Universidad de Alcalá
Escuela Politécnica Superior

Grado en Ingeniería Electrónica de Comunicaciones

Trabajo Fin de Grado

Estudio del módulo “OpenMV Cam H7” para aplicaciones de visión artificial

ESCUELA POLITECNICA SUPERIOR

Autor: Ángel Rayo Fernández

Tutor: José Manuel Villadangos Carrizo

2021

UNIVERSIDAD DE ALCALÁ

Escuela Politécnica Superior

GRADO EN INGENIERÍA ELECTRÓNICA DE COMUNICACIONES

Trabajo Fin de Grado

Estudio del módulo “OpenMV Cam H7” para aplicaciones de visión artificial

Autor: Ángel Rayo Fernández

Tutor: José Manuel Villadangos Carrizo

TRIBUNAL:

Presidente: Francisco Huerta Sánchez

Vocal 1º: Juan Manuel Miguel Jiménez

Vocal 2º: José Manuel Villadangos Carrizo

FECHA: 21 de septiembre de 2021

Índice

Tabla de Figuras

RESUMEN

En este proyecto se aborda el estudio del módulo *OpenMV Cam H7*. Este módulo consta de un procesador STM32H743VI (*Cortex M7*) y de una cámara de visión (OV7725). Se estudian todas sus características técnicas, eléctricas y también sus opciones de programación. Se trata de un dispositivo comercial reciente, por lo que, es necesario realizar este estudio previo para poder después crear una aplicación en su entorno de desarrollo (*OpenMV IDE*). Se aborda la programación de este módulo para realizar una aplicación que se encargue de reconocer mediante la cámara, la cara de un dado en tiempo real al ser lanzado sobre un tablero y representar su número en un *display*. Todo esto se realizará utilizando técnicas de visión artificial, programando en *micropython*.

SUMMARY

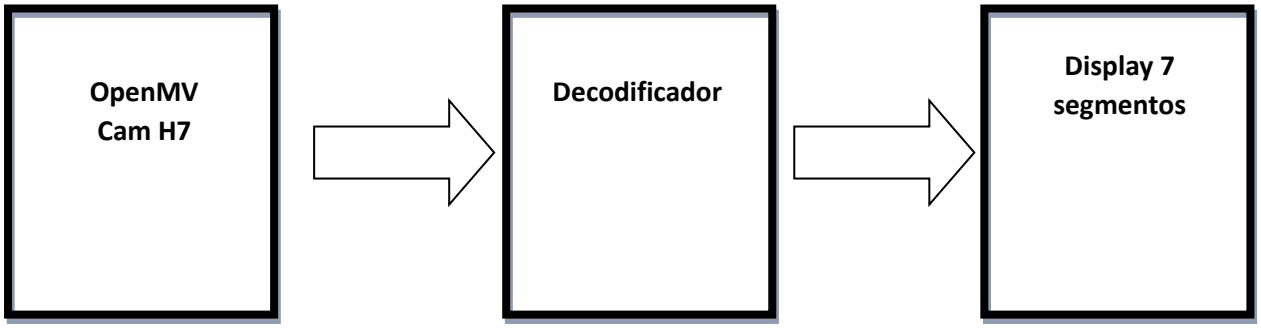
This project will address the study of the OpenMV Cam H7 module. This module consists of a STM32H743VI processor (Cortex M7) and a camera vision (OV7725). All its technical and electrical characteristics and also its programming options will be studied. It is a recent commercial device, so it is necessary to carry out this preliminary study in order to later create an application in its development environment (OpenMV IDE). The programming of this module is approached to make an application that is responsible for recognizing, through the camera, the face of a die in real time when thrown on a board and representing its number on a display. All this will be done using artificial vision techniques, programming in micropython.

RESUMEN EXTENDIDO

En la actualidad existen gran cantidad de aplicaciones relacionadas con la visión artificial. La visión artificial según la Asociación de Imágenes Automatizadas (AIA) se basa en la captura y procesamiento de imágenes combinadas con un software y un hardware específico. Los sistemas de visión artificial se basan en sensores digitales protegidos dentro de cámaras industriales con ópticas especializadas en adquirir imágenes, para que el hardware y el software puedan procesar, analizar y medir diferentes características para tomar decisiones. Estos sistemas se usan en niveles industriales, académicos, gubernamentales y militares. Suelen implicar un coste bajo, una buena precisión, alta solidez y confiabilidad, y una elevada estabilidad térmica y mecánica.

El estudio de la tarjeta *OpenMV Cam H7* consiste en abordar todas sus posibilidades a nivel de software del mundo de los sistemas electrónicos digitales. Se explica cómo y para qué sirven todos sus pines. Además, se explican los recursos que se pueden utilizar como: *timers*, control de servomotor, ADC, DAC, protocolo I2C, UART, etc. Una vez se tiene una idea general del funcionamiento de este módulo, se pasa a realizar un pequeño ejemplo relacionado con el mundo de la visión artificial.

Este ejemplo consiste en la detección de la cara de un dado en tiempo real, mediante la cámara que incorpora el módulo (OV7725) y representación en un *display* de 7 segmentos del número de la cara. Se utilizan técnicas de visión artificial, tales como, el reconocimiento de figuras planas (cuadrados y círculos). Para conseguir la detección de la cara del dado se utilizará la técnica de "entrenamiento" de la cámara del módulo. También se utilizan técnicas como la detección de manchas (*find blobs*). Además, se introduce un filtro cuando las condiciones de luz son bajas, para poder realizar dicha detección. Todo esto se programa en *micropython*, uno de los lenguajes de programación más conocidos. Por último, se muestra el diagrama de bloques del sistema completo.



Capítulo 1

ESTADO DEL ARTE

Este apartado se va a dedicar a la comparativa de los diferentes módulos que existen actualmente en el mercado para trabajar con aplicaciones de visión artificial utilizando *micropython*. En primer lugar, se va a comparar la familia de *OpenMV Cam*, de la cual se ha elegido uno de sus módulos más potentes para la realización de este proyecto. Una vez se ha realizado esta comparativa se mencionarán las características de otros microcontroladores programables en *micropython* y conocidos para aplicaciones de visión artificial o proyectos de IoT (Internet of Things).

Actualmente existe una gran variedad de opciones para realizar proyectos con microcontroladores. A continuación, se va a mostrar una breve comparativa de las características que estos poseen.

1.1. OpenMV Cam M4

Existen dos modelos donde su diferencia más notable es la cámara ya que una tiene mejores características y prestaciones que la otra. En cuanto a su procesador, cuentan con un *Cortex M4*, más concretamente el procesador que lleva es el STM32F427VG, que funciona a 180 MHz con 256 KB de RAM y 1 MB de *flash*. Los modelos de sus cámaras son OV2640 y OV7725. En la Figura 1 se muestra este modelo.

Figura 1. OpenMV M4 [1]

1.2. OpenMV Cam M7

La diferencia de este modelo con las dos versiones anteriores es su procesador. El procesador STM32F765VI ARM *Cortex M7* funciona a 216 MHz con 512 KB de RAM y 2 MB de *flash*. Hay que decir, que el fabricante ya no proporciona este modelo y los dos anteriores, solo el *OpenMV Cam H7* y *H7 Plus*, pero han dejado toda la información disponible para consultar documentación. En la Figura 2 se muestra este modelo.

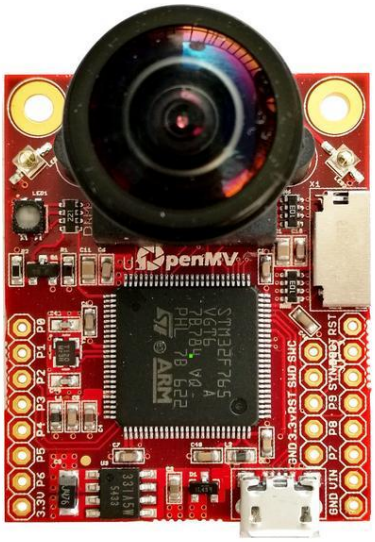


Figura 2. OpenMV H7 [1]

1.3. OpenMV Cam H7 y H7 Plus

Aquí están los dos modelos más modernos y potentes de esta familia. En cuanto al H7, el procesador STM32H743VI ARM *Cortex M7* funciona a 480 MHz con 1 MB de SRAM y 2 MB de *flash*. El H7 Plus cuenta con un STM32H743II, que funciona a 480 MHz con 32 MB de SDRAM + 1 MB de SRAM y 32 MB de *flash* externo + 2 MB de *flash* interno. En el caso de este proyecto se ha optado por el H7 ya que era un microcontrolador bastante potente con unas muy buenas prestaciones y más económico que el H7 Plus.

Estos modelos se diferencian por sus cámaras ya que el H7 Plus tiene una mayor resolución que el H7. En la Figura 3 se muestran los dos.

Figura 3. OpenMV H7 y H7 Plus [1]

1.4.Pyboard

Existen otros módulos como es el caso de este microcontrolador conocido como *pyboard*. Tiene un STM32F405RG CPU *Cortex M4* de 168 MHz con ROM *flash* de 1024 KB y RAM de 192 KB. Existen varias versiones de *pyboard*. En la Figura 4 se muestra la versión 1.1 (la más actual hoy en día). La diferencia de este modelo con el resto es que no tiene una cámara ya incorporada.

Figura 4. Pyboard [2]

1.5.Pixy2

Por último (aunque en el mercado existen miles de modelos más) se va a ver *Pixy2*. *Pixy2* puede aprender a detectar objetos. Tiene algoritmos que detecta y rastrea líneas. También pueden detectar

intersecciones y "señales de tráfico". Tiene un procesador NXP LPC4330 de 204 MHz y de doble núcleo, cuenta con 264 KB de RAM y 2MB de *flash*. En la Figura 5 se muestra este módulo.

Figura 5. Pixy 2 [3]

Hoy en día en la industria se utiliza a menudo la visión artificial, mediante el uso de módulos como los que se ha hecho la comparativa. A continuación, se muestran algunos ejemplos de aplicaciones utilizadas en el sector tecnológico que hacen la vida mucho más fácil.

Detección de defectos: es una de las aplicaciones más usadas en el mundo de la visión artificial. Se necesitan cámaras de una gran resolución para poder detectar defectos en objetos que el ojo humano no pueda percibir. Se pueden detectar defectos como rajas en metales, manchas… Para ello, es necesario entrenar la cámara de visión mostrando imágenes con defecto y sin defecto. Esto se conoce como "cerebro inteligente". En la Figura 6 se muestra un ejemplo de detección de defectos.

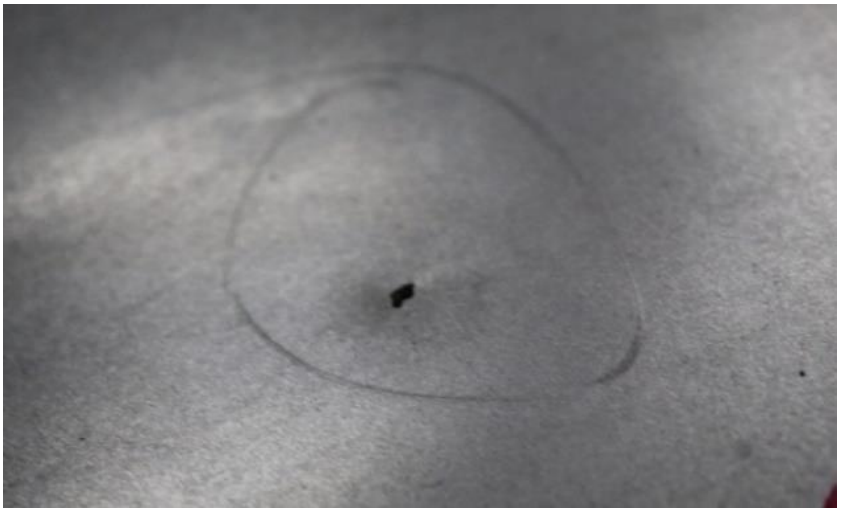


Figura 6. Detección de defectos [4]

Detección de "intrusos": en este caso se utilizan cámaras hiperespectrales, las cuales pueden diferenciar entre un tipo de alimento y un objeto. Esto hace que los productos sean más seguros para

el consumidor. Estas cámaras son capaces de diferenciar el tipo de material a través de la medida que hacen de la longitud de onda. En la Figura 7 se muestra una imagen hiperespectral.

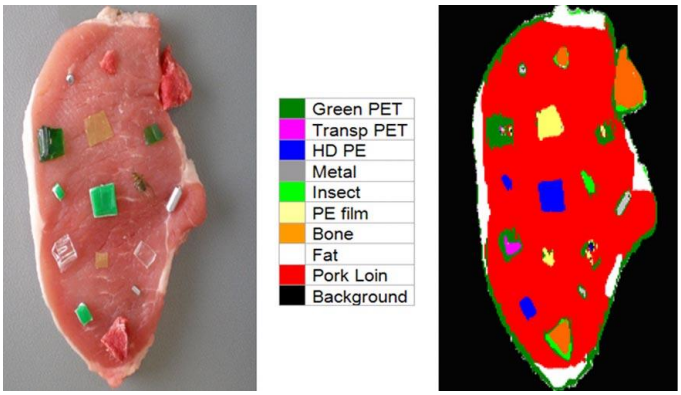


Figura 7. Detección de intrusos en imagen hiperespectral [5]

Verificación de montajes: mediante el uso de cámaras y la visión artificial se pueden comprobar montajes paso a paso o al finalizar un proceso de producción. Estos sistemas reducen considerablemente tiempos de ciclo de operaciones muy complejas y es muy útil para el montaje de maquinaria, equipos o placas electrónicas. En la Figura 8 se muestra un ejemplo de esto.

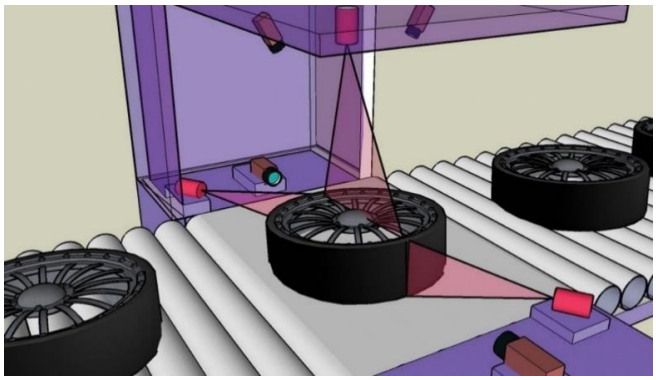


Figura 8. Verificación de montajes [4]

Lector de pantallas (códigos y caracteres): en algunos casos es imposible o resulta difícil extraer información de una pantalla. Una solución a este problema es instalar una cámara de visión artificial para leer la pantalla y extraer los datos que en ella aparecen. Para ello, buscamos las regiones de interés. En la Figura 9 se muestra un ejemplo de esto.

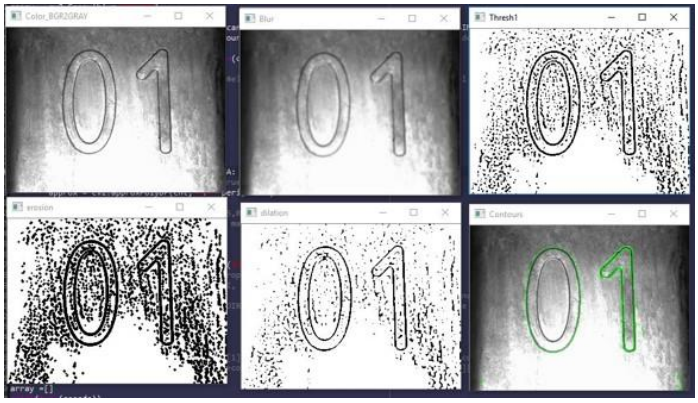


Figura 9. Lector de caracteres en pantallas [4]

En la actualidad existen miles de cámaras que se encargan de solucionar problemas diarios en el mundo de la visión artificial.

1.6. Tipos de módulos de cámaras en PCB

A continuación, se muestran algunos ejemplos compatibles con la familia STM32.

1.6.1. OV2640

Módulo de cámara OV2640 de 200W pixel, hecho con sensor CMOS HD OV2640 de 1/4 pulgadas. Tiene alta sensibilidad, alta flexibilidad y soporte para salida JPEG. Puede admitir exposición, balance de blancos, croma, saturación, contraste y muchos otros ajustes de parámetros, admite Salida de formato JPEG / RGB565.

Figura 10. OV2640 [6]

1.6.2. OV5642

Píxeles del Sensor: 5 millones de píxeles, admite salida de imagen de tamaño 2592*1944.
Compatible con RawRGB, RGB (GRB4: 2: 2, RGB565/555/444), YUV422, JPEG y otros formatos de salida de datos de imagen.
El módulo integra 24MHZ activo, oscilador de cristal, que es más cómodo de usar.
Tamaño óptico 1/4 ", área de píxeles 1,4 μm x 1,4 μm, sensibilidad de hasta 600mV/Lux-seg.
La lente está hecha de alta calidad y de alta transmitancia de luz.
Lente de cristal gran angular 1080P 2,5 MM 1/4 más lente de aleación de magnesio completa.

Figura 11.OV5642 [6]

1.6.3. OV7670

Alta sensibilidad para aplicaciones de baja iluminación.
Bajo voltaje para aplicaciones integradas.
Interfaz estándar SCCB, compatible con interfaz I2C.
Formatos de salida: RawRGB, RGB (GRB4: 2:2, RGB565/555/444), YUV (4:2:2) y YCbCr (4:2:2).
Admite VGA, CIF y varios tamaños desde CIF a 40x30.
Lente de compensación de pérdidas. Detección automática de 50/60Hz.
Ajuste automático de saturación (ajuste UV).

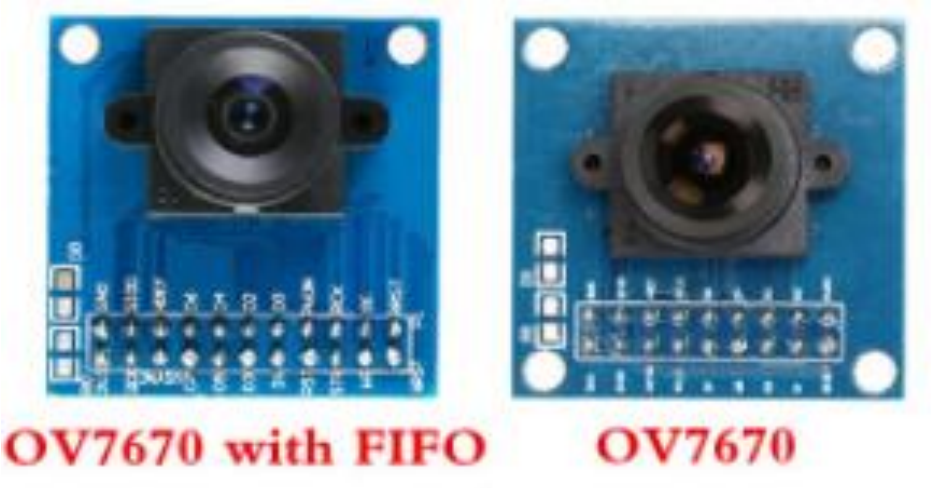


Figura 12. OV7670 [6]

1.6.4. OV7725

OV7725 es una cámara HD de 30W pixel. Similar al tipo anterior. Esta es la cámara de nuestro modulo por lo que, se explicarán sus características más adelante.

Figura 13.OV7725 [6]

Capítulo 2

ESTUDIO DEL MÓDULO "OPENMV CAM H7"

La cámara *OpenMV Cam H7* es una pequeña placa de microcontrolador. Este módulo se caracteriza por su baja potencia. Además, permite implementar aplicaciones de visión artificial en el mundo real.

2.1. Características generales

Las características de *OpenMV Cam H7*:

- El procesador STM32H743VI ARM *Cortex M7* funciona a 480 MHz con 1 MB de SRAM y 2 MB de *flash*. Todos los pines de E / S dan salida a 3,3 V y son tolerantes a 5 V. El procesador tiene las siguientes interfaces de E / S:
- Una interfaz USB (12Mbs) de velocidad completa para su computadora. Su *OpenMV Cam* aparecerá como un puerto COM virtual y una unidad *flash* USB cuando se conecte.
- Un zócalo de tarjeta µSD capaz de lecturas / escrituras de 100Mbs que permite que su *OpenMV* Cam tome fotografías y extraiga fácilmente los activos de visión artificial de la tarjeta µSD.

- Un bus SPI que puede funcionar hasta 80 Mbs, lo que le permite transmitir fácilmente datos de imágenes desde el sistema a LCD *Shield*, WiFi *Shield* u otro microcontrolador.
- Un bus I2C (hasta 1 Mb / s), un bus CAN (hasta 1 Mb / s) y un bus serie asíncrono (TX / RX, hasta 7,5 Mb / s) para interactuar con otros microcontroladores y sensores.
- Un ADC de 12 bits y un DAC de 12 bits.
- Tres pines de E / S para servocontrol.
- Interrupciones y PWM en todos los pines de E / S (hay 10 pines de E / S en la placa).
- Y, un LED RGB y dos LED IR de 850 nm de alta potencia.

En la Figura 14 se muestra el módulo *OpenMV Cam H7*.

Figura 14.Módulo OpenMV Cam H7 [7]

El módulo viene con una cámara integrada. Esta cámara es la OV7725 y se muestra en la Figura 15. Esta cámara es capaz de tomar imágenes en escala de grises de 640x480 de 8 bits o imágenes RGB565 de 640x480 de 16 bits a 75 FPS cuando la resolución es superior a 320x240 y 150 FPS cuando está por debajo.

La mayoría de los algoritmos simples se ejecutarán entre 75-150 FPS en resoluciones QVGA (320x240) e inferiores. Su sensor de imagen viene con una lente de 2.8 mm en una montura de lente M12 estándar.

Figura 15. OV7725 [6]

Un dato para tener en cuenta en el estudio de este microcontrolador es su potente procesador, ya que, cuenta con un ARM *Cortex M7*. A continuación, se muestra una comparativa con las diferentes características de los *Cortex M0* a *M7*.

En primer lugar, la familia de *Cortex M0* y *M0+* son la versión más básica, por lo que, no pueden producirse microcontroladores de alto rendimiento. Por otro lado, el más convencional es el *Cortex M3* ya que tiene una amplia gama de aplicaciones. Pero, al compáralo con el *Cortex M4* este es mucho más rápido en flujos de datos y cálculos.

Por último, el *Cortex M7* tiene mejor rendimiento que todos los citados anteriormente, aunque con un alto consumo de energía, es perfecto para proyectos con rendimiento extremo. En la Figura 16 se muestra el rendimiento y consumo de energía.

Figura 16. Rendimiento y consumo de procesadores Cortex [8]

También existen otros tipos de *Cortex-M* no tan conocidos. Pero, en la Figura 17 se muestra una tabla de comparación con características más técnicas y que pueden ser de interés.

Feature	Cortex-M0	Cortex-M0+	Cortex-M1	Cortex-M23	Cortex-M3	Cortex-M4	Cortex-M33	Cortex-M35P	Cortex-M55	Cortex-M7
Instruction Set Architecture	Armv6-M	Armv6-M	Armv6-M	Armv8-M Baseline	Armv7-M	Armv7-M	Armv8-M Mainline	Armv8-M Mainline	Armv8.1-M Mainline	Armv7-M
TrustZone for Armv8-M	No	No	No	Yes (option)	No	No	Yes (option)	Yes (option)	Yes (option)	No
Digital Signal Processing (DSP) Extension	No	No	No	No	No	Yes	Yes	Yes	Yes	Yes
Hardware Divide	No	No	No	Yes	Yes	Yes	Yes	Yes	Yes	Yes
Arm Custom Instructions	No	No	No	No	No	No	Yes	No	Yes	No
Coprocessor Interface	No	No	No	No	No	No	Yes	Yes	Yes	No
DMIPS/MHz*	0.87	0.95	0.8	0.98	1.25	1.25	1.5	1.5	1.6	2.14
CoreMark®/MHz*	2.33	2.46	1.85	2.64	3.34	3.42	4.02	4.02	4.2	5.01
Maximum # External Interrupts	32	32	32	240	240	240	480	480	480	240
Maximum MPU Regions	0	8	0	16	8	8	16	16	16	16
Bus Protocol	AHB Lite	AHB Lite	AHB Lite	AHB5	AHB Lite	AHB Lite	AHB	AHB	AXI	AXI
Instruction Cache	No	No	No	No	No	No	No	2-16kB	0-64kB	0-64kB
Data Cache	No	No	No	No	No	No	No	No	0-64kB	0-64kB
Instruction TCM	No	No	No	No	No	No	No	No	0-16MB	0-16MB
Data TCM	No	No	No	No	No	No	No	No	0-16MB	0-16MB
Dual Core Lock-Step (DCLS)	No	No	No	Yes	No	No	Yes	Yes	No	Yes
Common Criteria Certification	No	No	No	No	No	No	Yes	Yes	No	No
Reference Package and/or System Example	Corstone -101	Corstone -101	-	Corstone -102	Corstone -101	Corstone -101	Corstone -201	-	Corstone -300	-

Figura 17. Comparación de procesadores Cortex [9]

El entorno de desarrollo que se va a utilizar para la programación de esta tarjeta se llama *OpenMV IDE* y tiene un potente editor de texto, una terminal serie (de depuración), una ventana para la visualización de la cámara en tiempo real y un búfer para la visualización de los histogramas de la imagen. En la Figura 18 se muestra una imagen del entorno de desarrollo.

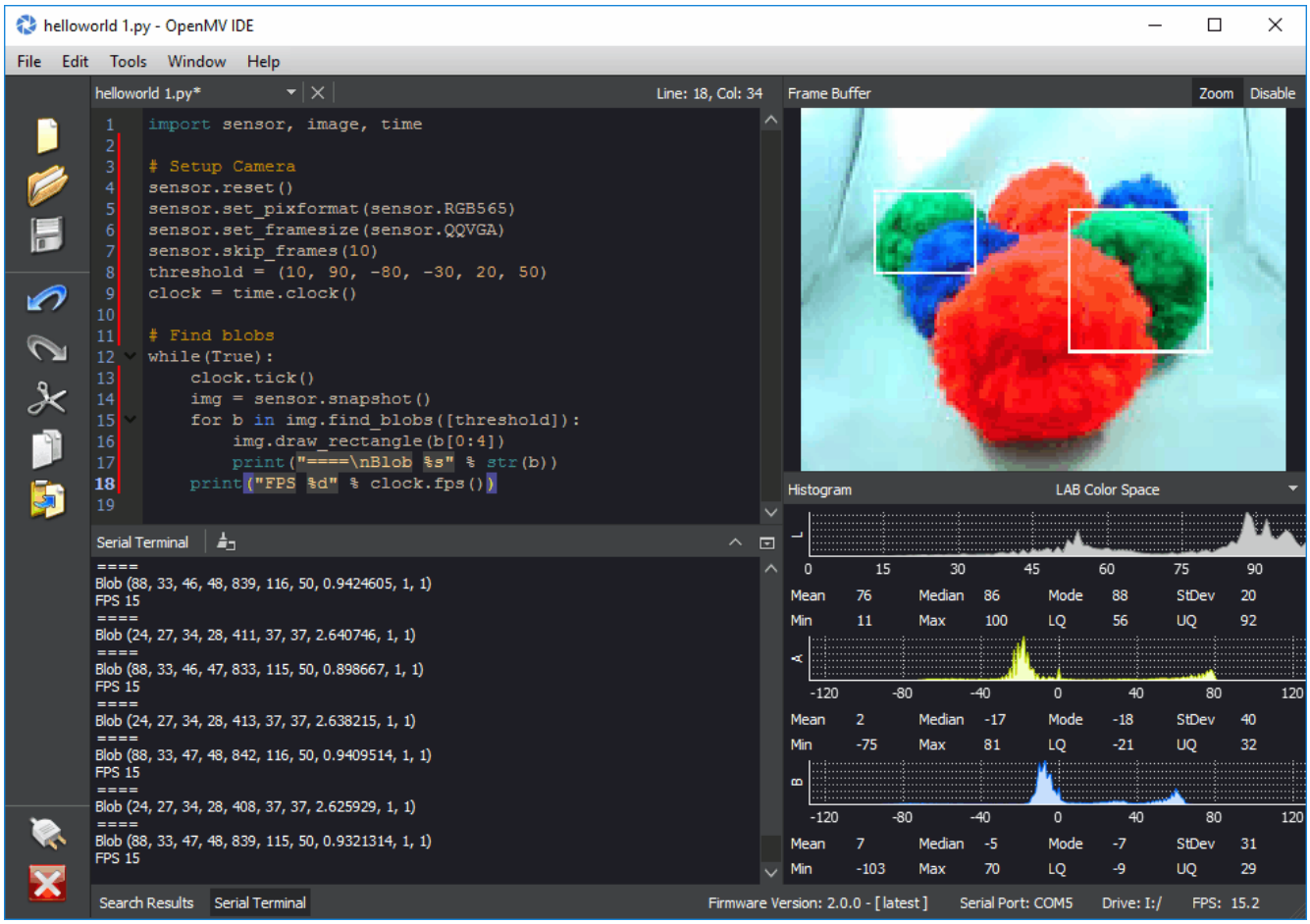


Figura 18. Entorno de desarrollo OpenMV IDE [10]

2.2. Pines y GPIO

En primer lugar, se va a mostrar un resumen de los pines y los nombres con que se les llamará. Además, todos los pines son tolerantes a 5V con una salida de 3.3V (P6 no es tolerante a 5V en modo ADC o DAC) y todos los pines pueden generar hasta 25 mA (hasta 120 mA en total entre todos los pines):

- Pin('P0') →P0 (PB15)
- Pin('P1') →P1 (PB14)
- Pin('P2') →P2 (PB13)
- Pin('P3') →P3 (PB12)
- Pin('P4') →P4 (PB10)
- Pin('P5') →P5 (PB11)
- Pin('P6') →P6 (PA5)

- Pin(‘P7’) →P7 (PD12)
- Pin(‘P8’) →P8 (PD13)
- Pin(‘P9’) →P9 (PD14) (OpenMV Cam M7/H7 Only)

Pin Functions

Pin			Description
Header	No	Name	
J1 Pin Configuration			
J1	1	P0	UART1 RX – TM1 CH3N – SPI 2 MOSI
	2	P1	UART1 TX – TM1 CH2N – SPI 2 MISO
	3	P2	CAN2 TX – TM1 CH1N – SPI 2 SCLK
	4	P3	CAN2 RX – SPI 2 SS
	5	P4	TIM2 CH3 – I2C 2 SCL – UART 3 TX
	6	P5	TIM2 CH4 – I2C 2 SDA – UART3 RX
	7	P6	TIM2 CH1 – DAC – ADC
	8	3.3	3.3V Rail (250 mA Supply MAX).
J2 Pin Configuration			
J2	1	RST	Reset (Connect to GND to reset).
	2	BOOT	Boot 0 (Connect to 3.3V for DFU mode).
	3	SYN	Frame synchronization pin (Use to frame sync cams).
	4	P9	Servo3 – TIM4 CH3
	5	P8	Servo2 – TIM4 CH2 – I2C4 SDA
	6	P7	Servo1 – TIM4 CH1 – I2C4 SCL
	7	VIN	VIN (3.6V – 5V).
	8	GND	GND Rail
J3 Pin Configuration			
J3	1	SWC	Serial wire debug clock.
	2	SWD	Serial wire debug data.
	3	RST	Reset (active low).
	4	3.3V	3.3V rail (500 mA Supply MAX)
	5	GND	GND rail

Figura 19.Pin functions [11]

En la Figura 20 se muestra una imagen del microcontrolador con todos sus pines y las diferentes funciones que podemos dar a cada pin.

