Trabajo Fin de Máster

Máster en Automática, Robótica y Telemática

Aplicaciones de percepción avanzada en el marco del proyecto HORUS

Autor: Manuel Baena Capilla

Tutor: Joaquín Ferruz Melero

Departamento de Ingeniería de Sistemas y Automática

Escuela Técnica Superior de Ingeniería

Universidad de Sevilla

Sevilla, 2017

Trabajo Fin de Máster

Máster en Automática, Robótica y Telemática

Aplicaciones de percepción avanzada en el marco del proyecto HORUS

Autor:
Manuel Baena Capilla

Tutor:
Joaquín Ferruz Melero

Departamento de Ingeniería de Sistemas y Automática
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla
Sevilla, 2017

Trabajo Fin de Máster: Aplicaciones de percepción avanzada en el marco del proyecto HORUS

Autor: Manuel Baena Capilla

Tutor: Joaquín Ferruz Melero

El tribunal nombrado para juzgar el TFM arriba indicado, compuesto por los siguientes miembros:

Presidente:

Vocales:

Secretario:

Acuerdan otorgarle la calificación de:

Sevilla, 2017

El Secretario del Tribunal

Agradecimientos

En primer lugar quiero agradecer a mi tutor Joaquín Ferruz por su ayuda para hacer posible la entrega del trabajo, y a Javier María Bustamante, investigador principal del proyecto Horus, por permitirme formar parte de su equipo de trabajo.

Y como no mencionar a mi familia y a mi amigo Jesús Marín con quien siempre puedo contar.

Resumen

En este Trabajo Fin de Máster se hace uso de las imágenes y vídeos que han sido obtenidas por el sistema de filmación de imágenes en el proyecto Horus, concretamente por las captadas por las cámaras ubicadas dentro de cajas nidos para cernícalos primilla, las cuales son grabadas por un servidor de vídeo IP (*Internet Protocol*) tras activarse una funcionalidad que permite detectar de forma somera cambios en la imagen, grabándose varias fotos y un vídeo de pequeño tamaño.

El alumno a través de algoritmos de percepción avanzada por computador, analiza los distintos tipos de ruido que afectan a las imágenes, que provocan que en muchas ocasiones que se active la detección por cambios en la imagen por el servidor de vídeo, y existan muchas grabaciones sin contenido útil. Se implementan distintos tipos de filtros para reducir el ruido y un algoritmo para cuantificar el nivel de nitidez.

Se testean distintos métodos de segmentación del fondo en distintas situaciones de ruido, con funciones proporcionadas por OpenCV, para implementar algoritmos más óptimos para detectar cuando los nidos son ocupados por aves. Se han propuesto métodos para determinar cambios de iluminación, y también, se ha desarrollado a través de la detección de objetos, con características definidas (*blob*), una función capaz de cotejar cuando la cámara ha sido movida.

Por último se ha utilizado una tarjeta de desarrollo de bajo coste “BeagleBone Black”, la cual puede ejecutar distintas versiones del sistema operativo Linux, y funcionar con las librerías de OpenCV, permitiendo ejecutar los algoritmos implementados en tiempo casi real, a través del acceso al streaming de vídeo.

Índice

1. Introducción

Las nuevas tecnologías tienen una penetración importante en numerosos aspectos de nuestras vidas; sin embargo, es todavía minoritaria su aplicación en ecología y en biología de la conservación. La monitorización de la fauna y flora se continúa realizando sobre todo mediante procedimientos manuales que requieren una gran dedicación de personal, generan información fragmentaria e incompleta, y son tremendamente costosos de mantener a largo plazo.

Hace unos años, la única manera de seguir la reproducción de muchas aves era instalando escondites cerca de sus nidos. Estas prácticas, en muchas ocasiones tenían como resultado el abandono del nido debido a las molestias de los observadores. Afortunadamente, gracias a la instalación de cámaras IP (*Internet Protocol*), ahora se puede hacer un seguimiento continuo a distancia sin que las aves se percaten de nuestra presencia.

La tecnología ha mejorado mucho y si antes estas cámaras emitían imágenes de una calidad muy pobre, en blanco y negro, y en muchas ocasiones sólo unos pocos fotogramas a la hora, actualmente pueden emitir imágenes continuas en alta definición, y ver en directo lo que están filmando a través de internet. Actualmente se puede seguir la reproducción de distintas aves encontradas en distintos puntos del mundo sin moverse de casa, por ejemplo, hay cámaras situadas en los bosques de Estonia (http://www.eenet.ee/EENet/kaamerad), filmando nidos de águilas pescadoras (*Pandion haliaetus*), cárabo común (*Strix aluco*), pigargo europeo (*Haliaetus albicilla*), cigüeña negra (*Ciconia nigra*), águila pomarina (*Aquila pomarina*). También otras de las cámaras más conocidas es la web holandesa de Beleef de Lente (https://www.vogelbescherming.nl/beleefdelente), en la que hay y ha habido cámaras instaladas en nidos de Mochuelo (*Athene noctua*), Buho real (*Bubo bubo*), Vencejo (*Apus apus*), Halcón peregrino (*Falco peregrinus*), Cernícalo común (*Falco tinnunculus*), Carbonero común (*Parus major*), Colirrojo real (*Phoenicurus phoenicurus*) y Cigüeña blanca (*Ciconia ciconia*).

En la ICTS (*Instalación Científica Tecnológica Singular*) de Doñana (http://icts.ebd.csic.es), se han instalado varias cámaras IP para seguir la reproducción de aves como el águila imperial ibérica (*Aquila adalberti*), ibis castaña (*Plegadis falcinellus*), búho real (*Bubo bubo*) y otras para el seguimiento de mamíferos como el lince ibérico (*Lynx pardinus*). También se han empleado cámaras IP en otros proyectos de investigación de la Estación Biológica de Doñana (EBD), como en el caso del proyecto Horus, con el cual se han obtenido numerosas fotos y vídeos que se utilizarán en este trabajo para sacar información útil para los investigadores de forma computacional, por ejemplo, obtener cuando un nido está ocupado sin necesidad de haber una persona mirando el nido todo el día.

1.1 Proyecto Horus

1.1.1 Descripción del proyecto

El proyecto ha sido bautizado de forma vulgar por HORUS, y su nombre viene de la mitología egipcia “El dios halcon”. Este proyecto ha sido desarrollado gracias a la financiación de la Junta de Andalucía, que concedío los siguientes proyectos de investigación: *“Seguimiento del cambio global en una especie amenazada: Desarrollo de un sistema automático de monitoreo remoto (HORUS)”* P06-RNM-01712 y *“El cernícalo primilla y el cambio global: “Aplicación de nuevas tecnologías al seguimiento remoto de una especie amenazada (HORUS 2009)”* P06- RNM 04588. El investigador principal del proyecto es Javier María Bustamante, investigador titular de la Estación Biológica de Doñana.

Se desarrolló un sistema automático de seguimiento de la reproducción en una colonia de aves. Para ello se construyó un conjunto de “*smart-nestboxes*” o caja-nido-inteligentes que detecta la entrada y salida de individuos mediante barreras infrarrojas, reconoce a los individuos marcados mediante etiquetas de identificación por radiofrecuencia (RFID) o transponders pasivos, pesa a los individuos que entran mediante una balanza electrónica, permite grabar el

comportamiento mediante cámaras de vídeo, monitoriza el microclima de la caja nido mediante sensores de temperatura y humedad, y además, tiene conectado un servomecanismo para activar una trampilla para capturar las aves. Toda la información se graba en un sistema informático accesible a los investigadores a través de internet. Con estos proyectos se han equipado 20 nidos con cajas nido inteligentes en una colonia de una especie protegida, "cernícalo primilla (*Falco naumanni*)", y actualmente se está depurando el sistema, y la toma de datos a largo plazo sobre la especie de manera automatizada.

La especie modelo del proyecto es el Cernícalo primilla (*Falco naumanni*), que es una pequeña rapaz colonial. Este ave acepta bien las cajas nidos para criar, y al encontrarse muchas de las colonias en edificios, su monitorización a través de sistemas tecnológicos es fácil de implementar. La especie viene siendo estudiada por grupos de investigación de la Estación Biológica de Doñana – CSIC, desde el año 1987 habiéndose realizado 8 tesis doctorales y publicado más de 70 trabajos científicos.

El edificio donde se ubica la colonia de cernícalos es un silo cerealista, que está conectado a la red eléctrica y ha sido dotado de un sistema eléctrico antideflagrante con tomas de corriente en cada una de las ventanas, así como de conexiones de red y una línea ADSL. Las cajas nidos inteligentes están instaladas en las ventanas de la planta sexta del edificio, y han sustituido a los nidos que hacían los cernícalos en los alféizares de las ventanas.

Figura 1.1. Pareja de cernícalos primilla.

Figura 1.2. Silo agrícola donde se situa la colonia.

La zona principal de monitorización en el proyecto es el nido, aunque además se complementa con una utilización de una cámara exterior con movimiento y potente zoom óptico, que permiten observar lo que ocurre en el exterior del nido. Toda la información recogida es procesada y almacenada por equipos informáticos de forma redundante.

Las cajas nido han sido enumeradas según la ubicación de la planta, la orientación y número. Por ejemplo, 6S3 es la caja ubicada en la sexta planta, con orientación sur y de ordenada como número tres contando de izquierda a derecha.

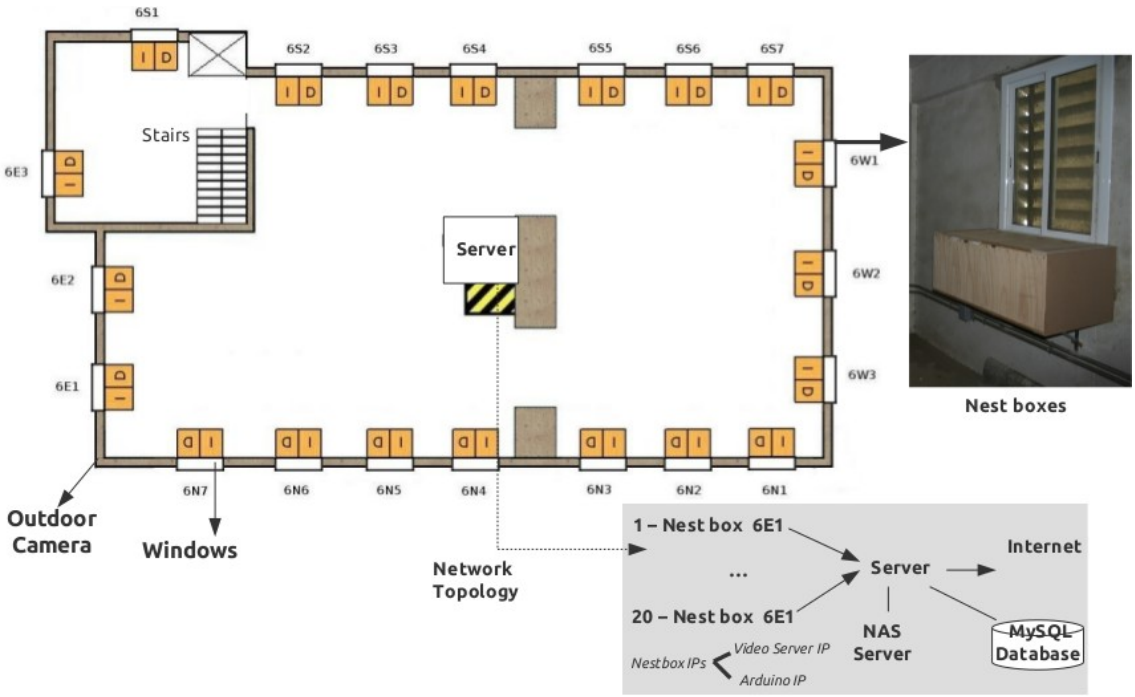


Figura 1.3. Estructura del sistema desarrollado.

Para monitorizar cada caja nido y enviar la información recogida a un servidor centalizado, se utilizó una placa microcontrolador Arduino Mega basado en el ATmega2560, que tiene 54 entradas/salidas digitales (de las cuales 14 proporcionan salida PWM), 16 entradas digitales, 4 UARTS (puertos serie por hardware), un cristal oscilador de 16MHz, conexión USB, entrada de corriente, conector ICSP y botón de reset. Para dotarla de conexión ethernet se le

incorporó la placa Arduino Ethernet, y además se realizarón placas propias para proporcionar funcionalidades necesarias para el proyecto, como pueden ser disponer de reloj RTC, barreras infrarrojas, y conexiones RS232 y RS485.

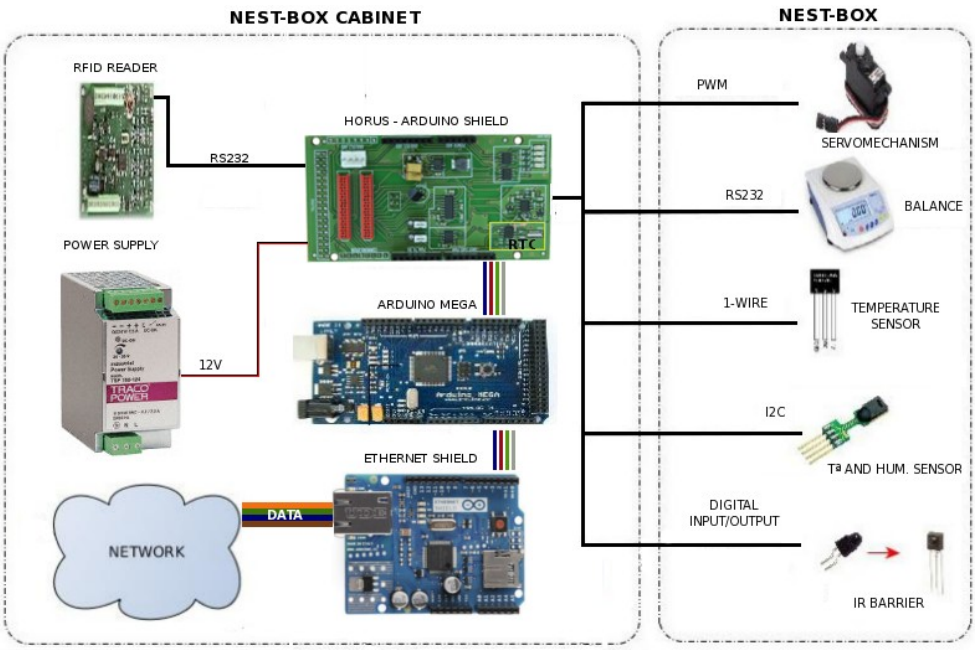


Figura 1.4. Esquema de la electrónica de la caja nido inteligente.

1.1.2 Sistema de grabación de imágenes en Horus

Para la filmación de imágenes dentro de la caja nido, se tuvo en cuenta que el interior está poco iluminado con lo cual es difícil obtener imágenes en color con buena calidad, y se optó por utilizar cámaras blanco y negro, con buena sensibilidad lumínica e iluminadas por luz infrarroja no visible (longitud de onda de 940 nm). Además era deseable que para algunos experimentos se pudiera cambiar la cámara, o cambiarla de lugar, por lo que se necesitaba una cámara de pequeñas dimensiones y que fuera independiente del servidor de vídeo IP (Internet Protocol). Por lo que se utilizó el modelo de cámara KPC-500 BEX, conjuntamente usada con el servidor de vídeo IP VS8102.

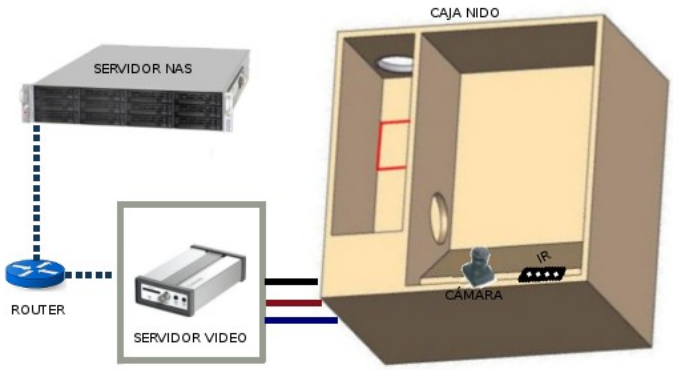


Figura 1.5. Elementos de grabación de imágenes y vídeos de la caja nido.

Las características más importantes de la cámara Camtronics KPC-500 BEX, son:

- Sensor: Sony 1/3-inch Ex View Had CCD
- Resolución:420 líneas
- Píxeles: 500 (H) x 582 (V).
- Sensibilidad (luminosidad): 0,0003 lux/F 2.0 .
- Relación Señal a Ruido (SNR): 48dB (AGC off).
- Óptica utilizada: 2.8mm (2.45mm por defecto).

Y la características más relevantes del servidor de vídeo VIVOTEK VS8102, son:

- Sistema: CPU: TI DM365 SoC, Flash: 128MB, RAM: 128MB, Embedded OS: Linux 2.6.
- Compresión: H.264, MPEG-4 Y MJPEG.
- Tamaño, calidad y rango de bits de la imagen ajustables.
- **Permite configurar hasta 4 formas de transmisión de streaming.**
- **Detección de eventos y grabación.**
- **Almacenamiento en tarjeta SD.**

Los servidores de vídeo se pueden configurar para que graben de forma contínua, o por grabación por activación de eventos (entrada digital, detección de movimiento, periodicamente, reinicion del servidor, pérdida de vídeo, restauración de vídeo y notificación de grabación).

Para tener un sistema de grabación más eficiente, se han configurado los servidores para que sólo graben en casos de la activación de un evento de detección de movimiento. En la siguiente figura se ilustra como está implementado esta opción.

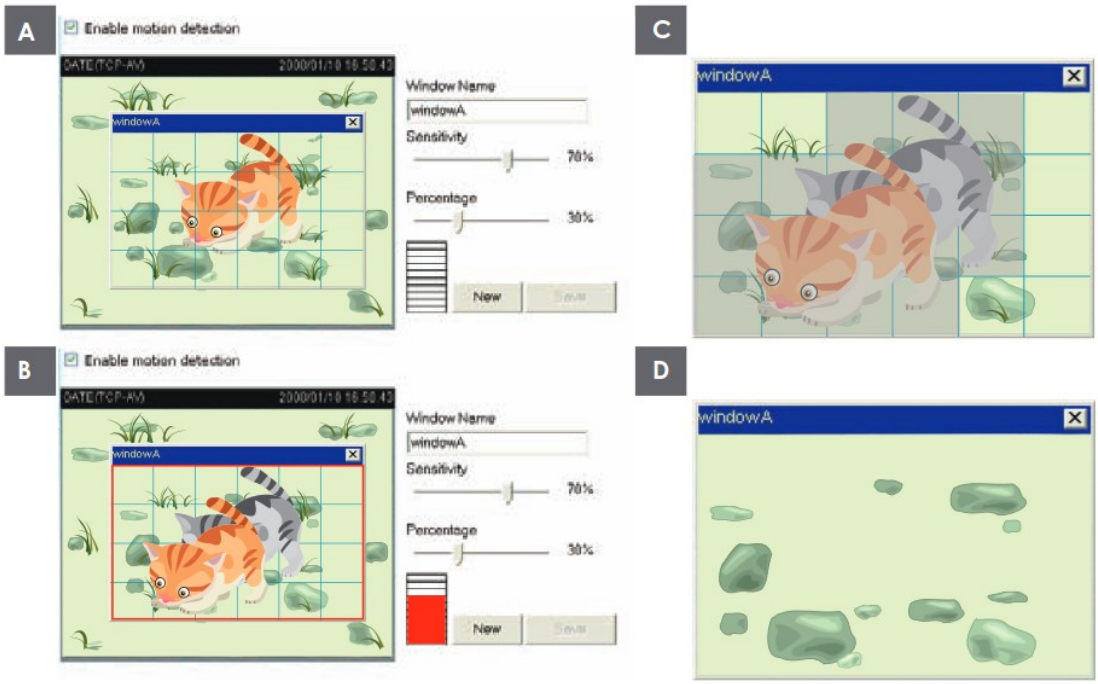


Figura 1.6. Configuración detección de movimiento en VS8102

Se disponen de dos parámetros la sensibilidad y el porcentaje. En la ilustración de arriba, los frames A y B son dos imágenes de una secuencia. La diferencia de píxeles entre los dos frames son detectados y resaltados en gris (frame C), y es comparada con la sensibiliad configurada. La sensibilidad es un parámetro que representa la magnitud de cambio de los valores de los píxeles que son detectado como objeto movido. Alta opciones de sensibilidad suponen detectar leves movimientos mientras que bajas opciones de sensibilidad rechazará gran parte de éstos. El porcentaje es el parámetro que expresa la proporción de píxeles que modifican su valor dentro del área selecciona para la detección de movimiento. En el caso de la figura, el 50% de los valores de los píxeles han cambiado, y como es superior al 30% se activa la detección de movimiento. Para aplicaciones que requieran un nivel alto de seguridad, se sugiere usar sensibilidad alta y porcentaje pequeño.

En el proyecto Horus se ha utilizado valores de sensibilidad del 95% y

porcentaje de 5%.

Una vez configurado el evento hay que definir el tipo de información que se va a almacenar y donde (tarjeta SD, servidor FTP, envío email, servidor SAMBA). Este servidor permite grabar imágenes, vídeo y archivos log. Permite hasta 15 imágenes del evento, con 7 imágenes anteriores y 7 posteriores a su activación. También se puede grabar un vídeo de 10 segundos de duración, con hasta 9 segundos de anterioridad a la activación del evento, y con una capacidad máxima de 800 Kbytes. En el proyecto Horus, en cada activación de un evento de movimiento se graba 1 imagen y un vídeo de duración de 10 segundos, de los cuales 1 segundo es anterior al evento.

