Meridian; otherwise, it is west. If no offset follows `dst`, summer time is assumed to be one hour ahead of standard time.

`start[/time], end[/time]`

Indicates when to change to and back from DST. The format of the start and end dates are one of the following:

`In`

The Julian day `n` (`1 <= n <= 365`). Leap days are not counted, so in all years February 28 is day 59 and March 1 is day 60.

`n`

The zero-based Julian day (`0 <= n <= 365`). Leap days are counted, and it is possible to refer to February 29.

Mm.n.d

The *d*'th day ($0 \leq d \leq 6$) or week *n* of month *m* of the year ($1 \leq n \leq 5$, $1 \leq m \leq 12$, where week 5 means “the last *d* day in month *m*” which may occur in either the fourth or the fifth week). Week 1 is the first week in which the *d*'th day occurs. Day zero is Sunday.

`time` has the same format as `offset` except that no leading sign ('-' or '+') is allowed. The default, if `time` is not given, is 02:00:00.

```
>>> os.environ['TZ'] = 'EST+05EDT,M4.1.0,M10.5.0'
>>> time.tzset()
>>> time.strftime('%X %x %Z')
'02:07:36 05/08/03 EDT'
>>> os.environ['TZ'] = 'AEST-10AEDT-11,M10.5.0,M3.5.0'
>>> time.tzset()
>>> time.strftime('%X %x %Z')
'16:08:12 05/08/03 AEST'
```

>>>

On many Unix systems (including *BSD, Linux, Solaris, and Darwin), it is more convenient to use the system’s zoneinfo (*tzfile(5)*) database to specify the timezone rules. To do this, set the `TZ` environment variable to the path of the required timezone datafile, relative to the root of the systems ‘zoneinfo’ timezone database, usually located at `/usr/share/zoneinfo`. For example, ‘US/Eastern’, ‘Australia/Melbourne’, ‘Egypt’ or ‘Europe/Amsterdam’.

```
>>> os.environ['TZ'] = 'US/Eastern'
>>> time.tzset()
>>> time.tzname
('EST', 'EDT')
>>> os.environ['TZ'] = 'Egypt'
>>> time.tzset()
>>> time.tzname
('EET', 'EEST')
```

>>>

See also:**Module `datetime`**

More object-oriented interface to dates and times.

Module `locale`

Internationalization services. The locale setting affects the interpretation of many format specifiers in `strftime()` and `strptime()`.

Module `calendar`

General calendar-related functions. `timegm()` is the inverse of `gmtime()` from this module.

Footnotes

[1] The use of `%z` is now deprecated, but the `%z` escape that expands to the preferred hour/minute offset is not supported by all ANSI C libraries. Also, a strict reading of the original 1982 [RFC 822](#) standard calls for a two-digit year (%y rather than %Y),

but practice moved to 4-digit years long before the year 2000. After that, [RFC 822](#) became obsolete and the 4-digit year has been first recommended by [RFC 1123](#) and then mandated by [RFC 2822](#).

ANEXO W: Manual de imshow de OpenCV

trackbar position and the second parameter is the user data (see the next parameter). If the callback is the NULL pointer, no callbacks are called, but only `value` is updated.

userdata – User data that is passed as is to the callback. It can be used to handle trackbar events without using global variables.

The function `createTrackbar` creates a trackbar (a slider or range control) with the specified name and range, assigns a variable `value` to be a position synchronized with the trackbar and specifies the callback function `onChange` to be called on the trackbar position change. The created trackbar is displayed in the specified window `winname`.

Note: [Qt Backend Only] `winname` can be empty (or NULL) if the trackbar should be attached to the control panel.

Clicking the label of each trackbar enables editing the trackbar values manually.

Note:

- An example of using the trackbar functionality can be found at [opencv_source_code/samples/cpp/connected_components.cpp](#)
-

getTrackbarPos

Returns the trackbar position.

C++: `int getTrackbarPos(const string& trackbarname, const string& winname)`

Python: `cv2.getTrackbarPos(trackbarname, winname) → retval`

C: `int cvGetTrackbarPos(const char* trackbar_name, const char* window_name)`

Python: `cv.GetTrackbarPos(trackbarName, windowName) → retval`

Parameters

trackbarname – Name of the trackbar.

winname – Name of the window that is the parent of the trackbar.

The function returns the current position of the specified trackbar.

Note: [Qt Backend Only] `winname` can be empty (or NULL) if the trackbar is attached to the control panel.

imshow

Displays an image in the specified window.