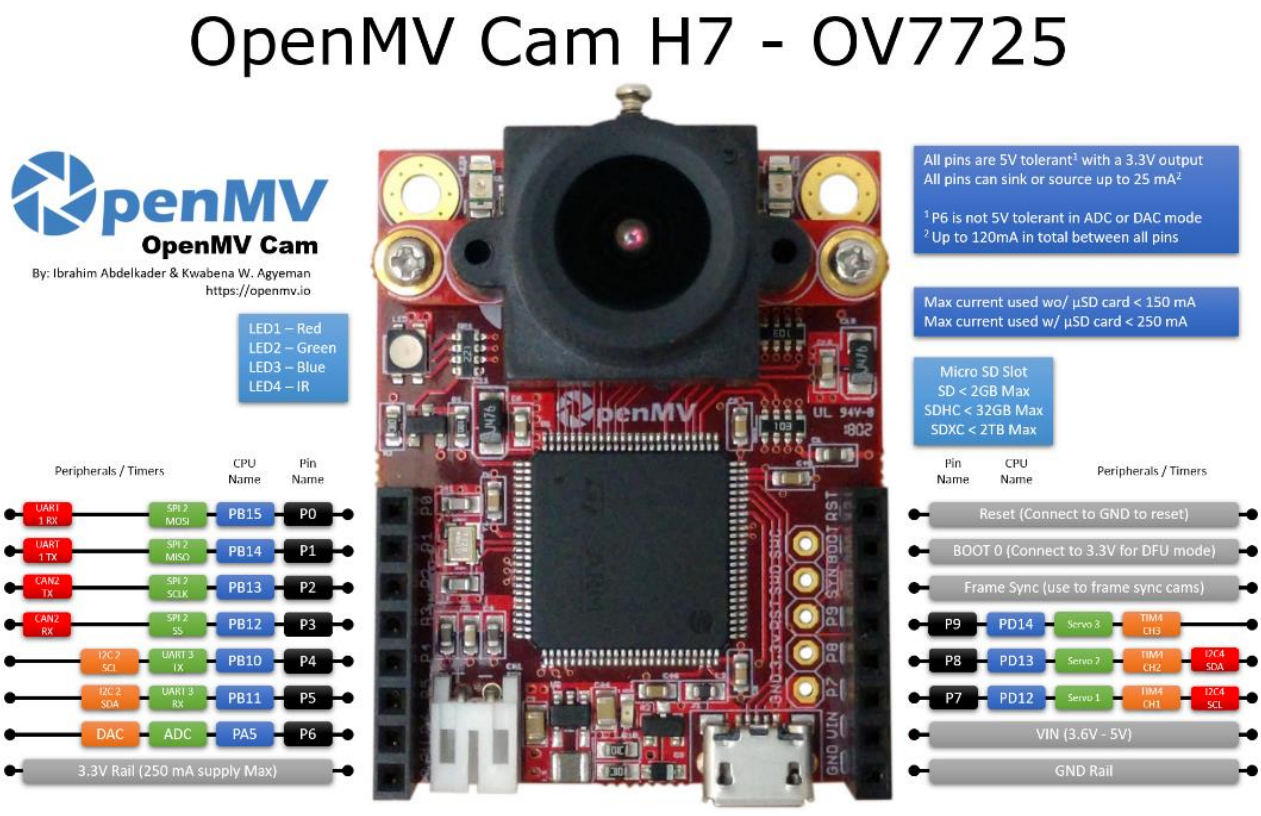


Figura 20. Pinout del módulo OpenMV Cam H7 [1]

Un pin es un objeto básico para controlar las entradas y salidas de nuestra placa. Tiene métodos para configurar el modo del pin (entrada, salida…) y métodos para controlar el nivel lógico digital. Para el control analógico de un pin se utiliza el ADC, que se estudiará más adelante.

A continuación, se muestran todas las opciones de configuración y de uso. Todos los pines están predefinidos con la siguiente estructura: *pyb.Pin.board.Name*. Para entenderlo mejor se explica un ejemplo de código de configuración del pin 0.

```
p0_pin = pyb.Pin.board.P0
```

Se ha configurado el pin 0 con el nombre *p0_pin*. Los pines de la CPU se pueden configurar de la misma forma, pero cambiando la palabra *board* por *cpu.* En este caso la diferencia sería que no habría que poner *cpu.P0* sino que habría que poner *cpu.PB15* ya que el pin 0 de la placa se corresponde con el PB15 de la CPU.

También se pueden utilizar cadenas:

```
g = pyb.Pin('P0', pyb.Pin.OUT_PP)
```

Se pueden agregar sus nombres personalizados, que sirvan de etiqueta:

```
MyMapperDict = {'LeftMotorDir': pyb.Pin.cpu.PB15}
pyb.Pin.dict(MyMapperDict)
g = pyb.Pin("LeftMotorDir", pyb.Pin.OUT_OD)
```

Además, se pueden consultar asignaciones:

```
pin = pyb.Pin("LeftMotorDir")
```

Y, por último, se puede agregar una función de mapeo:

```
def MyMapper(pin_name):
        if pin_name == "LeftMotorDir":
                return pyb.Pin.cpu.PB15

pyb.Pin.mapper(MyMapper)
```

Para resumir, el siguiente orden muestra como asignar la configuración de un pin de nuestra placa:

- Especificar un objeto pin
- Proporcionar la función de mapeo
- Escribir una cadena (*string*) que coincida con un pin de la placa.
- Escribir una cadena (*string*) que coincida con un puerto de la CPU.

2.2.1. Programación de pines y GPIO

- **CONSTRUCTORES**

class pyb.Pin(id, ...)

Esta clase se encarga de crear un nuevo objeto pin. Si se dan argumentos adicionales se utilizan para inicializar el pin, lo que se verá más adelante con *pin.init()*.

- **MÉTODOS DE CLASE**

***classmethod* Pin.debug([*state*])**

Se obtiene el estado de depuración (*true* o *false*) para activarlo o desactivarlo. Con el estado de depuración podemos obtener información sobre como se asigna un objeto en particular a un pin.

***classmethod* Pin.dict([*dict*])**

Se configura el diccionario del *mapeador* de pines.

***classmethod* Pin.mapper([*fun*])**

Se configura la función de mapeo de pines.

- **MÉTODOS**

Pin.init(mode, pull=Pin.PULL_NONE, af=-1)

Inicializa el pin. Podemos configurar el modo de las siguientes formas:

- *Pin.IN*: configurar el pin para la entrada.
- *Pin.OUT_PP*: configurar el pin para salida, con control push-pull.
- *Pin:OUT_OD*: configurar el pin para salida, con control de drenaje abierto.
- *Pin.AF_PP*: configurar el pin para función alternativa, pull-pull.
- *Pin.AF_OD*: configurar el pin para función alternativa, drenaje abierto.
- *Pin.ANALOG*: configurar el pin en modo analógico.

También se puede configurar la resistencia interna como resistencia pull-up (*Pin.PUL_UP*), como resistencia de pull-down (*Pin.PULL_DOWN*) o sin resistencia (*Pin.PULL_NONE*). Cuando el modo es *Pin.AF_PP* o *Pin.AF_OD*, entonces AF puede ser el índice o el nombre de una de las funciones alternativas asociadas con un pin.

Pin.value([value])

Establece el nivel lógico de un pin digital. Si no se le pasa argumento a la función se devuelve un 0 o un 1 en función del valor del pin. Con *value* dado, donde *value* puede ser cualquier cosa que se convierta en booleano. Si *value* se convierte en *True*, entonces el pin pasa a nivel alto, de lo contrario, el pin pasaría a nivel bajo.

Pin.__str__()

Devuelve una cadena que describe el objeto pin.

Pin.af()

Devuelve la función alternativa del pin. El entero devuelto coincidirá con una de las constantes permitidas para el argumento AF con la función init.

Pin.af_list()

Devuelve una matriz de funciones alternativas disponibles para este pin.

Pin.gpio()

Devuelve la dirección base del bloque GPIO asociado con este pin.

Pin.mode()

Devuelve el modo de configuración actual del pin. El entero devuelto coincidirá con una de las constantes permitidas para el argumento de modo de la función init.

Pin.name()

Devuelve el nombre del pin.

Pin.names()

Devuelve los nombres del pin correspondientes a la placa y CPU.

Pin.pin()

Devuelve el número del pin.

Pin.port()

Devuelve el puerto del pin.

Pin.pull()

Devuelve el tipo de resistencia de pull configurada del pin. El entero devuelto coincidirá con una de las constantes permitidas para el argumento de extracción de la función init.

- **CONSTANTES**

Pin.AF_OD

Inicializa el pin en el modo de función alternativa con una unidad de drenaje abierto.

Pin.AF_PP

Inicializa el pin en el modo de función alternativa con un impulsor push-pull.

Pin.ANALOG

Inicializa el pin en modo analógico.

Pin.IN

Inicializa el pin al modo de entrada.

Pin.OUT_OD

Inicializa el pin al modo de salida con una unidad de drenaje abierto.

Pin.OUT_PP

Inicializa el pin al modo de salida con una unidad push-pull.

Pin.PULL_DOWN

Habilita la resistencia desplegable en el pin.

Pin.PULL_NONE

No habilita ninguna resistencia de subida o bajada en el pin.

Pin.PULL_UP

Habilita la resistencia pull-up en el pin.

- **FUNCIONES ALTERNATIVAS (*classAF*)**

Los pines de nuestra placa pueden tener diferentes funciones, ya que, se pueden configurar como GPIO, I2C, ADC… Para determinar la función que queremos dar a cada pin usaremos estas funciones alternativas.

pinaf.__str__( )

Devuelve una cadena que describe la función alternativa.

pinaf.index( )

Devuelve el índice de la función alternativa.

pinaf.name( )

Devuelve el nombre de la función alternativa.

pinaf.reg( )

Devuelve el registro base asociado con el periférico asignado a esta función alternativa. Por ejemplo, si la función alternativa fuera TIM2_CH3, esto devolvería stm.TIM2.

2.3. Retardos y tiempos

Para controlar retardos y tiempos en el OpenMV Cam H7 es necesario importar la librería *utime* en nuestro OpenMV IDE. Por lo que, se va a estudiar a fondo esta librería para conocer todas sus funciones y posibilidades.

2.3.1. Programación de retardos

A continuación, se muestran las diferentes funciones que podemos programar en nuestro módulo.

***utime.localtime*([*segundos*]) y *utime.gmtime*([*segundos*])**

La función *localtime* devuelve una tupla de 8 digitos de fecha y hora en hora local, mientras que la *gmtime* la devuelve en UTC. El formato de entrada de la tupla de 8 dígitos.

utime.mktime( )

Esta función es inversa a la de la hora local. Su argumento es una tupla completa de 8 dígitos que expresa una hora según la hora local. Devuelve un número entero que es el número de segundos transcurridos desde el 1 de enero de 2000.

***utime.sleep*(*segundos*)**

Duerme la cantidad de segundos indicada. Algunas placas pueden aceptar *segundos* como un número de punto flotante para dormir durante una fracción de segundos. Tenga en cuenta que es posible que otras placas no acepten un argumento de punto flotante, por lo que existen otras funciones.

utime.sleep_ms(ms)

Retardo para un número determinado de milisegundos. Debe ser mayor o igual que 0.

utime.sleep_us(us)

Retardo para un número determinado de microsegundos. Debe ser mayor o igual que 0.

utime.ticks_ms( )

Esta función devuelve un contador en milisegundos creciente con un punto de referencia aleatorio, que se envuelve después de un valor. A este "valor de envoltura" podemos referirnos como TICKS_MAX

utime.ticks_us( )

Como la función anterior, pero en microsegundos.

utime.ticks_cpu( )

Función similar a *ticks_ms* pero con una resolución mayo, ya que se trata de relojes de la CPU. Esta función está pensada para evaluaciones comparativas de tiempos muy precisos, aunque no todos los puertos están disponibles para usar está función. Por lo que, solo sería necesaria en casos de una precisión temporal muy ajustada.

utime.ticks_add(ticks,delta)

Esta función desplaza el valor de los ticks por un número dado, que puede ser positivo o negativo. Permite calcular los *ticks delta* del valor de ticks antes o después de él, siguiendo la definición aritmética modular que se ha explicado en la función anterior *ticks_ms.* Es útil para calcular fechas límites de eventos o interrupciones.

utime.ticks_diff(ticks1,ticks2)

Mide la diferencia entre los valores devueltos por las funciones *ticks_ms, ticks_us* o *ticks_cpu.* Esta función nos devuelve la diferencia entre *ticks1 y ticks2.*

utime.time( )

Devuelve el número de segundos, como un número entero. Esta función devuelve el número de segundos desde un punto de referencia específico del puerto en el tiempo, es decir, desde su encendido o reinicio.

utime.time_ns( )

Similar a la función anterior, pero devuelve nanosegundos, como un número entero.

2.4. LED

En primer lugar, se va a mostrar cuantos LEDs tiene incorporado este microcontrolador. En este caso como se aprecia en la Figura 21 se ve que tiene un LED, pero que se puede configurar de cuatro formas diferentes.

Como se ve en la Figura 21, el PINOUT de los LEDs es el siguiente:

- LED (1) → Color rojo.
- LED (2) → Color verde.

- LED (3) → Color azul.
- LED (4) → LED IR.

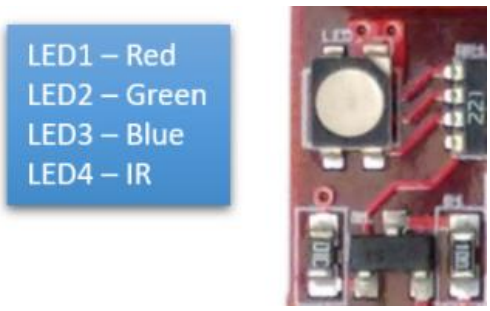


Figura 21. LED de la placa OpenMV Cam H7[1]

2.4.1. Programación del LED de la placa

La librería en la que se puede encontrar toda la información necesaria sobre este recurso es *pyb.LED.* a continuación, se muestran los comandos para controlar los LEDs

- **CONSTRUCTORES**

class ***pyb.LED(id)***

Crea un objeto LED, donde id es el número (1-4) de tipo de LED.

- **MÉTODOS**

Existen tres métodos para el control de LED:

- Función *LED.off()*: Apaga el LED
- Función *LED.on()*: Enciende el LED, a máxima intensidad.
- Función *LED.toggle()*: Alterna el modo del LED entre encendido (intensidad máxima) y apagado. Si el LED tiene una intensidad distinta de cero se considera que está "encendido", y entonces se apagará.

2.5. Control de servomotor

También se puede controlar hasta tres servomotores distintos con este microcontrolador. Para ello, los pines encargados de realizar esta función serían los que se muestran en la Figura 22.

Figura 22.Pines para control de servomotor [1]

Como se puede comprobar en la imagen los pines encargados de controlar los servomotores son el P7, P8, P9. Aunque dichos pines también se pueden configurar para otras funcionalidades que ya se verá más adelante (TIM4 e I2C). Una vez se conoce esta información se va a pasar al estudio de las diferentes funciones para el control de un servo usando *micropython*.

2.5.1. Programación para el control de un servomotor

- **CONSTRUCTORES**

class pyb.Servo(id)

Crea un objeto servo, donde id es el número del servomotor a controlar (1-3) y se corresponde con los pines P7-P9 respectivamente.

- **MÉTODOS**

Existen tres métodos o funciones para controlar cada servo:

Servo.angle([angle, time=0])

Si no se dan argumentos, esta función devuelve el ángulo actual. Si, por el contrario, se le pasan argumentos a la función; *angle* es el ángulo que se moverá el servo en grados y *time* es el tiempo empleado, en milisegundos, en llegar a ese ángulo (si se omite se moverá lo más rápido posible).

Servo.speed([speed, time=0])

Si no se dan argumentos se devuelve la velocidad actual. Si le damos argumentos *speed* es la velocidad a la que cambiar entre -100 y 100 y *time* el número de milisegundos para alcanzar dicha velocidad (si se omite *time* el tiempo en alcanzar la velocidad será el mínimo posible).

Servo.pulse_width([value])

Si no se dan argumentos se devuelve el valor del ancho del pulso sin procesar. Si se proporciona un argumento esta función establece el valor del ancho del pulso sin procesar.

Servo.calibration([pulse_min, pulse_max, pulse_centre[, pulse_angle_90, pulse_speed_100]])

Si no se proporcionan argumentos esta función devuelve los argumentos como un grupo de 5 datos. Si se proporcionan argumentos:

- *pulse_min:* es el ancho del pulso mínimo permitido.
- *pulse_max:* es el ancho del pulso máximo permitido.
- *pulse_centre:* es el ancho del pulso correspondiente a la posición central/cero.
- *pulse_angle_90:* es el ancho de pulso correspondiente a 90 grados.
- *pulse_speed_100:* es el ancho de pulso correspondiente a una velocidad de 100.

2.6. Interrupciones externas

Para estudiar las interrupciones para eventos externos se necesita estudiar la librería *pyb.ExtInt*. Hay un total de 22 interrupciones, 16 de ellas pueden ser de pines (GPIO) y otras 6 pueden venir de fuentes internas. Con el uso del comando *ExtInt* se configura el pin como GPIO para entrada

(preparándose así para la llegada de un flanco de bajada o subida que nos indique que deberá producirse una interrupción). En este ejemplo de código se llamará a la función cada vez que llegue un flanco de bajada al pin:

```
extint = pyb.ExtInt(pin, pyb.ExtInt.IRQ_FALLING, pyb.Pin.PULL_UP, callback)
```

2.6.1. Programación de interrupciones

- **CONSTRUCTORES**

class pyb.ExtInt(pin, mode, pull, callback)

Crea una interrupción externa. Los parámetros que se pueden configurar son:

- *pin*: es el pin en el que se habilita la interrupción.
- *mode*: puede ser de tres tipos (*falling, rising o rising_falling*).
- *pull*: pueden ser *pull_NONE, pull_UP o pull_DOWN*.
- *callback*: es la función a la que se llama cuando se produce la interrupción.

- **MÉTODOS**

ExtInt.disable()

Deshabilita la interrupción.

ExtInt.enable()

Habilita la interrupción deshabilitada.

ExtInt.line()

Devuelve el número de línea al que está asignado el pin.

ExtInt.swint()

Activa la devolución de llamada desde el software.

- **CONSTANTES**

ExtInt.IRQ_FALLING

Interrumpe en flanco de bajada.

ExtInt.IRQ_RISING

Interrumpe en flanco de subida.

ExtInt.IRQ_RISING_FALLING

Interrumpe en flanco de subida y bajada.

2.7. Temporizadores (TIMERS y PWM)

Los temporizadores o TIMERS se pueden utilizar para una amplia variedad de tareas, pero en este caso la función que tienen es la de interrumpir una función periódicamente, que es el caso que implementa este microcontrolador. Cada uno de los temporizadores de esta placa tiene un contador que cuenta a un determinado ritmo y su velocidad de cuenta es la frecuencia del reloj de la CPU dividida entre el *preescaler*. Cuando el contador alcanza el período del temporizador, se activa un evento y el contador vuelve a ponerse a cero. Al usar el método de devolución de llamada, el evento del temporizador puede llamar a una función de *micropython*.

A continuación, se muestra un pequeño ejemplo para entender mejor el funcionamiento de los temporizadores. Este ejemplo consiste en alternar el encendido y apagado de un LED a una frecuencia fija.

```
tim = pyb.Timer(4)              # create a timer object using timer 4
tim.init(freq=2)                # trigger at 2Hz
tim.callback(lambda t:pyb.LED(1).toggle())
```

En primer lugar, se ha creado un objeto *TIMER*, usando el TIMER4. Después, se le ha pasado la frecuencia a la cual se llama a la función del TIMER. Por último, se llama a la función *tim.callback*, en la cual, se hace un *toggle* al LED, para que de esta forma cada medio segundo (2Hz) se encienda o apague.

Nota: El temporizador (1) se utiliza para la cámara. De manera similar, el temporizador (5) controla el servo, el temporizador (6) se utiliza para la lectura / escritura cronometrada de ADC / DAC. Se recomienda utilizar los otros temporizadores en sus programas. En total se pueden configurar los TIMER del 1 al 14, pero solo se pueden controlar en nuestra placa 9 (3 canales de TIM1, TIM2 y TIM4).

2.7.1. Programación de TIMERS y PWM

- **CONSTRUCTORES**

class pyb.Timer(id, ...)

Construye un nuevo objeto timer, donde id puede ser 1-14. Si se dan elementos adicionales, el temporizador se inicializa con la función *init()*.

- **MÉTODOS**

Timer.init(* , freq , prescaler , period , mode=Timer.UP ,div = 1, callback = None , deadtime=0)

Inicializa el timer. La inicialización puede ser por frecuencia (en Hz) o por preescaler y periodo:

```
tim.init(freq=100)                  # set the timer to trigger at 100Hz
tim.init(prescaler=83, period=999)  # set the prescaler and period directly
```

Ahora vamos a explicar los argumentos que se la pasan a esta función:

- *freq*- Especifica la frecuencia periódica del TIMER. Tambié puede verse como la frecuencia por la que el TIMER realiza una cuenta entera y se resetea, es decir, vuelve a cero.

- *preescaler* [0-0xFFFF]- es el valor que se va a cargar en el prescaler del TIMER. El valor del reloj de la CPU se divide entre *prescaler + 1*, para ser este el valor al que tiene que llegar la cuenta del TIMER. Los TIMER 2-7 y 12-14 tienen una fuente de reloj de 84MHz, y los TIMER 1 y 8-11 tienen una fuente de reloj de 168 MHz.
- *Period* [0-0xFFFF]- para los TIMER 1,3,4 y 6-14. [0-0x3FFFFFFF] para los TIMER 2 y 5. Este valor se cargará en el registro ARR (AutoReload Register) del TIMER. Este determina el período del TIMER, es decir, cuando realiza un ciclo. El contador del TIMERse reiniciará después de *period + 1* ciclos.
- *mode* – el modo puede ser:
  - *Timer.UP:* Configura el TIMER para que cuente desde 0 hasta el ARR (opción por defecto).
  - *Timer.DOWN:* Configura el TIMER para que cuente desde ARR hasta 0 (cuenta descendente).
  - *Timer.CENTER:* Configura el TIMER para que cuente desde 0 hasta el ARR y después de forma descendente.
- *div* – puede tomar el valor 1,2 o 4. Divide el reloj del TIMER para determinar la frecuencia de muestreo.
- *callback* – como la función *timer.callback* (se verá a continuación).
- *Deadtime* – nos da la cantidad de tiempo inactivo. Este valor solo está disponible en los TIMER 1 y 8.

Timer.deinit( )

Sirve para desinicializar el TIMER. Desactiva la devolución a la llamada (callback), desactiva todos los *callback* de cualquier canal, para el TIMER y deshabilita el puerto del TIMER.

Timer.callback(fun)

Esta es la función que se llamará cuando se active el TIMER. Se le pasa a *fun* el argumento del objeto del TIMER, si *fun* es *None* entonces la función queda desactivada.

Timer.channel(channel, mode, ...)

Si solo se pasa un número de canal, esta función devuelve un objeto del canal (inicializado antes o *none* si no hay un canal anterior). En el caso contrario, se inicializa y se devuelve el objeto *TimerChannel*. Cada cana se puede configurar para usarlo como comparación a la salida (modo contador), captura a la entrada (modo captura) y para utilizar la PWM.

Argumentos que se le pasan a *Timer.channel*:

- *mode* – puede tomar los siguientes valores:
  - *Timer.PWM*: configura el TIMER como PWM (activo alto).
  - *Timer.PWM_INVERTED*: configura el TIMER como PWM (activo bajo).
  - *Timer.OC_TIMING:* no se active ningún pin.
  - *Timer.OC_ACTIVE:* se activa el pin cuando se produzca en la salida una coincidencia en la comparación.
  - *Timer.OC_INACTIVE:* se desactiva el pin cuando se produzca en la salida una coincidencia en la comparación.
  - *Timer.OC_TOGGLE:* se alterna el estado del pin cuando se produzca en la salida una coincidencia en la comparación.
  - *Timer.OC_FORCED_ACTIVE:* se activa el pin sin mirar la comparación, es decir, de forma forzada.

    - *Timer.OC_FORCED_INACTIVE:* se desactiva el pin sin mirar la comparación, es decir, de forma forzada.
    - *Timer.IC:* configura el TIMER en modo captura.
    - *Timer.ENC_A:* se configura el TIMER en modo *Encoder*. El TIMER solo cambiará cuando cambie el canal 1 (CH1).
    - *Timer.ENC_B:* se configura el TIMER en modo *Encoder*. El TIMER solo cambiará cuando cambie el canal 2 (CH2).
    - *Timer.ENC_AB:* se configura el TIMER en modo *Encoder*. El TIMER solo cambiará cuando cambie el canal 1 (CH1) o el canal 2 (CH2).
- *callback* – Igual que *Timer.callback()*.
- *Pin* – la opción por defecto es *None*. Si se especifica un numero de pin (se configura su función alternativa) y este no admite una función alternativa, se producirá un error.

A continuación, se muestran los argumentos para el modo PWM.

- *pulse_width* – determina el valor inicial del ancho del pulso.
- *pulse_width_percent* – porcentaje del ancho de pulso inicial.

Para el modo TIMER.OC los argumentos son los siguientes:

- *compare* – es el valor de inicio del registro comparación.
- *polarity* – hay dos modos de configuración:
    - *Timer.HIGH:* salida activa a nivel alto.
    - *Timer.LOW:* salida activa a nivel bajo.

Para el modo TIMER.IC los argumentos son:

- *polarity* – existen tres modos configurables:
    - Timer.RISING: captura en flanco de subida.
    - Timer.FALLING: captura en flanco de bajada.
    - Timer.BOTH: captura en ambos flancos.

El modo captura solo funciona en el canal principal, no en los secundarios. Una vez explicado estos conceptos se muestra un pequeño ejemplo de uso de PWM.

```
timer = pyb.Timer(2, freq=1000)
ch2 = timer.channel(2, pyb.Timer.PWM, pin=pyb.Pin.board.X2, pulse_width=8000)
ch3 = timer.channel(3, pyb.Timer.PWM, pin=pyb.Pin.board.X3, pulse_width=16000)
```

Timer.counter([value])

Configura el valor del contador del TIMER.

Timer.freq([value])

Configura el valor de la frecuencia del TIMER.

Timer.period([value])

Configura el valor del período del TIMER.

Timer.prescaler([value])

Configura el valor del prescaler del TIMER.

Timer.source_freq( )

Configura el valor de la frecuencia de la fuente.

2.7.2. Configuración de un canal del TIMER

Los canales de los temporizadores se utilizan para generar o capturar señales, se crean usando el método *timmerchannel()*.