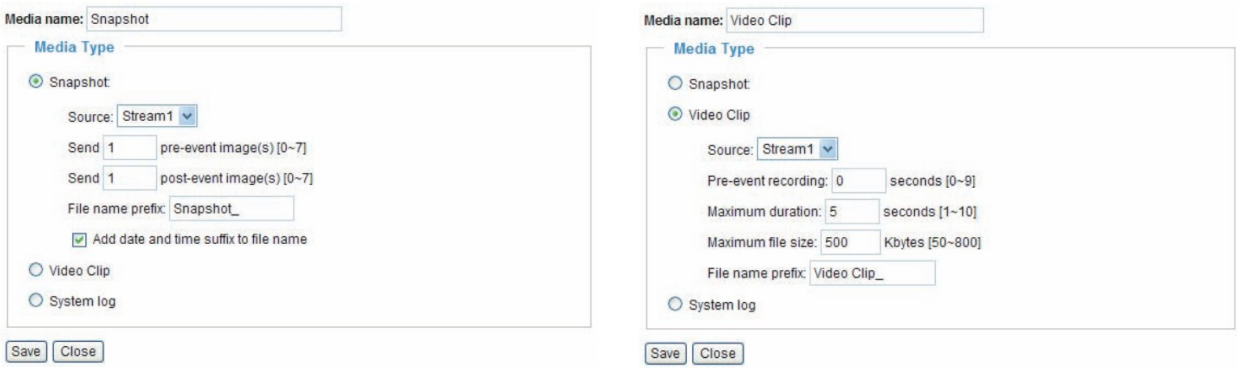


Figura 1.7. Configuración de Snapshot (captura imagen) y vídeo.

El primer año de funcionamiento de los servidores de vídeo fueron configurados para que grabasen los 5 segundos anteriores a la detección del evento, y vimos que al intentar unir los fragmentos de vídeos en una secuencia completa, faltaban trozos, y llegamos a la conclusión que era debido a utilizar muchos segundos predecesores, ya que cuando el evento siguiente era activado, necesitaba leer el buffer de memoria que todavía estaba dedicado a ensamblar el vídeo del anterior evento, lo cual provocaba que no se grabase el vídeo siguiente.

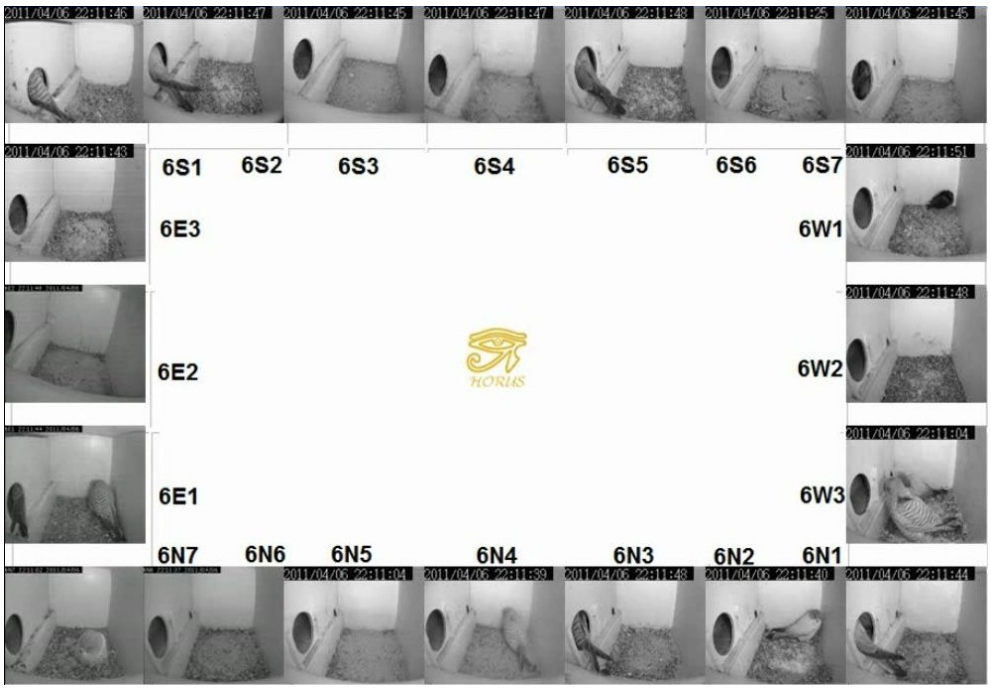
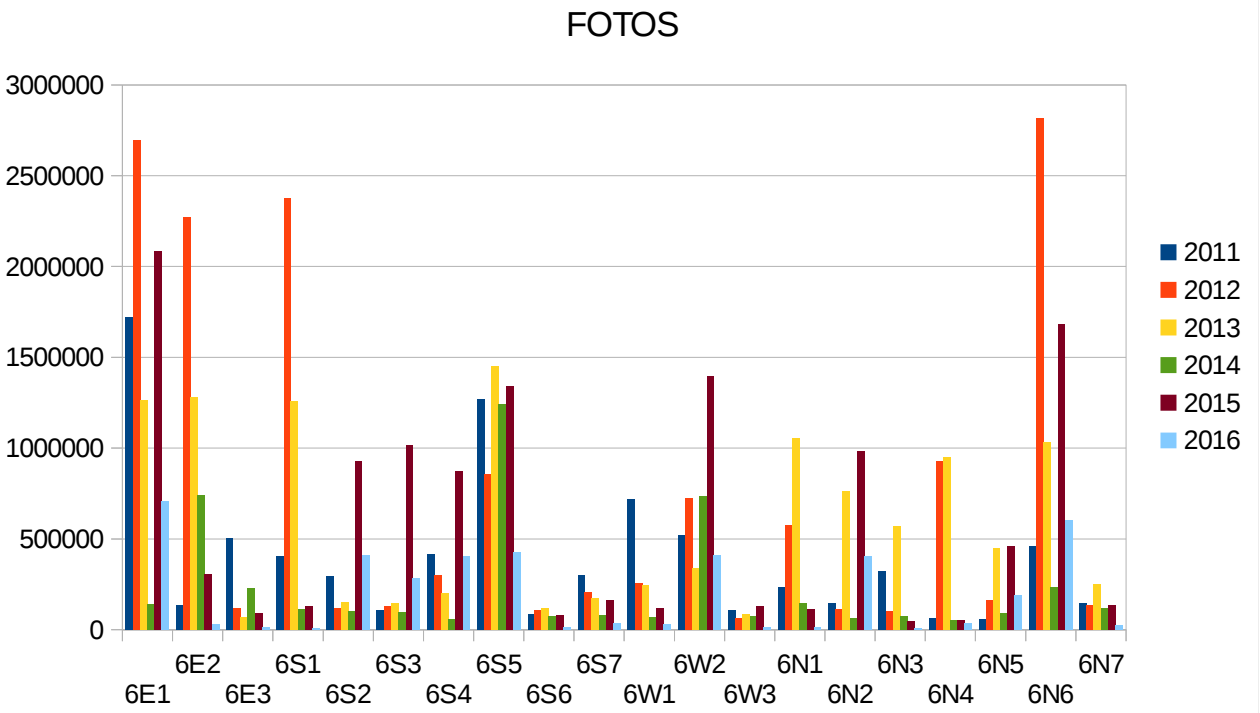


Figura 1.8. Mosaico de las cámaras de los 20 nidos.

1.2 Análisis de la batería de datos

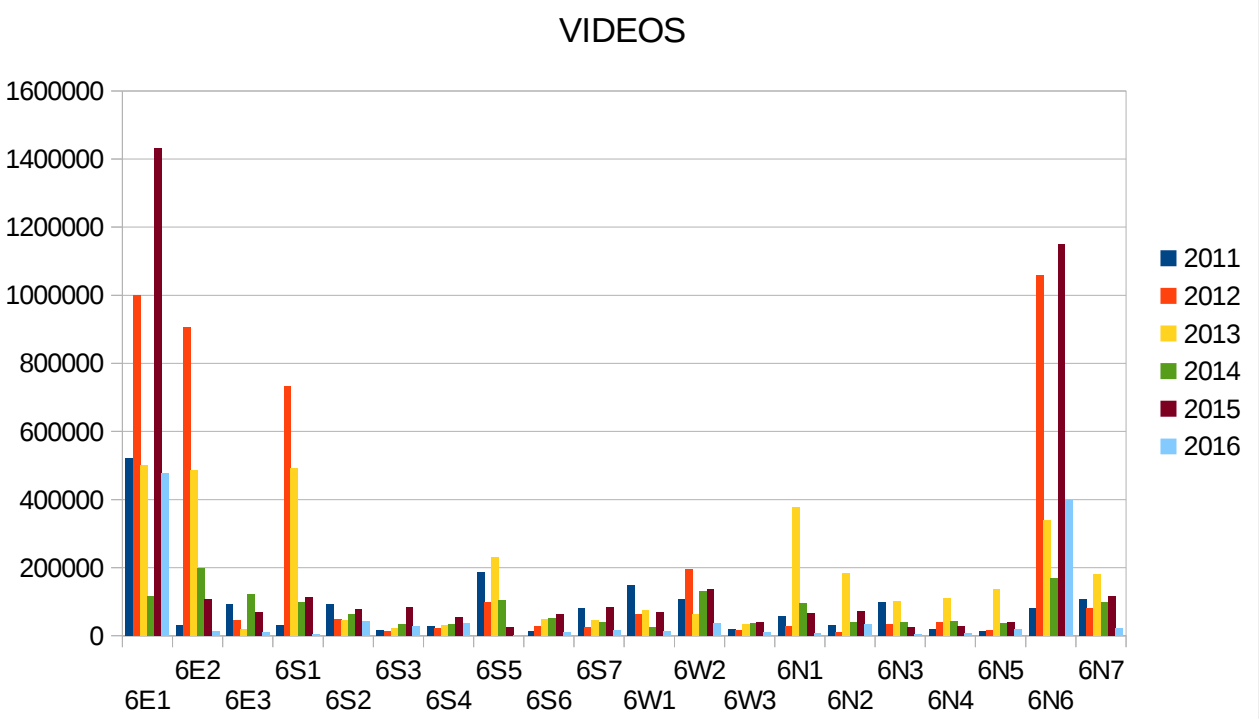
Se disponen de fotos y vídeos de los años 2011 hasta la actualidad. Y se han guardado en un servidor de almacenamiento en red (NAS), organizado por directorios de la siguiente forma: año/nido/fecha(YYYYMMDD)/hora, un ejemplo de directorio sería /2011/6E1-Izq/20110912/09 . La hora de los vídeos y fotos está en horario UTC.

A continuación se muestran las tablas de datos y gráficos, correspondientes al números de archivos almacenados de fotos y vídeos por nido y año. En algunos nidos se han borrado fotos y vídeos sin información para ahorrar espacio.



NIDO	2011	2012	2013	2014	2015	2016
6E1	1719287	2694511	1264437	141311	2085201	709538
6E2	134656	2269761	1278267	739922	306390	28786
6E3	502759	119036	69472	231643	89320	14485
6S1	405459	2374210	1255891	109915	129682	6279
6S2	297624	118714	152408	101248	926360	411965
6S3	108116	131646	143984	98667	1014233	283644
6S4	417126	298257	198225	55438	872455	406638
6S5	1269593	857708	1449099	1241889	1341136	424303
6S6	83468	106623	118627	72058	82263	15198
6S7	300214	205348	173980	81316	163048	37777
6W1	718178	254646	248016	70320	116355	28201
6W2	523500	724736	339622	737020	1398275	411063
6W3	108656	64039	86137	75586	131201	16312
6N1	232871	573976	1052593	144023	115227	14369
6N2	143812	110202	761000	64931	984937	402641
6N3	323425	100110	571763	72823	45158	8931
6N4	65233	925596	952636	51295	50691	35269
6N5	60987	160729	451654	90797	461891	192152
6N6	458669	2818717	1031811	235067	1683382	601760
6N7	147714	135827	248249	120388	137313	26892

Figura 1.9. Gráfica y tabla de fotos almacenadas



NIDO	2011	2012	2013	2014	2015	2016
6E1	522818	1000323	500633	115136	1433482	479093
6E2	31828	906607	485875	198442	106963	12618
6E3	93791	46080	19127	123376	69458	11032
6S1	29611	732498	492751	97034	112270	6150
6S2	91580	47189	46614	63902	79427	42536
6S3	15102	12726	21697	34348	84953	28450
6S4	28251	22945	31761	33692	53070	37334
6S5	186614	97466	231263	104027	25192	16
6S6	14062	29442	49006	50281	63204	11082
6S7	80840	25316	45698	39966	85058	14992
6W1	147377	64301	73915	24234	69061	11687
6W2	108278	195250	61819	131050	138493	36637
6W3	19518	17534	34705	38143	41219	10481
6N1	56289	27661	376340	94289	65797	7334
6N2	29991	10275	183810	39145	71110	34480
6N3	97191	34544	100550	39499	23800	3760
6N4	19348	40423	111610	42428	27081	8284
6N5	14426	17047	136281	36589	39664	17823
6N6	79744	1058499	339633	167809	1148943	398839
6N7	106118	81540	180413	98802	115091	23135

Figura 1.10. Gráfica y tabla de vídeos almacenados

1.3 Objetivos

Este trabajo final de máster tiene como objetivo principal la evaluación de algoritmos de tratamiento digital de imágenes con el catalogo de vídeos y fotos del proyecto Horus, con las siguientes finalidades:

1. Reducir el ruido de las imágenes.
2. Detectar imágenes borrosas.
3. Filtrar imágenes sin contenido útil para los investigadores, por ejemplo en las situaciones donde se ha activado la detección de movimiento debido a niveles altos de ruido o cambios de luminosidad.

Por último estos algoritmos se implementarán en una BeagleBone Black, que es un computador de placa reducida de hardware libre y bajo consumo. A través del procesado el streaming de vídeo obtenido del servidor de vídeo IP, permitirá realizar en tiempo casi real una detección de eventos más perfeccionada para evitar almacenar información no útil, y además producir alertas cuando ocurra algún suceso como imágenes borrosas, que no haya imagen de la cámara, el iluminador infrarrojo se haya roto, la cámara se haya movido y otros eventos.

2. Procesado de las imágenes

2.1 Ruido

2.1.1 Fundamentos

Todas las imágenes tienen una cierta cantidad de ruido. Las causas de esta distorsión pueden ser diversas, debiéndose habitualmente al sensor de la cámara y al medio físico de transmisión de la señal. Habitualmente, este ruido se manifiesta en píxeles aislados que toman valores diferentes al ideal. El ruido se puede clasificar en:

- Ruido aleatorio: se distribuye de forma aleatoria a lo largo del rango de brillo de la imagen.
- Ruido periódico: se distribuye periódicamente lo largo del rango de brillo de la imagen. Típicamente se origina por interferencias electromagnéticas. Se identifica y elimina por técnicas basadas en el dominio de la frecuencia.

Figura 2.1. Ruido aleatorio

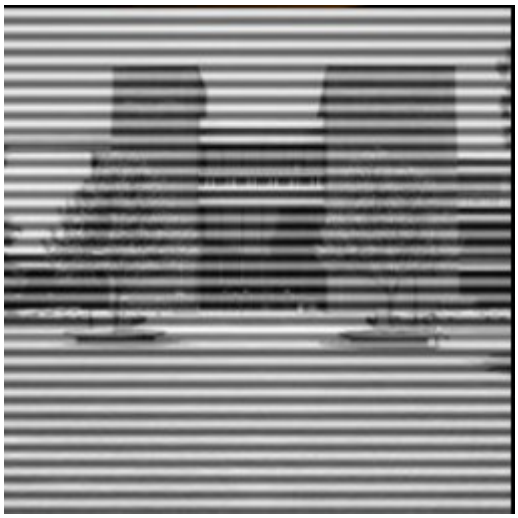


Figura 2.2. Ruido periódico

A) Ruido aleatorio

En imágenes convencionales el ruido aleatorio puede ser modelado básicamente con tres tipos de ruido comunes:

- distribución Gaussiana (normal,ruido electrónico).
- distribución Uniforme.
- distribución Sal y Pimienta (impulso).

Figura 2.3. Imagen original, con ruido gaussiano, uniforme y sal y pimienta

Con la distribución del ruido gaussiano, los valores de nivel de gris del ruido están distribuidos a lo largo de una campana gaussiana.

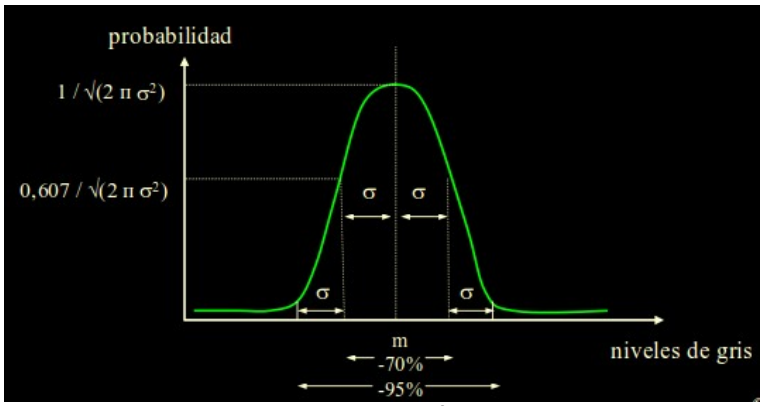


Figura 2.4. Distribución gaussiana

La distribución del ruido Gaussiano como una función de los niveles de gris puede ser modelada como un histograma de la siguiente forma:

$$histograma_{gaussiano} = \frac{1}{2\pi\sigma^2} e^{\frac{-(g-m^2)}{2\sigma}}$$

, donde:

g	nivel de gris
m	media
σ	distribución estándar (σ²=varianza)

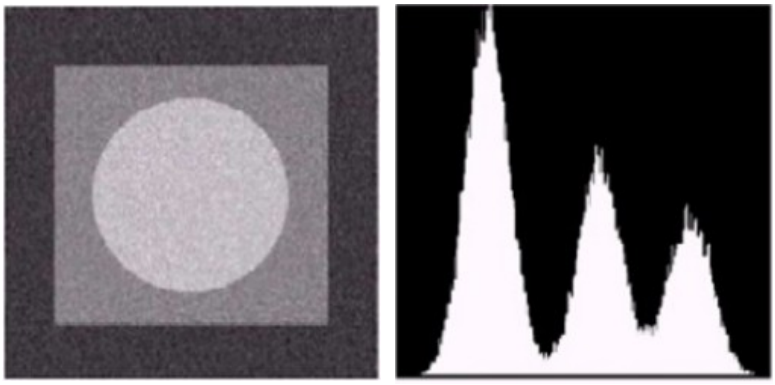


Figura 2.5. Imagen con ruido gaussiano y su histograma

Con la distribución del ruido uniforme, los valores de nivel de gris del ruido están distribuidos uniformemente a lo largo de un rango específico(menor o igual al rango [0,255], 8 bits).

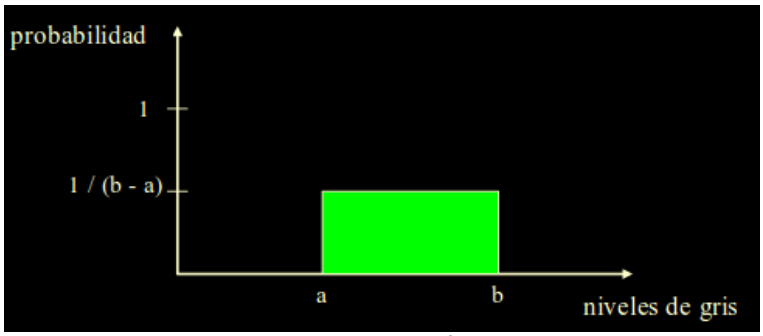


Figura 2.6. Distribución uniforme

La distribución del ruido uniforme está definida como:

$$histograma_{uniforme} = \begin{cases} \frac{1}{b-a}, para\ a \leqslant g \leqslant b, media = \frac{(a+b)}{2} \\ 0, por\ otro\ lado, varianza = \frac{(b-a)^2}{12} \end{cases}$$

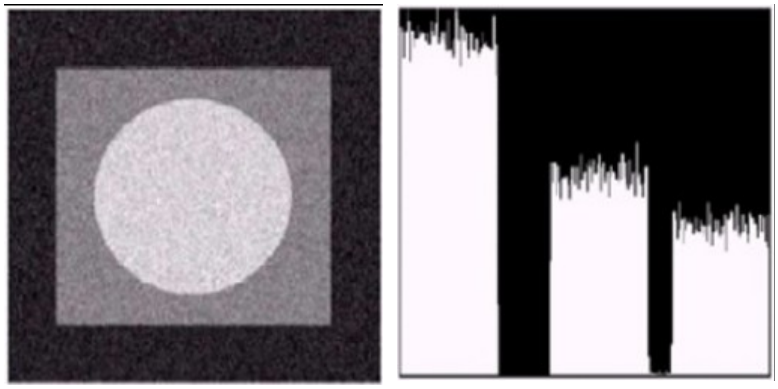


Figura 2.7. Imagen con ruido uniforme y su histograma

Con la distribución del ruido sal y pimienta, sólo existen 2 posibles valores para los niveles de gris del ruido, a y b.

