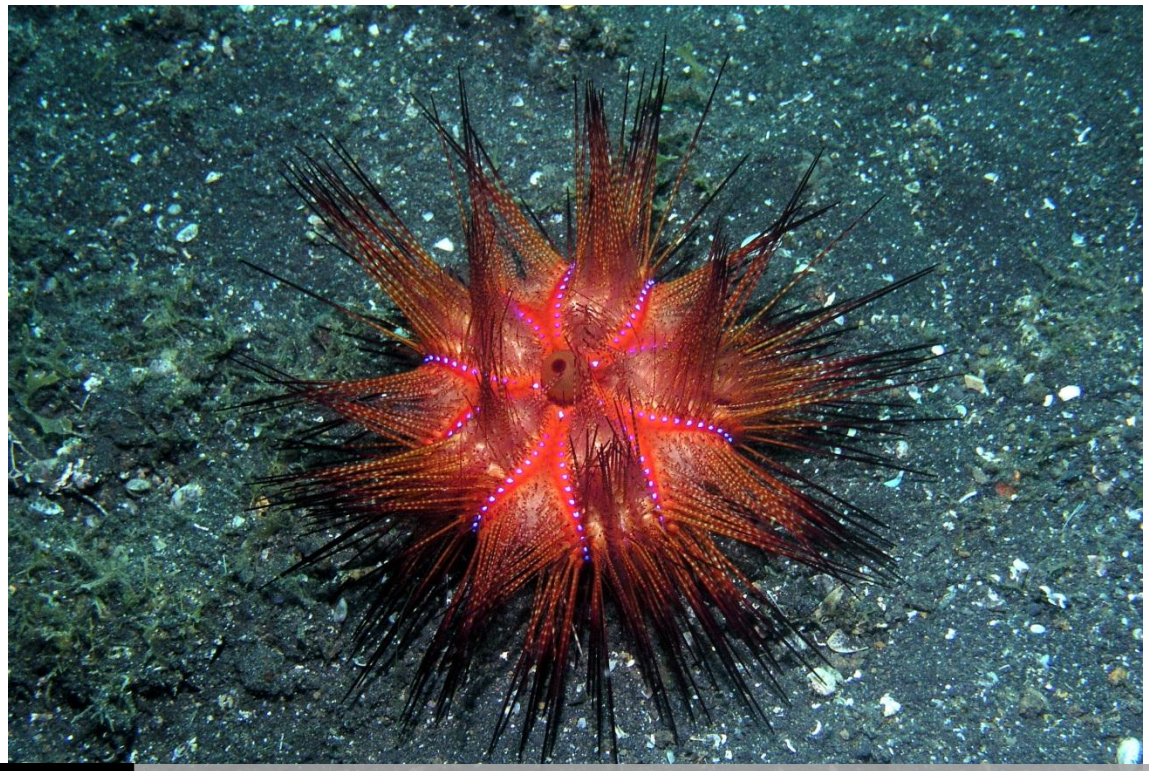

30 DE NOVIEMBRE DE 2011

TRABAJO FIN DE GRADO. IDENTIFICACIÓN VISUAL EN AMBIENTES SUBACUÁTICOS

ESCUELA DE INGENIERÍA INFORMÁTICA

UNIVERSIDAD DE LAS PALMAS DE GRAN CANARIA

Autor: Federico Maniscalco Martín
Tutores: Jorge Cabrera Gámez
Modesto F. Castrillón Santana
Profesores Titulares de la Universidad

Índice general

Agradecimientos

Quisiera dedicar este proyecto a mis padres, y a mi hermana, ya que, sin ellos no habría llegado a este punto.

También me gustaría nombrar a mis tíos Julio y Diana, que han estado apoyándome y dándome ánimos en todo momento.

Me gustaría agradecer también a mis tutores Jorge Cabrera Gámez y Modesto F. Castrillón Santana, que han sido muy pacientes y sumamente atentos, y por haber estado continuamente encima de mí para ayudarme en todo momento.

Por último, a la fuente de todas mis inspiraciones y la que ha estado siempre presente, alimentando mi interés por este proyecto y apoyándome continuamente, me gustaría más que agradecer a Margarita Martina Bilbao Martín.

Capítulo 1. Estado actual del tema

1.1. Introducción

Los erizos de mar[19] han servido como modelo prototípico de organismo en el desarrollo de la Biología. Este uso se origina a partir de la década de 1800, cuando su desarrollo embrionario fue fácilmente visualizado mediante microscopio. Los erizos de mar fueron la primera especie en la que se demostró que los espermatozoides fecundan el óvulo.

La principal característica de este grupo de *Equinodermos*[46] es la ausencia de brazos, por lo que tienen forma globosa, más o menos aplanada en su eje oral – aboral. El nombre del grupo *Echinoidea*[47], al que pertenecen unas 950 especies, significa "semejantes a los erizos", y esto hace referencia a las púas o espinas que tienen sobre todo su cuerpo.