Timerchannel.callback(fun)

Es la función que se llamará cuando se active el canal del TIMER. A *fun* se le pasa el objeto del TIMER, si *fun* es *None* entonces se desactiva esta función.

Timerchannel.capture([value])

Establece o configura el valor cuando el canal está en modo captura a la entrada.

Timerchannel.compare([value])

Establece o configura el valor cuando el canal está en modo comparación a la salida.

Timerchannel.pulse_width([value])

Esta función establece el valor del ancho del pulso del canal. Cuando esté en modo PWM se utilizará el nombre *pulse_width*.

Timerchannel.pulse_width_percent([value])

Establece el valor del porcentaje del ancho del pulso. Este valor puede estar entre 0 y 100 y es el período durante el cual el pulso del canal está activo, es decir, el porcentaje de tiempo que el pulso está activo. Por ejemplo, un valor 50, corresponde con un 50% (el número puede ser entero o float).

2.8. ADC (CONVERSIÓN ANALÓGICA-DIGITAL)

El pin que se encarga de controlar el ADC de nuestra placa es el P6. Este pin es tolerante a 3V3 pero en modo ADC no es tolerante a 5V, con lo que, se debe tener precaución si se usa el ADC de la placa.

2.8.1. Programación del ADC

- **CONSTRUCTORES**

class pyb.ADC(pin)

Crea un objeto ADC en el pin dado. En este caso, hay que seleccionar el pin P6 o PA5 si usamos el nombre de la CPU.

- **MÉTODOS**

ADC.read()

Lee y devuelve el valor del ADC, este valor estará entre 0 y 4095.

ADC.read_timed(buf, timer)

Lee los valores del ADC (*buf*) a una velocidad establecida por el TIMER. *Buf* puede ser de 8 a 16 bits, es decir, *bytearray* o *array.array*. a continuación, se muestra un ejemplo de uso:

```
adc = pyb.ADC(pyb.Pin("P5"))        # create an ADC on pin P5
tim = pyb.Timer(6, freq=10)         # create a timer running at 10Hz
buf = bytearray(100)                # create a buffer to store the samples
adc.read_timed(buf, tim)            # sample 100 values, taking 10s
```

2.8.2. El objeto ADCAll

El uso de este objeto cambia todos los pines del ADC a entradas analógicas. Se puede acceder a los datos del ADC por los canales 16,17 y 18 para ver los resultados de los registros VBAT, MCU y VREF (datos de temperaturas). Para escalar bien los datos, hay que saber que se utiliza un voltaje de referencia, que en este caso es 3V3. La placa tiene un sensor interno de temperatura, que nos mide la temperatura de MCU, no siempre esta temperatura será la misma que la temperatura ambiente.

Los métodos ADCAll *read_core_vbat()*, *read_vref()* y *read_core_vref()* leen la tensión de la batería y la tensión de referencia. Los resultados son números tipo *float* y dan valores directos de tensión.

- *Read_core_vbat()* devuelve la tensión de la batería de respaldo.
- *Read_vref()* se encarga de medir la referencia de tensión interna.

2.9. DAC (CONVERSIÓN DIGITAL-ANALÓGICA)

El DAC se utiliza para generar valores analógicos (un voltaje específico) en el pin P6. El voltaje estará entre 0 y 3,3 V. Para el uso del DAC es necesario inicializarlo, para ello, existen constructores y métodos.

2.9.1. Programación del DAC

- **CONSTRUCTORES**

class pyb.DAC(port, bits=8, *, buffering=None)

Esta instrucción construye un nuevo objeto DAC.

- *port*: es un objeto pin, en este caso, siempre deberá ser el pin P6.
- *bits*: se refiere al número de bits que especifica la resolución. Pueden ser 8 u 12 bits.
- *almacenamiento en búfer*: puede ser *None* (para seleccionar un valor predeterminado del búfer como *DAC.noise*, *DAC.triangle* o *DAC.write_timed*), *True* (permite el almacenamiento en búfer de salida) o *False* (desactiva el búfer completo).

- **MÉTODOS**

DAC.init(bits=8, *, buffering=None)

Inicializa el DAC.

DAC.deinit()

Desinicializa el DAC.

DAC.noise(freq)

Genera una señal de ruido a una frecuencia especifica.

DAC.triangle(freq)

Genera una onda triangular.

DAC.write(value)

Acceso a la salida del DAC. El rango es [0, 2^{bits}-1].

DAC.write_timed(data, freq, *, mode=DAC.NORMAL)

Inicia una ráfaga de RAM a DAC mediante una transferencia de DMA. Los datos de entrada se tratan como una matriz de bytes en el modo de 8 bits y una matriz de medias palabras sin signo (código de tipo de matriz 'H') en el modo de 12 bits. Se le pasa el parámetro *freq* que especifica la frecuencia para escribir las muestras.

Ejemplo de uso del DAC para generar una onda sinusoidal continua con una resolución de 12 bits:

```
import math
from array import array
from pyb import DAC
# create a buffer containing a sine-wave, using half-word samples
buf = array('H', 2048 + int(2047 * math.sin(2 * math.pi * i / 128)) for i in range(128))
dac = DAC(pyb.Pin("P6"), bits=12) # output the sine-wave at 400Hz
dac.write_timed(buf, 400 * len(buf), mode=DAC.CIRCULAR)
```

2.10. UART

Este módulo implementa el protocolo de comunicaciones serie dúplex UART / USART estándar. A nivel físico consta de 2 líneas: RX y TX. El *pinout* para el control de este recurso es el siguiente:

- UART 3 RX -> P5 (PB11)
- UART 3 TX -> P4 (PB10)
- UART 1 RX -> P0 (PB15) (OpenMV Cam M7 / H7 solamente)
- UART 1 TX -> P1 (PB14) (OpenMV Cam M7 / H7 solamente)

Para inicializar los objetos de la UART:

```
from pyb import UART

uart = UART(3, 9600, timeout_char=1000)
uart.init(9600, bits=8, parity=None, stop=1, timeout_char=1000)
```

Los bits pueden ser 7, 8 o 9. La paridad puede ser ninguna, 0 (par) o 1 (impar). Los bits de STOP pueden ser 1 o 2. Para el control de lectura o escritura se utilizan estas dos funciones básicas:

```
uart.read()         # read all available characters
uart.write('abc')   # write the 3 characters
```

2.10.1. Programación de la UART

- **CONSTRUCTORES**

class pyb.UART(bus, ...)

Crea un objeto UART en el bus dado. En este caso puede ser 1 u 3. Se crea el objeto, pero no se inicializa y los pines físicos en este módulo son:

- UART 1 → P0(RX) y P1(TX)
- UART 3 → P4(TX) y P5(RX)

- **MÉTODOS**

UART.init(baudrate, bits=8, parity=None, stop=1, *, timeout=1000, flow=0, timeout_char=0, read_buf_len=64)

Inicializa todos los parámetros que se pueden usar en la UART. Los parámetros son: *baudrate* es la frecuencia de reloj; *bits* es el número de bits por carácter 7-9; *parity* es la paridad par o impar; *stop* son los bits de parada 1 u 2; *flow* establece el control de flujo; *timeout* es el tiempo de espera para leer o escribir el primer carácter; *timeout_char* es el tiempo de espera entre caracteres; *read_buf_len* es la longitud de caracteres del buffer de lectura.

UART.deinit()

Desinicializa el bus de la UART.

UART.any()

Devuelve el número de bytes que están en espera.

UART.read([nbytes])

Lee caracteres, sin *nbytes* está dado se leen esos caracteres como máximo. Si no se pasa *nbytes* se leen la mayor cantidad de datos posibles.

UART.readchar()

Lee solo un carácter.

UART.readinto(buf[, nbytes])

Lee los bytes en *buf*. Si no se especifica *nbytes* se lee *len(buf)*.

UART.readline()

Lee una línea de caracteres.

UART.write(buf)

Escribe el búfer de bytes en el bus. Si los caracteres tienen 7 u 8 bits de ancho, cada byte es un carácter. Si los caracteres tienen un ancho de 9 bits, se utilizan dos bytes para cada carácter (little endian).

UART.writechar(char)

Escribe solo un carácter, donde *char* es un número entero.

UART.sendbreak()

Envía un *descanso,* que tiene una duración de 13 bits.

2.11. BUS SPI

El protocolo SPI es un protocolo controlado por un maestro en serie. A nivel físico hay 3 líneas: SCK, MOSI, MISO. Es un protocolo muy similar al I2C, que se verá en el siguiente apartado. El pinout de este recurso es:

- P0→ SPI 2 MOSI (Master-Out-Slave-In)
- P1→ SPI 2 MISO (Master-In-Slave-Out)
- P2→ SPI 2 SCLK (Serial Clock)
- P3→ SPI 2 SS (Serial Select)

2.11.1. Programación del bus SPI

- **CONSTRUCTORES**

class pyb.SPI(bus, ...)

Construye un objeto SPI, pero no lo inicializa.

- **MÉTODOS**

SPI.init(mode, baudrate=328125, *, prescaler, polarity=1, phase=0, bits=8, firstbit=SPI.MSB, ti=False, crc=None)

Inicializa el objeto SPI. Los parámetros que se le pasan a esta función son: *mode* puede ser maestro o esclavo; *baudrate* es la frecuencia de reloj de SCK; *prescaler* es la cuenta que se utilizará para derivar SCK de la frecuencia del bus APB; *polarity* puede ser 0 y 1; *phase* puede ser 0 y 1 para muestrear datos en primer o segundo flanco; *bits* es el número de bits y pueden ser 8 0 16; *firstbit* indica si es MSB o LSB (mayor o menor peso); *ti* y *crc* no son muy relevantes para su estudio.

SPI.deinit()

Apaga o desinicializa el SPI.

SPI.recv(recv, *, timeout=5000)

Recibe datos, donde *recv* es el número de bytes o datos a recibir y *timeout* el tiempo de espera para la recepción.

SPI.send(send, *, timeout=5000)

Envía datos, donde *send* es el número de bytes o datos a enviar y *timeout* el tiempo de espera para el envío.

SPI.send_recv(send, recv=None, *, timeout=5000)

Envía y recibe datos al mismo tiempo, es una combinación de las dos funciones anteriores.

2.12. I2C

El protocolo I2C es un protocolo *two-wire* para conseguir la comunicación entre dos dispositivos. A nivel físico consta de dos cables (SCL y SDA) el reloj y las líneas de datos.

Los métodos básicos son enviar y recibir:

```
i2c.send('abc')      # send 3 bytes
i2c.send(0x42)       # send a single byte, given by the number
data = i2c.recv(3)   # receive 3 bytes
```

Para recibir en el lugar, primero cree un *bytearray*:

```
data = bytearray(3)  # create a buffer
i2c.recv(data)       # receive 3 bytes, writing them into data
```

Puede especificar un tiempo de espera (en ms):

```
i2c.send(b'123', timeout=2000)   # timeout after 2 seconds
```

Un maestro debe especificar la dirección del destinatario:

```
i2c.init(I2C.MASTER)
i2c.send('123', 0x42)        # send 3 bytes to slave with address 0x42
i2c.send(b'456', addr=0x42)  # keyword for address
```

2.12.1. Programación del protocolo I2C

- **CONSTRUCTORES**

class pyb.I2C(bus, ...)

Construye un objeto I2C, pero no se inicializa ningún parámetro.

- **MÉTODOS**

I2C.init(mode, *, addr=0x12, baudrate=400000, gencall=False, dma=False)

Inicializa el objeto I2C creado. Los parámetros son: *mode* puede ser *I2C.MASTER* o *I2C.SLAVE*; *addr* es la dirección de 7 bits; *baudrate* es la frecuencia de SCL; *gencall* sirve para saber si admite el modo de llamada general; *dma* activa o desactiva las transferencias por DMA.

I2C.deinit()

Apaga el bus I2C.

I2C.is_ready(addr)

Comprueba si un dispositivo I2C responde a una dirección, solo es válido en modo maestro.

I2C.mem_read(data, addr, memaddr, *, timeout=5000, addr_size=8)

Se encarga de leer la memoria de un dispositivo I2C.

I2C.mem_write(data, addr, memaddr, *, timeout=5000, addr_size=8)

Se encarga de escribir en la memoria de un dispositivo I2C.

Ambas funciones (*mem_read* y *mem_write*) tienen estos cinco parámetros:

- *data*: buffer desde el que leer/escribir.
- *addr*: dirección del dispositivo I2C.
- *memaddr*: es la ubicación de la memoria.
- *timeout*: es el tiempo de espera (en ms).
- *addr_size*: es el ancho de la memoria (8 o 16 bits).

I2C.recv(recv, addr=0x00, *, timeout=5000)

Sirve para recibir datos en el bus.

I2C.send(send, addr=0x00, *, timeout=5000)

Sirve para enviar datos al bus.

I2C.scan()

Escanea todas las direcciones I2C (0x01-0x7f) y devuelve una lista de las que responden.

Capítulo 3

DETECCIÓN DE FIGURAS PLANAS UTILIZANDO TÉCNICAS DE VISIÓN ARTIFICIAL APLICADAS AL RECONOCIMIENTO DE LA CARA DE UN DADO

En este apartado se va a estudiar de forma teórica los pasos iniciales que se han dado para la posterior realización de la detección de la cara de un dado y representación en tiempo real. Se muestra un pequeño pseudocódigo a modo de "mapa mental inicial" en la Figura 23.

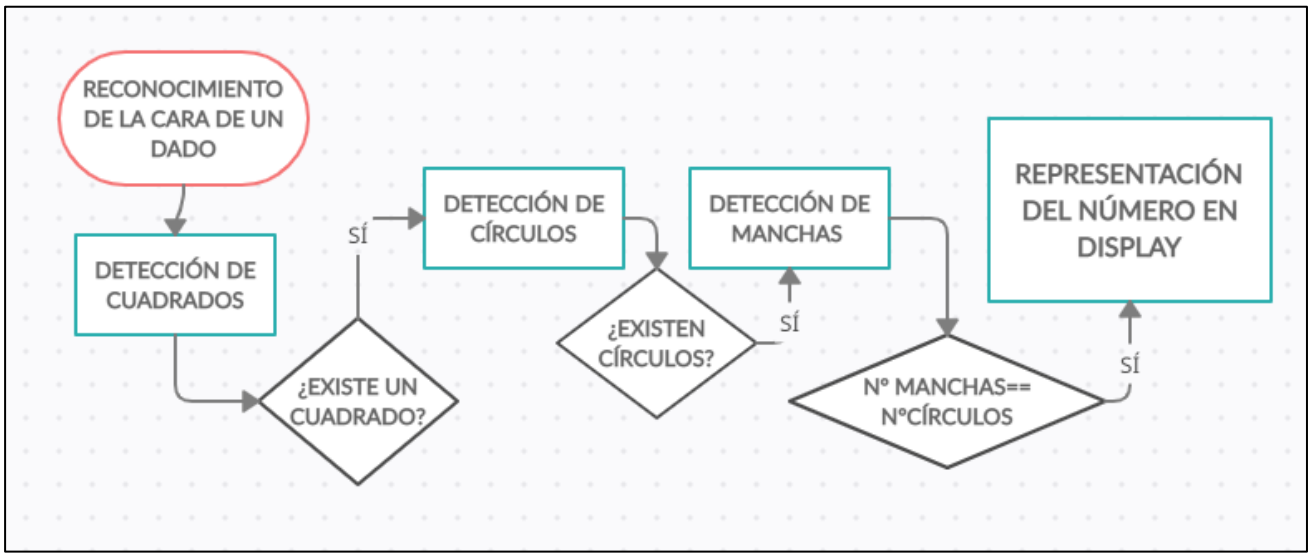


Figura 23.Pseudocódigo a modo de mapa mental inicial

3.1. Detección de cuadrados

Para la detección de cuadrados o rectángulos es necesario la utilización de un algoritmo de visión artificial capaz de llevar a cabo dicha tarea. Para ello, se va a explicar los pasos necesarios para conseguirlo. En visión artificial existen una amplia variedad de funciones que se encargan de encontrar contornos, puntos clave, etc. En este caso, se está trabajando con *micropython* por lo que existe una función que se encarga de encontrar contornos de forma cuadrada o rectangular. Para abordar este problema, se emplea el uso de la función *img.find_rects(threshold)*. Una vez encontrado la cámara puede dibujarlo si fuera necesario con la función *img.draw_rectangle()*. A continuación, se muestra un breve ejemplo de código y los resultados obtenidos al usar nuestro módulo.

```
for r in img.find_rects(threshold = 10000):
    img.draw_rectangle(r.rect(), color = (255, 0, 0))
    for p in r.corners(): img.draw_circle(p[0], p[1], 5, color = (0, 255, 0))
                print(r)
```

Código que dibuja un contorno cuando detecta cuadrados.

Como se puede observar se usa un bucle *for* para estar continuamente comprobando si existen figuras cuadradas o rectangulares. Se guarda esta imagen en r para después poder dibujar su contorno con la función *draw_rectangle*, ya que, el argumento que le pasamos a esta función es r. También, se le pasa el color con el que va a representar el contorno. En este caso, el color el rojo. En la Figura 24 se muestran los resultados obtenidos.

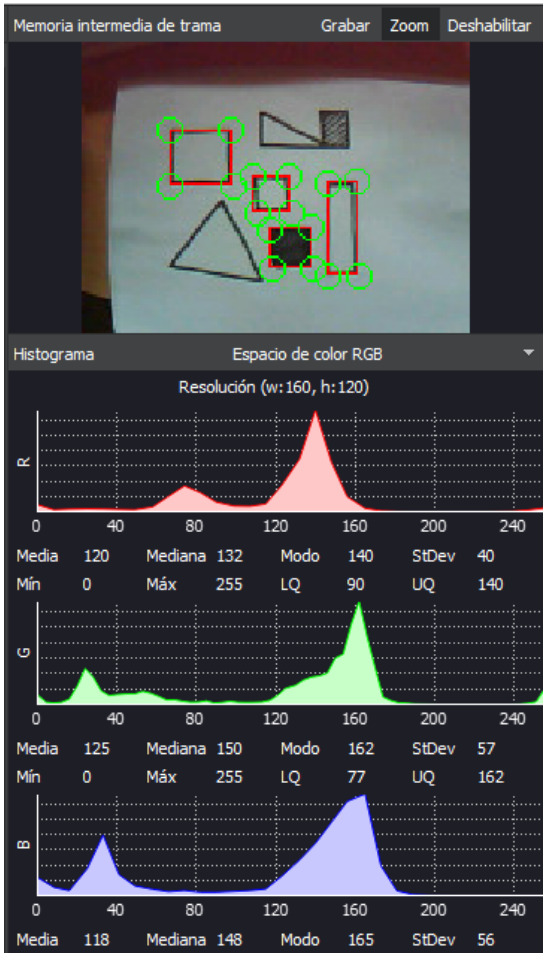


Figura 24. Reconocimiento de cuadrados en OpenMV IDE

3.2. Detección de círculos

Otra de las muchas técnicas de detección de objetos e imágenes usada en la visión artificial es el reconocimiento de círculos. Se trata de un algoritmo parecido al explicado en el apartado anterior. La diferencia es que en este caso se tienen muchos más parámetros en los que fijarse. A continuación, se muestra la función que se encargar de detectar círculos.

```
img.find_circles(threshold, x_margin, y_margin, r_margin, r_min, r_max, r_step )
```

Como se puede ver en esta función entran en juego varios parámetros, que se van a explicar.

- *threshold*: controla el número de círculos detectados. Incrementando este valor decrementa el número de círculos detectados. Este control se lleva a cabo, ya que cuanto menor sea el valor menos pixeles detectará.
- *x_margin*, *y_margin*, *r_margin*: controla la convergencia de los círculos
- *r_min*, *r_max*, *r_step*: controla el radio del circulo que se está encontrando o "testeando".

A parte de detectar el círculo, para saber que se ha detectado, se puede dibujar su contorno con la función:

```
img.draw_circle(c.x(), c.y(), c.r(), color = (255, 0, 0))
```

El parámetro "c" es el círculo que ya se han detectado antes y por eso se pasan las coordenadas del centro (x,y) y el valor del radio. El último parámetro es el color del que se dibuja el contorno. En la Figura 25 se muestran los resultados obtenidos de un ejemplo de código para detectar círculos.

```
import sensor, image, time

sensor.reset()
sensor.set_pixformat(sensor.RGB565) # grayscale is faster
sensor.set_framesize(sensor.QQVGA)
sensor.skip_frames(time = 2000)
clock = time.clock()

while(True):
        clock.tick()
        img = sensor.snapshot().lens_corr(1.8)
        for c in img.find_circles(threshold = 2000, x_margin = 10,     y_margin
        = 10, r_margin = 10,
             r_min = 2, r_max = 100, r_step = 2):
         img.draw_circle(c.x(), c.y(), c.r(), color = (255, 0, 0))
         print(c)
```

Código que dibuja un contorno cuando detecta círculos.

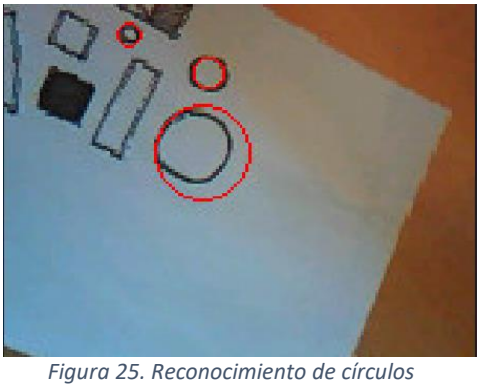


Figura 25. Reconocimiento de círculos

3.3. Detección de manchas o "blobs"

Esta técnica se conoce como *find blobs* o *descriptores de manchas* y se encarga de detectar áreas del mismo color. Los blobs nos dan al detalle una descripción de cómo es la estructura de la imagen (aunque existen también otras técnicas que se encargan de detectar esquinas o bordes, estas son más puntuales). Los descriptores de mancha a menudo funcionan comparando puntos que son iguales, es decir, recorren todos los pixeles de la imagen comparándolos con el color que deseamos encontrar. Cuando encuentran todos los pixeles del mismo color que queremos detectar machas, los agrupan y devuelven un *TRUE*. En la Figura 26 se muestran los pixeles de una imagen y como ha encontrado dos tipos de blobs (blob1 y blob2) y los clasifica. Esto ayudará a detectar mejor los círculos del dado, ya que serán del mismo color.

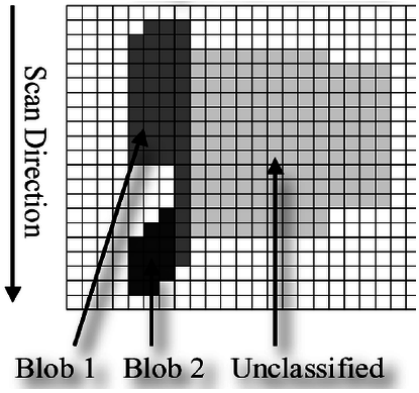


Figura 26. Pixeles de una imagen [12]

Una vez analizado y estudiado estos tres elementos se puede pasar a realizar el caso con un dado real teniendo una base teórica sobre algunas de las técnicas que nos pueden ser útiles. En este caso se hicieron pruebas con figuras de dados como las que se muestran en la Figura 27.

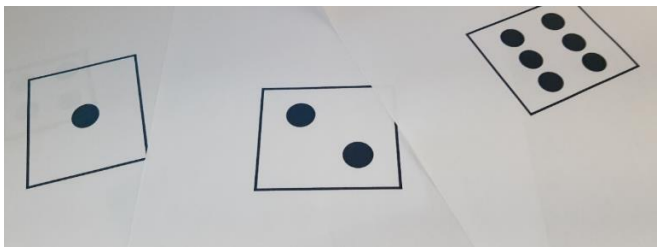


Figura 27.Imagenes de dado de prueba

En la Figura 28 se muestra uno de los resultados obtenidos en la cámara y como consigue detectar los círculos del número 5.

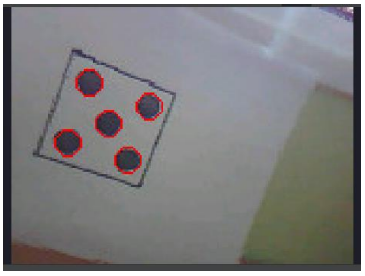


Figura 28. Prueba del número 5

3.4. Detección y reconocimiento de la cara de un dado

En primer lugar, se va a "entrenar" la cámara para la clasificación de objetos. Una vez hecho esto, se empezarán a aplicar las técnicas de visión artificial vistas en el capítulo anterior. Y, por último, se estudiará la electrónica para la representación del número de la cara del dado.

3.4.1. Entrenamiento de la cámara

En visión artificial esta técnica es muy utilizada para la detección de objetos, animales, matriculas, etc. En este caso queremos clasificar y detectar un dado. De esta forma no hará falta utilizar la técnica de detección del cuadrado del dado ya que se sabrá cuando habrá un dado o un objeto distinto. En la Figura 29 se muestran los dados que se han usado para realizar este entrenamiento, que serán los utilizados para demostrar el funcionamiento del sistema.

Figura 29. Dado de madera [13]

Para realizar esta tarea es importante conocer la siguiente función de *OpenMV*:

objeto_cascade = image.HaarCascade("/nombre.cascade", stages)

La función *HaarCascade* se encarga de clasificar objetos, para ello es necesario la creación de un archivo *.cascade*. Este archivo deberá contener la información de los objetos que queremos detectar. En *OpenMV* existen algunos clasificadores a modo de ejemplo, pero si se quiere específicamente uno, como es este caso, deberemos crear un *HaarCascade* propio. El proceso para conseguirlo es bastante complejo y se explica a continuación.

Para conseguir una clasificación de objetos se necesita una gran cantidad de imágenes positivas y negativas. A imágenes positivas nos referimos con aquellas imágenes en las que está el objeto que queremos detectar y las imágenes negativas son todos lo contrario. En las imágenes negativas es necesario que no tengan ningún elemento en común con las imágenes positivas. Además, cuanto mayor sea el número de imágenes se obtendrá un mejor resultado.

Esta clasificación no se puede realizar con nuestro programa *OpenMV* IDE, por lo que se ha utilizado un nuevo software llamado *Cascade Trainger GUI*. *Cascade Trainger GUI* es un programa que se puede utilizar para entrenar, probar y mejorar modelos de clasificadores en cascada. Utiliza una interfaz gráfica para establecer los parámetros y facilitar el uso de herramientas *OpenCV* para entrenar y probar clasificadores. En la Figura 30 se muestra el entorno de esta herramienta.

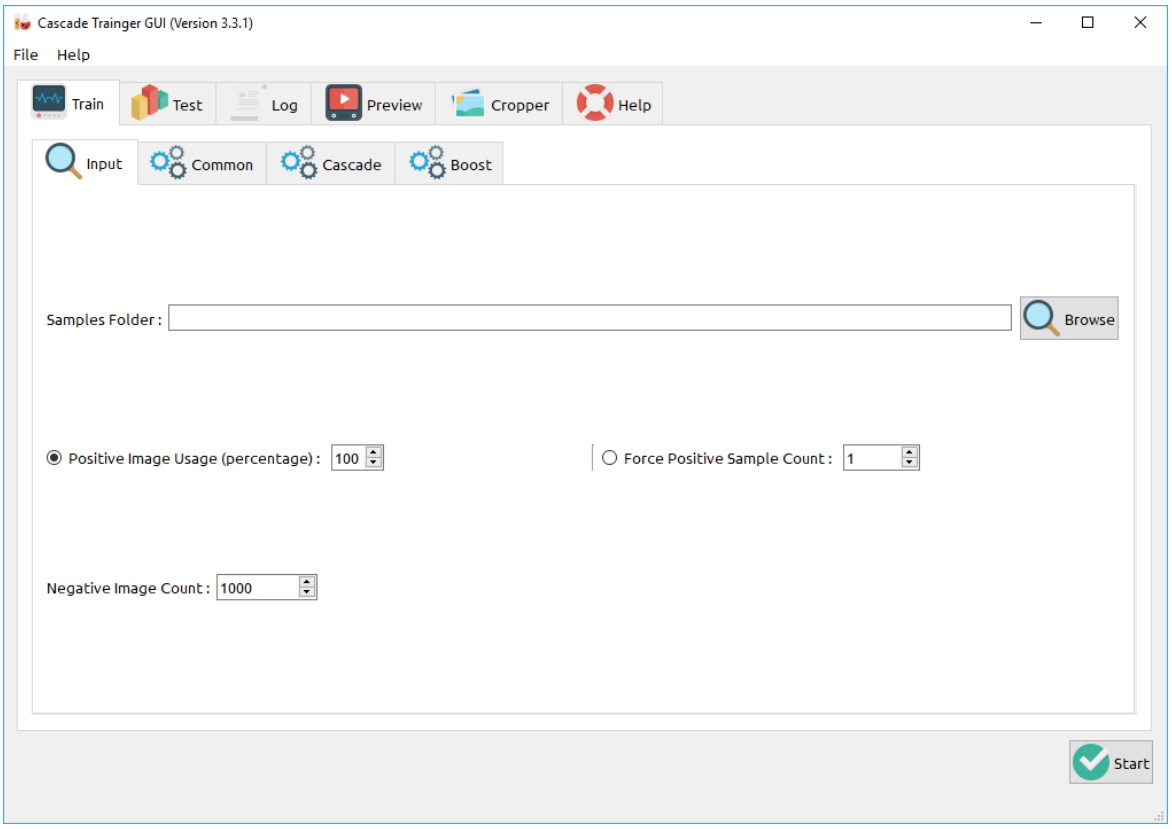


Figura 30. Entorno Cascade Trainger GUI

En este caso existe un problema ya que este software está pensado para *OpenCV*. *OpenCV* trabaja con Python y en este caso se utiliza *micropython*. Esta herramienta proporciona un archivo de salida *.xml* y como se ha contado anteriormente *OpenMV* utiliza un tipo de archivo *.cascade* por lo que se tiene que encontrar la forma de realizar una conversión de archivos, ya que no existe una herramienta para realizar entrenamientos para *OpenMV*. Más adelante se explicará cómo realizar esta conversión.