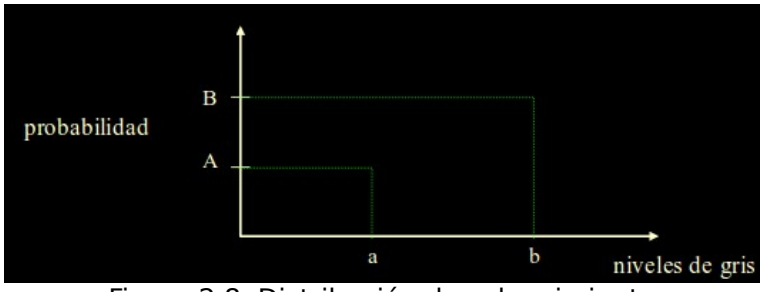


Figura 2.8. Distribución de sal y pimienta

$$histograma_{sal\ y\ pimienta} = \begin{cases} A\,, para\, g = a\,, (pimienta) \\ B\,, para\, g = b\,, (sal) \end{cases}$$

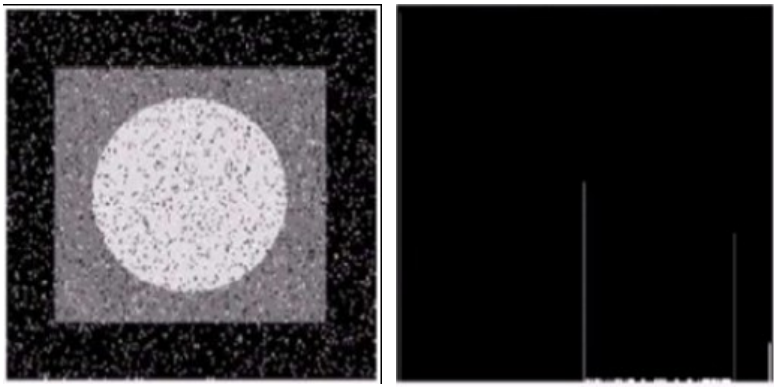


Figura 2.9. Imagen con ruido sal y pimienta, y su histograma

Existen más tipos de ruido aleatorios, como por ejemplo el ruido rayleigh, el ruido exponencial negativo, el ruido gamma y otros, que no se van a explicar en este trabajo.

B) Ruido periódico

El ruido periódico en muchas ocasiones se puede considerar como una señal sinusoidal vista en dos dimensiones finitas. caracterizada por su fase, su frecuencia de oscilación y su dirección de oscilación. Una función sinusoidal en un espacio de dos dimensiones tiene la forma:

$$f(m,n) = \sin(2\pi(Um + Vn))$$

donde *m*, *n* son las coordenadas espaciales (en píxeles), *U* y *V* son las dos

frecuencias (*ciclos/pixel*). El primer argumento del seno son radianes

$\frac{2\pi rad}{ciclo}\frac{ciclo}{pixel}pixel = rad$.

La frecuencia y la dirección máxima de oscilación son:

$$\Omega = \sqrt{U^2 + V^2} \qquad \theta = \arctan(U/V)$$

Se suele trabajar con frecuencias normalizadas al tamaño de la imagen *M x N*.

$$(u, v) = (MU, NV)$$

donde ahora las dimensiones son ciclos/imagen. $\frac{M\ pixels}{1\,imagen}\frac{ciclos}{pixel} = \frac{ciclos}{imagen}$

La expresión de la sinusoide es ahora:

$$f(m, n) = \sin 2\pi\left(\left(\frac{u}{M}\right)m + \left(\frac{v}{N}\right)n\right)$$

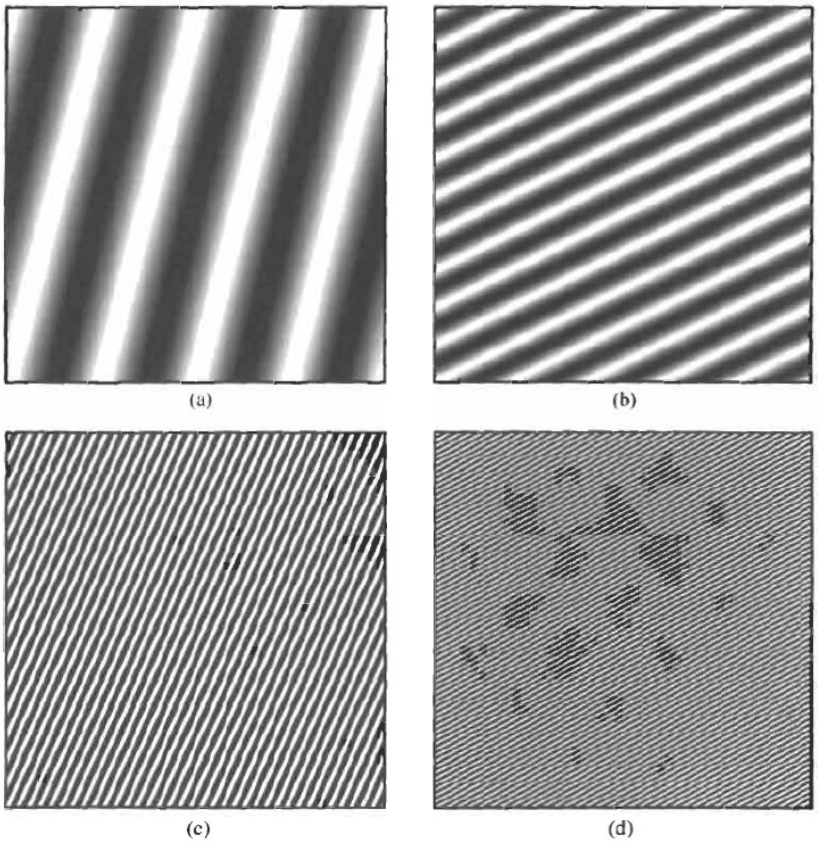


Figura 2.10. Ejemplo de funciones sinusoidales de dos dimensiones finitas. Las frecuencias escaladas medidas en ciclos/imágenes son, (a) u=1, v=4; (b) u=10, v=5; (c) u=15, Y=35; y (d) u=65, v=35.

Con la transformada de Fourier podemos transformar del dominio del espacio al dominio de frecuencia sin perder información de la imagen. El representar la información de la imagen en el dominio de la frecuencia tiene ventajas a la hora de aplicar algunos algoritmos y de determinar ciertas propiedades de la imagen. Cada armónico va a recoger un nivel de detalle, de cómo es la variación de los niveles de gris espacialmente.

Si construimos una imagen cuyo nivel de gris sea proporcional a la magnitud de la transformada de Fourier podemos representar gráficamente la transformada de Fourier. La forma del espectro revela información sobre la naturaleza de la imagen.

¿Qué características podemos obtener de la imagen observando su espectro de frecuencias?

- Zonas homogéneas en la imagen dará lugar a que la energía del espectro esté concentrada mayoritariamente en las bajas frecuencias.
- Zonas con muchos bordes, transiciones frecuentes de los niveles de gris dará lugar a un espresto con componentes de alta frecuencia.
- Si en la imagen existe alguna regularidad (patrones que se repiten) dará lugar a picos de igual intensidad separados una distancia regular.

Se suele representar como $\log_2(1+|F(u,v)|)$ para que su visualización sea mejor.

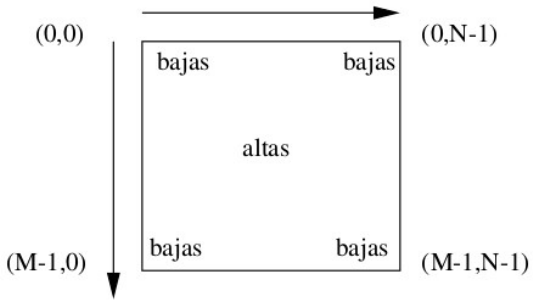


Figura 2.11. Representación DFT (*Transformada Discreta de Fourier*) de una imagen

Si se multiplica la DFT por una máscara con las formas que aparecen en la

figura 2.12, donde el valor blanco representa (1) y el negro (0), al calcular la inversa de la DFT obtendremos una imagen que contendrá sólo las frecuencias bajas, medias y altas.

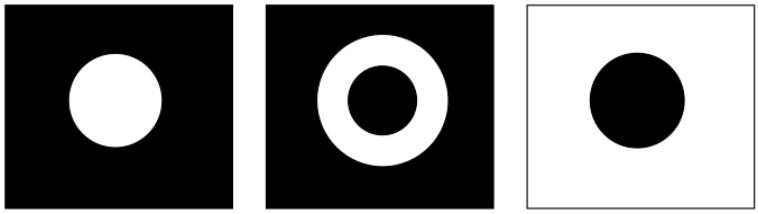


Figura 2.12. Representación de filtros pasa-baja, pasa-banda y pasa alta

En este trabajo se ha programado un tipo de filtro paso banda del tipo notch, este filtro en lugar de eliminar un anillo entero de frecuencias en el espectro, únicamente selecciona una parte de la frecuencia.

Estos filtros trabajan bien con ruido senoidal.

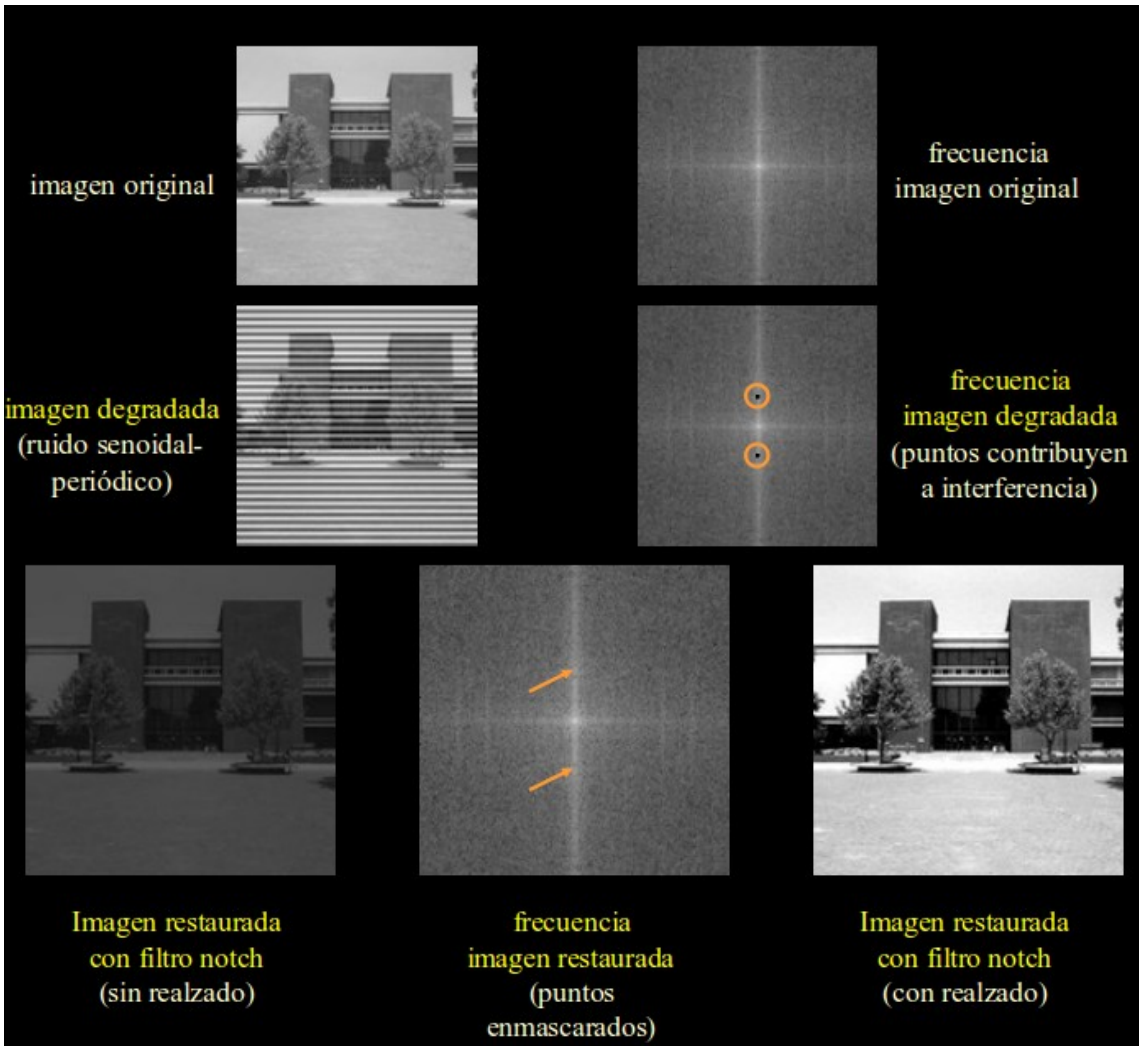


Figura 2.13. Aplicación filtro de notch

2.1.2. Ruido encontrado en las imágenes de HORUS

Las imágenes se ven afectadas principalmente por ruido frecuencial y gaussiano, como se puede ver con las utilizadas a continuación como ejemplos:

Figura 2.14. Ejemplos de imágenes con ruido

Dada la forma de funcionamiento de la detección de eventos en el servidor de vídeo IP, que compara el número de píxeles que han cambiado de un fotograma con el siguiente, dentro del área de seleccionada, con un porcentaje de sensibilidad. Niveles altos de ruido producirán que guarden una cantidad muy grande de falsos eventos, sin información útil para los investigadores.

En los apartados siguientes se describen los diferentes tipos de filtros que se han evaluado para la eliminación del ruido. La mayor parte de las imágenes que se han almacenado por la detección de movimiento, ha sido por ruido frecuencial. Este ruido afecta a determinadas zonas de la imagen, por ejemplo en la figura 2.15, la imagen está afectada por ruido frecuencial provocando líneas horizontales de mayor intensidad de gris.

Con el siguiente algoritmo desarrollado en matlab, podemos obtener el valor normalizado del sumatorio de los valores de los píxeles horizontales y verticales.

```
% Acondicionamiento de la imagen
I = imread('foto3.jpg'); % cargando la imagen a analizar
J = rgb2gray(I); % asegurando tener imagenes en escala de grises
J_r=imrotate(J,90);

a = sum(J_r);% la imagen se convierte en una matriz
a = a/max(a); % normalizando datos
b = sum(J);
b = b/max(b);

CH = (max(a) -min(a))/(max(a) +min(a)); % contraste de las franjas,
CV = (max(b) -min(b))/(max(b) +min(b));

T1=1:length(a);
T2=1:length(b);

%encuentra maximos locales una separacion entre ellos de 5 muestras
[peak_value, peak_location] = findpeaks(a,'MINPEAKDISTANCE',5);

figure;
subplot(2,2,1),imshow(J); title('Imagen original')
subplot(2,2,2),imshow(J_r); title('Imagen rotada')
subplot(2,2,3),plot(T2,b); title('Acumulacida columnas ')
subplot(2,2,4),plot(T1,a); title('Acumulacida filas')
```

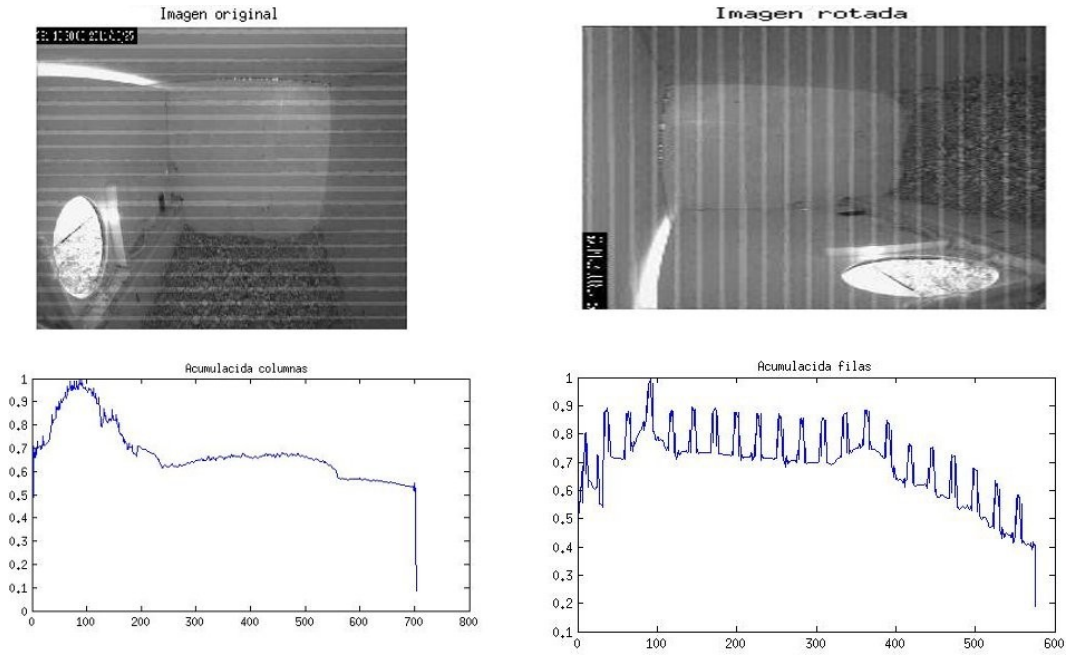


Figura 2.15. Gráficas de valores acumulados por columnas y filas en la imagen

Analizando como cambian los píxeles en una secuencia de vídeo, se observa que el ruido no afecta a todos los píxeles de forma aleatoria, las líneas horizontales se mantienen en las mismas posiciones de forma aproximada y no siempre con el mismo nivel de ruido. Como ejemplo de estudio se ha analizado

6 fotogramas de un vídeo y se han comparado los valores de las columnas 8 y 696. No se ha tenido en cuenta las primeras 40 líneas para evitar comparar el cuadro negro superpuesto con datos de fecha y hora.

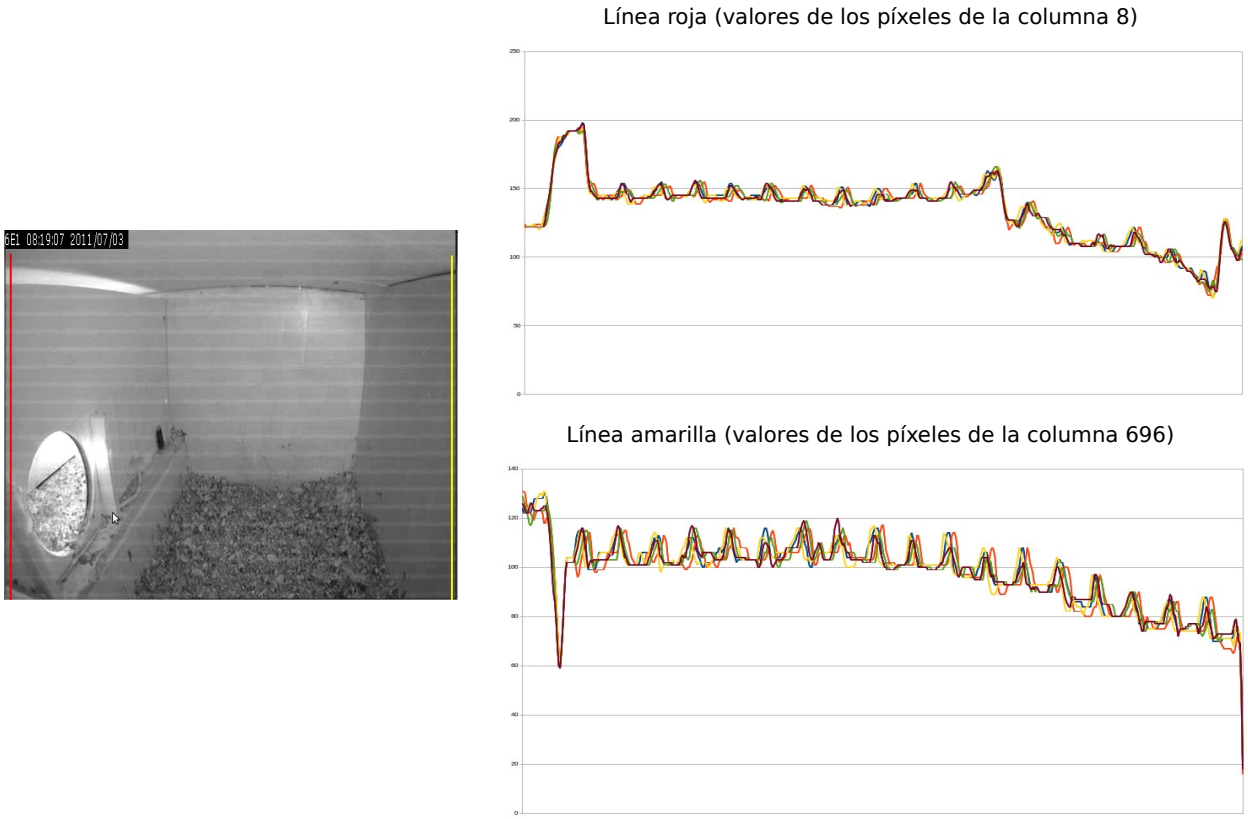


Figura 2.16. Gráficas de variaciones en los valores de los píxeles en una secuencia de vídeo

2.1.3 Filtros para reducir el ruido

A) Filtros en el dominio del espacio

Existen un conjunto de filtros que utilizan la propiedad de convolución en el dominio del espacio para suavizar el ruido, el valor resultante de un píxel se calcula como una función de su vecindad.

El concepto de máscara o *kernel* se entiende como una matriz de coeficientes donde el entorno del punto *(x,y)* que se considera en la imagen para obtener *g(x,y)* está determinado por el tamaño y forma del kernel seleccionado. Aunque la forma y tamaño de esta matriz es variable y queda a elección de cada usuario, es común el uso de kernels cuadrados *nxn*. Dependiendo de la implementación, en los límites de la imagen se aplica un tratamiento especial (se asume un marco exterior de ceros o se repiten los valores del borde) o no se aplica ninguno. Es por ello, que el tipo de filtrado queda establecido por el contenido de dicho kernel utilizado.