A pesar de tener un cuerpo casi esférico, mantienen la simetría pentámera característica de los *Equinodermos*, aunque algunos grupos hayan evolucionado hacia una simetría bilateral secundaria (los erizos corazón) al vivir en fondos arenosos. Su esqueleto interno está formado por numerosos osículos que se han aplanado y fusionado entre sí para formar un caparazón compacto; estos osículos presentan, generalmente, numerosos tubérculos y perforaciones que ayudan a reducir el peso de estas placas y que son caracteres fundamentales para diferenciar las especies.

La locomoción en los *Equinoideos* se realiza gracias a la acción de las espinas y los *pies ambulacrales*[48]. Los pies actúan de manera similar a como lo hacen en las estrellas de mar, pero la principal función de las espinas es levantar la zona oral del sustrato.

Los pies de los erizos están bastante desarrollados y en su extremo existe una ventosa provista de músculos y osículos para reforzarla, ya que de ellos depende el desplazamiento. En los erizos irregulares, los *pies ambulacrales* se han especializado en otras funciones, por lo que su estructura es también diferente. A continuación se muestra un gráfico que muestra estas partes.

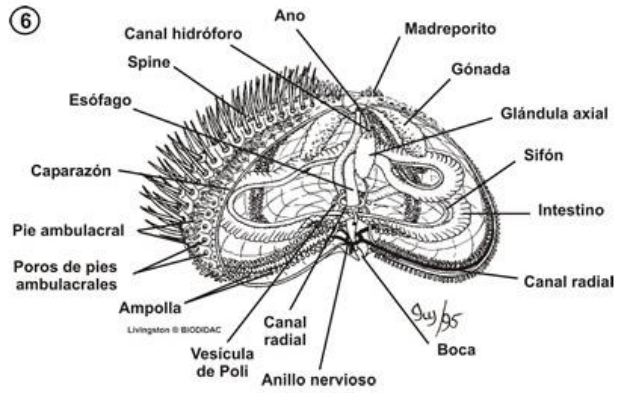

Fig. 1.1. Erizo de mar por dentro

Los erizos de mar se alimentan principalmente de algas que obtienen raspando con su aparato bucal la superficie de las rocas; de esta forma, ejercen una gran presión sobre las poblaciones de algas, ya que en su ausencia, la cobertura algal es mucho más densa. Los erizos irregulares que viven enterrados en la arena se alimentan de sedimento, cuyas partículas seleccionan mediante sus *pies ambulacrales*.

Los erizos son consumidos por una gran variedad de animales, entre los que destacan algunos peces, *gasterópodos*[49] y otros *equinodermos*, como las estrellas de mar. Muchos son consumidos por el hombre, y bien sean las huevas o el animal entero, su recogida ha supuesto en muchas ocasiones un descenso del número de poblaciones que ha puesto a estos animales al borde de la desaparición en muchas zonas, por lo que se han adoptado leyes que regulan su recogida.

Todas las especies de erizos de mar son *dioicas*[50], esto es, hay individuos masculinos y femeninos. Tienen sus *gónadas*[51] dispuestas en la cara interna del caparazón, directamente bajo las *placas interambulacrales*, existiendo por tanto cinco *gónadas*, excepto en los erizos acorazonados, cuya *gónada* del extremo posterior ha desaparecido. A través del *gonoporo* (poro genital de invertebrados, especialmente insectos), situado en las *placas genitales aborales* (zona opuesta de la boca), se expulsan los *gametos*[52], produciéndose la fecundación en el agua. Al igual que otros grupos de *equinodermos*, hay especies que incuban sus huevos, los erizos regulares entre las espinas que rodean el *periprocto* (alrededor del ano), y los irregulares en los *ambulacros petaloideos*, pero son las que menos.

Como norma general, los erizos de mar responden a un *fototactismo negativo*[53], por lo que buscan refugios entre grietas y agujeros, y muchos individuos se colocan encima conchas o algas para evitar la alta iluminación, especialmente en las zonas más altas del sublitoral.

No obstante, dentro de los erizos irregulares, los erizos acorazonados viven enterrados en el sedimento en una cámara que excavan y se comunican con el exterior por medio de los *pies ambulacrales* que usan para respirar y alimentarse. Asimismo, el dólar de arena vive en sedimentos de arena gruesa. Estudios recientes[54] sugieren que todo el cuerpo del erizo actúa como un enorme ojo compuesto, mediante receptores dispersos en su piel que actúan como las retinas. Así, al percibir la luz, son capaces de "ver" lo que tienen delante.

La elucidación de su secuencia genómica ha sido de gran interés para la comunidad científica para el estudio de todo, desde la economía y la ecología de los ecosistemas marinos a las preguntas fundamentales de la evolución y el desarrollo[1].