C++: `void imshow(const string& winname, InputArray mat)`

Python: `cv2.imshow(winname, mat) → None`

C: `void cvShowImage(const char* name, const CvArr* image)`

Python: `cv.ShowImage(name, image) → None`

Parameters

winname – Name of the window.

image – Image to be shown.



The function `imshow` displays an image in the specified window. If the window was created with the `CV_WINDOW_AUTOSIZE` flag, the image is shown with its original size. Otherwise, the image is scaled to fit the window. The function may scale the image, depending on its depth:

- If the image is 8-bit unsigned, it is displayed as is.
- If the image is 16-bit unsigned or 32-bit integer, the pixels are divided by 256. That is, the value range [0,255*256] is mapped to [0,255].
- If the image is 32-bit floating-point, the pixel values are multiplied by 255. That is, the value range [0,1] is mapped to [0,255].

If window was created with OpenGL support, `imshow` also support `ogl::Buffer`, `ogl::Texture2D` and `gpu::GpuMat` as input.

namedWindow

Creates a window.

C++: `void namedWindow(const string& winname, int flags=WINDOW_AUTOSIZE)`

Python: `cv2.namedWindow(winname[, flags])` → None

C: `int cvNamedWindow(const char* name, int flags=CV_WINDOW_AUTOSIZE)`

Python: `cv.NamedWindow(name, flags=CV_WINDOW_AUTOSIZE)` → None

Parameters

name – Name of the window in the window caption that may be used as a window identifier.

flags – Flags of the window. The supported flags are:

- **WINDOW_NORMAL** If this is set, the user can resize the window (no constraint).
- **WINDOW_AUTOSIZE** If this is set, the window size is automatically adjusted to fit the displayed image (see `imshow()`), and you cannot change the window size manually.
- **WINDOW_OPENGL** If this is set, the window will be created with OpenGL support.

The function `namedWindow` creates a window that can be used as a placeholder for images and trackbars. Created windows are referred to by their names.

If a window with the same name already exists, the function does nothing.

You can call `destroyWindow()` or `destroyAllWindows()` to close the window and de-allocate any associated memory usage. For a simple program, you do not really have to call these functions because all the resources and windows of the application are closed automatically by the operating system upon exit.

Note: Qt backend supports additional flags:

- **CV_WINDOW_NORMAL or CV_WINDOW_AUTOSIZE:** `CV_WINDOW_NORMAL` enables you to resize the window, whereas `CV_WINDOW_AUTOSIZE` adjusts automatically the window size to fit the displayed image (see `imshow()`), and you cannot change the window size manually.
- **CV_WINDOW_FREERATIO or CV_WINDOW_KEEP_RATIO:** `CV_WINDOW_FREERATIO` adjusts the image with no respect to its ratio, whereas `CV_WINDOW_KEEP_RATIO` keeps the image ratio.
- **CV_GUI_NORMAL or CV_GUI_EXPANDED:** `CV_GUI_NORMAL` is the old way to draw the window without statusbar and toolbar, whereas `CV_GUI_EXPANDED` is a new enhanced GUI.

By default, `flags == CV_WINDOW_AUTOSIZE | CV_WINDOW_KEEP_RATIO | CV_GUI_EXPANDED`

ANEXO X: Manual de circle de OpenCV

The function `transpose()` transposes the matrix `src`:

$$\text{dst}(i, j) = \text{src}(j, i)$$

Note: No complex conjugation is done in case of a complex matrix. It should be done separately if needed.

2.5 Drawing Functions

Drawing functions work with matrices/images of arbitrary depth. The boundaries of the shapes can be rendered with antialiasing (implemented only for 8-bit images for now). All the functions include the parameter `color` that uses an RGB value (that may be constructed with `CV_RGB` or the `Scalar` constructor) for color images and brightness for grayscale images. For color images, the channel ordering is normally *Blue, Green, Red*. This is what `imshow()`, `imread()`, and `imwrite()` expect. So, if you form a color using the `Scalar` constructor, it should look like:

```
Scalar(blue_component, green_component, red_component[, alpha_component])
```

If you are using your own image rendering and I/O functions, you can use any channel ordering. The drawing functions process each channel independently and do not depend on the channel order or even on the used color space. The whole image can be converted from BGR to RGB or to a different color space using `cvtColor()`.

If a drawn figure is partially or completely outside the image, the drawing functions clip it. Also, many drawing functions can handle pixel coordinates specified with sub-pixel accuracy. This means that the coordinates can be passed as fixed-point numbers encoded as integers. The number of fractional bits is specified by the `shift` parameter and the real point coordinates are calculated as $\text{Point}(x, y) \rightarrow \text{Point2f}(x * 2^{-\text{shift}}, y * 2^{-\text{shift}})$. This feature is especially effective when rendering antialiased shapes.