Para realizar un filtrado en el dominio del espacio se realiza una convolución del kernel sobre la imagen. Para ello se sigue el Teorema de Convolución en el espacio:

$$g(x,y)=h(x,y)*f(x,y)$$

1. Cada píxel de la nueva imagen se obtiene mediante el sumatorio de la multiplicación del kernel por los píxeles contiguos.
2. Generalmente se divide sobre cierto valor constante para normalizar que suele obtenerse de la suma de los valores del kernel empleado.

Dependiendo del kernel utilizado se pueden obtener los siguientes filtros:

- ***Promedio****:* promedio de píxeles vecinos (kernel de unos). La figura 2.16 muestra el resultado de aplicar la función de opencv “cvSmooth(image, promedio, CV_BLUR, 11, 11)”, en la cual se utiliza un kernel de 11x11.

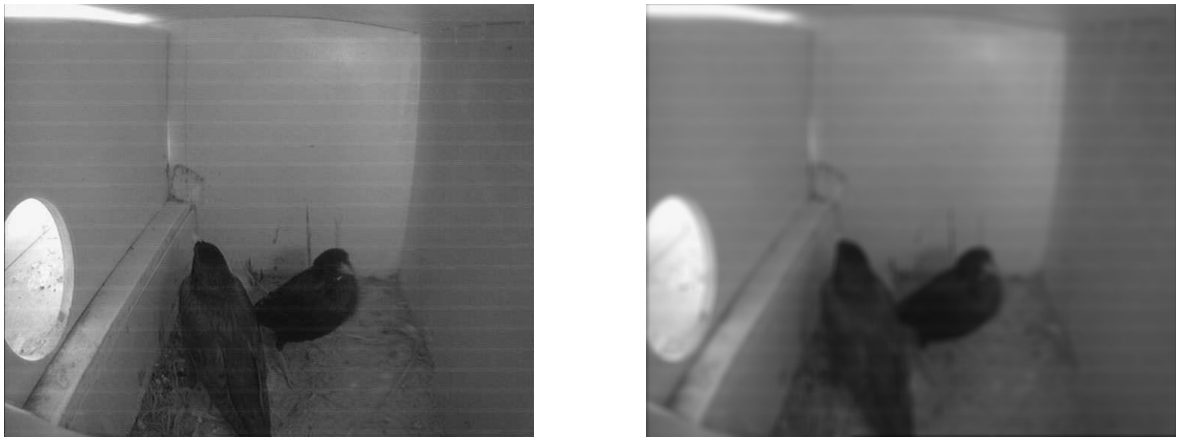


Figura 2.16. Imagen original y el resultado del filtro promedio

- ***Mediana****:* sustituye por el valor de la mediana de los píxeles vecinos (normalmente se comporta mejor que el de promedio). La figura 2.17 muestra el resultado de aplicar la función de opencv “cvSmooth(image, mediana, CV_MEDIAN, 11, 11)”.

Figura 2.17. Imagen original y el resultado del filtro promedio

- ***Gaussiano****:* utiliza un kernel con valores aproximados a la distribución gaussiana. La figura 2.18 muestra el resultado de aplicar la función de opencv “cvSmooth(image, gaussiano, CV_GAUSSIAN, 11, 11)” y en la figura 2.19 se ha utilizado un kernel de 1x11 para que el suavizado solo se realice con los píxeles de la misma columna, en opencv se realiza con la función “cvSmooth(image, gaussiano, CV_GAUSSIAN, 1, 11)” .

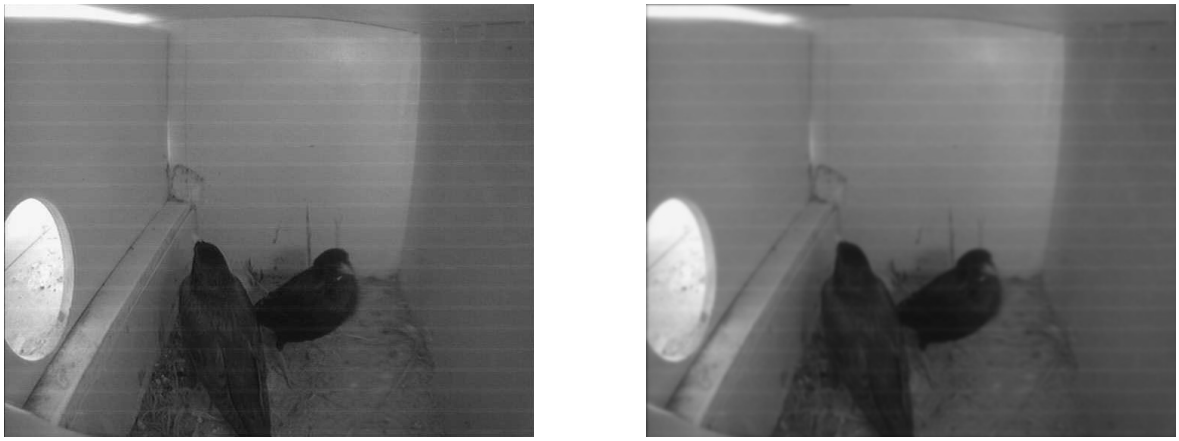


Figura 2.18. Imagen original y el resultado del filtro gaussiano (11x11)

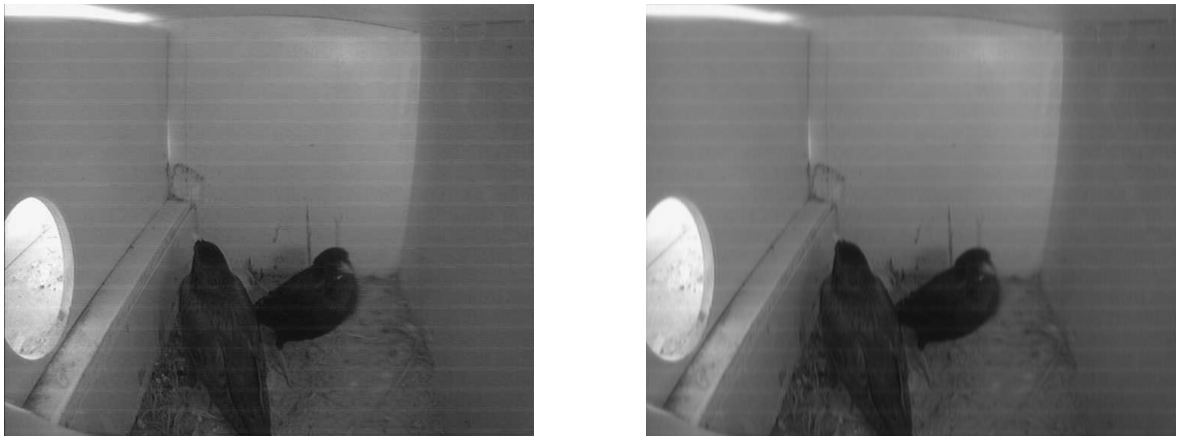


Figura 2.19. Imagen original y el resultado del filtro gaussino (1x11)

B) Filtrado a través de un conjunto de imágenes o vídeo.

Si se dispone de un conjunto de imágenes del mismo nido, sin objetos en movimiento en su interior, bajo las mismas situaciones de iluminación, y afectadas por ruido gaussiano de media cero, se puede calcular la imagen resultando como la media de todas las imágenes dando como resultado una imagen sin apenas ruido. Si por el contrario, el ruido es frecuencial aditivo, no vamos a poder filtrarlo al no tener media cero en este caso.

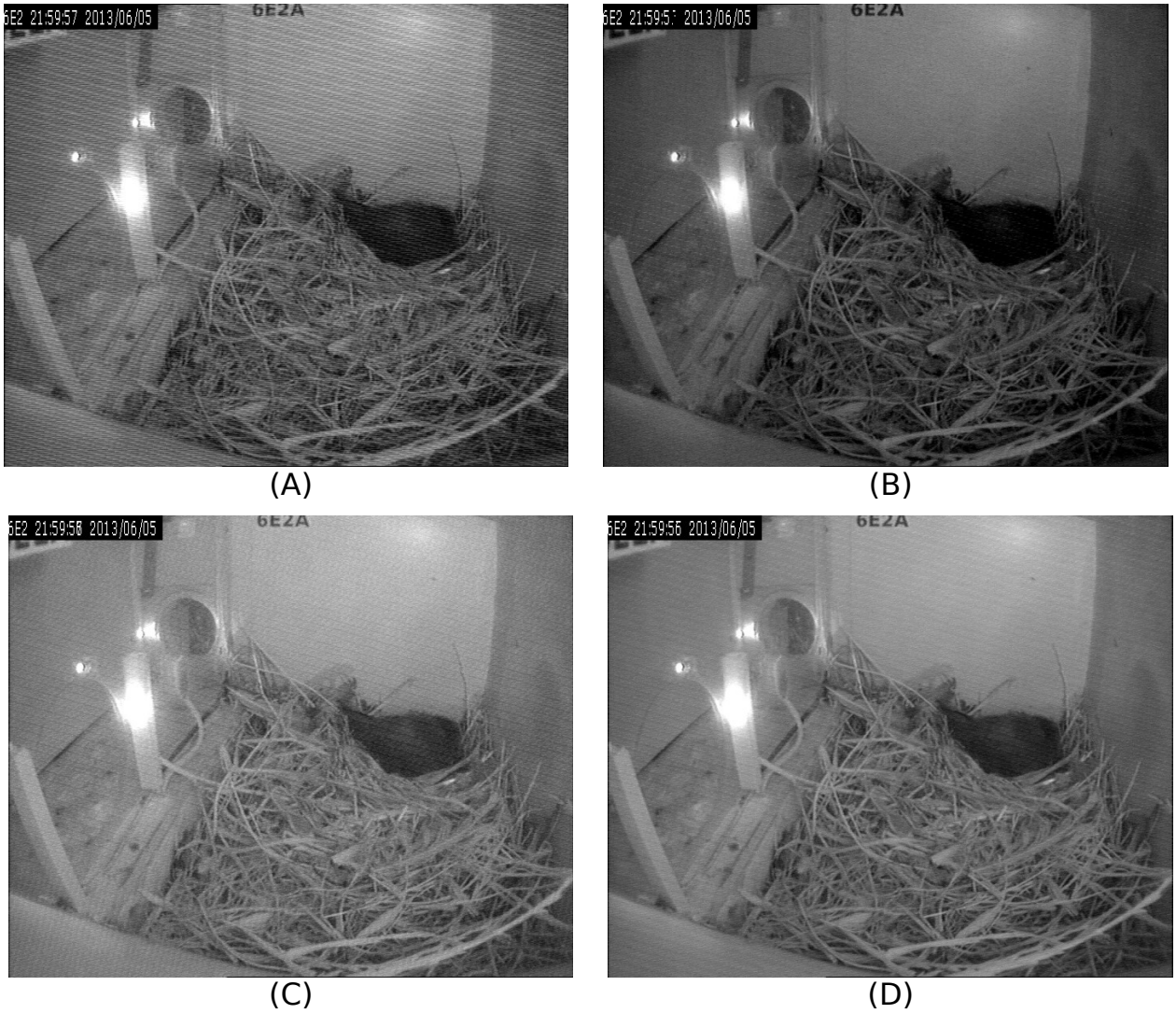


Figura 2.20. (A) Original (B) Reconstrucción con los valores mínimos, (C) Reconstrucción con los valores máximos (D) Reconstrucción con la media de los valores máximos y mínimos.

C) Filtros frecuenciales

Se ha implementado un tipo de filtrado frecuencial paso banda con los fundamentos del filtro notch, usando la transformada discreta del coseno (DCT). Se ha optado por implementar el filtro usando la DCT, porque el algoritmo es algo más rápid que el algoritmo del la transformada fápida de Fourier (FFT) y se evita usar números complejos.

```
RGB = imread('foto.jpg');
figure, imshow(RGB);
I = rgb2gray(RGB);
J = dct2(I);
figure, imshow(1+log(abs(J)),[]);
L=log(1 + abs(J));


for i=1:length(I(:,1))
```

```
   Umbral_h=max(abs(J(i,:)))*0.95;
   for j=1:length(I(1,:))
      % Umbral_v=max(abs(J(:,j)))*0.9;

       if ((i>20 || j>20))

          %quita líneas verticales
%               if  abs(J(i,j))>max(abs(J(:,j)))*0.95;
%                   J(i,j)=0;%mean(L(i,:));
%               end

              if  abs(J(i,j))>Umbral_h; %quita líneas horizontales
                  J(i,j)=0;%mean(L(:,j));
              end
       end
   end
end

K = idct2(J);
figure, imshow(log(1+abs(J)),[]);
figure, imshow(K,[0 255]);
```

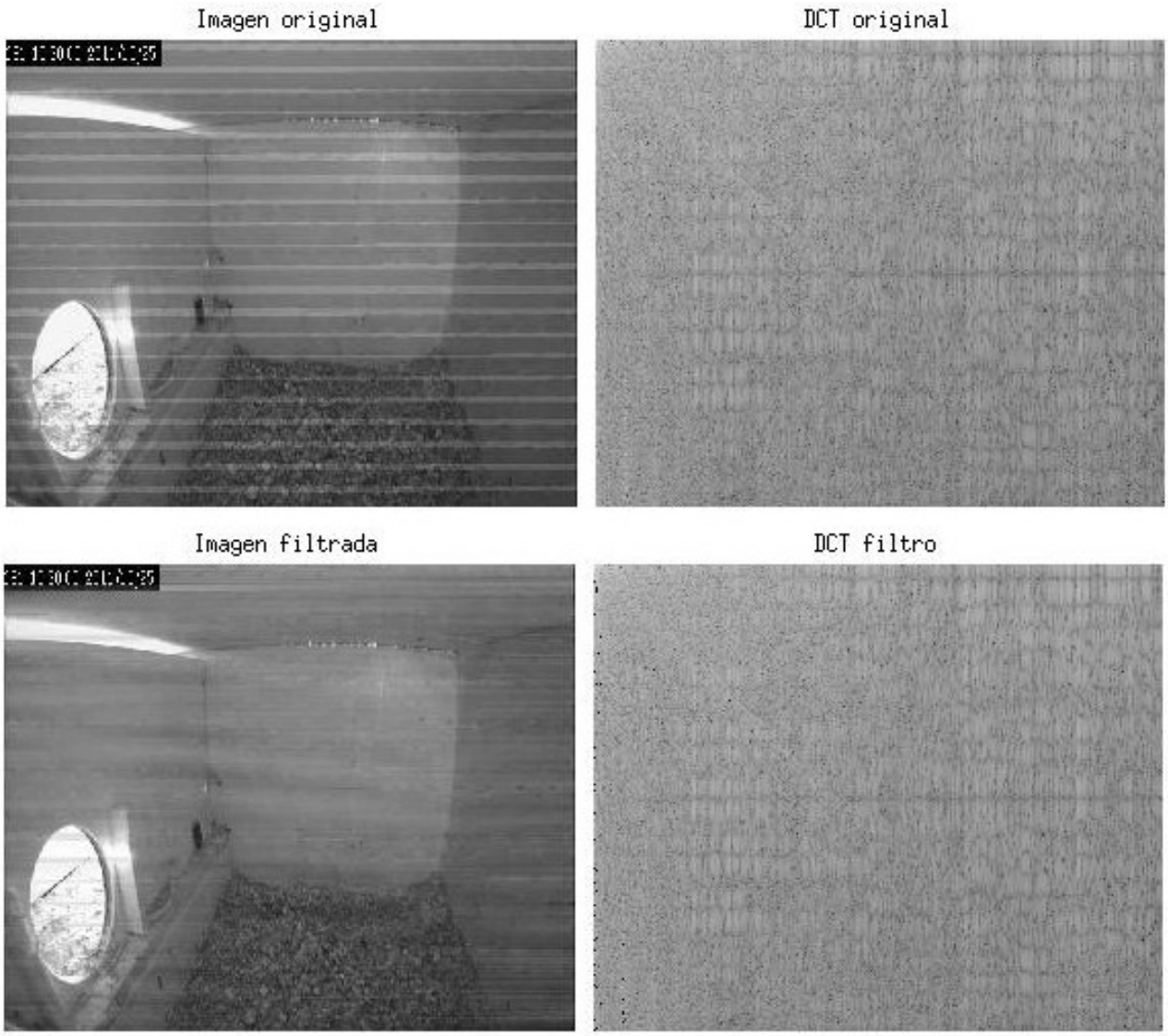


Figura 2.21. Resultados filtrado con DCT de una imagen afectada por ruido en líneas horizontales.

El algoritmo lo que hace es modificar los valores máximos de la DCT, a partir de un valor de frecuencia elegido, por un valor medio o cero, dado que la DCT tiene parte positiva y negativa.

Dependiendo del tipo ruido que afecte a la imagen conviene que sólo filtre las líneas horizontales o todas. Se ha implementado una función en opencv que realiza el filtrado y tiene como parámetros (imagen origen, imagen destino, frecuencia de corte=20, umbral horizontal en %, y umbral vertical en %).

Figura 2.22. Resultado de filtrar una imagen con mucho ruido periódico.

D) *Otros filtros estudiados*

Filtro bilateral

El filtro bilateral fue desarrollado por [1], y tiene la ventaja que realiza el suavizado del ruido conservando los bordes: el valor de un píxel se calcula en base a una media ponderada de los píxeles vecinos y con valores similares.

En las regiones más uniformes, los píxeles vecinos son similares entre sí y el filtro actúa eliminando las pequeñas diferencias atribuibles al ruido. Cuando el píxel central se encuentra en un borde entre zonas oscuras y claras el filtro reemplaza su valor por la media de los píxeles brillantes, ignorando los oscuros (cuando se centra en uno oscuro se promedian los oscuros e ignoran los brillantes). Este comportamiento permite mantener los bordes. En un caso

particular, las funciones de ponderación de cercanía c(x) y similitud s(x) son gausianas:

$$c(\xi - x) = \exp\left\{-\frac{1}{2}\left(\frac{d(\xi - x)}{\sigma_d}\right)\right\}$$

$$s(f(\xi) - f(x)) = \exp\left\{-\frac{1}{2}\left(\frac{\delta(\xi - x)}{\sigma_r}\right)\right\}$$

siendo $d(\xi - x)$ la distancia euclídea y $\delta(\xi - x)$ la distancia entre valores de intensidad de los píxeles, $|f(\xi) - f(x)|$. El filtro combinado toma entonces la forma:

$$h(x) = \frac{1}{k(x)} \int_{-\infty}^{\infty} f(\xi)\, c(\xi - x)\, s(f(\xi) - f(x))\, d\xi$$

siendo $k(x)$ el factor de normalización:

$$k(x) = \int_{-\infty}^{\infty} c(\xi - x)\, s(f(\xi) - f(x))\, d\xi$$

A continuación se describen los atributos usados de la función **bilateralFilter** que está disponible en OpenCV.

Atributos de la función bilateralFilter:

- **src:** imagen de entrada.
- **dst:** imagen resultado.
- **D:** Diametro de distancia de cada píxel vecino que es usado durante el filtrado. Si se le da un valor negativo, este valor es calculado por el valor de sigmaSpace.
- **SigmaColor:** valor sigma del filtro en el espacio de intensidad de color. Un valor alto de este parámetro significa que dentro del conjunto de píxeles vecinos se mezclaran aunque hayan niveles de intensidad de color muy superiores.
- **sigmaSpace** – valor sigma del filtro en el espacio de coordenadas. Un valor alto de este parámetro significa que los píxeles más lejanos se influenciarán mutuamente siempre que sus intensidades de color sean lo suficientemente parecidos.

Figura 2.23 . Imagen con ruido y resultado tras aplicarle bilateralFilter(src, dst, 31, 20, 10)

Filtro fastNlMeansDenoising (filtrado de ruido con medias no locales)

Este filtrado fue descrito en [2], y se basa en un principio simple: sustituye el valor del píxel por la media de valores de píxeles similares encontrados en la imagen.

Atributos de la función fastNlMeansDenoising:

- **src:** imagen de entrada.
- **dst:** imagen resultado.
- **TemplateWindowSize:** Tamaño en píxeles de la parte de la platilla que se utiliza para calcular los pesos. Debe ser impar, se recomienda el valor de 7 píxeles.
- **SearchWindowSize:** Tamaño en píxeles de la ventana que es usada para computar el promedio ponderado de un píxel determinado. Debería ser impar. Afecta linealmente al rendimiento: mayor tamaño de la ventana es equivalente a mayor tiempo para la eliminación del ruido. El valor recomendado es de 21 píxeles.
- **h:** Parámetro que regula la intensidad del filtro. El gran valor de h elimina perfectamente el ruido pero también elimina los detalles de la imagen, el valor h más pequeño conserva los detalles pero también conserva el ruido

Ventajas:

Funciona bien con imágenes que están afectadas por ruido en zonas determinadas, logrando suavizarlo en las zonas afectadas y no alterando las zonas donde no hay ruido. Elimina bien el ruido gaussiano.

Inconvenientes:

Costoso computacionalmente, y no útil para casos donde la imagen tiene mucho ruido frecuencial.

Figura 2.24. A la izquierda tres imágenes distintas afectadas con ruido, a la derecha el resultado de aplicar el filtro `fastNlMeansDenoising` con h=3,templateWindowSize=7 y searchWindowSize=21.