En 1983, el 95% de la población de erizo de Lima o erizo de púas largas (Diadema antillarum) del Caribe desapareció en tan solo unos meses a causa de una enfermedad desconocida. La epidemia comenzó en San Blas, en Panamá, y se extendió a Curaçao contra corriente, siguiendo la ruta de los petroleros. Esto sugiere que el agente causante podría proceder del Océano Pacífico, a través del Canal de Panamá. Desde Curaçao se propagó, llevado por las corrientes, al resto del Caribe. La tasa de mortandad media en Curaçao fue del 98% en menos de 10 días desde que se detectó la enfermedad en una zona determinada. Se redujo la población de erizos a apenas el 5% de su número original en todo el Caribe. Esta epidemia tuvo grandes consecuencias para los arrecifes de coral poco profundos del Caribe.

Los erizos regulan el número de algas que se adhieren al coral y limpian las superficies donde las larvas de coral pueden establecerse. Como resultado de este brote, la totalidad del arrecife coralino de poca profundidad desapareció por completo en determinadas áreas. Existen muchos otros factores que contribuyen a la degradación de los corales del Caribe, pero hay una cosa segura, la desaparición de los erizos de Lima fue la causa principal. Más de 20 años después del primer brote, los erizos de Lima se han reestablecido despacio, y su población ha aumentado en los últimos años. Con toda probabilidad, esta especie ha desarrollado defensas ante la enfermedad.

Al cambiar la distribución de algunos patógenos, el cambio climático posiblemente facilite la propagación de enfermedades como la que arrasó la población de erizos de Lima; mientras que los ecosistemas que ya están afectados por el blanqueamiento del coral son incluso más vulnerables y presentarán poca resistencia a estas agresiones en el futuro.

A diferencia de en el Caribe, la irrupción de este animal como especie invasora en los fondos canarios, combinada con el éxito reproductivo que ha tenido en nuestras aguas, ha creado un problema medioambiental importante que se ha intentado atajar con la puesta en marcha de proyectos e iniciativas orientados a su erradicación (matanzas masivas) o su contención con intentos de estimular su explotación comercial para uso gastronómico.

Algunos de los estudios realizados sobre los erizos de mar son:

- Estudios para poder seguir a los erizos y poder descubrir sus patrones de comportamiento[2][3]. Marcan estos erizos (*tagging*[20]) fijándoles una etiqueta al caparazón. Para ello, suelen tener que taladrar de alguna manera el caparazón, lo que implica que el erizo debe tener un tamaño mínimo. Esta técnica tiene dos inconvenientes:

  - Es relativamente fácil que la marca quede enganchada y se desprenda del animal.

  - Muchos de los animales marcados suelen morir porque el caparazón no cicatriza.

- En el campo de la ecología marina:

  - Relaciones de esta especie con otras e influencias sobre su densidad de población[4][5]. Las observaciones se realizan mediante varias inmersiones de un grupo variable de buceadores, equipados con escafandra autónoma, linternas e instrumental de medición, anotando luego la densidad de erizas.

  - Densidad de individuos: Tasas de recuperación en el Caribe y densidad de ciertas especies de algas[6].

  - Análisis de los patrones de movimiento durante el día y por la noche. En este tipo de estudios, el marcado de los animales es siempre necesario[7][8][9].

- Estudios sobre la capacidad de retorno (*homing*[21]) o de "sentirse expuestos" y moverse en grupo de estos animales a pesar de la extrema simplicidad de su sistema sensorial mediante la grabación continua de vídeos y su posterior análisis en el laboratorio[10].

En este Trabajo de Fin de Grado se aborda el problema de la detección automática de erizos de Lima en imágenes y vídeos subacuáticos con la intención, a medio plazo, de poder facilitar los trabajos de campo que requieren este tipo de estudios.

1.2. Finalidad del proyecto

Existen numerosas aplicaciones en el ámbito subacuático donde la clasificación visual de texturas y formas puede tener una gran importancia. Así, tareas de identificación y recuento de ejemplares, de clasificación de suelos o sustratos o de evaluación de hábitats requieren de esta capacidad. En la inmensa mayoría de las ocasiones, estas tareas se realizan en la actualidad mediante técnicas de conteo manual, bien in-situ con buceadores o bien sobre imágenes de vídeos, con elevado coste en términos de tiempo, capacidad de muestreo y mano de obra.

En el transcurso de este trabajo se pretende explorar la robustez con la que se pueden clasificar visualmente diferentes tipos de erizos a partir tanto de imágenes estáticas como de secuencias de vídeo; se abordaría la detección y clasificación de dos grandes grupos de erizos marinos. Concretamente, se trataría de localizar erizos y clasificarlos en dos grupos:

1. *Diadema antillarum*, vulgarmente llamados erizas, una especie invasora procedente del Caribe y que está ocasionando un problema medioambiental grave en los fondos costeros de Canarias.