A continuación, se muestra un pequeño tutorial de uso de esta herramienta. Como primer paso hay que crear una carpeta donde almacenar nuestras imágenes positivas y negativas.

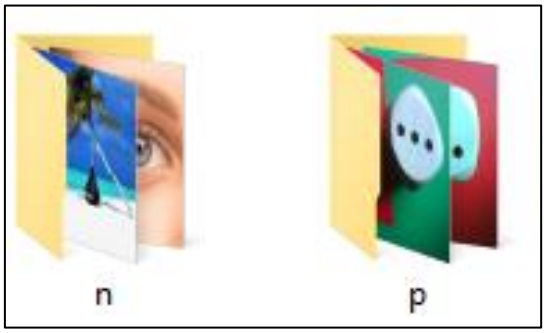


Figura 31. Imágenes P y N

En P se almacenarán todas las imágenes donde se encuentra el dado y en N todas las imágenes negativas donde no aparece. Se muestra en la Figura 32 el contenido de ambas carpetas.

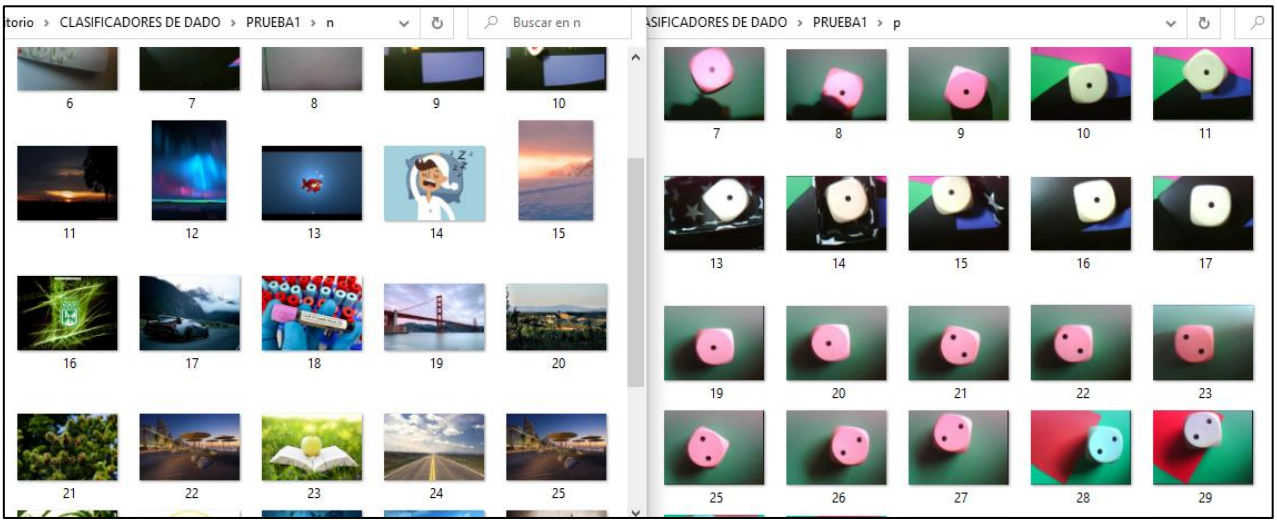


Figura 32. Carpetas p y n

Una vez creadas hay que seleccionar la ubicación en la que se encuentran estas dos carpetas y ponerla en la herramienta como se muestra en la Figura 33.

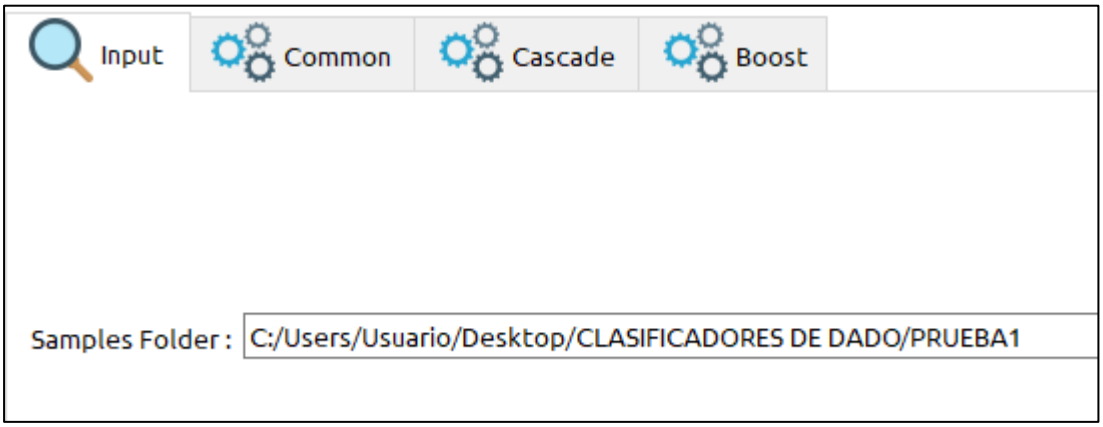


Figura 33. Ubicación de las carpetas

Después se debe indicar el número de imágenes negativas que se va a usar para que la herramienta optimice su funcionamiento al máximo. En la Figura 34 se muestra donde cambiar este parámetro.

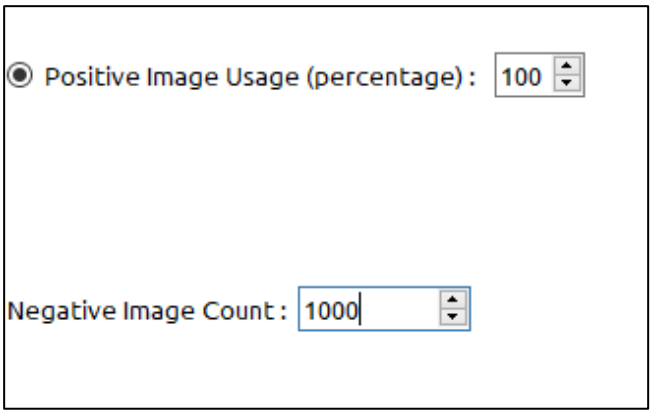


Figura 34. Número de imágenes negativas

Por defecto vienen 1000, pero se cambiará en función al número de imágenes negativas que haya en nuestra carpeta N.

Una vez hecho esto, se pueden ajustar más parámetros, aunque se van a dejar los que vienen por defecto, ya que el objetivo es crear un clasificador básico para poder usarlo. El único parámetro que se va a seleccionar (en la pestaña *Common*) es el que se muestra en la Figura 35.

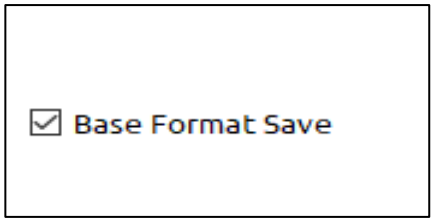


Figura 35. Parámetro a modificar

Servirá para poder realizar la conversión de la que se ha hablado anteriormente. Una vez se tienen todos los parámetros ajustados y la carpeta seleccionada pulsamos el botón START. Este proceso suele tardar un par de minutos. Cuando ha finalizado se generará en la carpeta un archivo llamado *cascade.xml* que es el que sirve para continuar con este proceso.

Una vez se tiene creado el archivo cascade.xml se utilizará Python 2.7 en Linux y el código que se muestra en el Anexo III para realizar la conversión de formatos. Para conseguirlo se utilizará una máquina virtual y desde el terminal se mandarán los siguientes comandos.

- `sudo apt update`
- `apt install python`
- `python2.7 -V`

Con estos comandos se instala Python 2.7 en nuestra máquina virtual. Una vez se tiene, se debe guardar en la carpeta personal el archivo que se encarga de convertir los formatos (*cascade_convert.py*) y el archivo que se ha generado antes (*cascade.xml*). Para finalizar se escribe el comando que se muestra en la Figura 36 (en este caso de ejemplo el archivo se llamaba unov3.xml).

```
programacion@programacion-VirtualBox:~$ python2.7 cascade_convert.py unov3.xml
Converting old XML format..
size:24x24
stages:20
features:42
rectangles:88
binary cascade generated
```

Figura 36. Comando para conversión

Una vez realizado todo este proceso ya se tiene el archivo *.cascade* para poder utilizar con la cámara *OpenMV*. A continuación, se muestran algunos resultados obtenidos para comprobar su funcionamiento en la Figura 37. El código para conseguir esto se muestra aquí:

```
#DETECCION DEL DADO
    objects=img.find_features(dado_cascade,threshold=0.85,scale_factor=1.2)
    #BUCLEQUE DIBUJA UN RECTANGULO CADA VEZ QUE DETETCTA EL DADO
    for r in objects:
        img.draw_rectangle(r,color=(0,255,255))
        print("DADO DETECTADO")
        #DIBUJAMOS UNA CADENA DE CARACTERES EN PANTALLA
        img.draw_string(10, 10, "DADO", color = (0, 255, 255), scale = 1,
mono_space = False,char_rotation = 0, char_hmirror = False, char_vflip =
False,string_rotation = 0, string_hmirror = False, string_vflip = False)
```

Código que dibuja un contorno cuando detecta la cara de un dado.

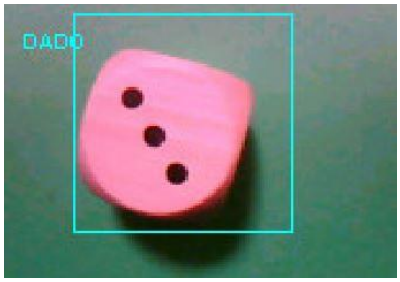


Figura 37. Detección del dado

3.5. Detección de los círculos del dado

Cuando ya se ha visto como detecta el dado de una forma correcta, se puede pasar a detectar los círculos cuando ha detectado que se está ante la cara de un dado y no cualquier otro objeto que tenga círculos. Para conseguir esta tarea se ha utilizado el algoritmo explicado anteriormente, pero ajustando los parámetros para las necesidades de esta tarea.

```
for c in img.find_circles(threshold = 2000, x_margin = 10, y_margin = 10,
r_margin = 10, r_min = 3, r_max = 10, r_step = 2):
      img.draw_circle(c.x(), c.y(), c.r(), color = (255, 0, 0))
      contador_circulos+=1
```

Código para detección de círculos cuando estamos ante la cara de un dado

Para mejorar la cuenta de los círculos se va a añadir la detección de manchas del color de los círculos del dado. Esto se consigue con la función *find.blobs()* a la cual necesita pasarle un parámetro que es la matriz del color que se quieren detectar las manchas. Esta matriz se ha conseguido usando el algoritmo del Anexo II. La matriz que se ha obtenido es la siguiente:

```
threshold = [8, 13, -3, 9, -4, 8]
```

Esta matriz es la que sale cuando en la cámara ponemos el color negro. Se puede comprobar el funcionamiento de cómo encontrar manchas de color negro con el siguiente ejemplo de código:

```
import sensor, image, time

sensor.reset()
sensor.set_pixformat(sensor.RGB565)
sensor.set_framesize(sensor.QQVGA)
sensor.skip_frames(time = 2000)
clock = time.clock()

threshold = [8, 13, -3, 9, -4, 8] # color negro.


while(True):
    clock.tick()
    img = sensor.snapshot()
    for blob in img.find_blobs([threshold], pixels_threshold=2,
area_threshold=2, merge=True, margin=5):
        img.draw_cross(blob.cx(), blob.cy())
```

Código para detección de manchas negras cuando estamos ante la cara de un dado

A continuación, se muestran los resultados obtenidos en la Figura 38 y 39.

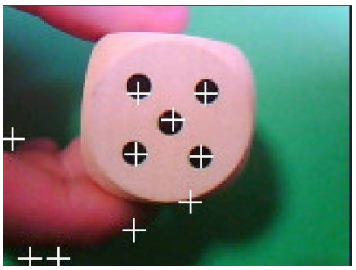


Figura 38. Find blobs ejemplo 1

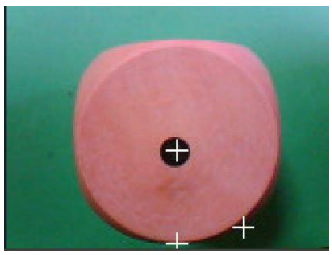


Figura 39. Find blobs ejemplo 2

Como se puede observar, al querer detectar manchas negras, las sombras también se detectan. Pero, esto no supone un problema, ya que el algoritmo principal para detectar el número del dado es la detección de círculos. Este algoritmo servirá de ayuda en la cuenta de círculos, ya que, en el momento que los círculos que se han contado sean igual al número de manchas, se tendrá de una forma muy acertada el número del dado.Para conseguir esto se han creado dos contadores: uno de círculos en el bucle *for* que detecta círculos y otro de manchas en el bucle que se encarga de encontrar manchas. Cuando el contador de círculos se va incrementando lo va guardando en una variable llamada *resultado_dado* y cuando *contador_circulos* coincida con *manchas*. La variable *resultado_dado* se guardará en *resultado_dado1* y está variable será el resultado final y correcto del dado. El algoritmo que acabamos de explicar se muestra a continuación:

```
if contador_circulos==0:
            resultado_dado=0
        if contador_circulos==1:
            resultado_dado=1
        if contador_circulos==2:
            resultado_dado=2
        if contador_circulos==3:
            resultado_dado=3
        if contador_circulos==4:
            resultado_dado=4
        if contador_circulos==5:
            resultado_dado=5
        if contador_circulos==5:
            resultado_dado=5
        if contador_circulos==6:
            resultado_dado=6
        elif contador_circulos>6:
            resultado_dado=6;
    if contador_circulos==manchas:
        resultado_dado1=resultado_dado
        print("resultado_dado",resultado_dado1)
        print("numero_manchas",manchas)
    contador_circulos=0
    color=0
    manchas=0
```

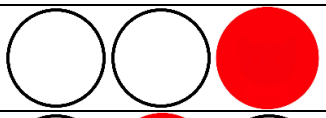
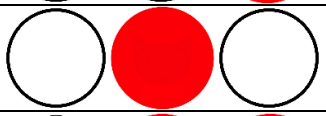
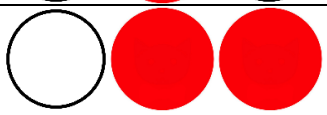
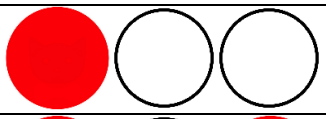
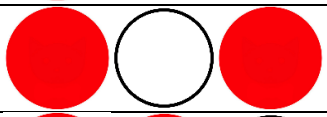
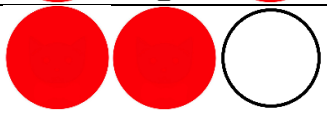
Código que almacena el resultado de la cara del dado

Por último, es necesario reiniciar a cero las variables para que estas no se incrementen hasta un número muy grande, ya que no deberían sobrepasar el número 6 (número máximo del dado).

3.6. Representación con *display*

Cuando ya se ha conseguido tener una cuenta bastante fiable y con poca probabilidad de fallo se puede pasar a representar este número mediante LEDs (en binario) o con un *display* de 7 segmentos. En primer lugar, se trabajó con LEDs, para representar el número en binario. Los números que hay que representar van del 1 al 6 por lo que con 3 dígitos se podrá conseguir esta tarea. Para poder entenderlo mejor se muestra una tabla con el resultado del dado, el código correspondiente en binario y los pines que se asigna a cada dígito binario.

Tabla 1. Código binario en función del resultado del dado

RESULTADO DADO	PIN P3 **MSB**	PIN P8	PIN P7 **LSB**	REPRESENTACIÓN LEDS	REPRESENTACION DISPLAY 7SEG
	**0**	**0**	**1**		
	**0**	**1**	**0**		
	**0**	**1**	**1**		
	**1**	**0**	**0**		
	**1**	**0**	**1**		
	**1**	**1**	**0**		

A continuación, se muestra la configuración de los tres pines elegidos:

```
pin_p7 = Pin('P7', Pin.OUT_PP, Pin.PULL_UP)
pin_p8 = Pin('P8', Pin.OUT_PP, Pin.PULL_UP)
pin_p3 = Pin('P3', Pin.OUT_PP, Pin.PULL_UP)
```

Como se ve en la configuración, se han configurado los pines como salida, en la Figura 40 se muestran los pines del decodificador (P7 se conecta con A, P8 con B y P3 con C).

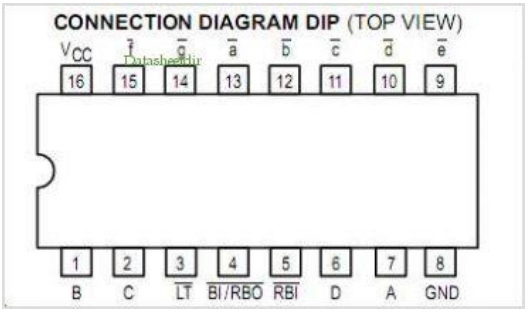


Figura 40. Diagrama de conexiones del decodificador [15]

Una vez se tienen los pines configurados se pueden conectar a los LEDs. Este ejemplo de los LEDs no se muestra, ya que, es más interesante mostrar el funcionamiento con el *display* de 7 segmentos. Antes de explicar esta configuración y conexionado del *display* se muestra el algoritmo que pone los pines a nivel alto ('1') o nivel bajo ('0') en función al resultado del dado que tengamos (se muestra solo el ejemplo del número 1).

```
if resultado_dado1==1: #código binario--> 001
    pin_p7.high()
    pin_p8.low()
    pin_p3.low()
```

Para poder conectar estos pines al *display* es necesario el uso de un decodificador. En este caso se ha elegido el modelo SN74LS47N de Texas Instruments. Hay que decir, que el *display* de 7 segmentos que se ha usado es de ánodo común, como se muestra en la Figura 41, hay que alimentarlo a *Vcc*.

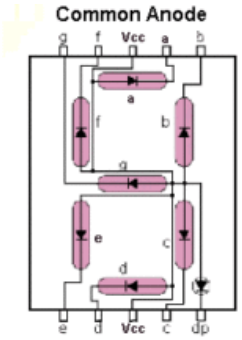


Figura 41. Display 7 segmentos de ánodo común [14]

El conexionado de ambos dispositivos, el *display* y el decodificador, se muestra en la Figura 42 (esta imagen sirve a modo de ejemplo, la conexión real se muestra en el Anexo IV).

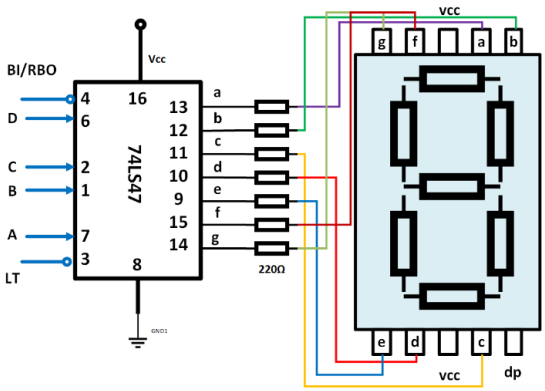


Figura 42. Conexión de decodificador y display [16]

3.7. Filtros y correcciones de imagen

En muchas situaciones se pueden encontrar problemas de luz. Uno de ellos es la falta de luz, la cual se podrá solucionar por medio de algún tipo de filtro de imagen que ayude a mejorar las características de los colores, el brillo, el contraste, etc. En temas de imagen o fotografía existe una corrección conocida como corrección *gamma*.

La corrección *gamma* sirve para codificar o decodificar luminosidad en una imagen. Este parámetro indica cuanto se oscurecerá o se aclarará una imagen. Es por esto, que, cuando se tenga una luminosidad baja y la imagen se oscurezca la cámara no podrá detectar bien el dado, por lo que esta corrección solucionará este problema. Al utilizar la aplicación en zonas interiores, será casi imposible tener el caso contrario (una luz del sol muy potente que impida ver la imagen), por lo que, el caso que hay que solucionar es la aparición de sombras.

La constante *gamma* [γ] siempre tendrá un valor positivo. Cuando, su valor está por debajo de 1, se denomina *gamma* de codificación y cuando está por encima *gamma* de decodificación. Cuando su valor es la unidad, es como si no aplicamos corrección alguna a la imagen, ya que veríamos la imagen original. En la Figura 43 se muestra un ejemplo de esta corrección.

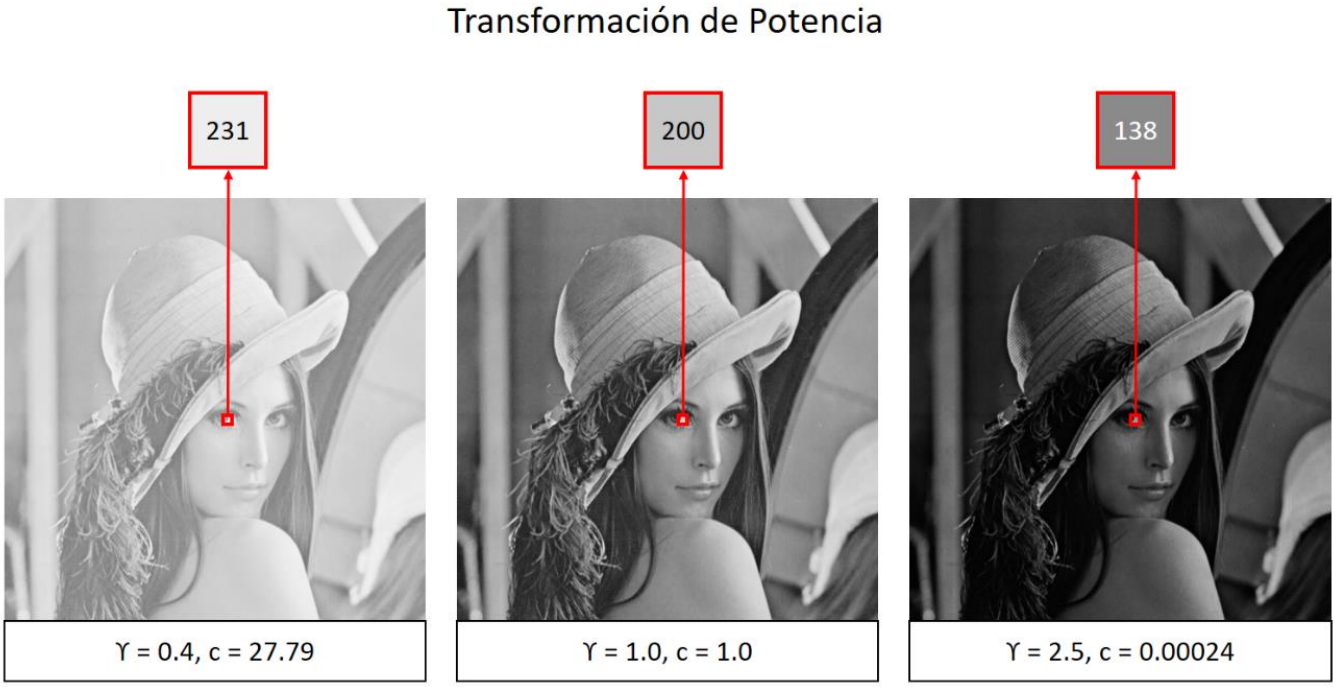


Figura 43. Corrección gamma aplicada a un retrato [17]

Una vez conocido el funcionamiento de esta técnica, el siguiente paso es llevar esto a la *OpenMV*. En primer lugar, hay que estudiar los parámetros de la imagen y saber a partir de que, punto la imagen está oscura y el dado no puede ser reconocido. Para ello se ha utilizado el siguiente comando:

```
stats=img.get_statistics() #ESTADISTICAS DEL HISTOGRAMA DE LA IMAGEN
```

Con el cual se puede analizar todos los parámetros de nuestro histograma. Una vez estudiados se ha visto que cuando se oscurece la imagen, la 'media' se decrementa considerablemente. Por lo tanto, este será el parámetro de la imagen en el cual habrá que fijarse para aplicar la corrección *gamma*. A continuación, se muestra el algoritmo utilizado para resolver este problema.

```
#PARAMETROS DEL FILTRO DE LA IMAGEN
#CON ESTOS PARAMETROS LA IMAGEN ES ORIGINAL
#COMO SI NO EXISTIERA NINGUN FILTRO O CORRECION
filtro=1.0
contraste=1.0
brillo=0.0
#CAPTURA IMAGEN Y LA DEVUELVE
    img = sensor.snapshot().gamma_corr(gamma = filtro, contrast = contraste,
brightness = brillo)
    #CONDICION APLICADA CUANDO EXISTE BAJA LUMINOSIDAD
    if mean<25:
        #PARAMETROS DEL FILTRO DE LA IMAGEN
        filtro=2.0
        contraste=1.0
        brillo=0.0
        #VALOR UMBRAL PARA DETECCION DE CIRCULOS
        cir_threshold=800
        stats=img.get_statistics() #ACTUALIZA LAS ESTADISTICAS DEL HISTOGRAMA
DE LA IMAGEN
        mean=stats.l_mean() #VALOR DE LA MEDIA DE LA IMAGEN
        mean=mean-25
        print('Filtro gamma en accion')
        threshold=[30, 36, -12, 8, 1, 27] #VALORES DE UMBRAL PARA COLOR NEGRO
CUANDO SI HAY CORRECION GAMMA
    #SI NO HAY BAJA LUMINOSIDAD NO APLICAMOS FILTROS
    else:
        #PARAMETROS DEL FILTRO DE LA IMAGEN
        filtro=1.0
        contraste=1.0
        brillo=0.0
        #VALOR UMBRAL PARA DETECCION DE CIRCULOS
        cir_threshold=2000
        stats=img.get_statistics() #ACTUALIZA LAS ESTADISTICAS DEL HISTOGRAMA
DE LA IMAGEN
        mean=stats.l_mean() #VALOR DE LA MEDIA DE LA IMAGEN
        threshold = [8, 13, -3, 9, -4, 8] #VALORES DE UMBRAL PARA COLOR NEGRO
CUANDO NO HAY CORRECION GAMMA
```

Código que aplica una corrección gamma cuando las condiciones de luz son bajas.

NOTA: Hay que destacar que cuando se introduce el filtro el valor de la media se incrementa en aproximadamente 25 unidades si no se da la orden de restar dicha cantidad a la media estará continuamente aplicando la corrección gamma y quitándola a modo de ráfaga.

Por último, se muestra en la Figura 44 el resultado de aplicar esta corrección y como, cuando se aplica, se pueden volver a detectar los círculos de forma correcta.

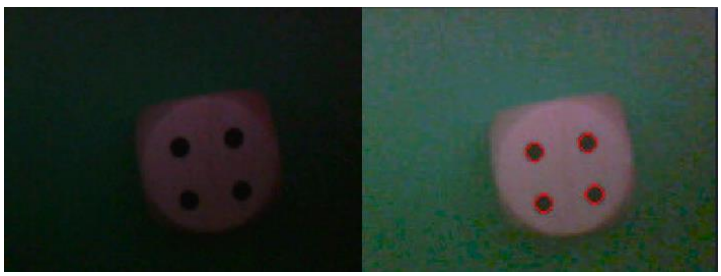


Figura 44. Resultado de aplicar la corrección gamma

Cuando en uno de los apartados anteriores se ha hablado de la matriz de colores para detectar manchas, en este caso, esa matriz es distinta. Siguiendo los mismos pasos contados anteriormente la matriz resultante es:

```
threshold=[30, 36, -12, 8, 1, 27] #VALORES DE UMBRAL PARA COLOR NEGRO CUANDO SI
HAY CORRECION GAMMA
```

3.8. Resultados experimentales

En las Figuras 45 y 46 se muestran los resultados finales del funcionamiento del sistema completo que se encarga de reconocer la cara de un dado y representar el resultado en un *display*. También aplica un filtro cuando las condiciones de luz son bajas, para que de esta forma el sistema completo siga funcionando con normalidad.