2.2 Obtención del valor de blur

2.2.1 Imágenes borrosas

Una imagen borrosa es aquella cuyos bordes y formas no están claramente definidos. Esta ausencia o degradación de bordes es utilizado para obtener el factor de nitidez ("*blur*") de las imágenes, en este trabajo es calculado utilizando la transformada wavelet de Haar propuesta en [3].

De la misma forma, en [4], se detectan los bordes verticales mediante un filtro de Sobel, y la medida de blur se relaciona con el ancho de los bordes detectados. El valor final que se adjudica a la imagen es la media del ancho de los bordes detectados.

En [5] la falta de bordes se entiende como la desaparición de las altas frecuencias de la imagen. Una herramienta muy útil para conseguir el factor de blur es la DCT, con cuyos coeficientes se elabora un histograma de frecuencias que determina si la imagen es borrosa o no.

2.2.2 Wavelets (transformada de Haar)

En 1909 Alfred Haar descubrió un conjunto de funciones ortonormales a partir de las que se podía representar otra función. El hallazgo de Haar no es más que la más simple de las wavelets ortogonales: la transformada de Haar. La aplicación de esta transformada permite esencialmente obtener un trazado de los bordes horizontales, verticales y diagonales que se encuentran en una imagen.

Basándose en varios estudios de bordes y con la ayuda de la transformada de Haar, los autores de [3] , son capaces de determinar si una imagen es borrosa o no y qué extensión de esa imagen no es nítida (*blur extent*).

En estos estudios de bordes se definen los tipos principales que se distinguen y se clasifica su comportamiento frente al efecto borroso y la transformada Haar. Los tipos de bordes considerados vienen definidos por la estructuras siguientes: la estructura delta de Dirac, la estructura Tejado, la estructura Escalón A y la estructura Escalón G.

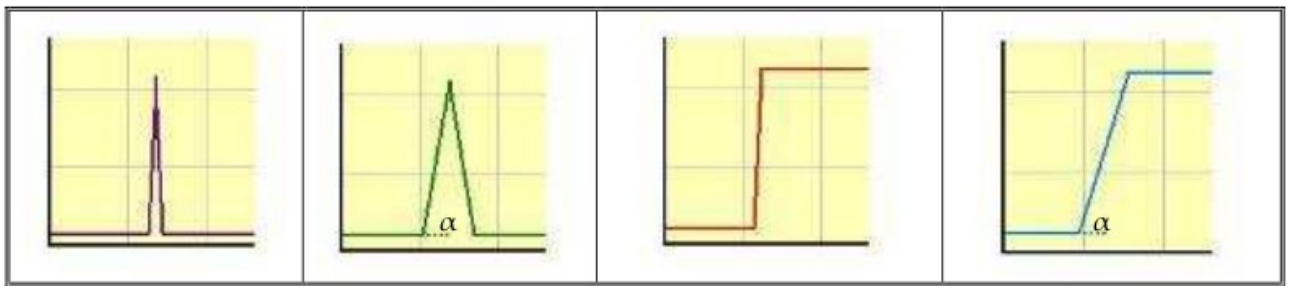


Figura 2.25. Tipos de estructuras de bordes: dirac, tejado, escalón A y escalón G.

Cada una de ellas cumple con unas características concretas: la estructura Dirac representa líneas finas en la imagen; la estructura Tejado se comporta de la misma forma que la Dirac pero en líneas más gruesas, en las que se note la variación progresiva del color o intensidad; por último, las estructuras Escalón representan los cambios de una zona de color o intensidad a otra, siendo la variación más abrupta en el caso de la estructura A y más progresiva en la G. Tanto en la estructura Tejado como en la estructura Escalón G se puede hablar de un ángulo α que representa la pendiente de la curva de variación de color o intensidad.

Según el estudio de bordes mencionado, los bordes tipo Dirac y Escalón A no existen en imágenes borrosas, (esta afirmación es la que les permite decidir si una imagen es borrosa o no) y los tipo Tejado y Escalón G pasan a tener un ángulo α que tiende a 0, es decir, estos bordes tienden a perder su nitidez.

Una vez conocido el comportamiento de los bordes frente al blur, es preciso ver cómo caracteriza la transformada Haar los distintos tipos de bordes para poder extraer conclusiones. A continuación se presentan algunos ejemplos de cómo trabaja la transformada Haar de 3 niveles con imágenes nítidas y borrosas:

Figura 2.26. Imagen nítida y borrosa

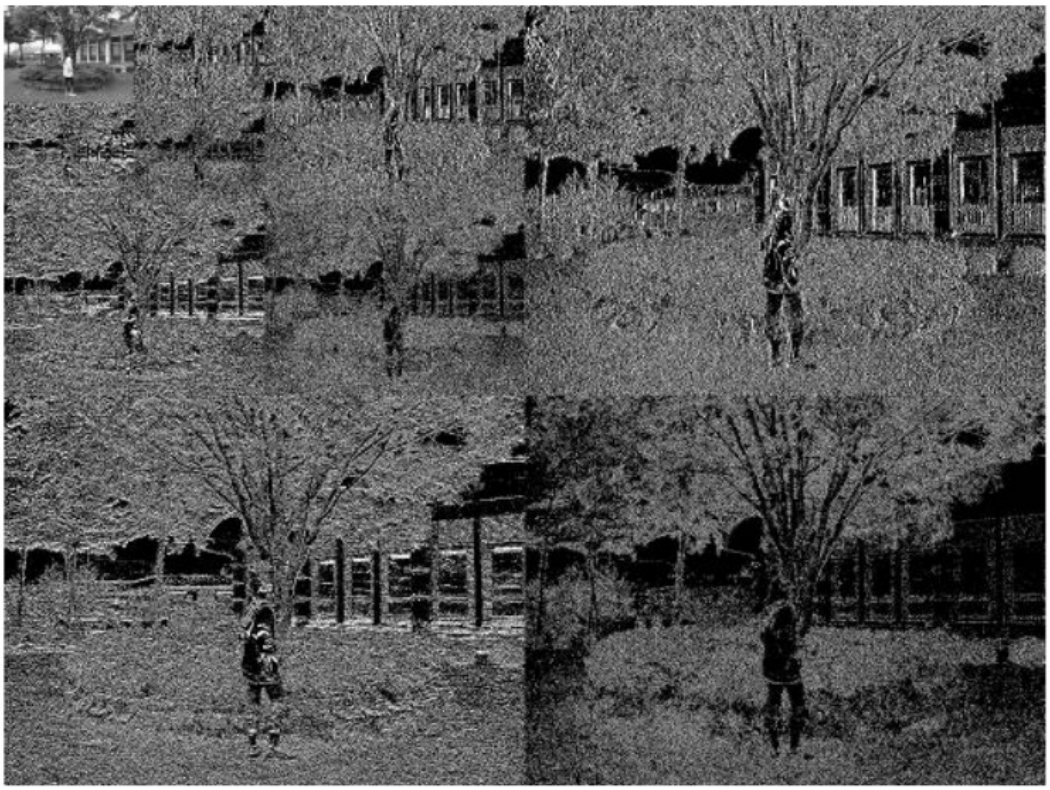


Figura 2.27. Transformada Haar de 3 niveles de una imagen nítida

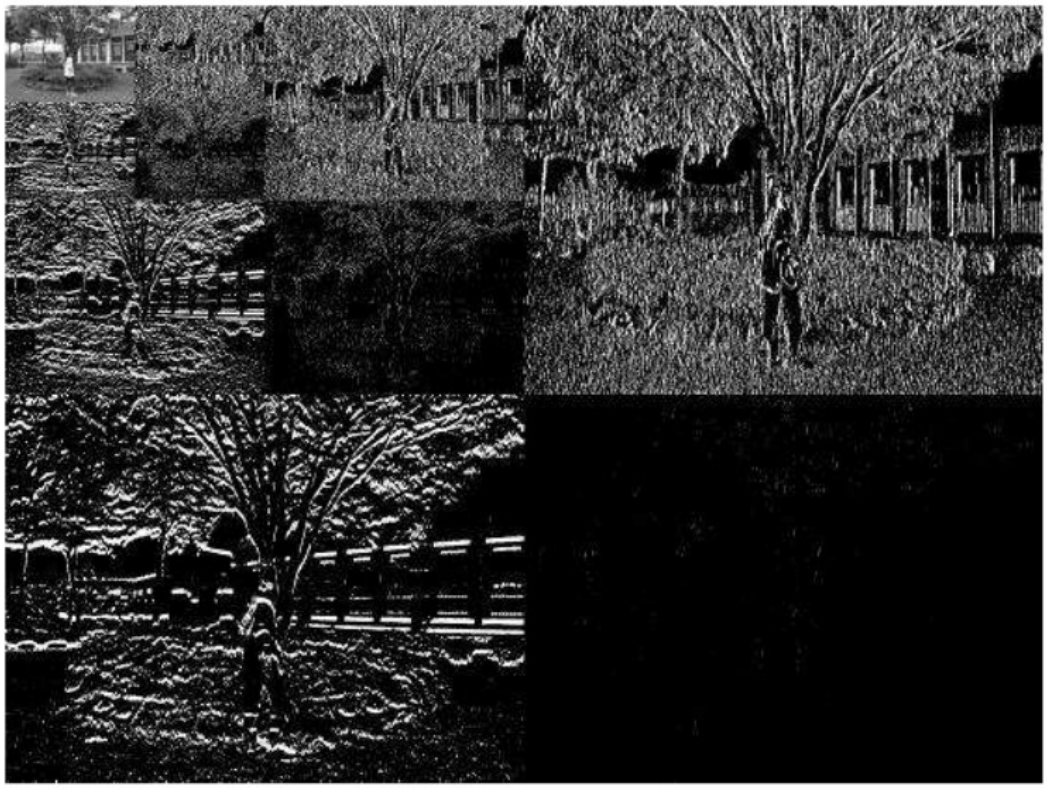


Figura 2.28. Transformada Haar de 3 niveles de una imagen borrosa

En la imagen nítida de la figura 2.26 se pueden considerar bordes tipo Delta, por ejemplo, algunas briznas de hierba, y bordes tipo Escalón A los troncos de los árboles o las columnas del edificio. El detalle de la imagen se mantiene en la imagen de la transformada de Haar (figura 2.27) hasta el tercer nivel, es decir, las estructuras características de las imágenes nítidas son definidas por la transformada en cada uno de los niveles estudiados, si bien son más claros en el primero.

Ahora, si volvemos borrosa la imagen de la figura 2.26 mediante un filtro gaussiano 3x3 y una desviación de 5.0, simulando un desenfoque, se obtienen los resultados que aparecen en la figura 2.28. Los bordes que antes representaban el tipo Delta y Escalón A han desaparecido por el efecto del desenfoque, y en la imagen de la transformada de Haar de tres niveles sólo se muestran las estructuras tipo Escalón G y Tejado, que se distinguen en el nivel 1 por sus líneas mucho más gruesas. La representación obtenida pierde detalle, que es justamente lo que sucede cuando una imagen se vuelve borrosa. Para completar el ejemplo se exponen seguidamente las imágenes de las transformadas de Haar de tres niveles de las fotografías que se muestran al

inicio de este apartado como modelo de imágenes borrosas.
Una vez observado el efecto de la transformada de Haar sobre imágenes nítidas y borrosas, los autores de [3] proponen el trazado de un mapa de bordes para la obtención del valor de la extensión de blur.

La imagen resultado derivada de la aplicación de la transformada es una matriz de valores en el rango de 0 (negro) a 255 (blanco). Cada nivel tiene un tamaño diferente y una representación de los bordes verticales, horizontales y diagonales de la imagen. Para construir un mapa de los bordes de cada nivel basta con recoger el valor de la matriz correspondiente al mismo punto de cada nivel y aplicar la siguiente fórmula:

$$Emap_i(k,l)=\sqrt{V_i(k,l)^2+H_i(k,l)^2+D_i(k,l)^2}$$

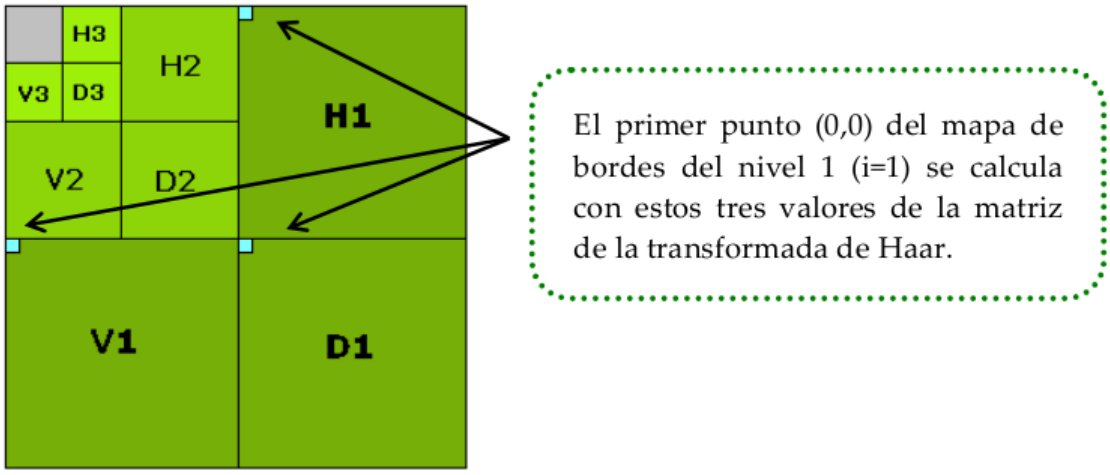


Figura 2.29. Estructura piramidal de la transformada de Haar

Pero como los Emaps tienen tamaños diferentes, los valores obtenidos en los Emaps han de ser reducidos mediante el uso de ventanas antes de obtener resultados. El objetivo de la ventana es determinar el valor del máximo local de los puntos de Emap que engloba la ventana, sin que ésta se solape en ningún caso. Precisamente, la ventana de Emap1 tendrá un tamaño de 8x8 celdas, la de Emap2 una de 4x4 celdas y la de Emap3 una de 2x2 celdas, de forma que los Emax que surjan del cálculo tengan todos el mismo tamaño.

Si $Emax_i$ (k,l) supera un cierto umbral, entonces el píxel (k,l) de la imagen se puede considerar un borde.

Según [3], el efecto de este algoritmo sobre los diferentes tipos de bordes es el siguiente:

	Emax1	Emax2	Emax3
Estructura Dirac	Mayor	Medio	Menor
Estructura Escalón A	Mayor	Medio	Menor
Estructura Escalón G	Menor	Medio	Mayor
Estructura Tejado	Menor	Medio	Mayor
	Menor	Mayor	Medio

Tabla 2.1: Valores de $Emax_i$ según el tipo de borde

Como se puede observar, la intensidad de los bordes que definen una imagen nítida disminuye a medida que aumentan los niveles de la transformada, al contrario que lo que sucede en los bordes que se encuentran en las imágenes borrosas, que la transformada tiende a hacer más finos (con mayor intensidad) según aumentan los niveles. Esta es una de las propiedades de la transformada Haar: la capacidad de recuperar la nitidez de los bordes borrosos. El comportamiento de los bordes con respecto al algoritmo propuesto permite al programador emparejar una región de la imagen con un tipo de borde diferente.

La clave para determinar la extensión de blur es clasificar qué bordes de las estructuras Tejado y Escalón G parecen pertenecer a una imagen borrosa. Para ello se hace uso de la propiedad de la transformada de Haar, primero buscando los puntos (k,l) en los que Emax1 < Emax2 < Emax3 (Tejado o Escalón G), o $Emax_2 > Emax_1$ y $Emax_2 > Emax_3$ (Tejado), que corresponderían con estructuras Tejado y Escalón G en la imagen. Como la transformada convierte este tipo de bordes borrosos en bordes más nítidos según se aumenta de nivel, si dichos puntos (k,l) tienen un Emax1 correspondiente menor que cierto umbral se considera que se encuentran en una imagen borrosa.

La fórmula para obtener la extensión de blur de la imagen será, por tanto:

$$BlurExtent = \frac{N_b}{N}$$

Siendo N el número de bordes Tejado y Escalón G encontrados en la imagen y Nb el número de bordes contabilizados en N que se han considerado borrosos.

La implementación de la transformada de Haar de tres niveles se puede realizar con el sencillo algoritmo mostrado en forma de diagrama en la figura 2.30.

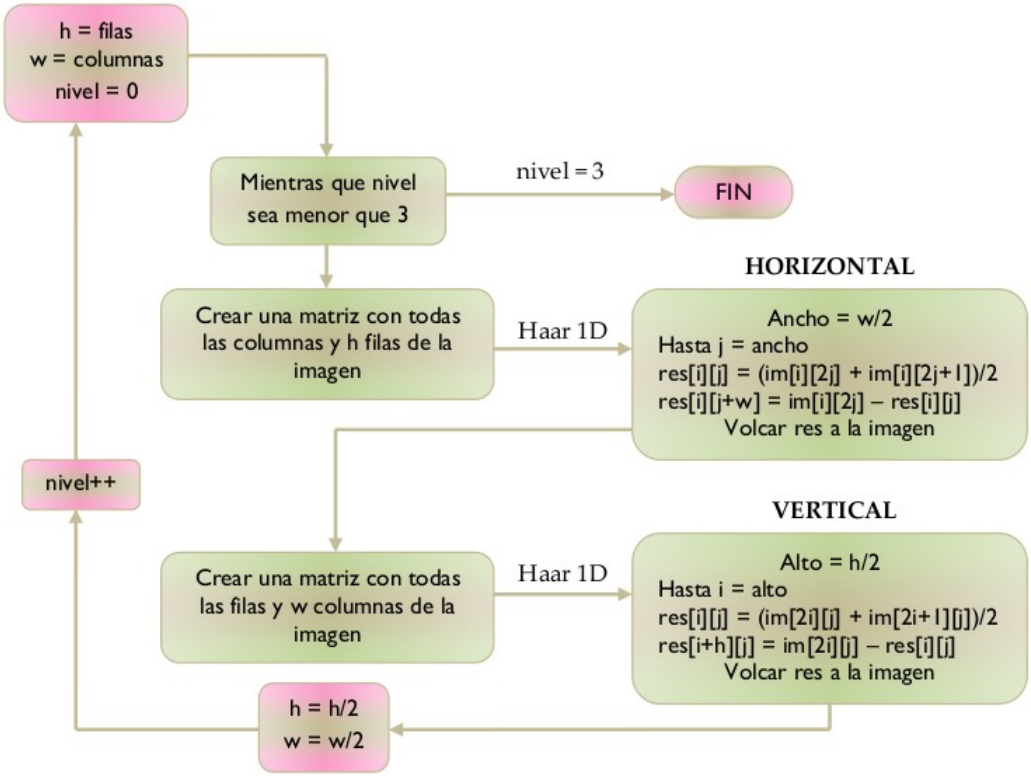


Figura 2.30. Diagrama de módulos para calcular la transformada de Haar de tres niveles de una imagen

He realizado su programación para el programa creado en el lenguaje C++ con las librerías de OpenCV.

```
void HAAR(IplImage* src, IplImage* dst){
    int altura,anchura,canales,w,h;
    uchar *data;
    int i,j, nivel;
    altura = src->height;
    anchura = src->width;
    canales  = src->nChannels;
    data= (uchar *)src->imageData;
```

```
    IplImage* tmp = cvCreateImage(
            cvSize( anchura,altura),
            src->depth,
            canales
        );
    IplImage* tmp1 = cvCreateImage(
            cvSize( anchura,altura),
            src->depth,
            canales
        );
    //Ecualizo el histograma para obtener mejor contraste
    cvEqualizeHist( src, tmp );
    printf("Transformada imagen HAAR de %dx%d píxeles con %d canales\n",altura, anchura, canales);
    w=anchura;
    h=altura;
    CvScalar pixel1,pixel2;;
    for(nivel=0;nivel<3;nivel++){
        w=w/2;
        for(i=0;i<altura;i++) {
            for(j=0;j<w;j++) {
                pixel1.val[0]=(cvGet2D(tmp,i,2*j).val[0]+cvGet2D(tmp,i,2*j+1).val[0])/2;
                pixel1.val[1]=(cvGet2D(tmp,i,2*j).val[1]+cvGet2D(tmp,i,2*j+1).val[1])/2;
                pixel1.val[2]=(cvGet2D(tmp,i,2*j).val[2]+cvGet2D(tmp,i,2*j+1).val[2])/2;
                pixel1.val[3]=(cvGet2D(tmp,i,2*j).val[3]+cvGet2D(tmp,i,2*j+1).val[3])/2;
                pixel2.val[0]=abs(cvGet2D(tmp,i,2*j).val[0]-pixel1.val[0]);
                pixel2.val[1]=abs(cvGet2D(tmp,i,2*j).val[1]-pixel1.val[1]);
                pixel2.val[2]=abs(cvGet2D(tmp,i,2*j).val[2]-pixel1.val[2]);
                pixel2.val[3]=abs(cvGet2D(tmp,i,2*j).val[3]-pixel1.val[3]);
                cvSet2D(tmp1,i,j,pixel1);
                cvSet2D(tmp1,i,j+w,pixel2);
            }
        }
        h=h/2;
        for(i=0;i<h;i++) {
            for(j=0;j<anchura;j++) {
                pixel1.val[0]=(cvGet2D(tmp1,2*i,j).val[0]+cvGet2D(tmp1,2*i+1,j).val[0])/2;
                pixel1.val[1]=(cvGet2D(tmp1,2*i,j).val[1]+cvGet2D(tmp1,2*i+1,j).val[1])/2;
                pixel1.val[2]=(cvGet2D(tmp1,2*i,j).val[2]+cvGet2D(tmp1,2*i+1,j).val[2])/2;
                pixel1.val[3]=(cvGet2D(tmp1,2*i,j).val[3]+cvGet2D(tmp1,2*i+1,j).val[3])/2;
                pixel2.val[0]=abs(cvGet2D(tmp1,2*i,j).val[0]-pixel1.val[0]);
                pixel2.val[1]=abs(cvGet2D(tmp1,2*i,j).val[1]-pixel1.val[1]);
                pixel2.val[2]=abs(cvGet2D(tmp1,2*i,j).val[2]-pixel1.val[2]);
                pixel2.val[3]=abs(cvGet2D(tmp1,2*i,j).val[3]-pixel1.val[3]);
                cvSet2D(dst,i,j,pixel1);
                cvSet2D(dst,i+h,j,pixel2);
            }
        }
        anchura=w;
        altura=h;
        tmp=cvCloneImage(dst);
    }
    cvNamedWindow( "HAAR3", 1 );
    cvShowImage( "HAAR3", dst );

    //libero memoria
    cvReleaseImage(&tmp);
    cvReleaseImage(&tmp1);
}
```