2. Erizos autóctonos.

En el ámbito de este trabajo se pretende explorar este problema para evaluar si, mediante el empleo de técnicas de visión por computador, es posible resolver estas tareas mediante la inspección automática de vídeos e imágenes.

1.3. Etapas del proyecto

Para realizar este proceso es necesario pasar por las siguientes etapas que son las que se irán abordando en este proyecto:

- **Análisis global. Revisión bibliográfica y análisis previo de Teoría de clasificadores.** Conlleva un estudio preliminar de los conceptos para tratar con clasificadores, del conjunto de librería utilizadas (como *OpenCV*[17][40][41][42]), la instalación del software para desarrollar clasificadores y utilización de ellos.

- **Análisis y diseño de modelo en MATLAB**[39]**.** Estudio profundo del lenguaje para la comprensión del modelo de programación en *MATLAB*.

- **Implementación en MATLAB de aplicaciones auxiliares.** Análisis de las aplicaciones auxiliares necesarias para la correcta utilización del conjunto de imágenes, jugando un papel de simbiosis con la fase en la que se utilizan las herramientas de la librería *OpenCV*. Se recurre de nuevo a esta etapa en la utilización de dichas herramientas.

- **Análisis y diseño de estructuras de datos y modelo de la librería OpenCV.** Estudio profundo de la librería *OpenCV* para la comprensión del modelo de programación básico para utilizarla correctamente.

- **Implementación en C**[35]**/C++**[36] **del uso de los clasificadores y pruebas.** Implementación de alguna manera del uso de los clasificadores generados en el paso anterior junto con varias pruebas para comprobar su bondad.

- **Documentación y defensa. Confección de la documentación del proyecto y preparación de la defensa.** Elaboración de la memoria y el manual de usuario. Explicaciones, apéndices y bibliografía.

1.4. Estructura de los capítulos

Acorde con la línea general que se pretende abordar en este trabajo, se expone a continuación la estructura del contenido de este documento:

- En el capítulo de *objetivos*, se hace una descripción general de los retos que se pretende alcanzar en cada una de las etapas en las que está dividido el proyecto.

- En el capítulo de *competencias* se comenta cómo se han cubierto dichas competencias con este trabajo.

- En el capítulo de *recursos* se comenta los principales recursos hardware y software necesarios.

- En el capítulo de *plan de trabajo y temporización* se hablará sobre la estimación inicial prevista y el tiempo que realmente se ha invertido.

A continuación, se abordará con más detalle cada una de las etapas en las que se ha dividido el trabajo. Esta descripción se ha dividido en cuatro capítulos, siguiendo un desglose por etapas, y supone el eje central de esta memoria.

En la **primera etapa**, se conocerán los conceptos generales de la *teoría sobre clasificadores*.

En la **segunda etapa,** se abordará una breve explicación del software necesario para la elaboración del proyecto (la instalación y puesta en marcha del software, lo cual abarca la instalación de la librería *OpenCV* se puede ver en el anexo **Instalación de OpenCV**).

Finalmente, se necesita compilar las aplicaciones escritas en C/C++ y para ello se ha utilizado el compilador *g++*, utilizando después una terminal de *Ubuntu* para ejecutarlas.

En la **tercera etapa,** se presentan las soluciones tomadas para obtener los clasificadores y usarlos, exponiéndose también el código de las aplicaciones que utilizan dichos clasificadores.

En la **cuarta etapa,** se muestran los resultados obtenidos y se exponen algunos experimentos interesantes que se han realizado.

Por último, se exponen las conclusiones obtenidas del proyecto. Se plantean también los posibles trabajos futuros que por falta de tiempo o material tuvieron que dejarse pendientes.

Se han añadido varios anexos a la memoria:

- **Manual de usuario:** Se explica cómo utilizar el software desarrollado. Nuestro manual de usuario revela las posibilidades que ofrecen las aplicaciones que utilizan los clasificadores y explica cómo utilizarlas.

- **Instalación de OpenCV**: Detalla los pasos a seguir para instalar la librería *OpenCV* y poder utilizar sus herramientas posteriormente.

- **Creación del conjunto de datos de entrenamiento**: Se muestran las herramientas utilizadas para el conjunto de datos usados para el entrenamiento de los clasificadores.

- **Código**: Se muestra el código de todas las funciones empleadas en este Trabajo de Fin de Grado.

Capítulo 2. Objetivos

2.1. Objetivos académicos

La meta a conseguir en cualquier educación superior se rige por la validez de sus estudiantes a la hora de incorporarse al mercado laboral, cobrando especial importancia la asimilación del rápido avance de la tecnología y su constante período de reciclaje, provocando la necesidad de que el alumno se familiarice con métodos, técnicas y tecnologías actuales.