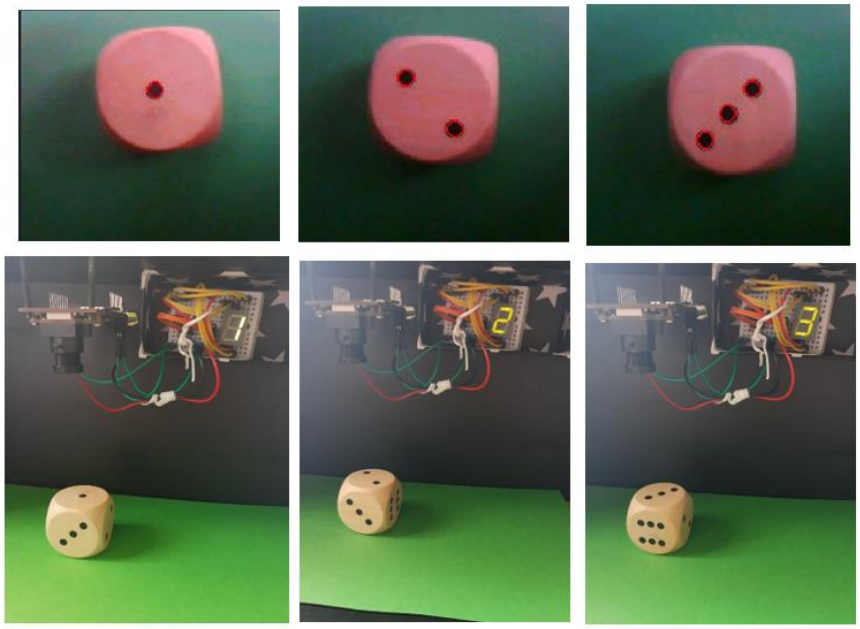


Figura 45. Resultados finales (1-3)

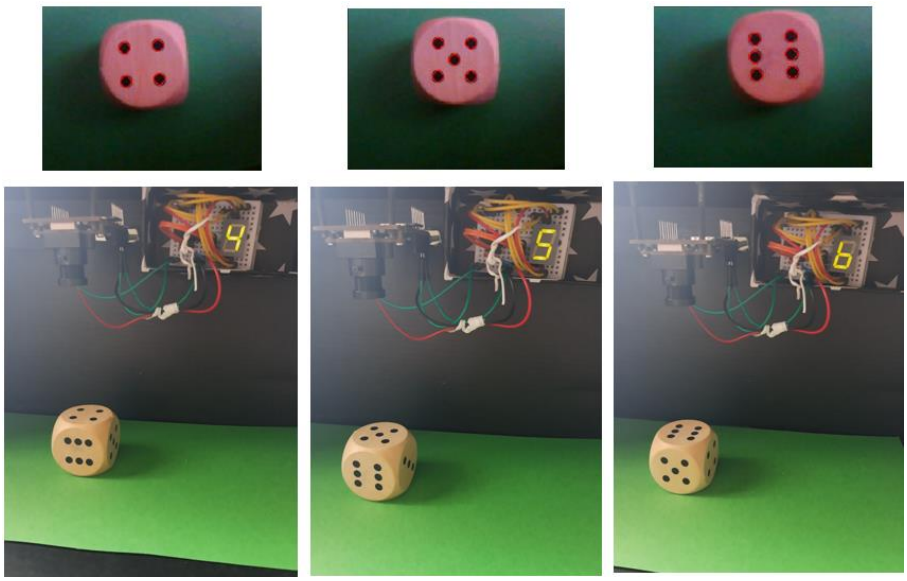


Figura 46. Resultados finales (4-6)

Por último, en la Figura 47 se muestra el resultado de aplicar el filtro *gamma* al bajar las condiciones de luz para poder ver la respuesta del sistema.

Figura 47. Resultado final con corrección gamma

PRESUPUESTO

En este apartado se van a estudiar todos los costes del proyecto.

COSTE DEL HARDWARE

Tabla 2. Coste del hardware

MATERIAL	PRECIO (€/ud)	CANTIDAD	COSTE TOTAL
Ordenador portátil	629€	1	(uso=4 meses) 40€
OpenMV Cam H7	76,21€	1	76,21€
Dado de madera	2,95€	1	2,95€
Cable USB-microUSB	1,95€	1	1,95€
Protoboard	1,60€	2	3,20€
Cables de conexión	0,03€	60	1,80€
Display BCD 7 segmentos	1,80€	1	1,80€
SN74LS47 (decodificador)	1,75€	1	1,75€
Tarjeta microSD	4,95€	1	4,95€
LEDs para pruebas	0,57€	3	1,71€
TOTAL			136,32€

COSTE DE MANO DE OBRA

Tabla 3. Coste de la mano de obra

CONCEPTO	CATEGORIA	€/HORA	HORAS TOTALES	COSTE TOTAL
Ángel Rayo Fernández	Ingeniero de investigación	35	180	6.300€

COSTE DE EJECUCIÓN DEL MATERIAL

Tabla 4. Coste de ejecución del material

CONCEPTO	COSTE TOTAL
COSTE DEL MATERIAL	136,32€
COSTE DE MANO DE OBRA	6.300€
TOTAL	6.436,32€

PRESUPUESTO DE EJECUCIÓN POR CONTRATA

Tabla 5. Coste de ejecución por contrata

CONCEPTO	COSTE TOTAL
COSTE DE EJECUCIÓN DEL MATERIAL	6.436,32€
GASTOS GENERALES (17%)	1.094,18€
BENEFICIO INDUSTRIAL (6%)	386,18€
TOTAL	7.916,68€

PRESUPUESTO TOTAL

Tabla 6. Presupuesto final

CONCEPTO	COSTE TOTAL
EJECUCIÓN POR CONTRATA	7.916,68€
I.V.A (21%)	1.662,51€
TOTAL	9.579,19€

BIBLIOGRAFÍA

[1] Documentación de *OpenMV*: https://docs.openmv.io/openmvcam/quickref.html

[2] Pyboard: https://store.micropython.org/product/PYBv1.1

[3] Pixy 2.0: https://tienda.bricogeek.com/sensores-imagen/1238-camara-pixy-2.html

[4] Aplicaciones de visión artificial en la industria: https://www.atriainnovation.com/siete-aplicaciones-vision-artificial/

[5] Imagen hiperespectral: https://www.interempresas.net/Industria-Carnica/Articulos/149980-Aplicaciones-en-la-industria-carnica-de-la-vision-hiperespectral.html

[6] Imágenes cámaras OV: https://es.aliexpress.com/item/32978201493.html?acnt=439-079-4345&aff_platform=aaf&ds_e_device=c&albcp=11405488241&ds_e_product_id=es32978201493&ds_url_v=2&ds_dest_url=https%3A%2F%2Fs.click.aliexpress.com%2Fdeep_link.htm%3Faff_short_key%3DUneMJZVf&ds_e_product_group_id=536572975094&sk=UneMJZVf&ds_e_adid=473728314619&terminal_id=b7242852194045afa92f4fe60cd63160&tmLog=new_Detail&needSmbHouyi=false&albbt=Google_7_shopping&ds_e_product_channel=online&src=google&ds_e_product_country=ES&aff_fcid=4f3efed3b46d40628bd515a3dbd2647a-1628761747745-05760-UneMJZVf&gclid=CjwKCAjwjdOIBhA_EiwAHz8xmxna94svjD5cfaka_EGTSCnhfT6AjXwUBa-4OplmUDYMq7GmyI_oLhoCqWcQAvD_BwE&albag=115612532150&aff_fsk=UneMJZVf&albch=shopping&ds_e_network=u&albagn=888888&ds_e_product_language=es&isSmbAutoCall=false&ds_e_product_merchant_id=106352398&aff_trace_key=4f3efed3b46d40628bd515a3dbd2647a-1628761747745-05760-UneMJZVf&gclsrc=aw.ds

[7] *OpenMV Cam H7*: https://www.open-electronics.org/openmv-cam-h7-the-open-source-micropython-powered-machine-vision-camera/

[8] Diferencias en procesadores *Cortex*: https://programmerclick.com/article/28461414149/

[9] Tabla comparación de *Cortex*: file:///C:/Users/Usuario/Downloads/Arm%20Cortex-M%20Comparison%20Table_v3.pdf

[10] Entorno de Desarrollo: https://openmv.io/pages/download

[11] Datasheet *OpenMV Cam H7*: https://cdn.sparkfun.com/assets/a/3/f/d/9/OpenMV-H7_Datasheet.pdf

[12] *Find blobs*: https://www.researchgate.net/figure/The-algorithm-must-find-blobs-without-knowing-how-many-there-are-It-first-finds-all_fig3_259660687

Cascade Trainger GUI: https://amin-ahmadi.com/cascade-trainer-gui/

[13] Dado de madera: https://www.amazon.es/Dados-madera-unidades-longitud-30/dp/B005M0GJC6

[14] Imágenes *display* 7 segmentos: https://www.cuantex.com/2016/10/11/contador-7-segmentos-cuantex/

[15] Datasheet SN74LS47: http://j5d2v7d7.stackpathcdn.com/wp-content/uploads/2019/03/SN74LS47.pdf

[16] Decodificador y *display*: https://www.diloentutospc.com/conectar-display-7-segmentos/

[17] Corrección *gamma*: https://bryanmed.github.io/correccionGamma/

Corrección *gamma*: http://www.efectohd.com/2008/09/entender-la-correcion-de-gamma.html

ANEXOS

ANEXO I. CÓDIGO DEL PROYECTO

```
# Aplicacion
#
# Reconocimiento y representacion
# de un dado utilizando tecnicas
# de vision artificial
# Creador: Angel Rayo

#DEFINICION DE LIBRERIAS
import sensor, image, time, pyb
from pyb import LED
from pyb import Pin
#CONFIGURACION DE LOS PINES UTILIZADOS
pin_p7 = Pin('P7', Pin.OUT_PP, Pin.PULL_UP) #LSB
pin_p8 = Pin('P8', Pin.OUT_PP, Pin.PULL_UP)
pin_p3 = Pin('P3', Pin.OUT_PP, Pin.PULL_UP) #MSB
#DEFINICION DE VARIABLES UTILIZADAS
resultado_dado_INICIAL=0; #resultado del dado inicial(poco fiable)
resultado_dado_FINAL=0;   #resultado del dado final(muy fiable)
contador_circulos=0;      #variable que almacena la cuenta de circulos
detector_color=0          #variable que detecta una mancha de color
manchas=0                 #numero de manchas detectadas
mean=100                  #valor de la media de la imagen (se inicializa a 100 pero cambia
despues)
cir_threshold=2000        #valor incial del umbral para detection de circulos
#PARAMETROS DEL FILTRO DE LA IMAGEN
#CON ESTOS PARAMETROS LA IMAGEN ES ORIGINAL
#COMO SI NO EXISTIERA NINGUN FILTRO O CORRECION
filtro=1.0
contraste=1.0
brillo=0.0
#DEFINICION DE UMBRALES
threshold = [8, 13, -3, 9, -4, 8] #umbral que detecta manchas negras

#inicializacion del sensor de la camara
sensor.reset()
sensor.set_pixformat(sensor.RGB565) # grayscale is faster
sensor.set_framesize(sensor.QQVGA)  #AL PONER UNA BAJA RESOLUCION OBTENEMOS MAYOR
VELOCIDAD
sensor.skip_frames(time = 2000)
#clasificador de objetos
dado_cascade=image.HaarCascade("/unov3.cascade")
#INICIALIZA EL RELOJ
clock = time.clock()

while(True):
    clock.tick()
    #CAPTURA IMAGEN Y LA DEVUELVE
    img = sensor.snapshot().gamma_corr(gamma = filtro, contrast = contraste, brightness =
brillo)
    #CONDICION APLICADA CUANDO EXISTE BAJA LUMINOSIDAD
    if mean<25:
        #PARAMETROS DEL FILTRO DE LA IMAGEN
        filtro=2.0
        contraste=1.0
        brillo=0.0
        #VALOR UMBRAL PARA DETECCION DE CIRCULOS
        cir_threshold=800
        stats=img.get_statistics() #ACTUALIZA LAS ESTADISTICAS DEL HISTOGRAMA DE LA IMAGEN
        mean=stats.l_mean() #VALOR DE LA MEDIA DE LA IMAGEN
        mean=mean-25
        print('Filtro gamma en accion')
        threshold=[30, 36, -12, 8, 1, 27] #VALORES DE UMBRAL PARA COLOR NEGRO CUANDO SI
HAY CORRECION GAMMA
    #SI NO HAY BAJA LUMINOSIDAD NO APLICAMOS FILTROS
```

```
    else:
        #PARAMETROS DEL FILTRO DE LA IMAGEN
        filtro=1.0
        contraste=1.0
        brillo=0.0
        #VALOR UMBRAL PARA DETECCION DE CIRCULOS
        cir_threshold=2000
        stats=img.get_statistics() #ACTUALIZA LAS ESTADISTICAS DEL HISTOGRAMA DE LA IMAGEN
        mean=stats.l_mean() #VALOR DE LA MEDIA DE LA IMAGEN
        threshold = [8, 13, -3, 9, -4, 8] #VALORES DE UMBRAL PARA COLOR NEGRO CUANDO NO
HAY CORRECION GAMMA
    #DETECCION DEL DADO
    objects=img.find_features(dado_cascade,threshold=0.85,scale_factor=1.2)
    #BUCLEQUE DIBUJA UN RECTANGULO CADA VEZ QUE DETETCTA EL DADO
    for r in objects:
        img.draw_rectangle(r,color=(0,255,255))
        print("DADO DETECTADO")
        #DIBUJAMOS UNA CADENA DE CARACTERES EN PANTALLA
        img.draw_string(10, 10, "DADO", color = (0, 255, 255), scale = 1, mono_space =
False,
                        char_rotation = 0, char_hmirror = False, char_vflip = False,
                        string_rotation = 0, string_hmirror = False, string_vflip = False)
    #BUCLE QUE SE ENCARGA DE DETECTAR MANCHAS
    for blob in img.find_blobs([threshold], pixels_threshold=20, area_threshold=50,
merge=True, margin=10):
        detector_color=1
        manchas+=1

    #SI EL DETECTOR DE MANCHAS FUNCIONA, COMENZAMOS A DETECTAR CIRCULOS
    if detector_color==1:
        #BUCLE QUE SE ENCARGA DE ENCONTRAR CIRCULOS EN LA IMAGEN
        for c in img.find_circles(threshold = cir_threshold, x_margin = 10, y_margin = 10,
r_margin = 10,
                r_min = 1, r_max = 5, r_step = 2):
            img.draw_circle(c.x(), c.y(), c.r(), color = (255, 0, 0))
            contador_circulos+=1
        #CUENTA LOS CIRCULOS QUE DETECTA EN LA IMAGEN
        if contador_circulos==0:
            resultado_dado_INICIAL=0
        if contador_circulos==1:
            resultado_dado_INICIAL=1
        if contador_circulos==2:
            resultado_dado_INICIAL=2
        if contador_circulos==3:
            resultado_dado_INICIAL=3
        if contador_circulos==4:
            resultado_dado_INICIAL=4
        if contador_circulos==5:
            resultado_dado_INICIAL=5
        if contador_circulos==6:
            resultado_dado_INICIAL=6
        elif contador_circulos>6:
            resultado_dado_INICIAL=6;

    #AJUSTE PARA CONSEGUIR MAYOR PRECISION
    if contador_circulos==manchas:
        resultado_dado_FINAL=resultado_dado_INICIAL
        #IMPRIMIMOS POR PANTALLA EL RESULTADO
        print("resultado_dado",resultado_dado_FINAL)
        print("numero_manchas",manchas)

    #REPRESENTACIÓN DEL DADO (CODIGO BINARIO)
    if resultado_dado_FINAL==1: #CODIGO BINARIO--> 001
        pin_p7.high()
        pin_p8.low()
        pin_p3.low()
    elif resultado_dado_FINAL==2: #CODIGO BINARIO--> 010
        pin_p7.low()
        pin_p8.high()
```

```
    pin_p3.low()
elif resultado_dado_FINAL==3: #CODIGO BINARIO--> 011
    pin_p7.high()
    pin_p8.high()
    pin_p3.low()
elif resultado_dado_FINAL==4: #CODIGO BINARIO--> 100
    pin_p7.low()
    pin_p8.low()
    pin_p3.high()
elif resultado_dado_FINAL==5: #CODIGO BINARIO--> 101
    pin_p7.high()
    pin_p8.low()
    pin_p3.high()
elif resultado_dado_FINAL==6: #CODIGO BINARIO--> 110
    pin_p7.low()
    pin_p8.high()
    pin_p3.high()
elif resultado_dado_FINAL==0: #CODIGO BINARIO--> 000
    pin_p7.low()
    pin_p8.low()
    pin_p3.low()

#REINICIALIZAMOS NUESTRAS VARIABLES
contador_circulos=0
detector_color=0
manchas=0
```

ANEXO II. CÓDIGO PARA OBTENER MATRIZ DE MANCHAS.

```
# Automatic RGB565 Color Tracking Example
#
# This example shows off single color automatic RGB565 color tracking using the OpenMV
Cam.

import sensor, image, time
print("Letting auto algorithms run. Don't put anything in front of the camera!")

sensor.reset()
sensor.set_pixformat(sensor.RGB565)
sensor.set_framesize(sensor.QVGA)
sensor.skip_frames(time = 2000)
sensor.set_auto_gain(False) # must be turned off for color tracking
sensor.set_auto_whitebal(False) # must be turned off for color tracking
clock = time.clock()

# Capture the color thresholds for whatever was in the center of the image.
r = [(320//2)-(50//2), (240//2)-(50//2), 50, 50] # 50x50 center of QVGA.

print("Auto algorithms done. Hold the object you want to track in front of the camera in
the box.")
print("MAKE SURE THE COLOR OF THE OBJECT YOU WANT TO TRACK IS FULLY ENCLOSED BY THE BOX!")
for i in range(60):
    img = sensor.snapshot()
    img.draw_rectangle(r)

print("Learning thresholds...")
threshold = [50, 50, 0, 0, 0, 0] # Middle L, A, B values.
for i in range(60):
    img = sensor.snapshot()
    hist = img.get_histogram(roi=r)
    lo = hist.get_percentile(0.01) # Get the CDF of the histogram at the 1% range (ADJUST
AS NECESSARY)!
    hi = hist.get_percentile(0.99) # Get the CDF of the histogram at the 99% range (ADJUST
AS NECESSARY)!
    # Average in percentile values.
    threshold[0] = (threshold[0] + lo.l_value()) // 2
    threshold[1] = (threshold[1] + hi.l_value()) // 2
    threshold[2] = (threshold[2] + lo.a_value()) // 2
    threshold[3] = (threshold[3] + hi.a_value()) // 2
    threshold[4] = (threshold[4] + lo.b_value()) // 2
    threshold[5] = (threshold[5] + hi.b_value()) // 2
    for blob in img.find_blobs([threshold], pixels_threshold=100, area_threshold=100,
merge=True, margin=10):
        img.draw_rectangle(blob.rect())
        img.draw_cross(blob.cx(), blob.cy())
        img.draw_rectangle(r)

print("Thresholds learned...")
print("Tracking colors...")

while(True):
    clock.tick()
    img = sensor.snapshot()
    for blob in img.find_blobs([threshold], pixels_threshold=100, area_threshold=100,
merge=True, margin=10):
        img.draw_rectangle(blob.rect())
        img.draw_cross(blob.cx(), blob.cy())
    print(clock.fps())
    print(threshold)
```

ANEXO III. CÓDIGO DE LA CONVERSIÓN .XML a .CASCADE

(https://github.com/openmv/openmv/blame/master/ml/haarcascade/cascade_convert.py)

```
#!/usr/bin/env python2
# This file is part of the OpenMV project.
#
# Copyright (c) 2013-2021 Ibrahim Abdelkader <iabdalkader@openmv.io>
# Copyright (c) 2013-2021 Kwabena W. Agyeman <kwagyeman@openmv.io>
#
# This work is licensed under the MIT license, see the file LICENSE for details.
#
# Haar Cascade binary converter.

import sys,os
import struct
import argparse
from xml.dom import minidom

def cascade_info_universal(path):
    xmldoc = minidom.parse(path)
    old_format = xmldoc.getElementsByTagName('stageNum').length == 0
    if old_format:
        print("Parsing old XML format..")
        cascade_info_old(path)
    else:
        print("Parsing new XML format..")
        cascade_info(path)

def cascade_info(path):
    #parse xml file
    xmldoc = minidom.parse(path)

    n_stages = int(xmldoc.getElementsByTagName('stageNum')[0].childNodes[0].nodeValue)

    # read stages
    stages_elements = xmldoc.getElementsByTagName('stages')
    stages = []
    for node in stages_elements[0].childNodes:
        if node.nodeType == 1:

stages.append(int(node.getElementsByTagName('maxWeakCount')[0].childNodes[0].nodeValue))
    stage_threshold = xmldoc.getElementsByTagName('stageThreshold')[0:n_stages]

    # total number of features
    n_features = sum(stages)

    #read rectangles
    feature = xmldoc.getElementsByTagName('rects')[0:n_features]

    #read cascade size
    size = [int(xmldoc.getElementsByTagName('width')[0].childNodes[0].nodeValue),
int(xmldoc.getElementsByTagName('height')[0].childNodes[0].nodeValue)]

    n_rectangles = 0
    for f in feature:
        rects = f.getElementsByTagName('_')
        n_rectangles = n_rectangles + len(rects)

    #print some cascade info
    print("size:%dx%d"%(size[0], size[1]))
    print("stages:%d"%len(stages))
    print("features:%d"%n_features)
    print("rectangles:%d"%n_rectangles)

def cascade_info_old(path):
    #parse xml file
    xmldoc = minidom.parse(path)
```

```
    trees = xmldoc.getElementsByTagName('trees')
    n_stages = len(trees)

    # read stages
    stages = [len(t.childNodes)//2 for t in trees][0:n_stages]
    stage_threshold = xmldoc.getElementsByTagName('stage_threshold')[0:n_stages]

    # total number of features
    n_features = sum(stages)
    # read features threshold
    threshold = xmldoc.getElementsByTagName('threshold')[0:n_features]
    #theres one of each per feature
    alpha1 = xmldoc.getElementsByTagName('left_val')[0:n_features]
    alpha2 = xmldoc.getElementsByTagName('right_val')[0:n_features]

    #read rectangles
    feature = xmldoc.getElementsByTagName('rects')[0:n_features]

    #read cascade size
    size = list(map(int,
xmldoc.getElementsByTagName('size')[0].childNodes[0].nodeValue.split()))

    n_rectangles = 0
    for f in feature:
        rects = f.getElementsByTagName('_')
        n_rectangles = n_rectangles + len(rects)

    #print some cascade info
    print("size:%dx%d"%(size[0], size[1]))
    print("stages:%d"%len(stages))
    print("features:%d"%n_features)
    print("rectangles:%d"%n_rectangles)

def cascade_binary_universal(path, n_stages, name):
    xmldoc = minidom.parse(path)
    old_format = xmldoc.getElementsByTagName('stageNum').length == 0
    if old_format:
        print("Converting old XML format..")
        cascade_binary_old(path, n_stages, name)
    else:
        print("Converting new XML format..")
        cascade_binary(path, n_stages, name)

def cascade_binary(path, n_stages, name):
    #parse xml file
    xmldoc = minidom.parse(path)

    max_stages = int(xmldoc.getElementsByTagName('stageNum')[0].childNodes[0].nodeValue)
    if n_stages > max_stages:
        raise Exception("The max number of stages is: %d"%(max_stages))

    if n_stages == 0:
        n_stages = max_stages

    # read stages
    stages_elements = xmldoc.getElementsByTagName('stages')
    stages = []
    for node in stages_elements[0].childNodes:
        if node.nodeType == 1:

stages.append(int(node.getElementsByTagName('maxWeakCount')[0].childNodes[0].nodeValue))
    stage_threshold = xmldoc.getElementsByTagName('stageThreshold')[0:n_stages]

    # total number of features
    n_features = int(sum(stages))

    # read features threshold
    internal_nodes = xmldoc.getElementsByTagName('internalNodes')[0:n_features]
```

```
    # theres one of each per feature
    leaf_values = xmldoc.getElementsByTagName('leafValues')[0:n_features]

    alpha1 = []
    alpha2 = []
    for val in leaf_values:
        alpha1.append(val.childNodes[0].nodeValue.split()[0])
        alpha2.append(val.childNodes[0].nodeValue.split()[1])

    # read rectangles
    feature = xmldoc.getElementsByTagName('rects')[0:n_features]

    # read cascade size
    size = [int(xmldoc.getElementsByTagName('width')[0].childNodes[0].nodeValue),
int(xmldoc.getElementsByTagName('height')[0].childNodes[0].nodeValue)]

    # open output file with the specified name or xml file name
    if not name:
        name = os.path.basename(path).split('.')[0]
    fout = open(name+".cascade", "wb")

    n_rectangles = 0
    for f in feature:
        rects = f.getElementsByTagName('_')
        n_rectangles = n_rectangles + len(rects)

    # write detection window size
    fout.write(struct.pack('i', size[0]))
    fout.write(struct.pack('i', size[1]))

    # write num stages
    fout.write(struct.pack('i', len(stages)))

    # write num feat in stages
    for s in stages:
        fout.write(struct.pack('B', s)) # uint8_t

    # write stages thresholds
    for t in stage_threshold:
        fout.write(struct.pack('h', int(float(t.childNodes[0].nodeValue)*256))) #int16_t

    # write features threshold 1 per feature
    for t in internal_nodes:
        fout.write(struct.pack('h',
int(float(t.childNodes[0].nodeValue.split()[3])*4096))) #int16_t

    # write alpha1 1 per feature
    for a in alpha1:
        fout.write(struct.pack('h', int(float(a)*256))) #int16_t

    # write alpha2 1 per feature
    for a in alpha2:
        fout.write(struct.pack('h', int(float(a)*256))) #int16_t

    # write num_rects per feature
    for f in internal_nodes:
        idx = int(f.childNodes[0].nodeValue.split()[2])
        rects = feature[idx].getElementsByTagName('_')
        fout.write(struct.pack('B', len(rects))) # uint8_t

    # write rects weights 1 per rectangle
    for f in internal_nodes:
        idx = int(f.childNodes[0].nodeValue.split()[2])
        rects = feature[idx].getElementsByTagName('_')
        for r in rects:
            l = list(map(int, r.childNodes[0].nodeValue[:-1].split()))
            fout.write(struct.pack('b', l[4])) #int8_t NOTE: multiply by 4096
```