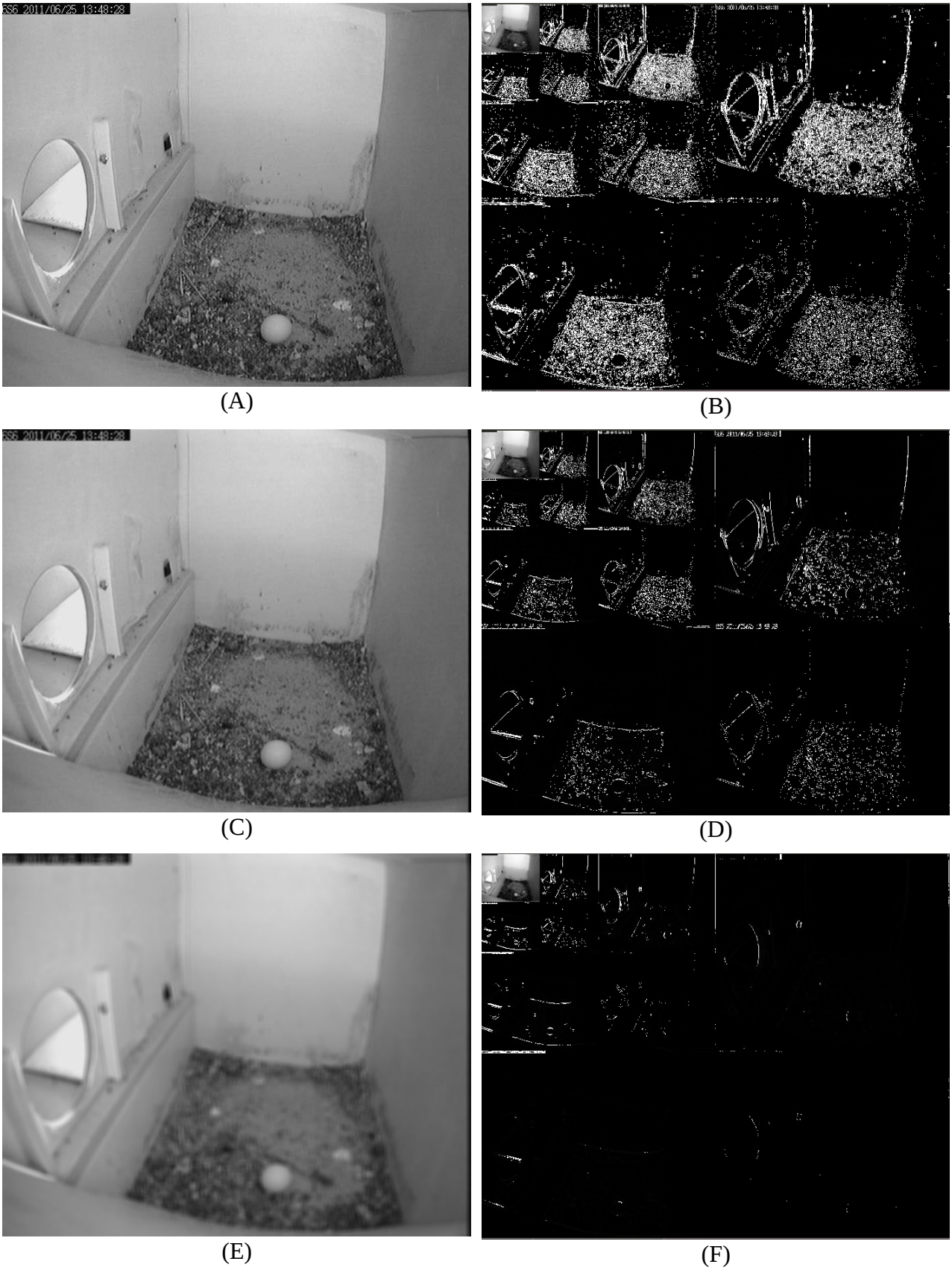
Además de realizar el algoritmo para calcular la transformada de Haar para tres niveles (aunque se puede aumentar fácilmente), se ha desarrollado otro procedimiento para calcular el número de bordes considerados como Tejado y Escalón G, y el nivel de blur extendido; utilizando como parámetros la imagen obtenida como transformada de Haar de tres niveles, el valor mínimo para considerar un valor como borde y el tamaño de la ventana.

```
double calcularEmap(IplImage* img, double blur, int size_windows){
    int altura,anchura,w,h;
    uchar *data;
    int i,j,nivel,ei,ej;
    int N=0,Nb=0;
    altura = img->height;
    anchura = img->width;
    data= (uchar *)img->imageData;
    //Calculo las dimensiones de los emap
    double emap[altura/2][anchura/2][3];
    double H=0,D=0,V=0;
    w=anchura;
    h=altura;
    //Paso 1: Recorro la imagen a la que se le ha aplicacdo la transformada HAAR de 3 niveles
calculando los valores de los emap
    for(nivel=0;nivel<3;nivel++){
       //Calculo para el tamaño de la ventana de (size_windows x size_windows = 8x8) el valor
máximo
        h=h/2;
        w=w/2;
        for(i=0;i<h;i++) {
            for(j=0;j<w;j++) {
                //Tengo que calcular los valores de Hi, Vi, Di y calcular el valor de su emap
correspondiente
                H=cvGet2D(img,i,j+w).val[0];
                V=cvGet2D(img,i+h,j).val[0];
                D=cvGet2D(img,i+h,j+w).val[0];
                //Calculo emap
                emap[i][j][nivel]=sqrt(pow(H,2)+pow(V,2)+pow(D,2));
            }
        }
    }
    //Ya tenemos los valores de los emaps para cada nivel de la transformada HAAR
    //Paso 2: Calculo el número de bordes detectados para cada nivel de HAAR segun el nivel de
blur seleccionado ajustando al máximo por ventana
    double emax1=0,emax2=0,emax3=0;
    h=altura/2;
    w=anchura/2;
    for(i=0;i<h;i=i+size_windows){
        for(j=0;j<w;j=j+size_windows){
            emax1=0,emax2=0,emax3=0;
            //Se calcula el valor máximo por ventana para emap1
            for(ei=0;ei<size_windows;ei++){
                for(ej=0;ej<size_windows;ej++){
                    if(emax1<emap[i+ei][j+ej][0]){
                        emax1=emap[i+ei][j+ej][0];
                    }
                }
            }

            //Se calcula el valor máximo por ventana para emap2
            for(ei=0;ei<size_windows/2;ei++){
                for(ej=0;ej<size_windows/2;ej++){
                    if(emax2<emap[i/2+ei][j/2+ej][1]){
                        emax2=emap[i/2+ei][j/2+ej][1];
                    }
                }
            }
            //Se calcula el valor máximo por ventana para emap3
            for(ei=0;ei<size_windows/4;ei++){
                for(ej=0;ej<size_windows/4;ej++){
                    if(emax3<emap[i/4+ei][j/4+ej][2]){
                        emax3=emap[i/4+ei][j/4+ej][2];
                    }
                }
            }

            //Buscamos bordes escalón o tejado totales N y el número de bordes Nb considerados
borrosos
            if(emax1 < emax2  && emax2 < emax3){
                N=N+1;
                if(emax1 > blur){
                    Nb=Nb+1;
                }
```

```
            }
        }
    }
    //Como ya tenemos el valor de Nb y N podemos calcular el valor de Blur extendido
    double blur_ext=0;
    if(N!=0){
        blur_ext=Nb;
        blur_ext=blur_ext/N;
    }
    printf("!!!!!!!!!!!Número de bordes tejado o escalón G %d \n",N);
    printf("!!!!!!!!!!!Blur extendido %f \n",blur_ext);
    return blur_ext;
}
```



(A) (B)

(C) (D)

(E) (F)

Figura 2.31. Ejemplos de transformadas de Haar de 3 niveles.

En la figura 2.31 se observa en la imagen original (A) y su transformada de

Haar (B) obteniéndose con el algoritmo empleado un total de **N=153** considerados tipo Tejado o Escalón G, si a la imagen original le aplicamos un filtro promedio 9x9 resultando la imagen (C) y su transformada de Haar (D) obteniendo **N=259**, y por último se ve la imagen resultante al usarse un filtro promedio 9x9 (E) y su transformada de Haar (F), dando un valor de **N=613**. Por tanto se puede comprobar que aumenta el número de bordes Tejado o Escalón G, a medida que la imagen se hace más borrosa.

3. Segmentación de objetos

3.1 Introducción

La segmentación de objetos habitualmente tiene como objetivo discriminar los objetos del primer plano (frente o foreground) de una imagen, que normalmente están moviéndose, del resto de los objetos o fondo (background).

Existen muchas técnicas distintas de segmentación de secuencias de vídeo dependiendo del tipo de secuencias y de los objetivos que se pretenden localizar. Un buen algoritmo capaz de detectar objetos en movimiento debe lograr de manera computacionalmente eficiente, y preferentemente en tiempo real, solventar los siguientes problemas:

- Cambios de iluminación de la escena a analizar: Son variaciones de iluminación que se pueden producir a lo largo del día si la escena ha sido capturada al aire libre, o tratándose de una secuencia de interior, por las distintas fuentes de iluminación que pueden estar apagadas o encendidas. Estos cambios pueden ser malinterpretados como parte de los objetos en movimiento o hacer que el fondo quede obsoleto si éste no se actualiza correctamente.
- Sombras y reflejos presentes en la escena: La interacción entre las fuentes luminosas y los diferentes objetos de la escena pueden producir efectos que suelen ser categorizados como objetos en movimiento (sombras y reflejos). Las sombras que acompañan a los objetos en movimiento no son parte del primer plano pero tampoco forman parte del fondo: deben ser eliminadas.
- Obtener y actualizar el fondo de la escena: A menudo, es difícil extraer una imagen de fondo adecuada durante la fase de inicialización de dicho fondo. Este hecho puede ser debido a que, en ese momento, existan objetos en movimiento en la escena que no puedan ser discriminados como fondo.
- Determinar los parámetros de funcionamiento de los algoritmos: Una de las tareas más importantes de este tipo de sistemas automáticos de

análisis es el ajuste de los parámetros de los algoritmos que determinan el fondo y los objetos en las secuencias. En cada algoritmo, podemos encontrar diferentes parámetros (umbrales, pesos,..) con distintas sensibilidades. Los parámetros más significativos serán los que alteren en mayor medida los resultados del sistema y por ello, tendremos que estudiarlos a fondo en función de las características de las secuencias.

- Ruido introducido en la secuencia de imágenes: El ruido que proviene de la captación de las imágenes por las cámaras de vídeo puede producir errores en la segmentación de objetos de vídeo y es necesario eliminarlo.
- Fondos multimodales: Son los que podemos encontrar en secuencias con objetos en movimiento lento y/o periódico (movimiento de las hojas de los árboles, movimiento ondulatorio del agua,...) y que, desde una perspectiva semántica, habitualmente se considera que pertenecen al fondo de la escena.
- Camuflaje: Este efecto aparece cuando los objetos del primer plano poseen el mismo color y textura que el fondo; por este motivo, el frente se confunde o camufla como fondo.

A fin de solucionar estos inconvenientes y extraer las regiones de interés con máxima fiabilidad, existen diversas aproximaciones. La mayoría de ellas, siguen un diagrama de flujo definido por [6], que identifica cuatro etapas:

- Preprocesado: En esta etapa se realizan simples tareas de procesamiento de imágenes que cambian la señal de entrada de vídeo en un formato que puede ser procesado por las etapas siguientes.
- Modelado de fondo: En esta etapa se realizan tanto la inicialización del fondo como su mantenimiento.
- Detección de frente: En esta etapa se realiza la substracción de frente; que permite obtener los objetos de interés en la secuencia de vídeo.
- Validación de datos o post-procesado: En esta etapa se realizan diversas

operaciones para mejorar los resultados obtenidos de las etapas anteriores.

En nuestro escenario donde las imágenes son captadas en el interior del nido, se puede deducir que la imagen tiene un fondo unimodal, que se ve afectado por ruido y cambios de luminosidad. Además el suelo del nido cambia lentamente por el trasiego de las pájaros.

Figura 3.1. Ejemplo de imagen, donde el cernícalo es el objeto a detectar (frente).

3.2 Modelos de representación de fondo

3.2.1 Introducción

En secuencias para las que ya se dispone de un fondo conocido resulta muy sencillo representar el fondo. En tales circunstancias, y si la cámara es fija, el diseño de los algoritmos de modelado de fondo se simplifica; los píxeles de fondo son valores constantes y, por tanto, el fondo se puede modelar con una imagen estática. Para representar esta imagen de fondo, en el caso de imágenes en escala de grises se puede utilizar un valor de intensidad para cada píxel. En el caso de una imagen a color, cada píxel sería representado con un canal R, G y B u otra escala (HSV,...).

No obstante, en la mayoría de las aplicaciones no se dispone del fondo o el

fondo no es constante. Incluso, en el ejemplo anterior, factores externos como el ruido de los sistemas de captación o internos como las variaciones de iluminación debidas a la interacción entre objetos móviles y fuentes de luz, hacen que en la práctica el fondo no se modele como constante. Por este motivo, algunos autores elaboran modelos matemáticos que permiten representar el fondo y adaptarlo a los cambios existentes.

Existen métodos básicos de sustracción de fondo que utilizan modelos matemáticos sencillos tales como, diferencias entre imágenes, promedios, máximos y mínimos,..etc., que permiten modelar de forma simple los píxeles de la imagen de fondo.

3.2.2 Modelos paramétricos

Los algoritmos basados en modelos paramétricos definen modelos de fondo más complejos, que permiten cierta tolerancia al ruido y a pequeñas fluctuaciones (hojas en movimiento, parpadeo de luces, pequeñas vibraciones de las cámaras, cambios bruscos de luz,..). Describen la imagen de fondo en base a parámetros de una distribución de probabilidad estándar (usualmente Gaussiana).

En función del tipo de fondo que se intenta modelar, se puede clasificar los métodos que usan un fondo unimodal, por ejemplo modelándolo como una distribución simple Gaussiana, y métodos que utilizan un fondo multimodal que lo modelan como un mezcla de distribuciones Gaussianas. Algunos de estos ejemplos son los que se muestran a continuación:

a) Gaussiana simple (*Simple Gaussian*)

El método de Gaussiana Simple (SG) *representa cada píxel* modela los pequeños cambios que ocurren en la imagen de fondo BG_t representando cada píxel x_t con una distribución unimodal Gaussiana definida por dos parámetros: media μ_t y varianza σ_t^2 .

$$\mu_t = \sum_{i=1}^{i=T} \frac{x_i}{t}$$

$$\sigma_t^2 = \sum_{i=1}^{i=T} \frac{x_i^2}{t} - \mu_t^2 \quad , \forall\, x_t \in BG_t$$

En cada instante *t*, se determina si un píxel pertenece al fondo BG_t si el valor de dicho píxel en la imagen está dentro de la Gaussiana definida para ese píxel, es decir, si la diferencia entre el valor del píxel y el de la media considera para dicho píxel μ_t es inferior a la desviación típica σ_t.

b) Mezcla de Gaussianas (*Mixture of Gaussian*)

En fondos multimodales, que contienen objetos no estáticos, hay píxeles cuyos valores de intensidad varían entorno a un conjunto finito de valores característicos. Por este motivo, un píxel no puede modelarse mediante una media y varianza, utilizando una distribución Gaussiana. La mezcla de Gaussianas propone una solución a este problema modelando la intensidad de los píxeles con una mezcla de *k* distribuciones Gaussianas definidas por los siguientes parámetros: media $\mu_{k,t}$, varianza $\sigma^2_{k,t}$ y peso $\omega_{k,t}$. En la práctica, *k* suele tomar un valor comprendido entre 3 y 5.

La probabilidad de observar el valor de un determinado pixel, *x*, en el instante de tiempo *t* mediante una mezcla de Gaussianas es:

$$P(x_t)=\sum_{i=i}^{k}\omega_{(i,t)}\eta(x_t-\mu_{(i,t)},\Sigma_{(i,t)})$$

donde Σ_i es la matriz de covarianzas y $\omega_{(i,t)}$ el peso asignado a cada distribución. Cada una de las *k* gaussianas describe sólo uno de los objetos observables de background o foreground.

Las gaussianas son multivariables para describir los valores en los canales RGB. Si se asume que estos valores son independientes, se simplifica la matriz de covarianza a una matriz diagonal. Además, si se asume lo mismo para la desviación típica de los tres canales, se simplifica todavía más $\sigma^2_{i,t}$.

Para convertir función de probabilidad de cada píxel en un único modelo de background, es necesario un criterio que distinga entre las distribuciones de foreground y background. Para ello, todas las distribuciones se clasifican en base a la relación entre la amplitud de pico ω_i y la desviación típica σ_i . Se

asume que cuanto mayor y más compacta sea la distribución, mayor probabilidad tiene de pertenecer al background. De esta manera, las primeras B distribuciones que cumplan lo anterior y verifiquen la desigualdad siguiente, se aceptan como background.

$$\sum_{i=1}^{B} \omega_i > T \text{ , donde } T \text{ es un umbral definido.}$$

Así, en el modelo de mezcla de Gaussianas, la combinación de k distribuciones Gaussianas cuya probabilidad de ocurrencia (suma de pesos $w_{k,t}(x,y)$) supere un determinado umbral, denominado umbral de frente, permitirá modelar cada píxel del fondo en cada instante.

En cada nuevo instante de tiempo t, hay que resolver dos problemas: asignar un nuevo valor, x_t , a la distribución que mejor se ajusta y estimar los parámetros para actualizar el modelo. De todas las distribuciones que satisfacen la desigualdad:

$$|x_t - \mu_i| > 2.5\,\sigma_{(i,t)}$$

La primera de ellas es la que mejor se ajusta a x_t . Por otra parte, los parámetros ($\mu_{i,t}$, $\sigma_{i,t}$, $\omega_{i,t}$) sólo se actualizan en la distribución de mejor ajuste y mediante el uso de promedios acumulativos simples. Si no se encuentra ninguna coincidencia, la última distribución se sustituye por una nueva centrada en x_t con peso ω_i bajo y varianza alta σ_i.

Por lo tanto, los píxeles de una imagen nueva se comparan con las k posibles subdistribuciones de fondo. Si se encuentra parecido, es decir, el píxel no difiere de alguna de las medias en más de la varianza asociada, se caracteriza el píxel como fondo y se actualizan los parámetros del modelo para ese píxel: la media $\mu_{k,t}$, la varianza $\sigma^2_{k,t}$ y un peso $\omega_{k,t}$) cuyo valor determina la cantidad de distribución que modela dicho píxel en cada instante.

La Mezcla de Gaussianas contiene multitud de variantes y existen otras medidas para clasificar los píxeles de fondo y realizar el seguimiento de los objetos en movimiento.

La Mezcla de Gaussianas es muy poco robusta a cambios repentinos de

iluminación. Por otro lado, los fondos multimodales requieren un número de distribuciones *k* elevado para modelar cada píxel, lo cual implica un incremento en la carga computacional. En este método de representación del fondo es muy importante la forma de actualizar las medias y las varianzas para adaptarse a los cambios del fondo. A pesar de los inconvenientes, es capaz de manejar una distribución multimodal de fondo ya que mantiene una función de densidad de probabilidad para cada píxel.

OpenCV implementa varias funciones de substracción de fondo modelado como mezclas de Gaussianas, como el algoritmo de segmentación **BackgroundSubtractorMOG** que es descrito en [7] y se basa en el trabajo realizado por [8][9][10]. Y el algoritmo **BackgroundSubtractorMOG2** que está basado en [11] y [12], y tiene como mejora importante que selecciona el número apropiado de distribuciones gaussianas para cada píxel, que en el algoritmo anterior se tomaba un número fijo de K distribuciones. Proporciona una mejor adaptabilidad a escenas cambiantes por ejemplo debido a cambios de iluminación, y además proporciona la opción de detectar las sombras de los objetos.

A continuación se describen los parámetros usados de cada función en OpenCV_2.4.

Atributos de BackgroundSubtractorMOG:

- **history:** Longitud del historial de valores usados para realizar el modelo.
- **Nmixtures:** Número de mezclas de gaussianas.
- **BackgroundRatio:** Umbral de Background.
- **NoiseSigma:** Desviación estandar de la luminosidad para cada canal de color. El valor “0” significa que toma el valor de forma automática.