Carácter integrador de conocimientos de la titulación:

- Algoritmos y Programación
- Sistemas Operativos
- Ingeniería del Software
- Interfaces Humanas

2.2. Objetivos generales del proyecto

- Estudiar las herramientas necesarias para generar clasificadores.
- Analizar la estructura de datos correcta para el uso de los clasificadores.
- Diseñar la aplicación que use los clasificadores generados y aporte la información necesaria.
- Estudiar dicha información y comparar los resultados de cada clasificador.

Capítulo 3. Competencias

En este capítulo se explican las competencias que han de ser cubiertas por este Trabajo de Fin de Grado. Para cada competencia, se muestra primero un pequeño párrafo, a modo de resumen y, posteriormente, se desarrolla dicha competencia.

3.1. CII01

Capacidad para diseñar, desarrollar, seleccionar y evaluar aplicaciones y sistemas informáticos, asegurando su fiabilidad, seguridad y calidad, conforme a principios éticos y a la legislación y normativa vigente.

Esta competencia queda cubierta en los capítulos *Estado actual*, donde se explica el organigrama a seguir en la elaboración del proyecto, en los capítulos de *Análisis*, *Desarrollo* y *Resultados y experimentos*.

Se encuentra en estos capítulos los motivos de diseño, desarrollo, etc, que explican exhaustivamente las decisiones tomadas en cada caso.

3.2. CII02

Capacidad para planificar, concebir, desplegar y dirigir proyectos, servicios y sistemas informáticos en todos los ámbitos, liderando su puesta en marcha y su mejora continua y valorando su impacto económico-social.

Esta competencia queda cubierta en los capítulos *Aportación a nuestro entorno socio-económico*, donde se explica el impacto del proyecto en la economía emergente del sector; en el capítulo *Plan de trabajo y temporización* se ven los pasos en la elaboración del proyecto y la temporización.

Como se observa en el capítulo *Aportación a nuestro entorno socio-económico*, los erizos *Diadema* están suponiendo un gran impacto medioambiental en nuestro entorno. La valoración y seguimiento de este impacto es muy costoso. La planificación y puesta en marcha de este proyecto se inspira en la necesidad de desarrollar herramientas que aminoren estos costes.

3.3. CII04

Capacidad para elaborar el pliego de condiciones técnicas de una instalación informática que cumpla los estándares y normativas vigentes.

Pliego de condiciones particulares para el desarrollo de una aplicación informática que facilite la clasificación de erizos Diadema en entornos subacuáticos.

3.3.1. Objeto del contrato

Este contrato tiene como objeto el diseño, desarrollo e implementación de un sistema informático que facilite la clasificación de erizos Diadema en entornos subacuáticos.

En líneas generales, se solicita un sistema que permita detectar erizos *Diadema* en imágenes o vídeos, de forma que se clasifiquen automáticamente los tipos de erizos en dicho entorno, con el objetivo de facilitar los estudios posteriores sobre los resultados obtenidos.

3.3.2. Presupuesto

El presupuesto máximo para la presente contratación es de 20000.00 € IGIC incluido. En el cálculo de este presupuesto se ha incluido las horas de programación, de búsqueda de material y de etiquetado de las muestras.

3.3.3. Especificaciones técnicas

- El sistema será capaz de procesar los siguientes formatos de imagen: bmp, dib, jpeg, jpg, jpe, png, pbm, pgm, ppm, sr, ras, tiff, tif.
- La aplicación permitirá el procesado de imágenes de cualquier tamaño.
- Existirá la posibilidad de clasificar los erizos acorde a diferentes criterios, a saber, tasa de aciertos o velocidad y mediante varios métodos diferentes: *Viola-Jones*[11][12], *LBP*[13][14], *Distancia Euclídea*[37], *PCA*[15][16] y *SVM*[38].
- El programa ofrecerá los resultados de la clasificación en modo imagen y en modo texto se mostrará el tiempo de ejecución empleado.
- El usuario podrá elegir las imágenes o vídeos utilizados en la clasificación.

El adjudicatario debe preparar un manual de usuario sin coste adicional alguno, con la suficiente información para que las personas interesadas puedan utilizar de manera fácil la aplicación.

3.3.4. Condiciones particulares

Plazo de ejecución

El contratista se compromete a elaborar el software de conformidad con lo establecido en este documento y con la oferta que hubiese presentado, así como a entregarlo al contratante. Dicha entrega deberá hacerse en el plazo máximo de 6 meses, a contar desde la fecha de firma del contrato.

El software se considerará debidamente entregado cuando se haya recibido la aplicación, esté disponible para su inmediato funcionamiento y se haya facilitado el manual y la documentación técnica correspondiente.

Penalizaciones

Cualquier incumplimiento del servicio técnico postventa y de mantenimiento y plazos de entrega, conllevará una penalización del 0.01% por cada día natural de incumplimiento sobre el precio del contrato. En caso de que el pago por parte del contratante estuviera realizado con anterioridad, la empresa contratada abonará al contratante el importe de la penalización. La penalización máxima por este concepto no excederá del 10% del importe final del Contrato.