Note: The functions do not support alpha-transparency when the target image is 4-channel. In this case, the `color[3]` is simply copied to the repainted pixels. Thus, if you want to paint semi-transparent shapes, you can paint them in a separate buffer and then blend it with the main image.

Note:

- An example on using variate drawing functions like `line`, `rectangle`, ... can be found at `opencv_source_code/samples/cpp/drawing.cpp`

circle

Draws a circle.

C++: `void circle(Mat& img, Point center, int radius, const Scalar& color, int thickness=1, int lineType=8, int shift=0)`

Python: `cv2.circle(img, center, radius, color[, thickness[, lineType[, shift]]]) → None`

C: `void cvCircle(CvArr* img, CvPoint center, int radius, CvScalar color, int thickness=1, int line_type=8, int shift=0)`

Python: `cv.Circle(img, center, radius, color, thickness=1, lineType=8, shift=0) → None`

Parameters



img – Image where the circle is drawn.
center – Center of the circle.
radius – Radius of the circle.
color – Circle color.
thickness – Thickness of the circle outline, if positive. Negative thickness means that a filled circle is to be drawn.
lineType – Type of the circle boundary. See the [line\(\)](#) description.
shift – Number of fractional bits in the coordinates of the center and in the radius value.

The function **circle** draws a simple or filled circle with a given center and radius.

clipLine

Clips the line against the image rectangle.

C++: bool **clipLine**(Size **imgSize**, Point& **pt1**, Point& **pt2**)
C++: bool **clipLine**(Rect **imgRect**, Point& **pt1**, Point& **pt2**)
Python: cv2.**clipLine**(imgRect, pt1, pt2) → retval, pt1, pt2
C: int **cvClipLine**(CvSize **img_size**, CvPoint* **pt1**, CvPoint* **pt2**)
Python: cv.**ClipLine**(imgSize, pt1, pt2) -> (point1, point2)

Parameters

imgSize – Image size. The image rectangle is Rect(0, 0, **imgSize.width**, **imgSize.height**).
imgRect – Image rectangle.
pt1 – First line point.
pt2 – Second line point.

The functions **clipLine** calculate a part of the line segment that is entirely within the specified rectangle. They return **false** if the line segment is completely outside the rectangle. Otherwise, they return **true**.

ellipse

Draws a simple or thick elliptic arc or fills an ellipse sector.

C++: void **ellipse**(Mat& **img**, Point **center**, Size **axes**, double **angle**, double **startAngle**, double **endAngle**, const Scalar& **color**, int **thickness**=1, int **lineType**=8, int **shift**=0)
C++: void **ellipse**(Mat& **img**, const RotatedRect& **box**, const Scalar& **color**, int **thickness**=1, int **lineType**=8)
Python: cv2.**ellipse**(img, center, axes, angle, startAngle, endAngle, color[, thickness[, lineType[, shift]]]) → None
Python: cv2.**ellipse**(img, box, color[, thickness[, lineType]]) → None
C: void **cvEllipse**(CvArr* **img**, CvPoint **center**, CvSize **axes**, double **angle**, double **start_angle**, double **end_angle**, CvScalar **color**, int **thickness**=1, int **line_type**=8, int **shift**=0)
Python: cv.**Ellipse**(img, center, axes, angle, start_angle, end_angle, color, thickness=1, lineType=8, shift=0) → None

ANEXO Y: Manual de subplot, plot y title de Matplotlib

Table 3.1 – continued from previous page

zorder	any number
--------	------------

To get a list of settable line properties, call the `set()` function with a line or lines as argument

In [69]: `lines = plt.plot([1,2,3])`

In [70]: `plt.setp(lines)`
alpha: float
animated: [True | False]
antialiased or aa: [True | False]
...snip