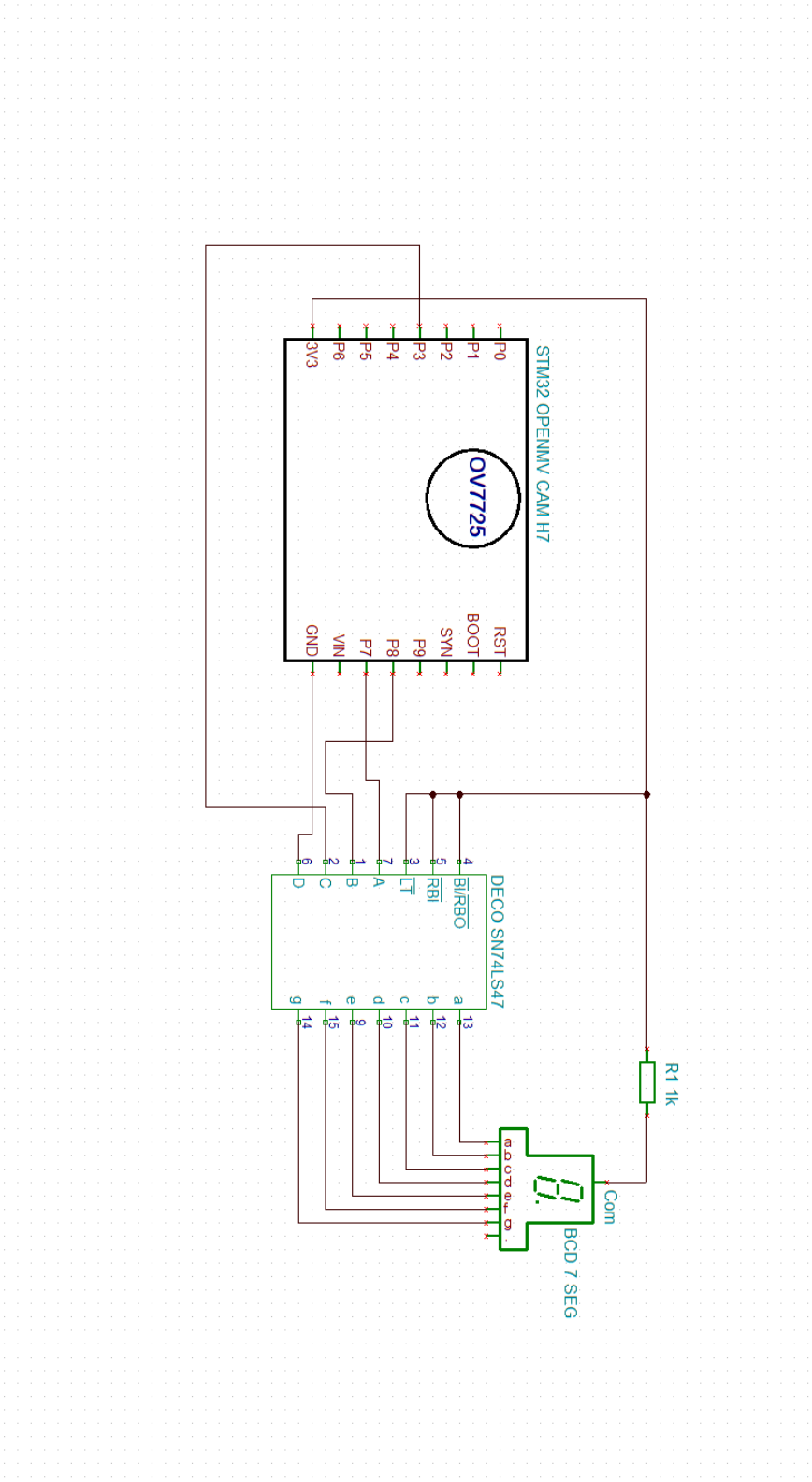
```
    # write rects
    for f in internal_nodes:
        idx = int(f.childNodes[0].nodeValue.split()[2])
        rects = feature[idx].getElementsByTagName('_')
        for r in rects:
            l = list(map(int, r.childNodes[0].nodeValue[:-1].split()))
            fout.write(struct.pack('BBBB', l[0], l[1], l[2], l[3])) #uint8_t

    # print cascade info
    print("size:%dx%d"%(size[0], size[1]))
    print("stages:%d"%len(stages))
    print("features:%d"%n_features)
    print("rectangles:%d"%n_rectangles)
    print("binary cascade generated")


def cascade_binary_old(path, n_stages, name):
    #parse xml file
    xmldoc = minidom.parse(path)

    trees = xmldoc.getElementsByTagName('trees')
    max_stages = len(trees)
    if n_stages > max_stages:
        raise Exception("The max number of stages is: %d"%(max_stages))

    if n_stages == 0:
        n_stages = max_stages

    # read stages
    stages = [len(t.childNodes)//2 for t in trees][0:n_stages]
    stage_threshold = xmldoc.getElementsByTagName('stage_threshold')[0:n_stages]

    # total number of features
    n_features = sum(stages)

    # read features threshold
    threshold = xmldoc.getElementsByTagName('threshold')[0:n_features]

    # theres one of each per feature
    alpha1 = xmldoc.getElementsByTagName('left_val')[0:n_features]
    alpha2 = xmldoc.getElementsByTagName('right_val')[0:n_features]

    # read rectangles
    feature = xmldoc.getElementsByTagName('rects')[0:n_features]

    # read cascade size
    size = list(map(int,
xmldoc.getElementsByTagName('size')[0].childNodes[0].nodeValue.split()))


    # open output file with the specified name or xml file name
    if not name:
        name = os.path.basename(path).split('.')[0]
    fout = open(name+".cascade", "wb")

    n_rectangles = 0
    for f in feature:
        rects = f.getElementsByTagName('_')
        n_rectangles = n_rectangles + len(rects)

    # write detection window size
    fout.write(struct.pack('i', size[0]))
    fout.write(struct.pack('i', size[1]))

    # write num stages
    fout.write(struct.pack('i', len(stages)))

    # write num feat in stages
    for s in stages:
```

```
            fout.write(struct.pack('B', s)) # uint8_t

        # write stages thresholds
        for t in stage_threshold:
            fout.write(struct.pack('h', int(float(t.childNodes[0].nodeValue)*256))) #int16_t

        # write features threshold 1 per feature
        for t in threshold:
            fout.write(struct.pack('h', int(float(t.childNodes[0].nodeValue)*4096))) #int16_t

        # write alpha1 1 per feature
        for a in alpha1:
            fout.write(struct.pack('h', int(float(a.childNodes[0].nodeValue)*256))) #int16_t

        # write alpha2 1 per feature
        for a in alpha2:
            fout.write(struct.pack('h', int(float(a.childNodes[0].nodeValue)*256))) #int16_t

        # write num_rects per feature
        for f in feature:
            rects = f.getElementsByTagName('_')
            fout.write(struct.pack('B', len(rects))) # uint8_t

        # write rects weights 1 per rectangle
        for f in feature:
            rects = f.getElementsByTagName('_')
            for r in rects:
                l = list(map(int, r.childNodes[0].nodeValue[:-1].split()))
                fout.write(struct.pack('b', l[4])) #int8_t NOTE: multiply by 4096

        # write rects
        for f in feature:
            rects = f.getElementsByTagName('_')
            for r in rects:
                l = list(map(int, r.childNodes[0].nodeValue[:-1].split()))
                fout.write(struct.pack('BBBB',l[0], l[1], l[2], l[3])) #uint8_t

        # print cascade info
        print("size:%dx%d"%(size[0], size[1]))
        print("stages:%d"%len(stages))
        print("features:%d"%n_features)
        print("rectangles:%d"%n_rectangles)
        print("binary cascade generated")

    def cascade_header(path, n_stages, name):
        #parse xml file
        xmldoc = minidom.parse(path)

        trees = xmldoc.getElementsByTagName('trees')
        max_stages = len(trees)
        if n_stages > max_stages:
            raise Exception("The max number of stages is: %d"%(max_stages))

        if n_stages == 0:
            n_stages = max_stages

        # read stages
        stages = [len(t.childNodes)/2 for t in trees][0:n_stages]
        stage_threshold = xmldoc.getElementsByTagName('stage_threshold')[0:n_stages]

        # total number of features
        n_features = sum(stages)

        # read features threshold
        threshold = xmldoc.getElementsByTagName('threshold')[0:n_features]

        # theres one of each per feature
        alpha1 = xmldoc.getElementsByTagName('left_val')[0:n_features]
        alpha2 = xmldoc.getElementsByTagName('right_val')[0:n_features]
```

```
    # read rectangles
    feature = xmldoc.getElementsByTagName('rects')[0:n_features]

    # read cascade size
    size = list(map(int,
xmldoc.getElementsByTagName('size')[0].childNodes[0].nodeValue.split()))

    # open output file with the specified name or xml file name
    if not name:
        name = os.path.basename(path).split('.')[0]
    fout = open(name+".h", "w")

    n_rectangles = 0
    for f in feature:
        rects = f.getElementsByTagName('_')
        n_rectangles = n_rectangles + len(rects)

    # write detection window size
    fout.write("const int %s_window_w=%d;\n" %( name, size[0]))
    fout.write("const int %s_window_h=%d;\n" %(name, size[1]))

    # write num stages
    fout.write("const int %s_n_stages=%d;\n" %(name, len(stages)))

    # write num feat in stages
    fout.write("const uint8_t %s_stages_array[]={%s};\n"
            %(name, ", ".join(str(x) for x in stages)))

    # write stages thresholds
    fout.write("const int16_t %s_stages_thresh_array[]={%s};\n"
            %(name, ", ".join(str(int(float(t.childNodes[0].nodeValue)*256)) for t in
stage_threshold)))

    # write features threshold 1 per feature
    fout.write("const int16_t %s_tree_thresh_array[]={%s};\n"
            %(name, ", ".join(str(int(float(t.childNodes[0].nodeValue)*4096)) for t in
threshold)))

    # write alpha1 1 per feature
    fout.write("const int16_t %s_alpha1_array[]={%s};\n"
            %(name, ", ".join(str(int(float(t.childNodes[0].nodeValue)*256)) for t in
alpha1)))

    # write alpha2 1 per feature
    fout.write("const int16_t %s_alpha2_array[]={%s};\n"
            %(name, ", ".join(str(int(float(t.childNodes[0].nodeValue)*256)) for t in
alpha2)))

    # write num_rects per feature
    fout.write("const int8_t %s_num_rectangles_array[]={%s};\n"
            %(name, ", ".join(str(len(f.getElementsByTagName('_'))) for f in feature)))

    # write rects weights 1 per rectangle
    rect_weights = lambda rects:", ".join(r.childNodes[0].nodeValue[:-1].split()[4] for r
in rects)
    fout.write("const int8_t %s_weights_array[]={%s};\n"
            %(name, ", ".join(rect_weights(f.getElementsByTagName('_')) for f in
feature)))


    # write rects
    rect = lambda rects:", ".join(", ".join(r.childNodes[0].nodeValue.split()[:-1]) for r
in rects)
    fout.write("const int8_t %s_rectangles_array[]={%s};\n"
            %(name, ", ".join(rect(f.getElementsByTagName('_')) for f in feature)))

    # print cascade info
    print("size:%dx%d"%(size[0], size[1]))
```

```
    print("stages:%d"%len(stages))
    print("features:%d"%n_features)
    print("rectangles:%d"%n_rectangles)
    print("C header cascade generated")

def main():
    # CMD args parser
    parser = argparse.ArgumentParser(description='haar cascade generator')
    parser.add_argument("-i", "--info",     action = "store_true",  help = "print cascade
info and exit")
    parser.add_argument("-n", "--name",     action = "store",       help = "set cascade
name", default = "")
    parser.add_argument("-s", "--stages",   action = "store",       help = "set the
maximum number of stages", type = int, default=0)
    parser.add_argument("-c", "--header",   action = "store_true",  help = "generate a C
header")
    parser.add_argument("file", action = "store", help = "OpenCV xml cascade file path")

    # Parse CMD args
    args = parser.parse_args()

    if args.info:
        # print cascade info and exit
        cascade_info_universal(args.file)
        return

    if args.header:
        # generate a C header from the xml cascade
        cascade_header(args.file, args.stages, args.name)
        return

    # generate a binary cascade from the xml cascade
    cascade_binary_universal(args.file, args.stages, args.name)

if __name__ == '__main__':
    main()
```

ANEXO IV. ESQUEMÁTICO DEL PROYECTO



ANEXO V. DATASHEET OPENMV CAM H7

OpenMV
https://openmv.io
AUGUST 2018

OpenMV-H7 A Python programmable machine vision camera.

1 Features

- 32-Bit Arm Cortex-M7 operating at 400MHz.
- High bandwidth 1MB SRAM / 2MB FLASH.
- Double-precision Floating Point Unit (FPU).
- Full DSP instructions, hardware JPEG encoding.
- 2 UARTs, 2 I2C, 1 SPI, 1 CAN, 3 TIM/PWM.
- 1 USB full speed (FS) for programming.
- 1 RGB LED and 2 IR LEDS on board.
- 1 uSD Card socket (supports up to 64GBs).
- High efficiency switching regulator (1A out).
- Low noise LDO for sensor analog supply.
- LiPo battery connector.
- Less than 150-mA power consumption.
- Modular sensor design supports multiple sensors:
  - OV7725 640x480.
  - MT9V034 (Global Shutter Sensor).
  - FLIR 1,2 and 3 thermal imaging sensors.

2 Description

The OpenMV cameras are low-power, Python3 programmable machine vision cameras that support an extensive set of image processing functions and neural networks. OpenMV cameras are programmed using a cross-platform IDE which allows viewing the camera's frame buffer, accessing sensor controls, uploading scripts to the camera via serial over USB (or WiFi/BLE if available).
The OpenMV-H7 camera base board is based on the STM32H7 Arm Cortex-M7 MCU operating at 400MHz, and featuring 1MB SRAM, 2MB FLASH, FPU, DSP and a hardware JPEG encoder. The base board has a modular sensor design, decoupling the sensor from the camera. The modular sensor design enables the camera to support multiple sensors including OV7725, MT9V03x global shutter sensor and FLIR Lepton 1, 2 and 3 thermal sensors.

Device Information

PART NUMBER	BODY SIZE (NOM)
OPENMV-H7	1.4 in × 1.75 in

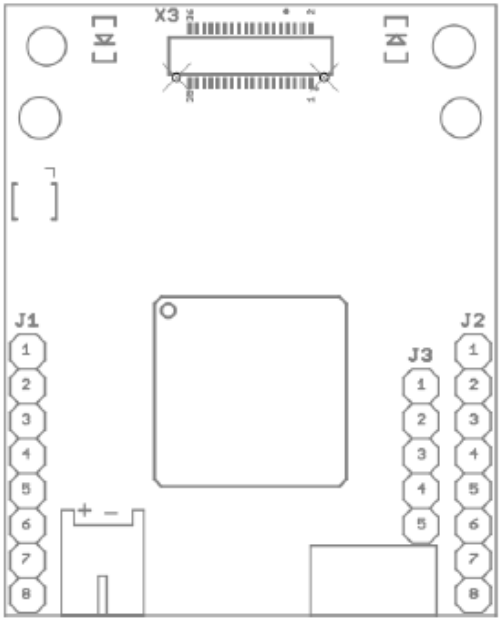
3 Applications

- Home automation.
- Robot guidance.
- Industrial Applications.
- Surveillance Applications.
- Object detection and tracking.

OpenMV
https://openmv.io
AUGUST 2018

Table of Contents

4 Pin Configurations and Functions



Pin Functions

Pin			Description
Header	No	Name	
J1 Pin Configuration			
J1	1	P0	UART1 RX – TM1 CH3N – SPI 2 MOSI
	2	P1	UART1 TX – TM1 CH2N – SPI 2 MISO
	3	P2	CAN2 TX – TM1 CH1N – SPI 2 SCLK
	4	P3	CAN2 RX – SPI 2 SS
	5	P4	TIM2 CH3 – I2C 2 SCL – UART 3 TX
	6	P5	TIM2 CH4 – I2C 2 SDA – UART3 RX
	7	P6	TIM2 CH1 – DAC – ADC
	8	3.3	3.3V Rail (250 mA Supply MAX).
J2 Pin Configuration			
J2	1	RST	Reset (Connect to GND to reset).
	2	BOOT	Boot 0 (Connect to 3.3V for DFU mode).
	3	SYN	Frame synchronization pin (Use to frame sync cams).
	4	P9	Servo3 – TIM4 CH3
	5	P8	Servo2 – TIM4 CH2 – I2C4 SDA
	6	P7	Servo1 – TIM4 CH1 – I2C4 SCL
	7	VIN	VIN (3.6V – 5V).
	8	GND	GND Rail
J3 Pin Configuration			
J3	1	SWC	Serial wire debug clock.
	2	SWD	Serial wire debug data.
	3	RST	Reset (active low).
	4	3.3V	3.3V rail (500 mA Supply MAX)
	5	GND	GND rail

5 Electrical Characteristics

5.1 Absolute Maximum Ratings[1]

SYMBOL	RATINGS	MIN	MAX	UNIT
V_{IN}	External input supply voltage range.	3.6	5.5	V
V_{OUT}	External output supply voltage range.		3.3	
$V_{I/O}$	Input voltage range on ADC/DAC pins.	-0.3	4.0	
	Input voltage range on any other pins.	-0.3	7.3	
I_{OUT}	External output supply current range.		600	mA
$I_{I/O}$	Output current sunk by any I/O and control pin		20	
	Output current sourced by any I/O and control pin		-20	
$\Sigma I_{I/O}$	Total output current sunk by all I/Os and control pins		140	mA
	Total output current sourced by all I/Os and control pins		140	
T_J	Junction temperature.		125	°C
T_{stg}	Storage temperature.	-65	150	

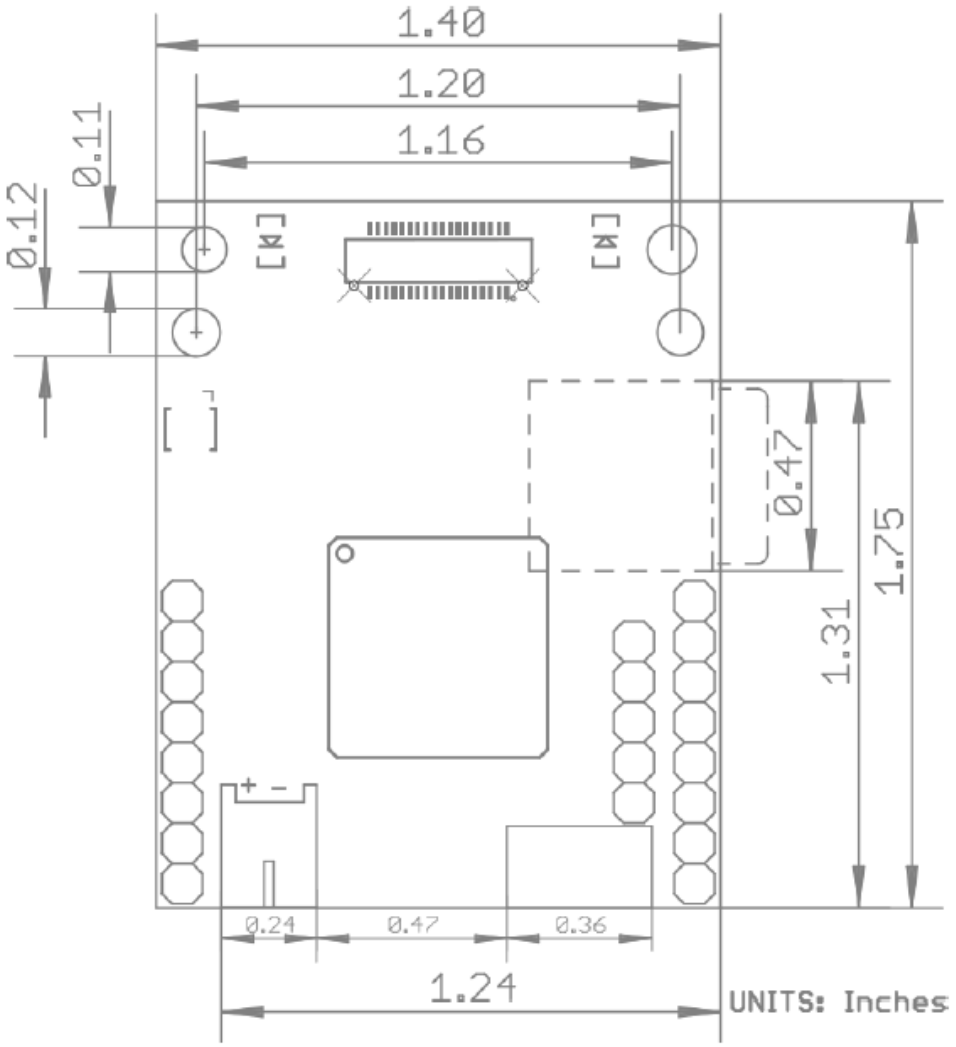
5.2 Recommended Operating Conditions

SYMBOL	RATINGS	MIN	MAX	UNIT
V_{IN}	External input supply voltage range.	3.6	5.0	V
V_{OUT}	External output supply voltage range.		3.3	
$V_{I/O}$	Input voltage range on ADC/DAC pins.	-0.3	3.6	
	Input voltage range on any other pins.	-0.3	5.0	
I_{OUT}	External output supply current range.		500	mA
T_J	Junction temperature.	-40	125	°C

(1) Stresses beyond those listed under *Absolute Maximum Ratings* may cause permanent damage to the device. These are stress ratings only, which do not imply functional operation of the device at these or any other conditions beyond those indicated under *Recommended Operating Conditions*. Exposure to absolute-maximum-rated conditions for extended periods may affect device reliability.

6 Mechanical Information

The following information is the most current data available for the designated device. This data is subject to change without notice and without revision of this document.



ANEXO VI. DATASHEET OV7725

OmniVision

Advanced Information
Preliminary Datasheet

OV7725 Color CMOS VGA (640x480) CAMERACHIP™ Sensor with OmniPixel2™ Technology

General Description

The OV7725 CAMERACHIP™ image sensor is a low voltage CMOS device that provides the full functionality of a single-chip VGA camera and image processor in a small footprint package. The OV7725 provides full-frame, sub-sampled or windowed 8-bit/10-bit images in a wide range of formats, controlled through the Serial Camera Control Bus (SCCB) interface.

This device has an image array capable of operating at up to 60 frames per second (fps) in VGA with complete user control over image quality, formatting and output data transfer. All required image processing functions, including exposure control, gamma, white balance, color saturation, hue control and more, are also programmable through the SCCB interface. In addition, OmniVision sensors use proprietary sensor technology to improve image quality by reducing or eliminating common lighting/electrical sources of image contamination, such as fixed pattern noise, smearing, blooming, etc., to produce a clean, fully stable color image.

Note: The OV7725 uses a lead-free package.

Features

- High sensitivity for low-light operation
- Standard SCCB interface
- Output support for Raw RGB, RGB (GRB 4:2:2, RGB565/555/444) and YCbCr (4:2:2) formats
- Supports image sizes: VGA, QVGA, and any size scaling down from CIF to 40x30
- VarioPixel® method for sub-sampling
- Automatic image control functions including: Automatic Exposure Control (AEC), Automatic Gain Control (AGC), Automatic White Balance (AWB), Automatic Band Filter (ABF), and Automatic Black-Level Calibration (ABLC)
- Image quality controls including color saturation, hue, gamma, sharpness (edge enhancement), and anti-blooming
- ISP includes noise reduction and defect correction
- Lens shading correction
- Saturation level auto adjust (UV adjust)
- Edge enhancement level auto adjust
- De-noise level auto adjust
- Frame synchronization capability

Ordering Information

Product	Package
OV07725-VL1A (Color, lead-free)	28-pin CSP2

Applications

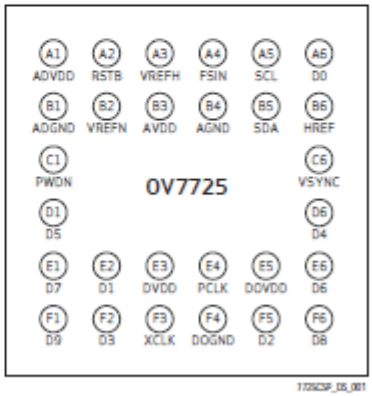
- Cellular and picture phones
- Toys
- PC Multimedia
- Digital still cameras

Key Specifications

	Array Size	640 x 480
Power Supply	**Digital Core**	1.8VDC ± 10%
	Analog	3.0V to 3.3V
	I/O[a]	1.7V to 3.3V
Power Requirements	**Active**	120 mW typical (60 fps VGA, YUV)
	Standby	< 20 µA
Temperature Range		-20°C to +70°C
Output Format (8-bit)		• YUV/YCbCr 4:2:2 • RGB565/555/444 • GRB 4:2:2 • Raw RGB Data
Lens Size		1/4"
Lens Chief Ray Angle		25° non linear
Max Image Transfer Rate		60 fps for VGA
Sensitivity		3.0 V/(Lux • sec)
S/N Ratio		50 dB
Dynamic Range		60 dB
Scan Mode		Progressive
Electronic Exposure		Up to 510:1 (for selected fps)
Pixel Size		6.0 µm x 6.0 µm
Dark Current		40 mV/s
Well Capacity		26 Ke⁻
Fixed Pattern Noise		< 0.03% of $V_{PEAK-TO-PEAK}$
Image Area		3984 µm x 2952 µm
Package Dimensions		5345 µm x 5265 µm

a. I/O power should be 2.45V or higher when using the internal regulator for Core (1.8V); otherwise, it is necessary to provide an external 1.8V for the Core power supply

Figure 1 OV7725 Pinout (Top View)



© 2007 OmniVision Technologies, Inc.
Version 1.1, March 30, 2007

VarioPixel, OmniVision, and the OmniVision logo are registered trademarks of OmniVision Technologies, Inc.
OmniPixel2 and CameraChip are trademarks of OmniVision Technologies, Inc.
These specifications are subject to change without notice.

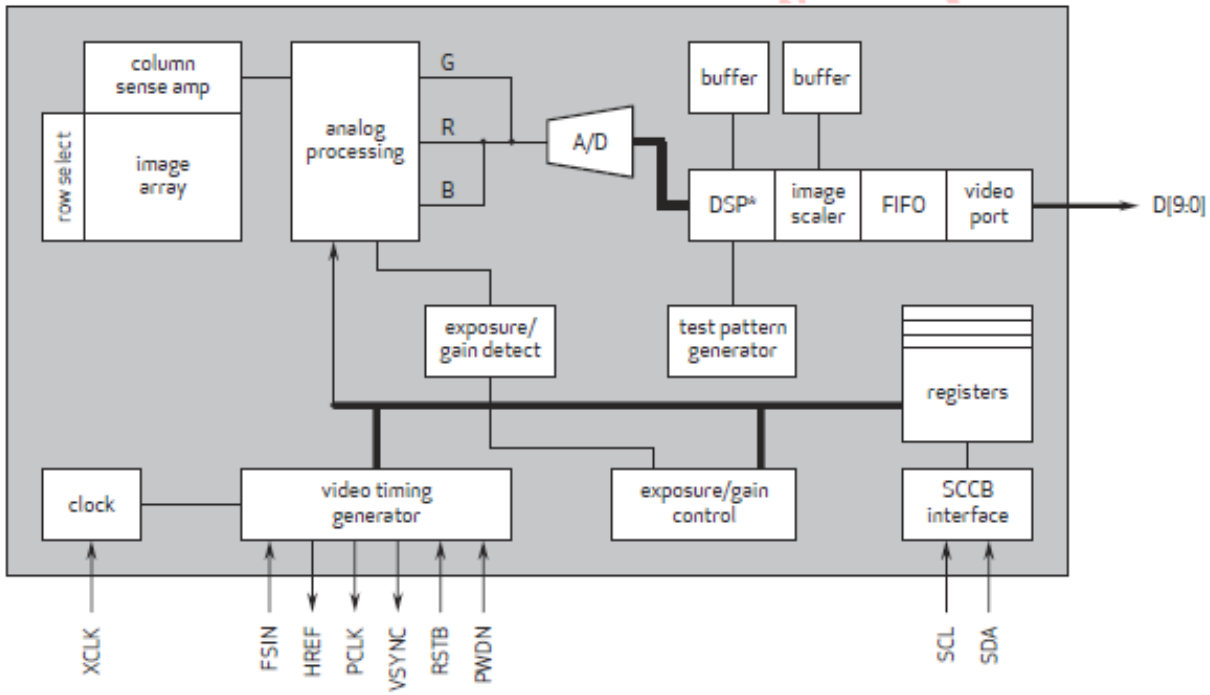
Preliminary

Functional Description

Figure 2 shows the functional block diagram of the OV7725 image sensor. The OV7725 includes:

- Image Sensor Array (total array of 656 x 488 pixels, with active pixels 640 x 480 in YUV mode)
- Analog Signal Processor
- A/D Converters
- Test Pattern Generator
- Digital Signal Processor (DSP)
- Image Scaler
- Timing Generator
- Digital Video Port
- SCCB Interface

Figure 2 Functional Block Diagram



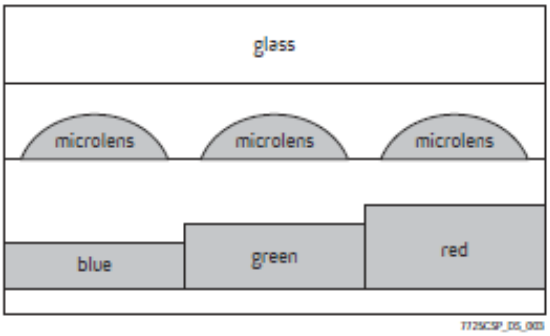
note 1 DSP* (lens shading correction, de-noise, white/black pixel correction, auto white balance, etc.)

7725CSP_DS_002

Image Sensor Array

The OV7725 sensor has an image array of 656 x 488 pixels for a total of 320,128 pixels, of which 640 x 480 pixels are active (307,200 pixels). Figure 3 shows a cross-section of the image sensor array.

Figure 3 Image Sensor Array



Timing Generator

In general, the timing generator controls the following functions:

- Array control and frame generation
- Internal timing signal generation and distribution
- Frame rate timing
- Automatic Exposure Control (AEC)
- External timing outputs (VSYNC, HREF/HSYNC, and PCLK)

Analog Signal Processor

This block performs all analog image functions including:

- Automatic Gain Control (AGC)
- Automatic White Balance (AWB)

A/D Converters

After the Analog Processing block, the bayer pattern Raw signal is fed to a 10-bit analog-to-digital (A/D) converter shared by G and BR channels. This A/D converter operates at speeds up to 12 MHz and is fully synchronous to the pixel rate (actual conversion rate is related to the frame rate).

In addition to the A/D conversion, this block also has the following functions:

- Digital Black-Level Calibration (BLC)
- Optional U/V channel delay
- Additional A/D range controls

In general, the combination of the A/D Range Multiplier and A/D Range Control sets the A/D range and maximum value to allow the user to adjust the final image brightness as a function of the individual application.

Test Pattern Generator

The Test Pattern Generator features the following:

- 8-bar color bar pattern
- Shift "1" in output pin

Digital Signal Processor (DSP)

This block controls the interpolation from Raw data to RGB and some image quality control.

- Edge enhancement (a two-dimensional high pass filter)
- Color space converter (can change Raw data to RGB or YUV/YCbCr)
- RGB matrix to eliminate color cross talk
- Hue and saturation control
- Programmable gamma control
- Transfer 10-bit data to 8-bit

Image Scaler

This block controls all output and data formatting required prior to sending the image out. This block scales YUV/RGB output from VGA to CIF and almost any size under CIF.

Digital Video Port

Register bits COM2[1:0] increase I_{OL}/I_{OH} drive current and can be adjusted as a function of the customer's loading.

SCCB Interface

The Serial Camera Control Bus (SCCB) interface controls the CAMERACHIP sensor operation. Refer to *OmniVision Technologies Serial Camera Control Bus (SCCB) Specification* for detailed usage of the serial control port.