Documentación

La parte contratada deberá facilitar un manual de usuario en el momento de entrega del software, así como la documentación técnica necesaria que permita el mantenimiento y reinstalación del sistema si fuera necesario. Además, se requiere un análisis del hardware necesario para la implantación del sistema desarrollado.

3.4. CII18

Conocimiento de la normativa y la regulación de la informática en los ámbitos nacional, europeo e internacional.

Un análisis de las legislaciones que se han promulgado en diversos países arroja que las normas jurídicas que se han puesto en vigor están dirigidas a proteger la utilización abusiva de la información reunida y procesada mediante el uso de computadoras, e incluso en algunas de ellas, se ha previsto formar órganos especializados que protejan los derechos de los ciudadanos amenazados por los ordenadores.

Desde hace aproximadamente diez años, la mayoría de los países europeos han hecho todo lo posible para incluir dentro de la ley la conducta punible penalmente, como el acceso ilegal a sistemas de cómputo o el mantenimiento ilegal de tales accesos, la difusión de virus o la interceptación de mensajes informáticos.

En la mayoría de las naciones occidentales existen normas similares a los países europeos. Todos estos enfoques están inspirados por la misma preocupación de contar con comunicaciones electrónicas, transacciones e intercambios tan confiables y seguros como sea posible.

Dar un concepto sobre delitos informáticos no es una labor fácil y esto, en razón de que su misma denominación alude a una situación muy especial, ya que para hablar de "delitos" en el sentido de acciones tipificadas o contempladas en textos jurídico-penales se requiere que la expresión "delitos informáticos" esté consignada en los códigos penales, lo cual en nuestro país, al igual que en muchos otros, no ha sido objeto de tipificación aún; sin embargo, muchos especialistas en derecho informático emplean esta alusión a los efectos de una mejor conceptualización.

Normativa y regulación de la informática en el ámbito internacional.

En el contexto internacional, son pocos los países que cuentan con una legislación apropiada. Entre ellos, se destacan Alemania, Argentina, Austria, Chile, España, Estados Unidos, Francia, Gran Bretaña y Holanda.

Dado lo anterior, a continuación se mencionan algunos aspectos relacionados con la ley en los diferentes países, así como con los delitos informáticos que persigue.

Alemania

Este país sancionó en 1986 la *Ley contra la Criminalidad Económica*, que contempla los siguientes delitos:

- Espionaje de datos.
- Estafa informática.
- Alteración de datos.
- Sabotaje informático.

Austria

La *Ley de reforma del Código Penal*, sancionada el 22 de Diciembre de 1987, sanciona a aquellos que con dolo causen un perjuicio patrimonial a un tercero influyendo en el resultado de una elaboración de datos automática a través de la confección del programa, por la introducción, cancelación o alteración de datos o por actuar sobre el curso del procesamiento de datos. Además, contempla sanciones para quienes comenten este hecho utilizando su profesión de especialistas en sistemas.

Chile

Chile fue el primer país latinoamericano en sancionar una *Ley contra delitos informáticos*, la cual entró en vigencia el 7 de junio de 1993. Esta ley se refiere a los siguientes delitos:

- La destrucción o inutilización de los datos contenidos dentro de una computadora es castigada con penas de prisión. Asimismo, dentro de esas consideraciones se encuentran los virus.

- Conducta maliciosa tendiente a la destrucción o inutilización de un sistema de tratamiento de información o de sus partes componentes o que dicha conducta impida, obstaculice o modifique su funcionamiento.

- Conducta maliciosa que altere, dañe o destruya los datos contenidos en un sistema de tratamiento de información.

Estados Unidos

Este país adoptó en 1994 el *Acta Federal de Abuso Computacional*, que modificó al *Acta Federal de Fraude y Abuso Computacional* de 1986.

Con la finalidad de eliminar los argumentos hipertécnicos acerca de qué es y qué no es un virus, un gusano, un caballo de Troya y en qué difieren de los virus, la nueva acta proscribe la transmisión de un programa, información, códigos o comandos que causan daños a la computadora, a los sistemas informáticos, a las redes, información, datos o programas. La nueva ley es un adelanto porque está directamente en contra de los actos de transmisión de virus.

Asimismo, en materia de estafas electrónicas, defraudaciones y otros actos dolosos relacionados con los dispositivos de acceso a sistemas informáticos, la legislación estadounidense sanciona con pena de prisión y multa a la persona que defraude a otro mediante la utilización de una computadora o red informática.

En el mes de julio del año 2000, el Senado y la Cámara de Representantes de este país, tras un año largo de deliberaciones, establece el *Acta de Firmas Electrónicas en el Comercio Global y Nacional*. La ley sobre la firma digital responde a la necesidad de dar validez a documentos informáticos –mensajes electrónicos y contratos establecidos mediante Internet- entre empresas (para el B2B) y entre empresas y consumidores (para el B2C).