3.2 Working with multiple figures and axes

MATLAB, and `pyplot`, have the concept of the current figure and the current axes. All plotting commands apply to the current axes. The function `gca()` returns the current axes (a `matplotlib.axes.Axes` instance), and `gcf()` returns the current figure (`matplotlib.figure.Figure` instance). Normally, you don't have to worry about this, because it is all taken care of behind the scenes. Below is a script to create two subplots.

```
import numpy as np
import matplotlib.pyplot as plt

def f(t):
    return np.exp(-t) * np.cos(2*np.pi*t)

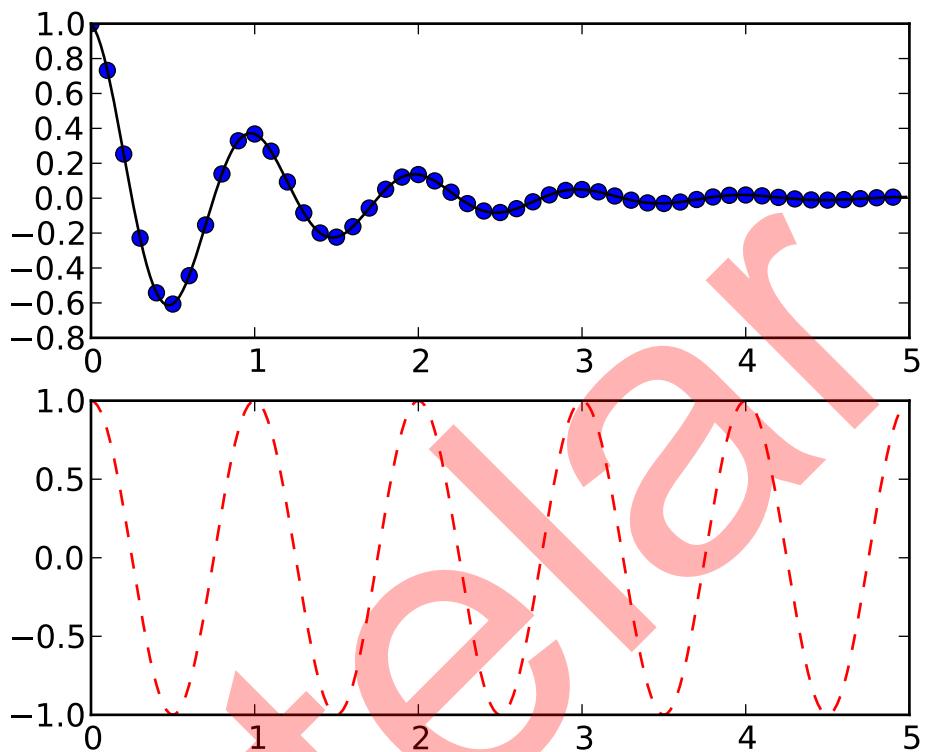
t1 = np.arange(0.0, 5.0, 0.1)
t2 = np.arange(0.0, 5.0, 0.02)

plt.figure(1)
plt.subplot(211)
plt.plot(t1, f(t1), 'bo', t2, f(t2), 'k')

plt.subplot(212)
plt.plot(t2, np.cos(2*np.pi*t2), 'r--')
```

The `figure()` command here is optional because `figure(1)` will be created by default, just as a `subplot(111)` will be created by default if you don't manually specify an axes. The `subplot()` command specifies `numrows`, `numcols`, `fignum` where `fignum` ranges from 1 to `numrows*numcols`. The commas in the `subplot` command are optional if `numrows*numcols<10`. So `subplot(211)` is identical to `subplot(2,1,1)`. You can create an arbitrary number of subplots and axes. If you want to place an axes manually, ie, not on a rectangular grid, use the `axes()` command, which allows you to specify the location as `axes([left, bottom, width, height])` where all values are in fractional (0 to 1) coordinates. See `pylab_examples example code: axes_demo.py` for an example of placing axes manually and `pylab_examples example code: line_styles.py` for an example with lots-o-subplots.

You can create multiple figures by using multiple `figure()` calls with an increasing figure number. Of course, each figure can contain as many axes and subplots as your heart desires:



```
import matplotlib.pyplot as plt
plt.figure(1)
plt.subplot(211)
# the first figure
# the first subplot in the first figure
plt.plot([1,2,3])
plt.subplot(212)
# the second subplot in the first figure
plt.plot([4,5,6])

plt.figure(2)
# a second figure
# creates a subplot(111) by default
plt.plot([4,5,6])

plt.figure(1)
# figure 1 current; subplot(212) still current
plt.subplot(211)
# make subplot(211) in figure1 current
plt.title('Easy as 1,2,3') # subplot 211 title
```

You can clear the current figure with `clf()` and the current axes with `cla()`. If you find this statefulness, annoying, don't despair, this is just a thin stateful wrapper around an object oriented API, which you can use instead (see [Artist tutorial](#))

If you are making a long sequence of figures, you need to be aware of one more thing: the memory required for a figure is not completely released until the figure is explicitly closed with `close()`. Deleting all references to the figure, and/or using the window manager to kill the window in which the figure appears on the screen, is not enough, because pyplot maintains internal references until `close()` is called.