Atributos de BackgroundSubtractorMOG2:

- **nmixtures:** número máximo permitido de mezclas de gaussianas. El actual número es determinado dinámicamente por píxel.
- **BackgroundRatio:** Umbral que define si el componente es lo suficientemente significativo como para ser incluido en el modelo de fondo.
- **VarThresholdGen:** Umbral para la distancia cuadrada de Mahalanobis que ayuda a decidir cuándo una muestra está cerca de los componentes existentes (corresponde a Tg). Si no está cerca de ningún componente, se genera un nuevo componente. Un valor de Tg más pequeño genera más componentes.
- **fvarInit:** Variación inicial para los componentes recién generados. Afecta a la velocidad de adaptación. El

valor del parámetro se basa en su estimación de la desviación típica estándar de las imágenes. OpenCV utiliza 15 como un valor razonable.

- **fvarMin:** Parámetro usado además para controlar la varianza.
- **fvarMax:** Parámetro usado además para controlar la varianza.
- **fCT:** Parámetro de reducción de complejidad. Este parámetro define el número de muestras necesarias para aceptar y justificar que el componente existe.
- **NshadowDetection:** Valor para marcado de píxeles de sombra en la salida de la máscara del primer plano.
- **fTau:** Umbral de sombra. La sombra es detectada si el pixel es una versión más oscura del fondo. Tau es un umbral que define como de oscura puede ser la sombra.

3.2.3 Modelos no paramétricos

Los modelos no paramétricos son métodos complejos en los que no se asumen distribuciones estándar de probabilidad para modelar a los píxeles de fondo, sino técnicas más generales, como pueden ser almacenamiento de los últimos valores del píxel, cálculo de rangos de valores del píxel, ajustes de funciones de predicción, etc. Algunos modelos no paramétricos son los siguientes:

a) Estimación por densidad de núcleo (Kernel Density Estimation)

El método de representación del fondo con Densidad de Núcleo (‘KDE’) descrito por [13], estima la función de densidad de probabilidad de cada píxel de la imagen de fondo en cada instante de tiempo. Esta operación se realiza gracias a la información de la historia reciente de dicho píxel que se haya almacenada en un buffer. El objetivo es obtener mayor sensibilidad de detección que utilizando un método de representación de fondo con una distribución de probabilidad fija.

$$B_t(x,y) = buffer\left\{I_{(t-L)}(x,y), I_{(t-L+1)}(x,y), I_t(x,y)\right\}$$

La pertenencia al fondo B_t se estima mediante el promedio ($Pr(x_t)$) de funciones de núcleo K (por ejemplo, de tipo Gaussiano) centradas en valores anteriores x_i y evaluados en el valor del píxel actual x_t; es decir, se calcula la probabilidad de parecido entre el píxel actual y los valores que dicho píxel ha tomado en las L imágenes anteriores y si esa probabilidad es suficientemente elevada (supera un determinado umbral U) se considera que el píxel es fondo; en caso contrario, será frente.

$$x_t \in B_t / Pr(x_t) = \sum_{i=t-1}^{i=t-1-L} K(x_i - x_t) > U$$

Por ejemplo, eligiendo un estimador de función de núcleo K de tipo Gaussiano, cada muestra de las L almacenadas se considera que posee una distribución $N(\mu_{k,t}, \sigma^2_{k,t})$.

$$Pr(x_t) = \left(\frac{1}{L} \sum_{i=t-1}^{i=t-1-L} \left(\frac{1}{\sqrt{2\pi\sigma^2}} \right) e^{\frac{-(x_i - x_t)^2}{2\sigma^2}} \right)$$

El modelo KDE soporta parpadeo de fondo, ruido en la imagen y es capaz de adaptarse a los cambios rápidos y progresivos del fondo. No obstante, posee un requisito muy importante que es la alta carga computacional.

OpenCV implementa varias funciones de substracción de fondo modelados como funciones de predicción probabilistica como la clasificación por K vecinos más cercanos (*K-nearest neigbours*) descrito en [12]. Muy eficiente si el número de píxeles que cambian del frente es bajo.

A continuación se describen los atributos usados de la función **BackgroundSubtractorKNN** que está disponible en la versión OpenCV3.0.

Atributos de la función BackgroundSubtractorKNN:

- **history:** Longitud del historial de valores usados para realizar el modelo.
- **Dist2Threshold:** Umbral en la distancia al cuadrado entre el píxel y la muestra para decidir si un píxel está cerca de esa muestra. Este parámetro no afecta a la actualización del background.
- **DetectShadows:** Si es verdadero, el algoritmo detectará sombras y las marcará. Esto decrementa la velocidad.
- **DefaultNsamples:** Número de muestras salvadas en memoria.

Otro algoritmo que implementa OpenCV, y que ha sido estudiado en este trabajo es **BackgroundSubtractorGMG** basado en el algoritmo dado en [14], combina estimación estadística del fondo de la imagen, segmentación bayesiana por píxel, y una solución aproximada al problema del seguimiento multiobjetivo usando un banco de filtros de kalman y el algoritmo de coincidencia de Gale-Shapley. Un modelo de confianza heurística permite el filtrado selectivo de seguimientos basado en datos dinámicos. Los autores

demostraron que su algoritmo ha mejorado la exhaustividad y precisión sobre los existentes métodos de OpenCV2.1 en una variedad de situaciones.

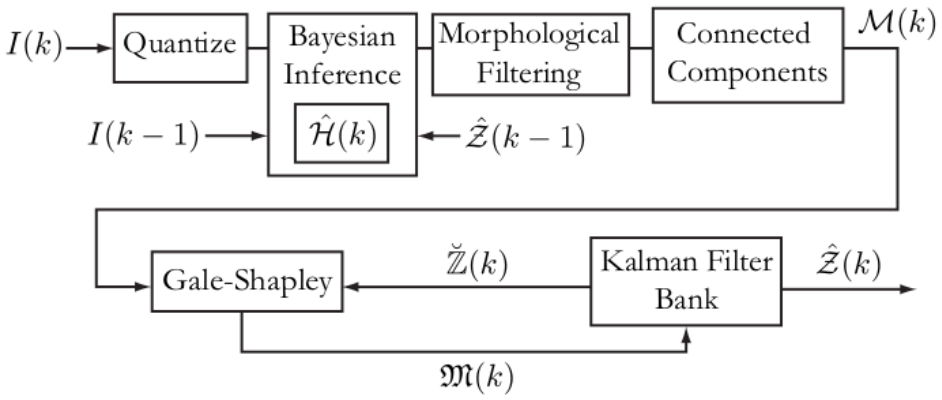


Figura 3.2. Diagrama de bloques del algoritmo GMG.

En la figura 3.2, se muestra el diagrama de bloques de funcionamiento del algoritmo, donde una imagen *I(k)* es cuantificada en el espacio de color RGB para cada píxel, y comparado con el modelo estadístico de la imagen por inferencia Bayesiana, *Ĥ(k)*, generando una nueva imagen de probabilidad. Esta imagen es filtrada con operaciones morfológicas y segmentada con cuadros limitadores (*bounding boxes*), mediante el algoritmo de componentes conectados obteniendo *M(k).* El banco de filtros de kalman mantiene un conjunto de seguimientos realizados Ẑ(k), y predice los cuadros limitadores de los objetos en movimiento para el tiempo k, $\breve{Z}(k)$. El algoritmo Gale-Shapley empareja elementos de M(k) con $\breve{Z}(k)$; Estos pares se utilizan para actualizar el banco de filtros de Kalman. El resultado es Ẑ(k), la colección de píxeles identificados como primer plano o frente. Esto junto con la imagen I(k), se utiliza para actualizar el modelo de imagen de fondo a Ĥ(k+1). Este paso actualiza selectivamente sólo los píxeles identificados como fondo.

Atributos de la función BackgroundSubtractorGMG:

- **maxFeatures:** Total de números de distinto color a mantener en el modelo de background.
- **LearningRate:** Rango entre 0.0 y 1.0, que determina como de rápido las características son olvidadas.
- **NumInitializationFrames**: Número de frames de vídeo usados para inicializar el modelo de background.
- **QuantizationLevels**: Número de niveles discretos en cada canal para ser usado en el el modelo de

background.

- **BackgroundPrior**: Probabilidad previa de que algún píxel dado sea background píxel.
- **DecisionThreshold**: Valor por encima del cual se determina que el píxel es frente.
- **smoothingRadius**: Radio de suavizado, en píxeles, para limpiar la imagen frente.
- **updateBackgroundModel**: Actualizar el modelo de fondo.

3.3 Comparativa de algoritmos

Se ha realizado una comparativa del funcionamiento de las cuatros funciones que implementa opencv para la substracción de background (MOG, MOG2, KNN y GMG), para el estudio del tiempo de ejecución y su funcionamiento en distintas condiciones de ruido e iluminación en las imágenes.

Se ha utilizado una secuencia de vídeo sin ningún objeto de primer plano (*foregroung*) para entrenar a los algoritmos, y se ha fijado un historial o número imágenes para calcular el modelo de background igual a 50 *frames* en la comparativa de la figura 3.3 y de 200 frames para la comparativa de la figura 3.4. En los algoritmos que permiten detectar sombras de los elementos foreground se ha optado por desactivar esta funcionalidad.

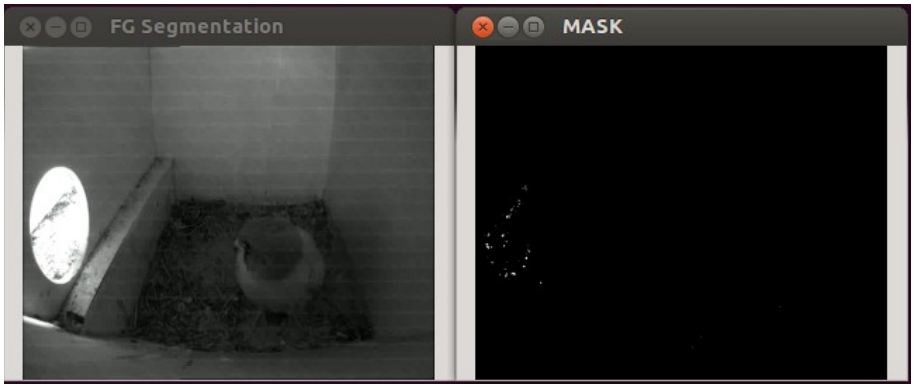
Probando con distintos vídeos se ha obtenido que el algoritmo más rápido es el MOG, seguido de KNN, MOG2 y GMG.

Tomando como referencia el **tiempo de ejecución** de MOG, se obtiene que:

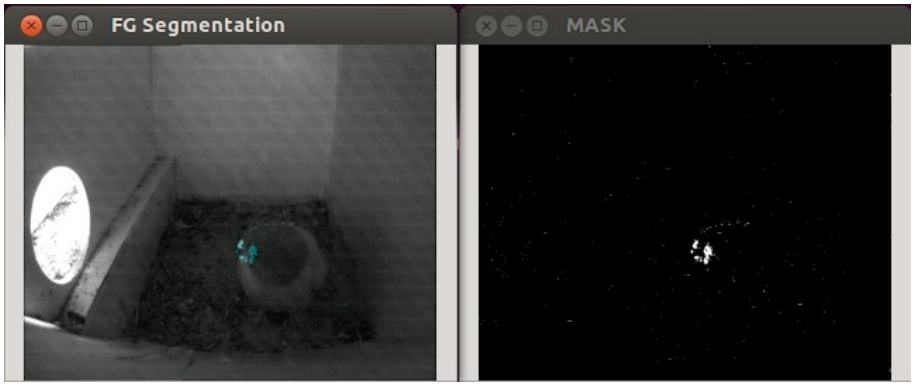
- KNN es 1,3 es veces más lento que MOG.
- MOG2 es 1,5 veces más lento que MOG.
- GMG es 5,1 veces más lento que MOG.

Observando las figuras 4.3 y 4.4, se puede ver que la función MOG es bastante sensible a los cambios de iluminación, dado que en las máscaras foreground aparecen píxeles en la entrada a la caja nido debido a cambios de iluminación; por otra parte cuando en las imágenes hay mucho ruido no es capaz de distinguir el objeto foreground. La función MOG2 es capaz de detectar parte de los píxeles del objeto foreground, aunque sigue siendo bastante sensible al ruido y necesitaría actualizar su modelo de background más rápido, para adaptarse al movimiento del pájaro. El algoritmo GMG en secuencias con

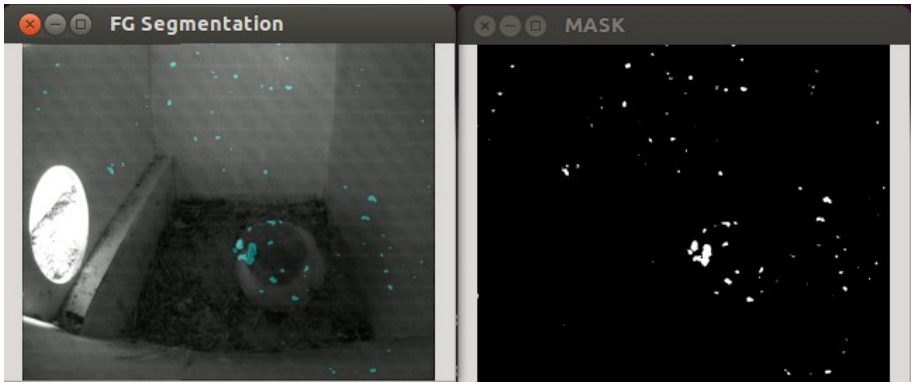
mucho ruido es capaz de detectar al objeto foreground, pero trabajaría mejor con un historial mayor de frames, para calcular el modelo óptimo de background. Y por último KNN es el que mejor funciona ante la secuencia de imágenes con bastante ruido, y si se utiliza un tamaño grande de frames para calcular el modelo de background necesita actualizarse más rápido, además le afecta más los cambios de iluminación que a GMG.



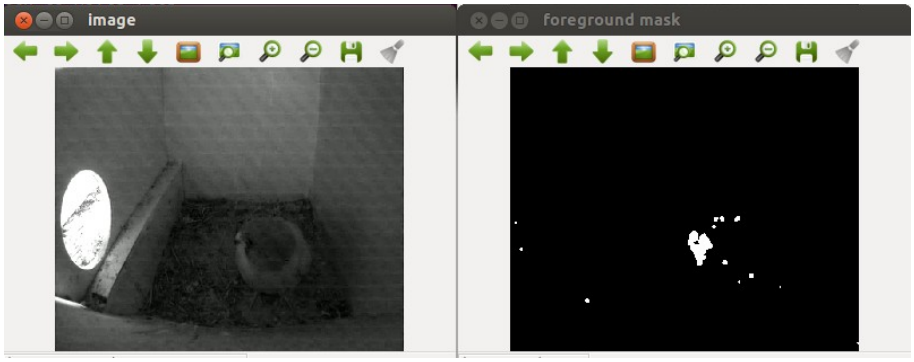
MOG



MOG2

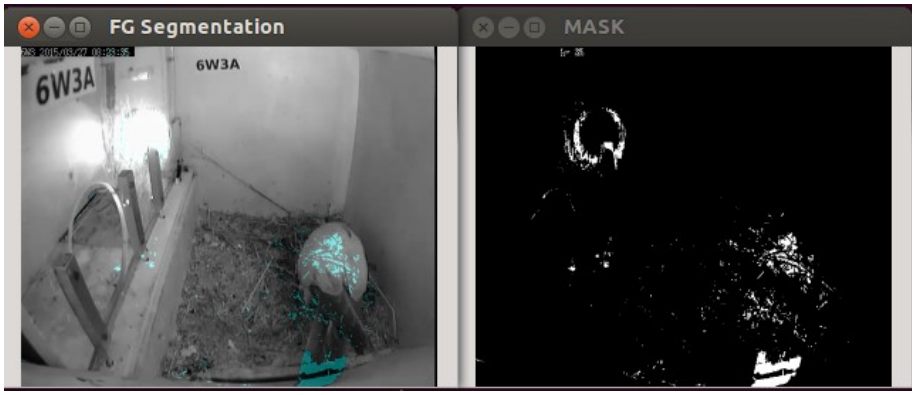


GMG

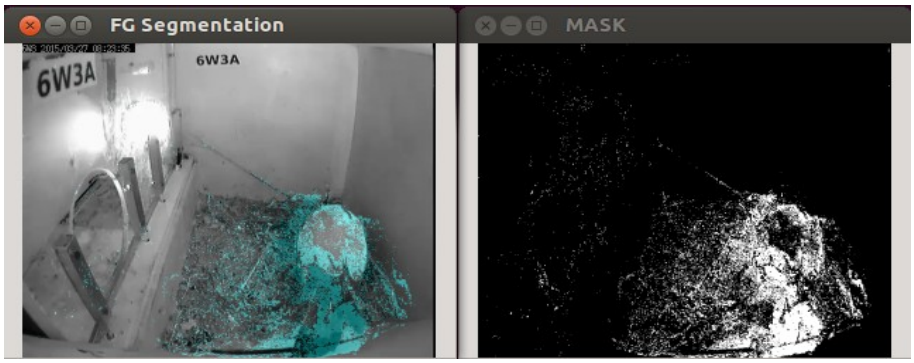


KNN

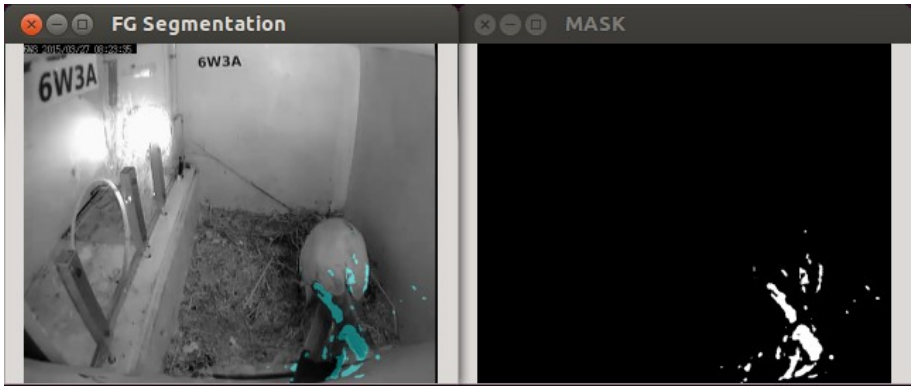
Figura 3.3. Comparativa vídeo con nivel de ruido alto



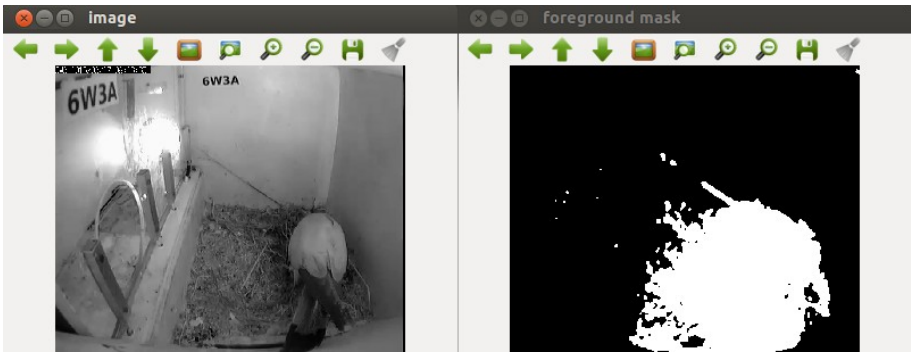
MOG



MOG2



GMG



KNN

Figura 3.4. Comparativa vídeo con nivel de ruido bajo

4. Detección de eventos

4.1 Detección de cambios de iluminación

Los cambios de iluminación producen que el servidor de vídeo active eventos y se produzcan grabaciones sin información útil. De forma general los cambios de iluminación se producen rápidamente, y alteran el nivel de intensidad de todos los píxeles de la imagen, aunque en la entrada de luz cambian más significativamente, ya que aumentan o disminuyen el número de píxeles con valor=255 (blancos).