Francia

En enero de 1988, este país dictó la *Ley relativa al fraude informático*, en la que se consideran aspectos como:

- Intromisión fraudulenta que suprima o modifique datos.

- Conducta intencional en la violación de derechos a terceros que haya impedido o alterado el funcionamiento de un sistema de procesamiento automatizado de datos.

- Conducta intencional en la violación de derechos a terceros, en forma directa o indirecta, en la introducción de datos en un sistema de procesamiento automatizado o la supresión o modificación de los datos que éste contiene, o sus modos de procesamiento o de transmisión.

- Supresión o modificación de datos contenidos en el sistema, o bien en la alteración del funcionamiento del sistema (sabotaje).

Gran Bretaña

Debido a un caso de hacking en 1991, comenzó a regir en este país la *Computer Misuse Act* (Ley de Abusos Informáticos). Mediante esta ley, el intento, exitoso o no, de alterar datos informáticos es penado con hasta cinco años de prisión o multas. Esta ley tiene un apartado que especifica la modificación de datos sin autorización.

Holanda

El 1 de Marzo de 1993 entró en vigencia la *Ley de Delitos Informáticos*, en la cual se penaliza los siguientes delitos:

- El hacking y el preacking (utilización de servicios de telecomunicaciones evitando el pago total o parcial de dicho servicio).

- La ingeniería social (arte de convencer a la gente de entregar información que en circunstancias normales no entregaría).

- La distribución de virus.

Normativa y regulación de la informática en el ámbito europeo

Hasta ahora, el principal esfuerzo europeo por regular el tema de los delitos informáticos dio como resultado el *Convenio sobre la Ciberdelincuencia*, de 21 de noviembre de 2001. Este documento fue firmado por los representantes de cada país miembro del Consejo de Europa, aunque su eficacia depende de su posterior refrendo por los órganos nacionales de cada país firmante.

El *Convenio sobre la Ciberdelincuencia* permitió la definición de los delitos informáticos y algunos elementos relacionados con éstos, tales como *sistemas informáticos*, *datos informáticos*, o *proveedor de servicios*. Estos delitos informáticos fueron clasificados en cuatro grupos:

1. Delitos contra la confidencialidad, la integridad y la disponibilidad de los datos y sistemas informáticos.
    - Acceso ilícito a sistemas informáticos.
    - Interceptación ilícita de datos informáticos.
    - Interferencia en el sistema mediante la introducción, transmisión, provocación de daños, borrado, alteración o supresión de éstos.
    - Abuso de dispositivos que faciliten la comisión de delitos.
2. Delitos informáticos.
    - Falsificación informática que produzca la alteración, borrado o supresión de datos informáticos que ocasionen datos no auténticos.
    - Fraudes informáticos.
3. Delitos relacionados con el contenido.
    - Delitos relacionados con la pornografía infantil.
4. Delitos relacionados con infracciones de la propiedad intelectual y derechos afines.

Conviene destacar que en el *Convenio sobre la Ciberdelincuencia* se encomienda a cada Parte que tome las medidas necesarias para tipificar como delito en su derecho interno cada uno de los apartados descritos en cada categoría.

En la Disposición 14221 del BOE núm. 226 de 2010 se encuentra el *Instrumento de Ratificación del Convenio sobre la Ciberdelincuencia*, hecho en Budapest el 23 de noviembre de 2001.

Normativa y regulación de la informática en el ámbito nacional

Leyes y Decretos Ley.

Ley Orgánica de Protección de datos de carácter personal.

Régimen sancionador aplicable (BOE No298 de 14/XII/99 que publicó la Ley Org. 15/1999 de 13 de Dic.).

Objeto: Proteger y garantizar las libertades públicas y derechos fundamentales de las personas, especialmente su HONOR e INTIMIDAD personal y familiar.

Aspectos de interés: Serán responsables: "Los responsables de los ficheros o de los tratamientos" y "los encargados de los tratamientos".

Tipos de infracciones:

- Leves (art.44.2): multas de 100.000 a 10M pts.
  Ej. Rectificar datos o no comunicarlos a la Agencia de Protección de Datos.

- Graves (art.43): multas de 10M a 50M pts.
  Ej. No mantener sistemas de seguridad, obstrucción o inspección, uso en provecho propio…

- Muy Graves (art.45): multas de 50M a 100M pts ("Conductas reprochables").
  Ej. Vulnerar a propósito el secretismo, etc…

Ley 7/1998 de 13 de abril que regula las condiciones generales de contratación.

R.D. 1906/1999 de 17/XII que regula la contratación telefónica.

R.D. Ley 14/1999 de 17/XII sobre Firma Electrónica (BOE No224 de 18/XII).

Firma electrónica: Dispositivo electrónico que permite la identificación del signatario de las operaciones realizadas por Internet.