Preliminary

Pin Description

Table 1 Pin Description

Pin Number	Name	Pin Type	Function/Description
A1	ADVDD	Power	ADC power supply
A2	RSTB	Input	System reset input, active low
A3	VREFH	Reference	Reference voltage - connect to ground using a 0.1 µF capacitor
A4	FSIN	Input	Frame synchronize input
A5	SCL	Input	SCCB serial interface clock input
A6	D0[a]	Output	Data output bit[0]
B1	ADGND	Power	ADC ground
B2	VREFN	Reference	Reference voltage - connect to ground using a 0.1 µF capacitor
B3	AVDD	Power	Analog power supply
B4	AGND	Power	Analog ground
B5	SDA	I/O	SCCB serial interface data I/O
B6	HREF	Output	HREF output
C1	PWDN	Input (0)[b]	Power Down Mode Selection 0: Normal mode 1: Power down mode
C6	VSYNC	Output	Vertical sync output
D1	D5	Output	Data output bit[5]
D6	D4	Output	Data output bit[4]
E1	D7	Output	Data output bit[7]
E2	D1	Output	Data output bit[1]
E3	DVDD	Power	Power supply (+1.8 VDC) for digital logic core
E4	PCLK	Output	Pixel clock output
E5	DOVDD	Power	Digital power supply for I/O (1.7V ~ 3.3V)
E6	D6	Output	Data output bit[6]
F1	D9[c]	Output	Data output bit[9]
F2	D3	Output	Data output bit[3]
F3	XCLK	Input	System clock input
F4	DOGND	Power	Digital ground
F5	D2	Output	Data output bit[2]
F6	D8	Output	Data output bit[8]

a. D[9:0] for 10-bit Raw RGB data (D[9] MSB, D[0] LSB)
b. Input (0) represents an internal pull-down resistor.
c. D[9:2] for 8-bit YUV or RGB565/RGB555 (D[9] MSB, D[2] LSB)

Preliminary

Electrical Characteristics

Table 2 Operating Conditions

Parameter	Min	Max
Operating temperature	-20°C	+70°C
Storage temperature[a]	-40°C	+125°C

a. Exceeding the stresses listed may permanently damage the device. This is a stress rating only and functional operation of the sensor at these and any other condition above those indicated in this specification is not implied. Exposure to absolute maximum rating conditions for any extended period may affect reliability.

Table 3 Absolute Maximum Ratings

Ambient Storage Temperature		-40°C to +95°C
Supply Voltages (with respect to Ground)	V_{DD-A}	4.5 V
	V_{DD-C}	3 V
	V_{DD-IO}	4.5 V
All Input/Output Voltages (with respect to Ground)		-0.3V to V_{DD-IO}+0.5V
Lead-free Temperature, Surface-mount process		245°C

NOTE: Exceeding the Absolute Maximum ratings shown above invalidates all AC and DC electrical specifications and may result in permanent device damage.

Table 4 DC Characteristics (-20°C < T_A < 70°C)

Symbol	Parameter	Condition	Min	Typ	Max	Unit
V_{DD-A}	DC supply voltage – analog	–	3.0	3.3	3.6	V
V_{DD-C}	DC supply voltage – digital core	–	1.62	1.8	1.98	V
V_{DD-IO}	DC supply voltage – I/O	–	2.5	–	3.3	V
I_{DDA}	Active (operating) current	See Note [a]		10 + 8[b]		mA
$I_{DDS-SCCB}$	Standby current	See Note [c]		1		mA
$I_{DDS-PWDN}$	Standby current			10	20	µA
V_{IH}	Input voltage HIGH	CMOS	0.7 x V_{DD-IO}			V
V_{IL}	Input voltage LOW				0.3 x V_{DD-IO}	V
V_{OH}	Output voltage HIGH	CMOS	0.9 x V_{DD-IO}			V
V_{OL}	Output voltage LOW				0.1 x V_{DD-IO}	V
I_{OH}	Output current HIGH	See Note [d]	8			mA
I_{OL}	Output current LOW		15			mA
I_L	Input/Output leakage	GND to V_{DD-IO}			± 1	µA

a. At 25°C, V_{DD-A} = 3.3V, V_{DD-C} = 1.8V, V_{DD-IO} = 3.3V
$I_{DDA} = \Sigma\{I_{DD-IO} + I_{DD-C} + I_{DD-A}\}$, f_{CLK} = 24MHz at 30 fps YUV output, no I/O loading

b. I_{DD-C} = 10mA, I_{DD-A} = 8mA, without loading

c. At 25°C, V_{DD-A} = 3.3V, V_{DD-C} = 1.8V, V_{DD-IO} = 3.3V
$I_{DDS-SCCB}$ refers to a SCCB-initiated Standby, while $I_{DDS-PWDN}$ refers to a PWDN pin-initiated Standby

d. Standard Output Loading = 25pF, 1.2KΩ

Preliminary

Table 5 Functional and AC Characteristics (-20°C < T_A < 70°C)

Symbol	Parameter	Min	Typ	Max	Unit
Functional Characteristics					
	A/D Differential non-linearity		± 1/2		LSB
	A/D Integral non-linearity		± 1		LSB
	AGC Range			30	dB
	Red/Blue adjustment range			12	dB
Inputs (PWDN, CLK, RESET#)					
f_{CLK}	Input clock frequency	10	24	48	MHz
t_{CLK}	Input clock period	21	42	100	ns
$t_{CLK:DC}$	Clock duty cycle	45	50	55	%
$t_{S:RESET}$	Setting time after software/hardware reset			1	ms
$t_{S:REG}$	Settling time for register change (10 frames required)			300	ms
SCCB Timing (see Figure 4)					
f_{SCL}	Clock frequency			400	KHz
t_{LOW}	Clock low period	1.3			µs
t_{HIGH}	Clock high period	600			ns
t_{AA}	SCL low to data out valid	100		900	ns
t_{BUF}	Bus free time before new START	1.3			µs
$t_{HD:STA}$	START condition hold time	600			ns
$t_{SU:STA}$	START condition setup time	600			ns
$t_{HD:DAT}$	Data in hold time	0			µs
$t_{SU:DAT}$	Data in setup time	100			ns
$t_{SU:STO}$	STOP condition setup time	600			ns
t_R, t_F	SCCB rise/fall times			300	ns
t_{DH}	Data out hold time	50			ns
Outputs (VSYNC, HREF, PCLK, and D[9:0] (see Figure 5, Figure 6, Figure 7, and Figure 8)					
t_{PDV}	PCLK[↓] to data out Valid			5	ns
t_{SU}	D[9:0] setup time	15			ns
t_{HD}	D[9:0] Hold time	8			ns
t_{PHH}	PCLK[↓] to HREF[↑]	0		5	ns
t_{PHL}	PCLK[↓] to HREF[↓]	0		5	ns
AC Conditions:	• V_{DD}: V_{DD-C} = 1.8V, V_{DD-A} = 3.3V, V_{DD-IO} = 3.3V • Rise/Fall Times: I/O: 5ns, Maximum SCCB: 300ns, Maximum • Input Capacitance: 10pf • Output Loading: 25pF, 1.2KΩ to 3.3V • f_{CLK}: 24MHz				

Preliminary

Timing Specifications

Figure 4 SCCB Timing Diagram

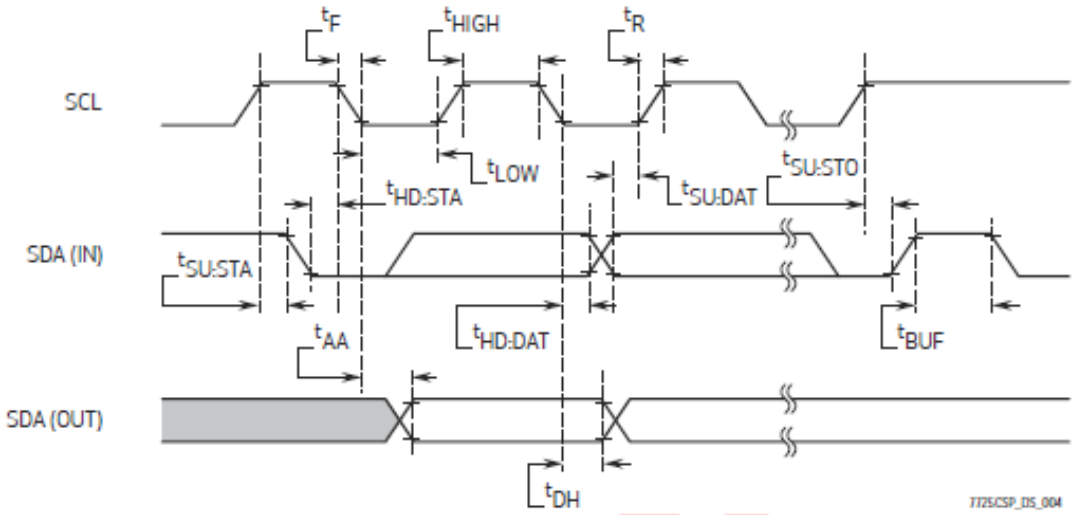


Figure 5 Horizontal Timing

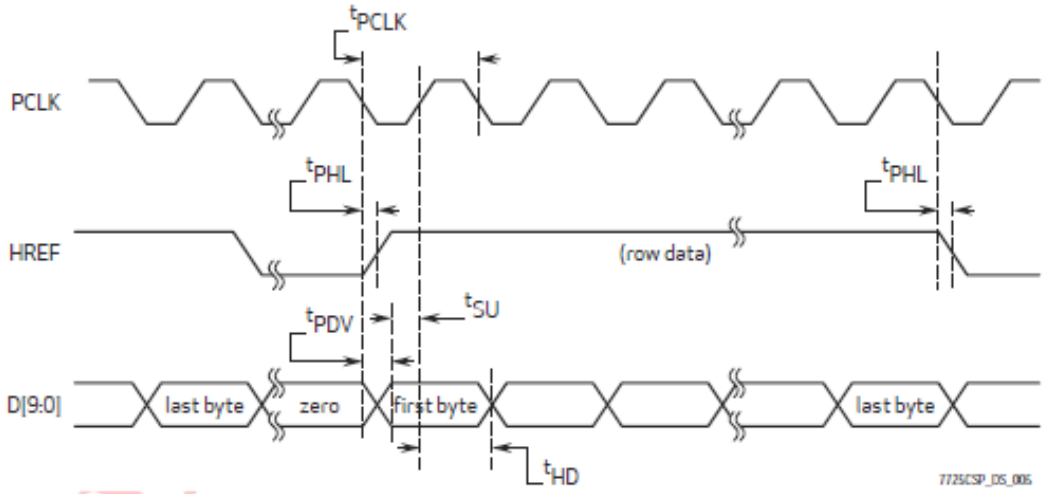
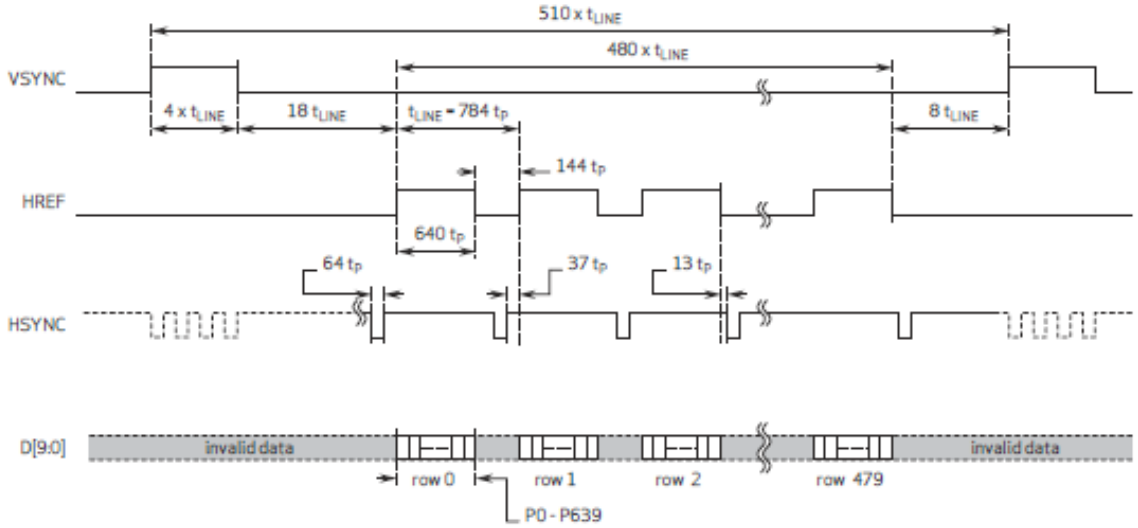


Figure 6 VGA Frame Timing

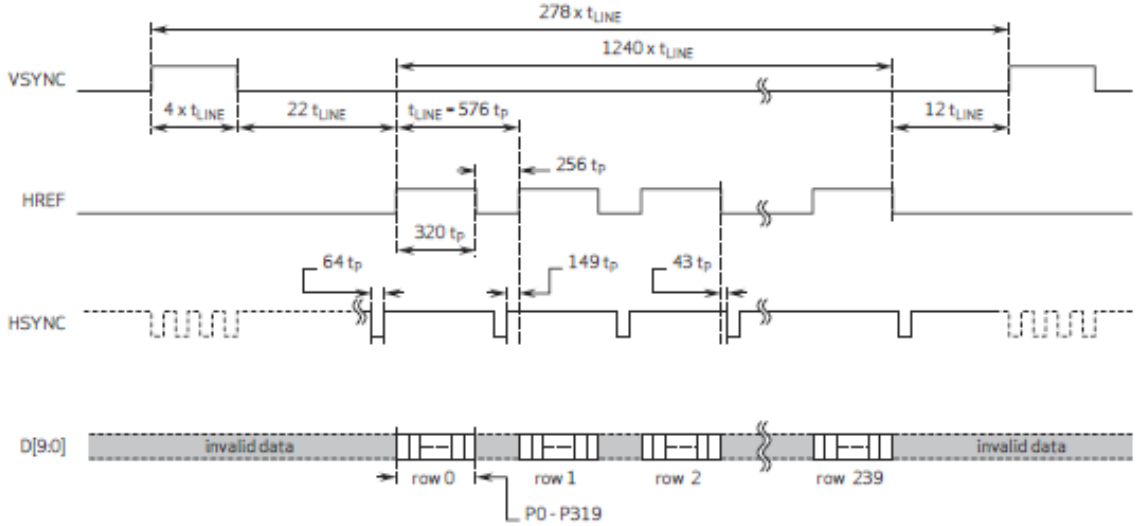


note 1 for raw data, $t_P = t_{PCLK}$

note 2 for YUV/RGB, $t_P = 2 \times t_{PCLK}$

7725CSP_DS_006

Figure 7 QVGA Frame Timing

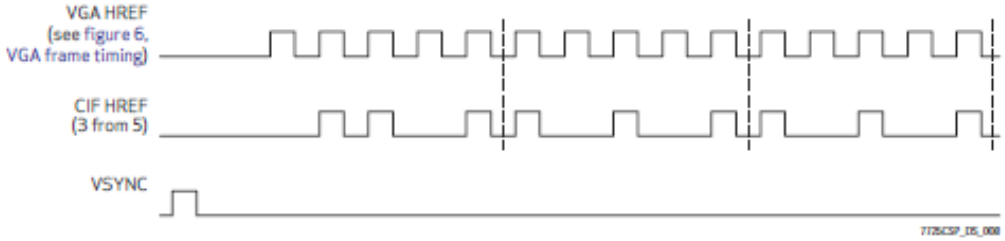


note 1 for raw data, $t_P = t_{PCLK}$

note 2 for YUV/RGB, $t_P = 2 \times t_{PCLK}$

7725CSP_DS_007

Figure 8 CIF Frame Timing



7725CSP_DS_008

Figure 9 RGB 565 Output Timing Diagram

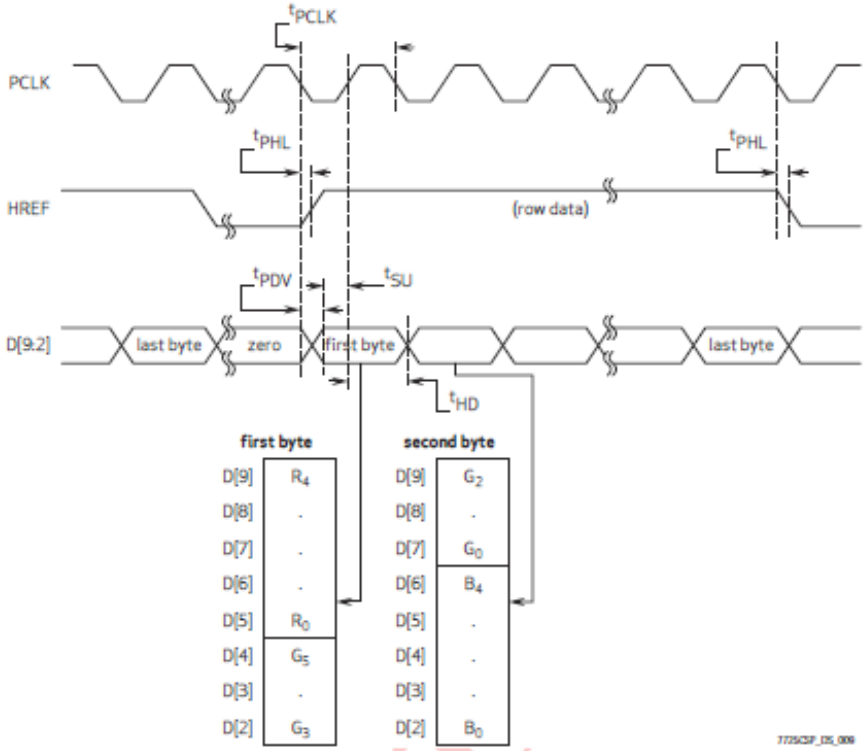


Figure 10 RGB 555 Output Timing Diagram

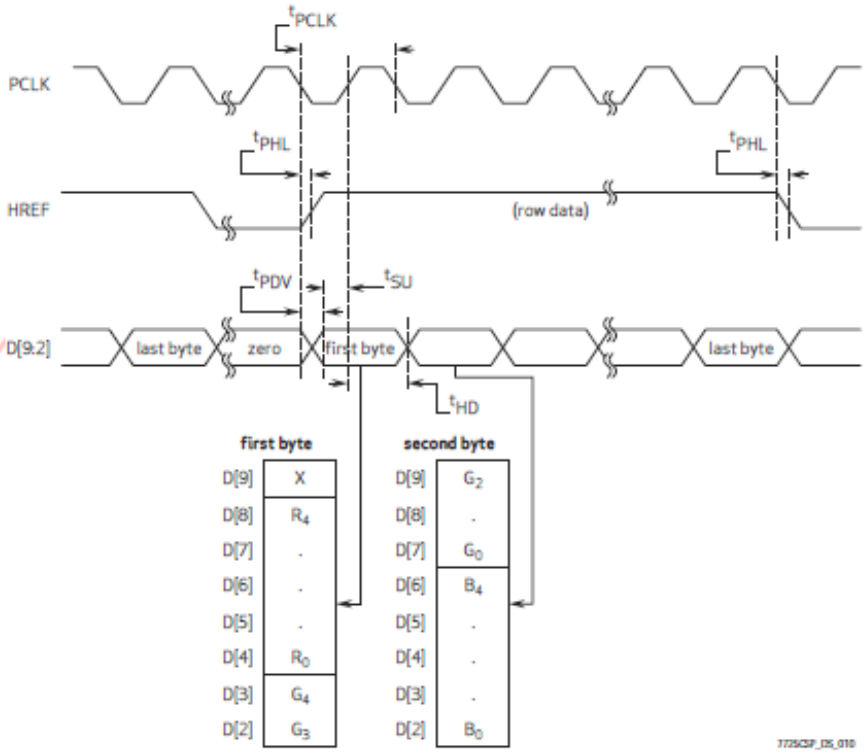
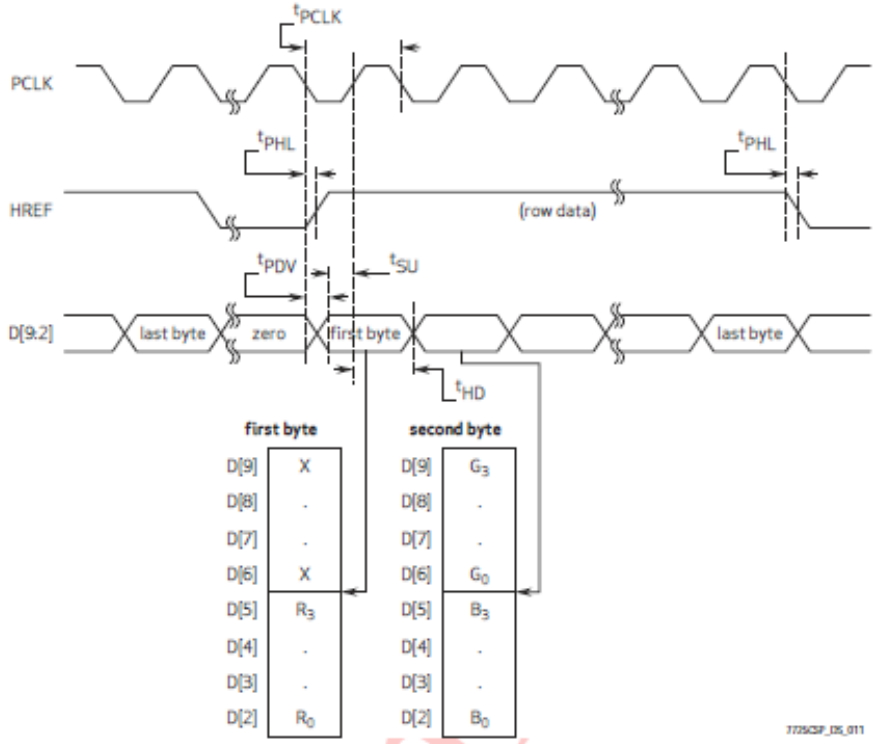


Figure 11 RGB 444 Output Timing Diagram



Preliminary

Register Set

Table 6 provides a list and description of the Device Control registers contained in the OV7725. For all register Enable/Disable bits, ENABLE = 1 and DISABLE = 0. The device slave addresses are 42 for write and 43 for read.

Table 6 Device Control Register List (Sheet 1 of 13)

Address (Hex)	Register Name	Default (Hex)	R/W	Description
00	GAIN	00	RW	AGC – Gain control gain setting Bit[7:0]: AGC[7:0] • Range: [00] to [FF]
01	BLUE	80	RW	AWB – Blue channel gain setting • Range: [00] to [FF]
02	RED	80	RW	AWB – Red channel gain setting • Range: [00] to [FF]
03	GREEN	00	RW	AWB – Green channel gain setting • Range: [00] to [FF]
04	RSVD	XX	–	Reserved
05	BAVG	00	RW	U/B Average Level Automatically updated based on chip output format
06	GAVG	00	RW	Y/Gb Average Level Automatically updated based on chip output format
07	RAVG	00	RW	V/R Average Level Automatically updated based on chip output format
08	AECH	00	RW	Exposure Value – AEC MSBs Bit[7:0]: AEC[15:8] (see register AEC for AEC[7:0]) Automatically updated based on chip output format
09	COM2	01	RW	Common Control 2 Bit[7:5]: Reserved Bit[4]: Soft sleep mode Bit[3:2]: Reserved Bit[1:0]: Output drive capability 00: 1x 01: 2x 10: 3x 11: 4x
0A	PID	77	R	Product ID Number MSB (Read only)
0B	VER	21	R	Product ID Number LSB (Read only)

Preliminary

Table 6 Device Control Register List (Sheet 2 of 13)

Address (Hex)	Register Name	Default (Hex)	R/W	Description
0C	COM3	10	RW	Common Control 3 Bit[7]: Vertical flip image ON/OFF selection Bit[6]: Horizontal mirror image ON/OFF selection Bit[5]: Swap B/R output sequence in RGB output mode Bit[4]: Swap Y/UV output sequence in YUV output mode Bit[3]: Swap output MSB/LSB Bit[2]: Tri-state option for output clock at power-down period 0: Tri-state at this period 1: No tri-state at this period Bit[1]: Tri-state option for output data at power-down period 0: Tri-state at this period 1: No tri-state at this period Bit[0]: Sensor color bar test pattern output enable
0D	COM4	41	RW	Common Control 4 Bit[7:6]: PLL frequency control 00: Bypass PLL 01: PLL 4x 10: PLL 6x 11: PLL 8x Bit[5:4]: AEC evaluate window 00: Full window 01: 1/2 window 10: 1/4 window 11: Low 2/3 window Bit[3:0]: Reserved
0E	COM5	01	RW	Common Control 5 Bit[7]: Auto frame rate control ON/OFF selection (night mode) Bit[6]: Auto frame rate control speed selection Bit[5:4]: Auto frame rate max rate control 00: No reduction of frame rate 01: Max reduction to 1/2 frame rate 10: Max reduction to 1/4 frame rate 11: Max reduction to 1/8 frame rate Bit[3:2]: Auto frame rate active point control 00: Not allowed 01: Add frame when AGC reaches 4x gain 10: Add frame when AGC reaches 8x gain 11: Add frame when AGC reaches 16x gain Bit[1]: Reserved Bit[0]: AEC max step control 0: AEC increase step has limit 1: No limit to AEC increase step
0F	COM6	43	RW	Common Control 6 Bit[7:1]: Reserved Bit[0]: Auto window setting ON/OFF selection when format changes

Preliminary

Table 6 Device Control Register List (Sheet 3 of 13)

Address (Hex)	Register Name	Default (Hex)	R/W	Description
10	AEC	40	RW	Exposure Value Bit[7:0]: AEC[7:0] (see register AECH for AEC[15:8])
11	CLKRC	80	RW	Internal Clock Bit[7]: Reserved Bit[6]: Use external clock directly (no clock pre-scale available) Bit[5:0]: Internal clock pre-scalar F(internal clock) = F(input clock)/(Bit[5:0]+1)/2 • Range: [0 0000] to [1 1111]
12	COM7	00	RW	Common Control 7 Bit[7]: SCCB Register Reset 0: No change 1: Resets all registers to default values Bit[6]: Resolution selection 0: VGA 1: QVGA Bit[5]: BT.656 protocol ON/OFF selection Bit[4]: Sensor RAW Bit[3:2]: RGB output format control 00: GBR4:2:2 01: RGB565 10: RGB555 11: RGB444 Bit[1:0]: Output format control 00: YUV 01: Processed Bayer RAW 10: RGB 11: Bayer RAW
13	COM8	8F	RW	Common Control 8 Bit[7]: Enable fast AGC/AEC algorithm Bit[6]: AEC - Step size limit 0: Step size is limited to vertical blank 1: Unlimited step size Bit[5]: Banding filter ON/OFF Bit[4]: Enable AEC below banding value Bit[3]: Fine AEC ON/OFF control Bit[2]: AGC Enable Bit[1]: AWB Enable Bit[0]: AEC Enable

Preliminary

Table 6 Device Control Register List (Sheet 4 of 13)

Address (Hex)	Register Name	Default (Hex)	R/W	Description
14	COM9	4A	RW	Common Control 9 Bit[7]: Histogram or average based AEC/AGC selection Bit[6:4]: Automatic Gain Ceiling - maximum AGC value 000: 2x 001: 4x 010: 8x 011: 16x 100: 32x 101 Not allowed 110: Not allowed 111: Not allowed Bit[3]: Reserved Bit[2]: Drop VSYNC output of corrupt frame Bit[1]: Drop HREF output of corrupt frame Bit[0]: Reserved
15	COM10	00	RW	Common Control 10 Bit[7]: Output negative data Bit[6]: HREF changes to HSYNC Bit[5]: PCLK output option 0: Free running PCLK 1: PCLK does not toggle during horizontal blank Bit[4]: PCLK reverse Bit[3]: HREF reverse Bit[2]: VSYNC option 0: VSYNC changes on falling edge of PCLK 1: VSYNC changes on rising edge of PCLK Bit[1]: VSYNC negative Bit[0]: Output data range selection 0: Full range 1: Data from [10] to [F0] (8 MSBs)
16	REG16	00	RW	Register 16 Bit[7]: Bit shift test pattern options Bit[6:0]: Reserved
17	HSTART	23 (VGA) 3F (QVGA)	RW	Horizontal Frame (HREF column) Start 8 MSBs (2 LSBs are at HREF[5:4])
18	HSIZE	A0 (VGA) 50 (QVGA)	RW	Horizontal Sensor Size (2 LSBs are at HREF[1:0])
19	VSTRT	07 (VGA) 03 (QVGA)	RW	Vertical Frame (row) Start 8 MSBs (1 LSB is at HREF[6])
1A	VSIZE	F0 (VGA) 78 (QVGA)	RW	Vertical Sensor Size (1 LSB is at HREF[2])
1B	PSHFT	40	RW	Data Format - Pixel Delay Select (delays timing of the D[9:0] data relative to HREF in pixel units) • Range: [00] (no delay) to [FF] (256 pixel delay which accounts for whole array)

Preliminary

Table 6 Device Control Register List (Sheet 5 of 13)