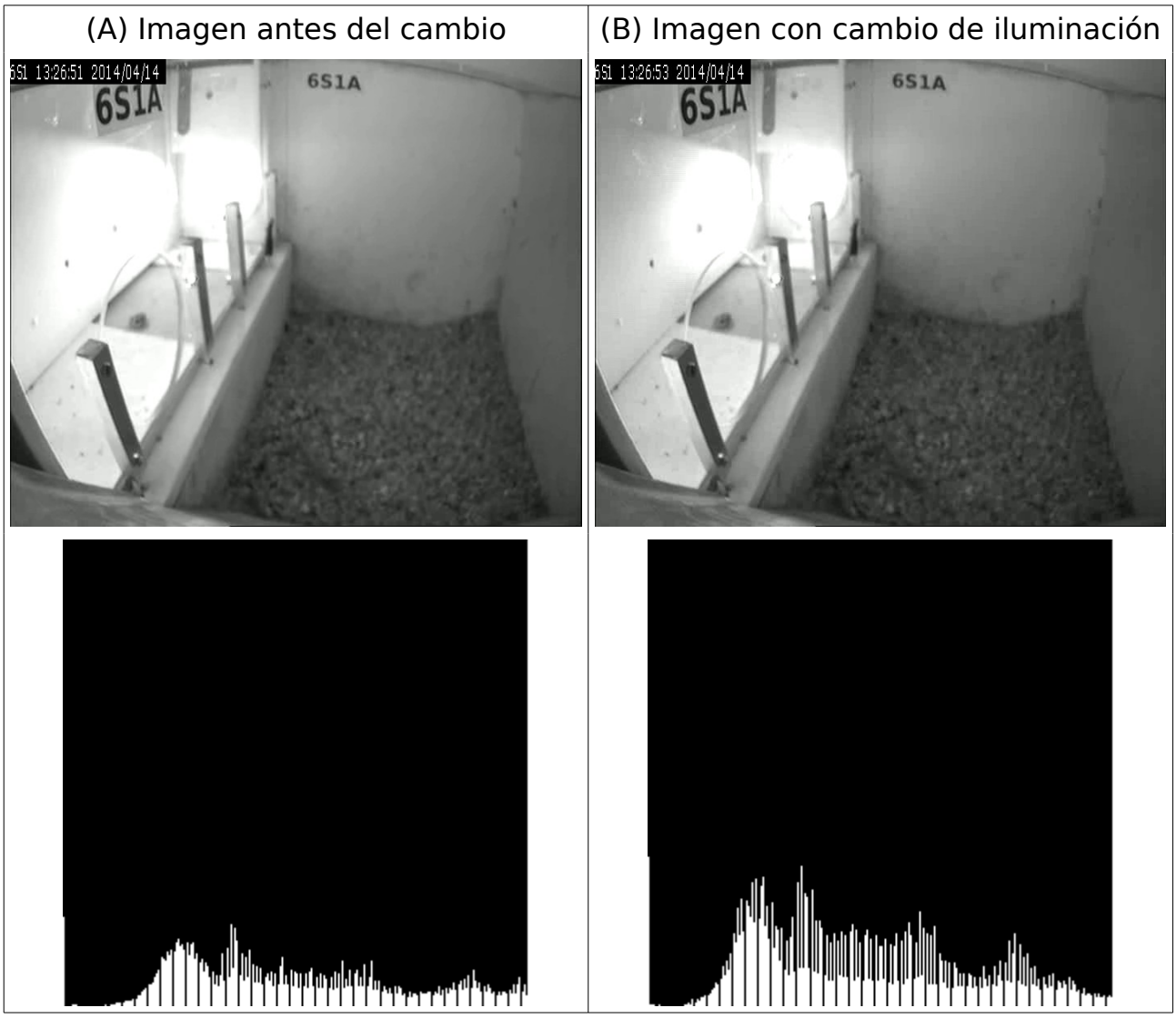


Figura 4.1. (A) imagen e histogramas de imágenes antes y durante cambio de iluminación

Simplemente observando como cambian los píxeles de color blanco en la imagen podemos conocer si la imagen está siendo afecta por cambios de iluminación o no. Por ejemplo en la figura 4.1, **el número de píxeles blancos es 78506 en la imagen (A) y 42140 en (B)**. Como los objetos a detectar no tienen píxeles blancos, se puede decir que los cambios bruscos del número de píxeles blancos es debido a cambios por la iluminación de la entrada. En cambio en el ejemplo de la figura 4.2, el número de píxeles blancos en la imagen (A) 1086 y 1077 en (B), dado el pájaro al entrar obstruye la entrada de luz exterior.

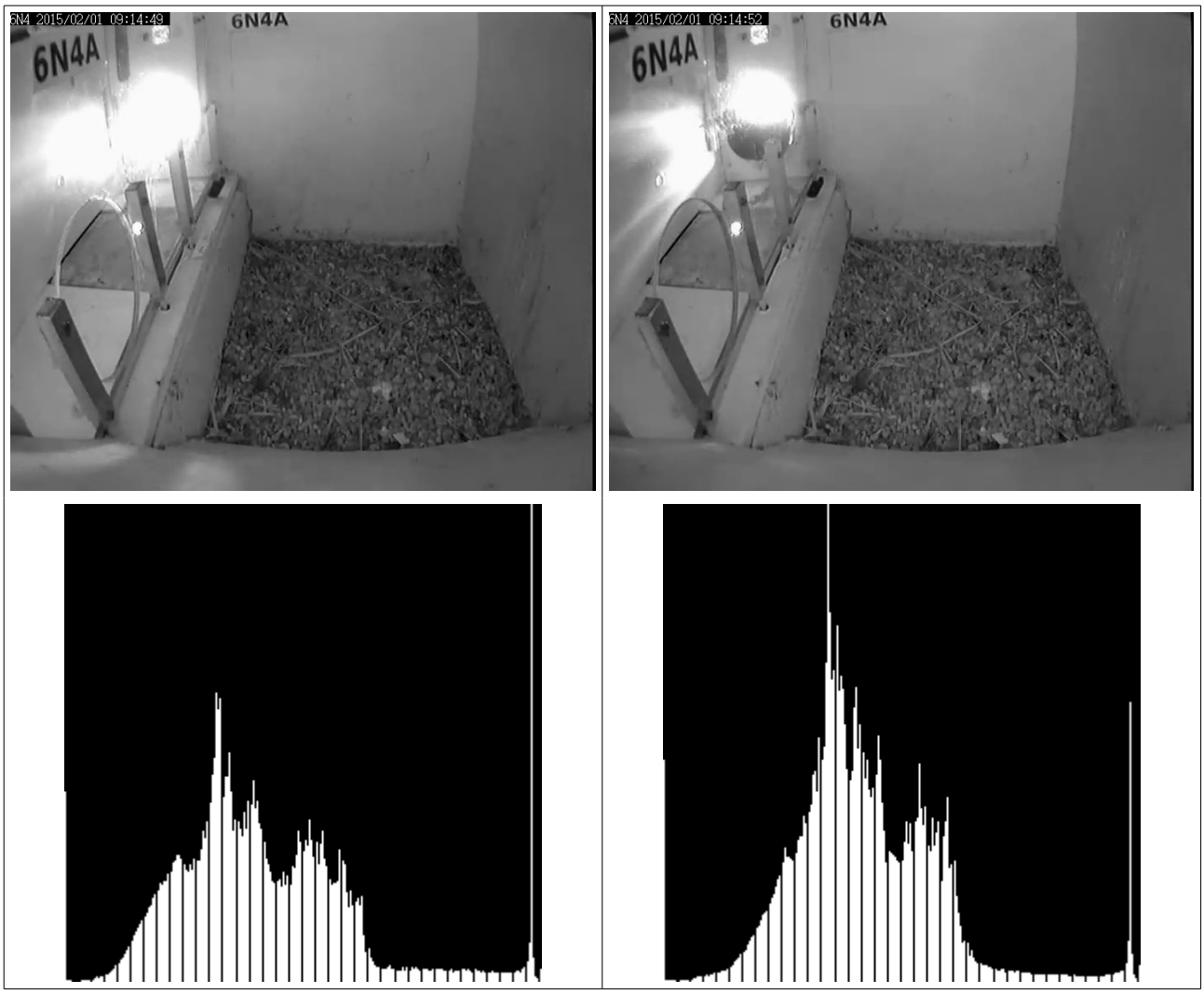


Figura 4.2. (A) imagen e histogramas de imágenes antes y durante cambio de iluminación

Para distinguir si el cambio de iluminación ha sido producido por el paso por la entrada de un ave, se compara la región de interes (ROI), mostrada en la figura

4.3, y si se obtiene un valor en tanto por ciento de píxeles blancos, si se obtiene un 100% de píxeles blancos podemos asegurar que el evento es provocado por un cambio de iluminación exterior.

Figura 4.3. Imagen con zona de ROI a comparar

4.2 Detección de cámara movida

Para la conseguir detectar movimientos de la cámara se han utilizado técnicas simples de extracción de blobs (*Binary Large Object*), que permiten caracterizar geométricamente objetos, definiendo sus parámetros de umbral de intensidad, forma y área, y consiguiendo como resultado su localización en la imagen. Un blob (*Binary Large Object*) no es más que un conjunto conectado y amorfo de píxeles de una imagen cuya intensidad es superior a un umbral.

Para realizar esta función se ha utilizado el objeto "SimpleBlobDetector" definido en las librerías de OpenCV. Se puede consultar más información en https://www.learnopencv.com/blob-detection-using-opencv-python-c/ .

A continuación se puede observar parte del código usado, y las características definidas para obtener el blob deseado.

```
SimpleBlobDetector::Params params;

// Niveles de umbrales de intensidad
params.minThreshold = 100;
params.maxThreshold = 200;


// Filtro por área
params.filterByArea = true;
params.minArea = 210;
params.maxArea = 250;


// Filtro por circularidad
params.filterByCircularity = true;
params.minCircularity = 0.1;


// Filtro por convexidad
params.filterByConvexity = true;
params.minConvexity = 0.87;


// Almacenamiento de blobs
vector<KeyPoint> keypoints;


// Asignación de parámetros
SimpleBlobDetector detector(params);


// Detección de blobs
detector.detect( im, keypoints);

// Almacenamos posición del blobs
Point2f p = keypoints.pt;
int x = keypoints[0].pt.x;
int y = keypoints[0].pt.y;

// Imprimimos por consola sus coordenadas
printf("x %d\n",x);
```

```
printf("y %d\n",y);
```

El blob elegido en la imagen, es el círculo formado en la parte inferior del número 6, que se encuentra en el fondo de la caja nido. Obteniendo las posiciones que se encuentran en la tabla 6.1, para las imágenes de la figuras siguientes.

Foto	X	Y
A	322	39
B	400	39

Tabla 4.1. Posiciones blobs

Figura 4.3. Imagen A, cámara centrada

Figura 4.4. Imagen B, cámara movida

5. Implementación práctica con BeagleBone Black

5.1 Introducción

"*Beaglebone Black*" es una tarjeta de desarrollo de bajo costo desarrollada por Beagleboard.org, enfocada a la utilización de software y hardware open source, que cuenta con una plataforma que utiliza el sistema operativo Linux. Cuenta con múltiples entradas y salidas de propósito general para realizar diversas funciones: entradas y salidas digitales, entradas analógicas, salidas PWM, soporte para I2C y SPI, etc. Así como un puerto ethernet y otro USB para la comunicación con otros dispositivos.

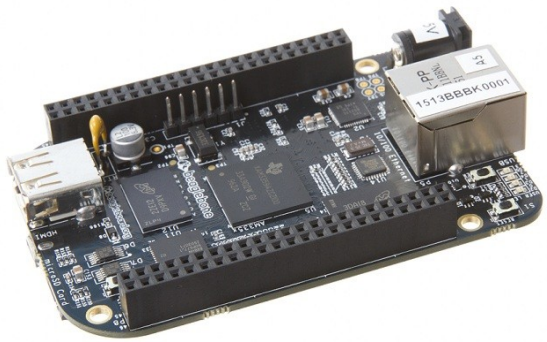


Figura 5.1 Placa Beaglebone Black (https://elinux.org/)

"*Beaglebone Black*" supone una evolución de su predecesora "*Beaglebone*" y está basada en procesadores ARM Cortex A8. Cuenta con numerosos accesorios y placas de circuitos modulares o shields, como por ejemplo para comunicación con otros dispositivos.

A continuación se indican su principales características (https://beagleboard.org/black):

- Procesador: AM335x 1GHz ARM® Cortex-A8
- Aceleradora gráfica 3D
- Acelerador de punto flotante NEON
- Microcontraladores 2x PRU 32-bit
- Memoria RAM: 512MB DDR3
- Memoria Flash: 4GB 8-bit eMMC on-board

- Conexiones HDMI, Ethernet y USB.
- Conectores: 2 de 46 pines
- Alimentación: 300-500 mA @ 5V

La placa "*Beaglebone Black*" incluye un chip eMMC integrado con una memoria de 2 GB con la distribución Debian preinstalada. Aunque también es posible "flashearla" y usar otros sistemas operativos como Angstrom, Ubuntu o Android, o arrancar directamente desde la tarjeta microSD

5.2 Conexión Beaglebone Black con servidor de vídeo IP

Uno de los principales usos de esta plataforma es el procesamiento de imágenes y visión artificial al ser compatible con librerías de visión artificial como OpenCV y cámaras USB. La instalación de OpenCV se realiza de forma similar que en ordenadores convencionales. En el caso de la utilización de una cámara analógica conectada a un servidor de vídeo como Vivotek VS8102, el acceso al stream de vídeo desde el programa de OpenCV se realiza a través de la dirección IP del servidor de vídeo correspondiente. Con el siguiente programa de ejemplo podemos obtener el streaming que emite cada servidor de vídeo y visualizarlo.

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

int main(int, char**) {
    cv::VideoCapture vcap;
    cv::Mat image;

    const std::string videoStreamAddress = "http://<user:password>@<ip_address>/video.cgi?.mjpg";

    //Abrir el stream de vídeo
    if(!vcap.open(videoStreamAddress)) {
        std::cout << "Error abriendo el stream de vídeo" << std::endl;
        return -1;
    }

    for(;;) {
        if(!vcap.read(image)) {
            std::cout << "No frame" << std::endl;
            cv::waitKey();
        }
        cv::imshow("Output Window", image);
        if(cv::waitKey(1) >= 0) break;
    }
}
```

Cada caja nido está prevista de un servidor de vídeo VS8102 que cuenta con una entrada y una salida digital, que puede conectarse con dispositivos externos, como puede ser la *Beaglebone Black* que cuenta con numerosas entrada y salidas digitales, como se muestra en la figura siguiente. Con la entrada digital del servidor de vídeo podemos activar grabaciones con la Beaglebone Black, y mejorar las grabaciones por detección de eventos del servidor, y consiguiendo vídeos completos de cada suceso, por ejemplo, una entrada y salida de un cernícalo, sin tener que agrupar varios vídeos evitando que existan trozos perdidos.

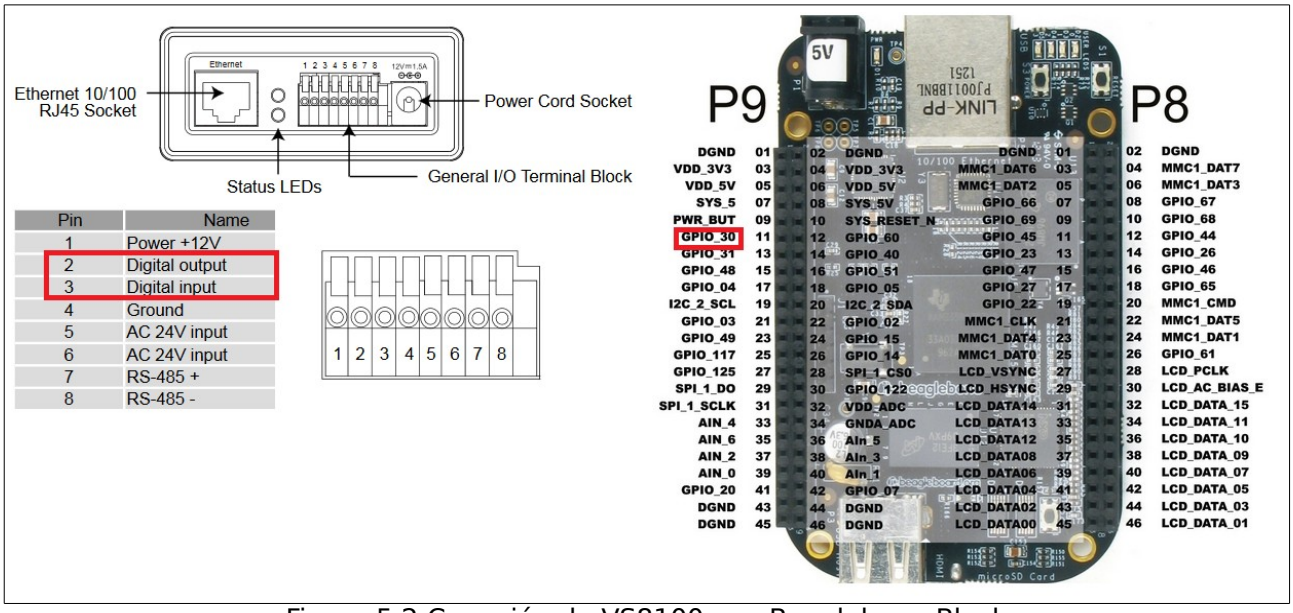


Figura 5.2 Conexión de VS8100 con Beaglebone Black

En la figura 5.2 se muestra la entrada y salida digital del servidor de vídeo VS810 (a la izquierda) y pinout de la Beaglebone Black donde se ha remarcado una de las entradas/salidas de propósito general (GPIO).

(Manual de referencia de VS8102 en http://download.vivotek.com/downloadfile/downloads/usersmanuals/vs8102manual_en.pdf y pinout Beaglebone Black de www.insigntech.com)

6. Conclusiones y mejoras

Tras el estudio del sistema de filmación de imágenes y vídeos del proyecto Horus, y de analizar la problemática ocasiona por la grabación de grandes volúmenes de contenido multimedia, cuyo análisis resulta muy costoso en tiempo para los investigadores. Se ha estudiado los distintos tipos de ruido que afectan a las imágenes, observando que principalmente se ven influidas por ruido periódico, y como se puede suavizar o atenuar. También se ha implementado un algoritmo para poder cuantificar el nivel de nitidez de las imágenes, a través de la transformada de Haar.

También se han testeado distintos métodos de segmentación del fondo, y se han comparado en distintas situaciones. Además se han implementado distintos métodos para determinar detección de cambios de iluminación, y cotejar cuando la cámara ha sido movida. Por último se ha utilizado una tarjeta de desarrollo de bajo coste “BeagleBone Black”, la cual puede ejecutar distintas versiones del sistema operativo linux, y funcionar con las liberias de OpenCV, permitiendo ejecutar los algoritmos implementados, a través del acceso al streaming de vídeo, en tiempo casi real.

Para trabajos futuros se pueden estudiar otros métodos de segmentación para uso en servidores con Unidades de Procesamiento Gráfico (GPU), con el uso de la librería CUDA de OpenCV. También recientemente se ha desarrollado un nuevo método de segmentación “BackgroundSubtractorCNT”, ya disponible en la versión beta de OpenCV3.3 que tiene como ventaja su gran velocidad.

Además se pueden implementar y probar con el sistema Horus, nuevos algoritmos para detectar eventos provocados por las aves, como entradas y salidas al nido, distinguir entre machos, hembras o otras especies, contar huevos, determinar cuando son los nacimientos de los pollos, …

7. Referencias

[1] Carlo Tomasi and Roberto Manduchi, “Bilateral filtering for gray and color images,” in Computer Vision, 1998. Sixth International Conference on . IEEE, 1998, pp. 839– 846.

[2] Antoni Buades, Bartomeu Coll, and Jean-Michel Morel, Non-Local Means Denoising, Image Processing On Line, 1(2011)].

[3] H. Tong, M. Li, H. Zhang, C. Zhang, “Blur Detection for Digital Images Using Wavelet Transform”, Proc. of ICME 2004 .

[4] X. Marichal, W. Ma, H. Zhang, “Blur Determination in the compressed domain using DCT information”, Proc. ICIP'99 .

[5] P. Marziliano, F. Dufaux, S. Winkler, T. Ebrahimi, “A no-reference perceptual blur metric”, Proc.of the ICIP 2002 .

[6] Cheung S-C, Kamath C. Robust techniques for background subtraction in urban traffic video. In S. Panchanathan and B. Vasudev, editors, Proc Elect Imaging: Visual Comm Image Proce 2004 (Part One) SPIE, 2004; 5308: 881-892.

[7] P KadewTraKuPong and R. Bowden. “An improved adaptive background mixture model for real-time tracking with shadow detection”, Proc. 2nd European Workshop on Advanced Video-Based Surveillance Systems, 2001: http://personal.ee.surrey.ac.uk/Personal/R.Bowden/publications/avbs01/avbs01.pdf

[8] Grimson Wel, Stauffer C. Romano R. Lee L. Using adaptive tracking to classify and monitor activities in a site. in Proceedings. 1998 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (Cat. No.98CB36231). IEEE Comput. Soc. 1998. 1998.

[9] Stauffer C, Grimson W. E. L. Adaptive background mixture models for real-time tracking. in Proceedings. 1999 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (Cat. No PR00149). IEEE Comput. Soc. Part Vol. 2, 1999.

[10] Stauffer C, Grimson W. E. L., Learning patterns of activity using real-time tracking. IEEE Transactions on Pattern Analysis & Machine Intelligence, 2000. 22(8): p. 747-57.

[11] Zivkovic. “Improved adaptive Gausian mixture model for background

subtraction”, International Conference Pattern Recognition, UK, August, 2004, http://www.zoranz.net/Publications/zivkovic2004ICPR.pdf. The code is very fast and performs also shadow detection. Number of Gausssian components is adapted per pixel.

[12] Z.Zivkovic, F. van der Heijden. “Efficient Adaptive Density Estimation per Image Pixel for the Task of Background Subtraction”, Pattern Recognition Letters, vol. 27, no. 7, pages 773-780, 2006.

[13] Elgammal A, Duraiswami R, Harwood D, Davis LS. Background and foreground modeling using nonparametric kernel density estimation for visual surveillance. Proc. IEEE Jul. 2002; 90(7): 1151-1163.

[14] Andrew B. Godbehere, Akihiro Matsukawa, Ken Goldberg, “Visual Tracking of Human Visitors under Variable-Lighting Conditions for a Responsive Audio Art Installation”, American Control Conference, Montreal, June 2012.

Otros artículos consultados:

[15] ESCALERA HUESO, Arturo de la. Visión por computador. Fundamentos y métodos. Madrid: Pearson Educación, 2001. 304 p. ISBN: 84-205-3098-0.

[16] LONG, Zhiling; YOUNANA, Nicolas H. Denoising of images with multiplicative noise curruption. 13th European Signal Processing Conference EUSIPCO 2005, September 4-8, 2005, Antalya, Turkey.

[17] Liyuan Li, Weimin Huang, Irene Y.H. Gu, and Qi Tian. *Foreground Object Detection from Videos Containing Complex Background*. ACM MM2003 9p, 2003.