Address (Hex)	Register Name	Default (Hex)	R/W	Description
1C	MIDH	7F	R	Manufacturer ID Byte – High (Read only = 0x7F)
1D	MIDL	A2	R	Manufacturer ID Byte – Low (Read only = 0xA2)
1E	RSVD	XX	–	Reserved
1F	LAEC	00	RW	Fine AEC Value - defines exposure value less than one row period
20	COM11	10	RW	Common Control 11 Bit[7:2]: Reserved Bit[1]: Single frame ON/OFF selection Bit[0]: Single frame transfer trigger
21	RSVD	XX	–	Reserved
22	BDBase	FF	RW	Banding Filter Minimum AEC Value
23	BDMStep	01	RW	Banding Filter Maximum Step
24	AEW	75	RW	AGC/AEC - Stable Operating Region (Upper Limit)
25	AEB	63	RW	AGC/AEC - Stable Operating Region (Lower Limit)
26	VPT	D4	RW	AGC/AEC Fast Mode Operating Region Bit[7:4]: High nibble of upper limit of fast mode control zone Bit[3:0]: High nibble of lower limit of fast mode control zone
27	RSVD	XX	–	Reserved
28	REG28	00	RW	Register 28 Bit[7:1]: Reserved Bit[0]: Selection on the number of dummy rows, N
29	HOutSize	A0 (VGA) 50 (QVGA)	RW	Horizontal Data Output Size MSBs (2 LSBs at register EXHCH[1:0])
2A	EXHCH	00	RW	Dummy Pixel Insert MSB Bit[7:4]: 4 MSB for dummy pixel insert in horizontal direction Bit[3]: Reserved Bit[2]: Vertical data output size LSB Bit[1:0]: Horizontal data output size 2 LSBs
2B	EXHCL	00	RW	Dummy Pixel Insert LSB 8 LSB for dummy pixel insert in horizontal direction
2C	VOutSize	F0 (VGA) 78 (QVGA)	RW	Vertical Data Output Size MSBs (LSB at register EXHCH[2])
2D	ADVFL	00	RW	LSB of Insert Dummy Rows in Vertical Sync (1 bit equals 1 row)
2E	ADVFH	00	RW	MSB of Insert Dummy Rows in Vertical Sync
2F	YAVE	00	RW	Y/G Channel Average Value
30	LumHTh	80	RW	Histogram AEC/AGC Luminance High Level Threshold
31	LumLTh	60	RW	Histogram AEC/AGC Luminance Low Level Threshold

Preliminary

Table 6 Device Control Register List (Sheet 6 of 13)

Address (Hex)	Register Name	Default (Hex)	R/W	Description
32	HREF	00	RW	Image Start and Size Control Bit[7]: Mirror image edge alignment Bit[6]: Vertical HREF window start control LSB Bit[5:4]: Horizontal HREF window start control LSBs Bit[3]: Data output bit shift test pattern ON/OFF control Bit[2]: Vertical sensor size LSB Bit[1:0]: Horizontal sensor size 2 LSBs
33	DM_LNL	00	RW	Dummy Row Low 8 Bits
34	DM_LNH	00	RW	Dummy Row High 8 Bits
35	ADoff_B	80	RW	AD Offset Compensation Value for B Channel
36	ADoff_R	80	RW	AD Offset Compensation Value for R Channel
37	ADoff_Gb	80	RW	AD Offset Compensation Value for Gb Channel
38	ADoff_Gr	80	RW	AD Offset Compensation Value for Gr Channel
39	Off_B	80	RW	Analog Process B Channel Offset Compensation Value
3A	Off_R	80	RW	Analog Process R Channel Offset Compensation Value
3B	Off_Gb	80	RW	Analog Process Gb Channel Offset Compensation Value
3C	Off_Gr	80	RW	Analog Process Gr Channel Offset Compensation Value
3D	COM12	80	RW	Common Control 12 Bit[7:6]: Reserved Bit[5:0]: DC offset compensation for analog process
3E	COM13	E2	RW	Common Control 13 Bit[7]: BLC enable Bit[6]: ADC channel BLC ON/OFF control Bit[5]: Analog processing channel BLC ON/OFF control Bit[4:3]: Reserved Bit[2]: ABLC gain trigger enable Bit[1:0]: Reserved
3F	COM14	1F	RW	Edge Enhancement Adjustment Bit[7:4]: Reserved Bit[3:2]: AD offset compensation option x0: Use R/Gr channel value for B/Gb 01: Use B/Gb channel value for R/Gr 11: Use B/Gb/R/Gr channel value independently Bit[1:0]: Analog processing offset compensation option x0: Use R/Gr channel value for B/Gb 01: Use B/Gb channel value for R/Gr 11: Use B/Gb/R/Gr channel value independently

Preliminary

Table 6 Device Control Register List (Sheet 7 of 13)

Address (Hex)	Register Name	Default (Hex)	R/W	Description
40	COM15	C0	RW	Common Control 15 Bit[7:4]: Reserved Bit[3]: AD add 128 bit offset Bit[2:0]: Reserved
41	COM16	08	RW	Common Control 16 Bit[7:2]: Reserved Bit[1:0]: BLC target 2 LSBs
42	TGT_B	80	RW	BLC Blue Channel Target Value
43	TGT_R	80	RW	BLC Red Channel Target Value
44	TGT_Gb	80	RW	BLC Gb Channel Target Value
45	TGT_Gr	80	RW	BLC Gr Channel Target Value
46	LC_CTR	00	RW	Lens Correction Control Bit[7:3]: Reserved Bit[2]: Lens correction control select 0: R, G, and B channel compensation coefficient is set by registers LC_COEF (0x49) 1: R, G, and B channel compensation coefficient is set by registers LC_COEFB (0x4B), LC_COEF (0x49), and LC_COEFR (0x4C), respectively Bit[1]: Reserved Bit[0]: Lens correction enable 0: Disable 1: Enable
47	LC_XC	00	RW	X Coordinate of Lens Correction Center Relative to Array Center Bit[7]: Sign bit 0: Positive 1: Negative Bit[6:0]: X coordinate of lens correction center relative to array center
48	LC_YC	00	RW	Y Coordinate of Lens Correction Center Relative to Array Center Bit[7]: Sign bit 0: Positive 1: Negative Bit[6:0]: Y coordinate of lens correction center relative to array center
49	LC_COEF	50	RW	Lens Correction Coefficient G channel compensation coefficient when LC_CTR[2] (0x46) is 1 R, G, and B channel compensation coefficient when LC_CTR[2] is 0
4A	LC_RADI	30	RW	Lens Correction Radius – radius of the circular section where no compensation applies
4B	LC_COEFB	50	RW	Lens Correction B Channel Compensation Coefficient (effective only when LC_CTR[2] is high)

Preliminary

Table 6 Device Control Register List (Sheet 8 of 13)

Address (Hex)	Register Name	Default (Hex)	R/W	Description
4C	LC_COEFR	50	RW	Lens Correction R Channel Compensation Coefficient (effective only when LC_CTR[2] is high)
4D	FixGain	00	RW	Analog Fix Gain Amplifier Bit[7:6]: Gb channel fixed gain Bit[5:4]: Gr channel fixed gain Bit[3:2]: B channel fixed gain Bit[1:0]: R channel fixed gain
4E	AREF0	EF	RW	Sensor Reference Control • Range: [00] to [FF]
4F	AREF1	10	RW	Sensor Reference Current Control Bit[7:4]: Sensor reference current control Bit[3]: Internal regulator ON/OFF selection Bit[2]: Reserved Bit[1:0]: Analog reference control
50	AREF2	60	RW	Analog Reference Control • Range: [00] to [FF]
51	AREF3	00	RW	ADC Reference Control • Range: [00] to [FF]
52	AREF4	00	RW	ADC Reference Control • Range: [00] to [FF]
53	AREF5	24	RW	ADC Reference Control • Range: [00] to [FF]
54	AREF6	7A	RW	Analog Reference Control • Range: [00] to [FF]
55	AREF7	FC	RW	Analog Reference Control • Range: [00] to [FF]
56-5F	RSVD	XX	–	Reserved
60	UFix	80	RW	U Channel Fixed Value Output
61	VFix	80	RW	V Channel Fixed Value Output
62	AWBb_blk	FF	RW	AWB Option for Advanced AWB
63	AWB_Ctrl0	F0	RW	AWB Control Byte 0 Bit[7]: AWB gain enable Bit[6]: AWB calculate enable Bit[5]: Reserved Bit[4:0]: WBC threshold 2

Preliminary

Table 6 Device Control Register List (Sheet 9 of 13)

Address (Hex)	Register Name	Default (Hex)	R/W	Description
64	DSP_Ctrl1	1F	RW	DSP Control Byte 1 Bit[7]: FIFO enable/disable selection Bit[6]: UV adjust function ON/OFF selection Bit[5]: SDE enable Bit[4]: Color matrix ON/OFF selection Bit[3]: Interpolation ON/OFF selection Bit[2]: Gamma function ON/OFF selection Bit[1]: Black defect auto correction ON/OFF Bit[0]: White defect auto correction ON/OFF
65	DSP_Ctrl2	00	RW	DSP Control Byte 2 Bit[7:4]: Reserved Bit[3]: Vertical DCW enable Bit[2]: Horizontal DCW enable Bit[1]: Vertical zoom out enable Bit[0]: Horizontal zoom out enable
66	DSP_Ctrl3	10	RW	DSP Control Byte 3 Bit[7]: UV output sequence option Bit[6]: Reserved Bit[5]: DSP color bar ON/OFF selection Bit[4]: Reserved Bit[3]: FIFO power down ON/OFF selection Bit[2]: Scaling module power down control 1 Bit[1]: Scaling module power down control 2 Bit[0]: Interpolation module power down control
67	DSP_Ctrl4	00	RW	DSP Control Byte 4 Bit[7:3]: Reserved Bit[2]: AEC selection 0: Before gamma 1: After gamma Bit[1:0]: Output selection 00: YUV or RGB 01: YUV or RGB 10: RAW8 11: RAW10
68	AWB_bias	00	RW	AWB BLC Level Clip
69	AWBCtrl1	5C	RW	AWB Control 1 Bit[7:4]: Reserved Bit[3]: G gain enable 0: AWB adjusts R and G gain 1: AWB adjusts R, G, and B gain Bit[2]: Max color gain 0: Max color gain is 2x 1: Max color gain is 4x Bit[1:0]: Reserved

Preliminary

Table 6 Device Control Register List (Sheet 10 of 13)

Address (Hex)	Register Name	Default (Hex)	R/W	Description
6A	AWBCtrl2	11	RW	AWB Control 2
6B	AWBCtrl3	A2	RW	AWB Control 3 Bit[7]: AWB mode select 0: Advanced AWB 1: Simple AWB
6C	AWBCtrl4	01	RW	AWB Control 4
6D	AWBCtrl5	50	RW	AWB Control 5
6E	AWBCtrl6	80	RW	AWB Control 6
6F	AWBCtrl7	80	RW	AWB Control 7
70	AWBCtrl8	0F	RW	AWB Control 8
71	AWBCtrl9	00	RW	AWB Control 9
72	AWBCtrl10	00	RW	AWB Control 10
73	AWBCtrl11	0F	RW	AWB Control 11
74	AWBCtrl12	0F	RW	AWB Control 12
75	AWBCtrl13	FF	RW	AWB Control 13
76	AWBCtrl14	FF	RW	AWB Control 14
77	AWBCtrl15	FF	RW	AWB Control 15
78	AWBCtrl16	10	RW	AWB Control 16
79	AWBCtrl17	70	RW	AWB Control 17
7A	AWBCtrl18	70	RW	AWB Control 18
7B	AWBCtrl19	F0	RW	AWB R Gain Range
7C	AWBCtrl20	F0	RW	AWB G Gain Range
7D	AWBCtrl21	F0	RW	AWB B Gain Range
7E	GAM1	0E	RW	Gamma Curve 1st Segment Input End Point 0x04 Output Value
7F	GAM2	1A	RW	Gamma Curve 2nd Segment Input End Point 0x08 Output Value
80	GAM3	31	RW	Gamma Curve 3rd Segment Input End Point 0x10 Output Value
81	GAM4	5A	RW	Gamma Curve 4th Segment Input End Point 0x20 Output Value
82	GAM5	69	RW	Gamma Curve 5th Segment Input End Point 0x28 Output Value
83	GAM6	75	RW	Gamma Curve 6th Segment Input End Point 0x30 Output Value
84	GAM7	7E	RW	Gamma Curve 7th Segment Input End Point 0x38 Output Value
85	GAM8	88	RW	Gamma Curve 8th Segment Input End Point 0x40 Output Value
86	GAM9	8F	RW	Gamma Curve 9th Segment Input End Point 0x48 Output Value
87	GAM10	96	RW	Gamma Curve 10th Segment Input End Point 0x50 Output Value

Preliminary

Table 6 Device Control Register List (Sheet 11 of 13)

Address (Hex)	Register Name	Default (Hex)	R/W	Description
88	GAM11	A3	RW	Gamma Curve 11th Segment Input End Point 0x60 Output Value
89	GAM12	AF	RW	Gamma Curve 12th Segment Input End Point 0x70 Output Value
8A	GAM13	C4	RW	Gamma Curve 13th Segment Input End Point 0x90 Output Value
8B	GAM14	D7	RW	Gamma Curve 14th Segment Input End Point 0xB0 Output Value
8C	GAM15	E8	RW	Gamma Curve 15th Segment Input End Point 0xD0 Output Value
8D	SLOP	20	RW	Gamma Curve Highest Segment Slope - calculated as follows: SLOP[7:0] = (0x100 - GAM15[7:0]) × 4/3
8E	DNSTh	00	RW	De-noise Threshold
8F	EDGE0	00	RW	Sharpness (Edge Enhancement) Control 0 Bit[7:5]: Reserved Bit[4:0]: Sharpness (edge enhancement) strength control
90	EDGE1	08	RW	Sharpness (Edge Enhancement) Control 1 Bit[7:4]: Reserved Bit[3:0]: Sharpness (edge enhancement) threshold detection
91	DNSOff	10	RW	Auto De-noise Threshold Control
92	EDGE2	1F	RW	Sharpness (Edge Enhancement) Strength Upper Limit
93	EDGE3	01	RW	Sharpness (Edge Enhancement) Strength Lower Limit
94	MTX1	2C	RW	Matrix Coefficient 1
95	MTX2	24	RW	Matrix Coefficient 2
96	MTX3	08	RW	Matrix Coefficient 3
97	MTX4	14	RW	Matrix Coefficient 4
98	MTX5	24	RW	Matrix Coefficient 5
99	MTX6	38	RW	Matrix Coefficient 6
9A	MTX_Ctrl	9E	RW	Matrix Control Bit[7]: Matrix double ON/OFF selection Bit[6]: Reserved Bit[5]: Sign bit for MTX6 Bit[4]: Sign bit for MTX5 Bit[3]: Sign bit for MTX4 Bit[2]: Sign bit for MTX3 Bit[1]: Sign bit for MTX2 Bit[0]: Sign bit for MTX1
9B	BRIGHT	00	RW	Brightness Control
9C	CNST	40	RW	Contrast Gain Gain × 0x20
9D	RSVD	XX	–	Reserved

Preliminary

Table 6 Device Control Register List (Sheet 12 of 13)

Address (Hex)	Register Name	Default (Hex)	R/W	Description
9E	UVADJ0	11	RW	Auto UV Adjust Control 0 Bit[7:4]: Auto UV adjust offset control 4 LSBs Bit[3:0]: Auto UV adjust threshold control
9F	UVADJ1	02	RW	Auto UV Adjust Control 1 Bit[7:3]: Auto UV adjust value Bit[2]: Reserved Bit[1]: Auto UV adjust stop control Bit[0]: Auto UV adjust offset control MSB
A0	SCAL0	00	RW	DCW Ratio Control Bit[7:4]: Reserved Bit[3:2]: Vertical down sampling select 00: Bypass 01: 1/2 vertical down sampling 10: 1/4 vertical down sampling 11: 1/8 vertical down sampling Bit[1:0]: Horizontal down sampling select 00: Bypass 01: 1/2 horizontal down sampling 10: 1/4 horizontal down sampling 11: 1/8 horizontal down sampling
A1	SCAL1	40	RW	Horizontal Zoom Out Control Horizontal zoom ratio = $\frac{0x40}{SCAL1[7:0]}$
A2	SCAL2	40	RW	Vertical Zoom Out Control Vertical zoom ratio = $\frac{0x40}{SCAL2[7:0]}$
A3	FIFOdlyM	06	RW	FIFO Manual Mode Delay Control
A4	FIFOdlyA	00	RW	FIFO Auto Mode Delay Control
A5	RSVD	XX	–	Reserved
A6	SDE	00	RW	Special Digital Effect Control Bit[7]: Reserved Bit[6]: Negative image enable Bit[5]: Gray scale image enable Bit[4]: V fixed value enable Bit[3]: U fixed value enable Bit[2]: Contrast/Brightness enable Bit[1]: Saturation enable Bit[0]: Hue enable
A7	USAT	40	RW	U Component Saturation Gain Gain × 0x40

Preliminary

Table 6 Device Control Register List (Sheet 13 of 13)

Address (Hex)	Register Name	Default (Hex)	R/W	Description
A8	VSAT	40	RW	V Component Saturation Gain Gain × 0x40
A9	HUECOS	80	RW	Cosine value × 0x80
AA	HUESIN	80	RW	Sine value × 0x80
AB	SIGN	06	RW	Sign Bit for Hue and Brightness Bit[7:4]: Reserved Bit[3]: Brightness sign bit Bit[2]: Reserved Bit[1]: Sign bit for HueSin (in Cr' equation) Bit[0]: Sign bit for HueSin (in Cb' equation)
AC	DSPAuto	FF	RW	DSP Auto Function ON/OFF Control Bit[7]: AWB auto threshold control Bit[6]: De-noise auto threshold control Bit[5]: Sharpness (edge enhancement) auto strength control Bit[4]: UV adjust auto slope control Bit[3]: Auto scaling factor control (register SCAL0 (0xA0)) Bit[2]: Auto scaling factor control (registers SCAL1 (0xA1 and SCAL2 (0xA2)) Bit[1:0]: Reserved
NOTE: All other registers are factory-reserved. Please contact OmniVision Technologies for reference register settings.				

Preliminary!

Package Specifications

The OV7725 uses a 28-ball Chip Scale Package 2 (CSP2). Refer to Figure 12 for package information, Table 7 for package dimensions and Figure 13 for the array center on the chip.

Note: For OVT devices that are lead-free, all part marking letters are lower case. Underlining the last digit of the lot number indicates CSP2 is used.

Figure 12 OV7725-CSP2 Package Specifications

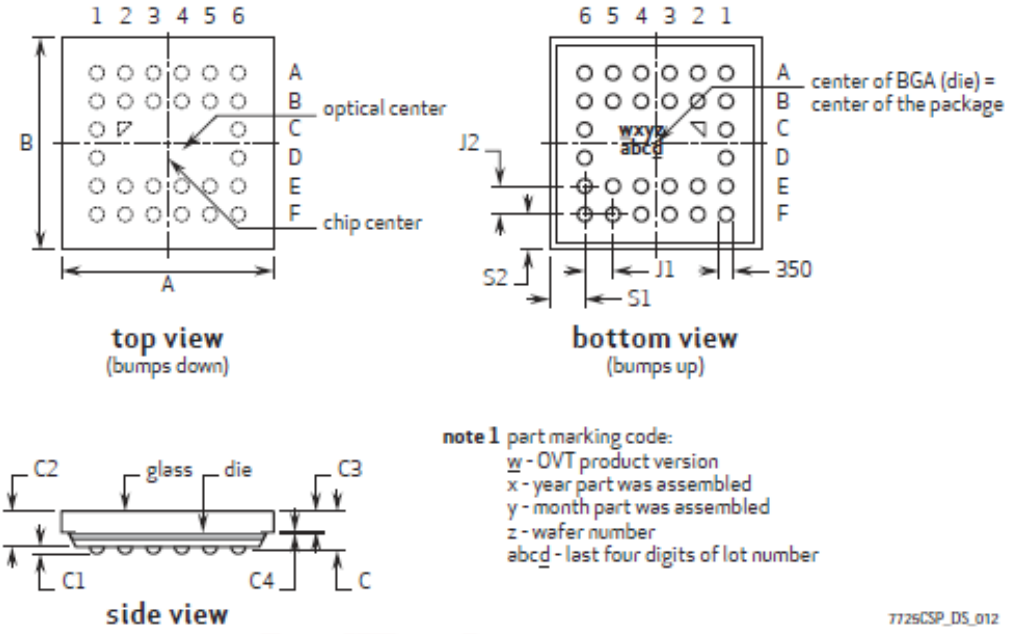


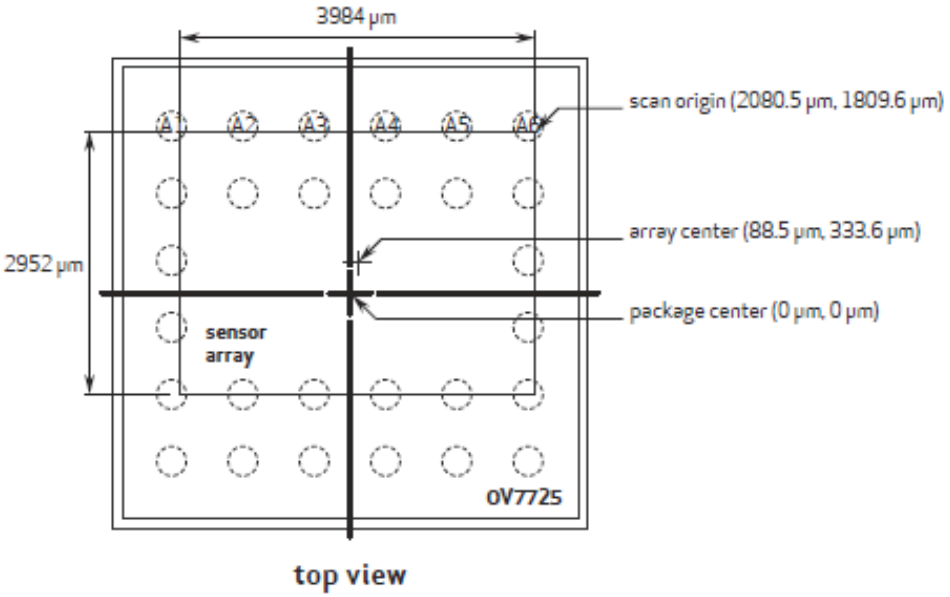
Table 7 OV7725-CSP2 Package Dimensions

Parameter	Symbol	Minimum	Nominal	Maximum	Unit
Package body dimension X	A	5320	5345	5370	µm
Package body dimension Y	B	5240	5265	5290	µm
Package height	C	845	905	965	µm
Ball height	C1	150	180	210	µm
Package body thickness	C2	680	725	770	µm
Cover glass thickness	C3	375	400	425	µm
Airgap between cover glass and sensor	C4	30	45	60	µm
Ball diameter	D	320	350	380	µm
Total pin count	N		28		
Pin count X-axis	N1		6		
Pin count Y-axis	N2		6		
Pins pitch X-axis	J1		800		µm
Pins pitch Y-axis	J2		750		µm
Edge-to-pin center distance analog X	S1	643	673	703	µm
Edge-to-pin center distance analog Y	S2	728	758	788	µm

Prel

Sensor Array Center

Figure 13 OV7725 Sensor Array Center



top view

note1 this drawing is not to scale and is for reference only.

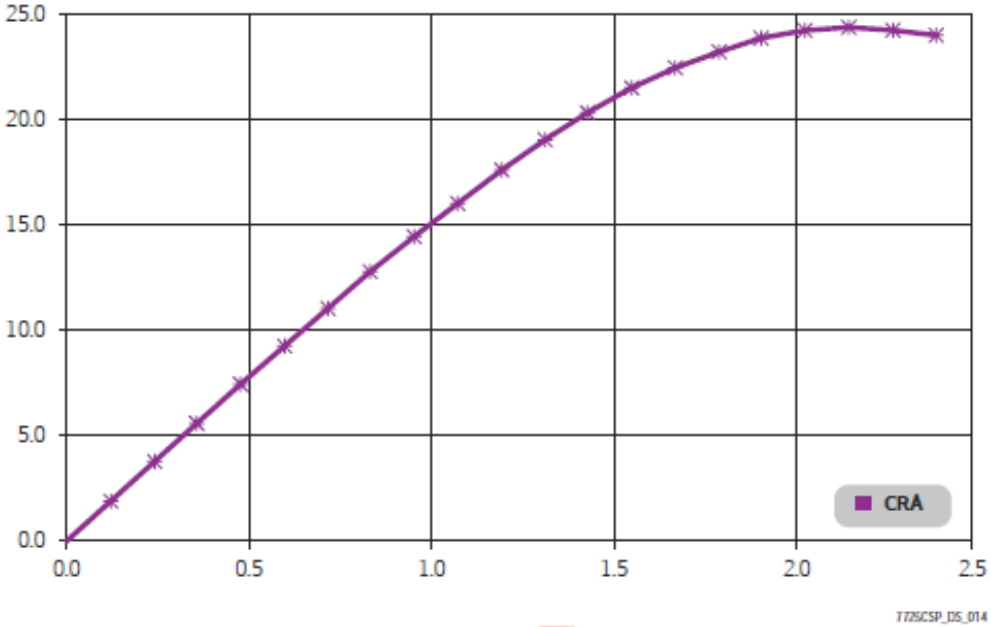
note2 as most optical assemblies invert and mirror the image, the chip is typically mounted with pins A1 to A6 oriented down on the PCB.

7725CSP_DS_013

Prelim

Chief Ray Angle

Figure 14 OV7725 Chief Ray Angle



Prelim

IR Reflow Ramp Rate Requirements

OV7725 Lead-Free Packaged Devices

Note: For OVT devices that are lead-free, all part marking letters are lower case

Figure 15 IR Reflow Ramp Rate Requirements

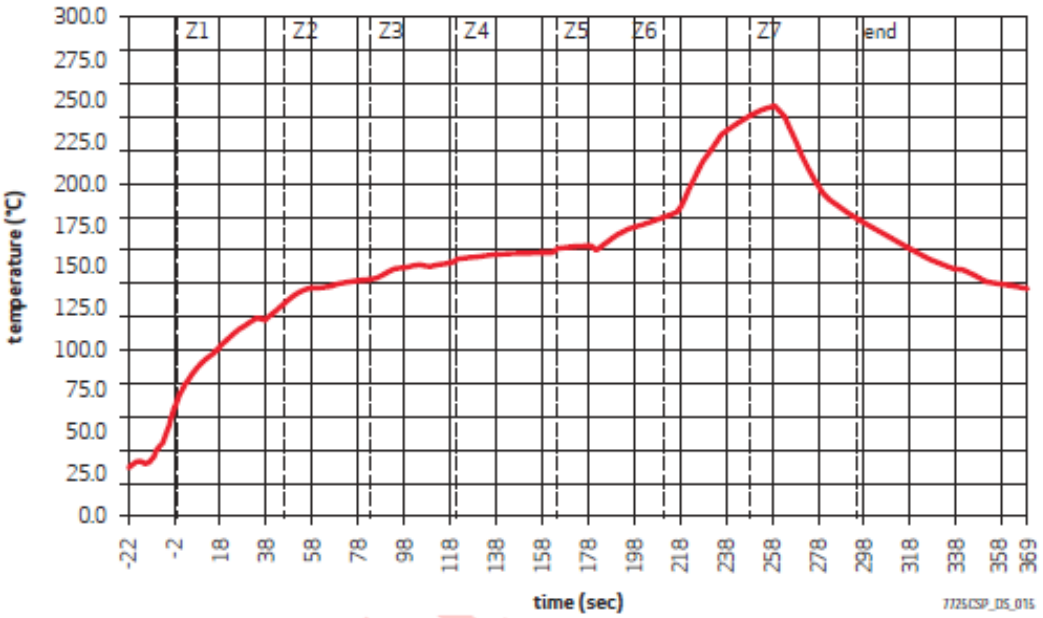


Table 8 Reflow Conditions

Condition	Exposure
Average ramp-up rate (30°C to 217°C)	Less than 3°C per second
> 100°C	Between 330 - 600 seconds
> 150°C	At least 210 seconds
> 217°C	At least 30 seconds (30 ~ 120 seconds)
Peak temperature	245°C
Cool-down rate (peak to 50°C)	Less than 6°C per second
Time from 30°C to 245°C	No greater than 390 seconds

Pre1.1

Note:

- *All information shown herein is current as of the revision and publication date. Please refer to the OmniVision web site (http://www.ovt.com) to obtain the current versions of all documentation.*

- *OmniVision Technologies, Inc. reserves the right to make changes to their products or to discontinue any product or service without further notice (It is advisable to obtain current product documentation prior to placing orders).*

- *Reproduction of information in OmniVision product documentation and specifications is permissible only if reproduction is without alteration and is accompanied by all associated warranties, conditions, limitations and notices. In such cases, OmniVision is not responsible or liable for any information reproduced.*

- *This document is provided with no warranties whatsoever, including any warranty of merchantability, non-infringement, fitness for any particular purpose, or any warranty otherwise arising out of any proposal, specification or sample. Furthermore, OmniVision Technologies, Inc. disclaims all liability, including liability for infringement of any proprietary rights, relating to use of information in this document. No license, expressed or implied, by estoppels or otherwise, to any intellectual property rights is granted herein.*

- *'OmniVision', 'VarioPixel' and the OmniVision logo are registered trademarks of OmniVision Technologies, Inc. 'OmniPixel2' and 'CameraChip' are trademarks of OmniVision Technologies, Inc. All other trade, product or service names referenced in this release may be trademarks or registered trademarks of their respective holders. Third-party brands, names, and trademarks are the property of their respective owners.*

Preliminary

For further information, please feel free to contact OmniVision at info@ovt.com.

OmniVision Technologies, Inc.
1341 Orleans Drive
Sunnyvale, CA USA
(408) 542-3000