Identifica: El firmante (autentificación) y Evita el retracto (no repudio).

Código Penal

Ley Orgánica 10/1995 de 23/XI

Tipifica delitos y faltas por el uso de la informática, concretamente contra la Intimidad, Patrimonio, Socioeconómicos y Propiedad Intelectual.

Título X: "Delitos contra la intimidad, derecho a la propia imagen y la inviolabilidad del Domicilio".

- Apoderarse de papeles, e-mails, mensajes, otros…
- Cracks: delitos.
- Obtener datos de terceros…

Recomendaciones de la APD

Información en la recogida de datos

- Cuando suministre datos personales a cualquier organización (proveedores de acceso, proveedores de contenido, vendedores a través de comercio electrónico, etc.) sea consciente de a quién se los facilita y con qué finalidad.
- Procure averiguar la política de sus proveedores y administradores de listas y directorios en lo que se refiere a venta, intercambio o alquiler de los datos que les suministra. Solicite que sus datos personales no vayan unidos a su identificación de acceso a Internet.

Finalidad para la que se recogen los datos

- Desconfíe si los datos que le solicitan son excesivos para la finalidad con la que se recogen o innecesarios para el servicio que se le presta.

- Tenga en cuenta que cuando introduce su dirección de correo electrónico en un directorio, lista de distribución o grupo de noticias, dicha dirección puede ser recogida por terceros para ser utilizada con una finalidad diferente, como por ejemplo, remitirle publicidad no deseada.

- Cuando navegue por Internet, sea consciente de que los servidores Web que visita pueden registrar tanto las páginas a las que accede como la frecuencia y los temas o materias por las que busca, aunque no le informen de ello. Asimismo, su pertenencia a determinados grupos de noticias y listas de distribución puede contribuir a la elaboración de perfiles más o menos detallados sobre su persona. En el caso de que no desee dejar constancia de sus actividades en la red, utilice los mecanismos para preservar el anonimato que se describen en el cuerpo de este documento.

Seguridad en el intercambio de datos

- Utilice, siempre que sea posible, las últimas versiones de los programas navegadores, ya que cada vez suelen incorporar mejores medidas de seguridad. Considere la posibilidad de activar en dichos programas las opciones que alerten sobre los intercambios de datos no deseados y no rellene aquellos datos que no desee hacer públicos (por ejemplo, dirección de correo electrónico, nombre, apellidos, etc.).

- No realice transacciones comerciales electrónicas a través de proveedores con sistemas inseguros o no fiables. Consulte el manual de su navegador para averiguar cómo informa de que se ha establecido una conexión con un servidor seguro.

- Recuerde que existen sistemas de dinero electrónico que preservan el anonimato de sus compras en Internet.

- Utilice los mecanismos de seguridad que tenga a su alcance para proteger sus datos de accesos no deseados. El medio más fiable para conseguirlo es el cifrado de los mismos.

- Salvo que se utilicen mecanismos de integridad, autenticación y certificación (firma digital, notarios electrónicos, etc.) no confíe ciegamente en que la persona u organización que le remite un mensaje es quien dice ser y en que el contenido del mismo no se ha modificado, aunque esto sea así en la inmensa mayoría de las ocasiones.

Para terminar

- Siempre que se le soliciten datos personales que no esté obligado legalmente a suministrar, sopese los beneficios que va a recibir de la organización que los recoge frente a los posibles riesgos de utilización irregular de los mismos.

- Ante cualquier duda sobre la legalidad de la utilización de sus datos de carácter personal, póngase en contacto con la *Agencia de Protección de Datos*.

La protección jurídica de programas de ordenador. Piratería informática

- El Real Decreto Legislativo 1/1996, por el que se aprueba el *Texto Refundido sobre Propiedad Intelectual*, la protección jurídica de los programas de ordenador, antes regulada por la *Ley de Protección Jurídica de Programas de Ordenador* y por la *Ley de Propiedad Intelectual*, crea un marco jurídico en contra de la piratería informática.

- El Texto Refundido desarrolla una serie de medidas para combatir la piratería informática, como la posibilidad de que los fabricantes de programas de ordenador soliciten a la justicia española la realización de un registro sorpresa en empresas en las que existan sospechas fundadas o evidencias de delito. España es uno de los países en los que se puede acudir a esta medida cautelar. De esta manera se erradica la posibilidad de que los presuntos infractores puedan destruir las pruebas existentes lo cual, indudablemente, ocurriría si se les notificase por adelantado la realización de un registro.

¿En qué casos se infringe la Ley?

- Al copiar o distribuir un programa de ordenador o la documentación que le acompaña, incluidas aplicaciones, datos, códigos y manuales, sin permiso expreso o licencia del propietario de los derechos de explotación.

- Al utilizar un programa sin la correspondiente licencia o autorización del fabricante, con independencia de que se utilice en un solo ordenador o en varios de forma simultánea.

- Al utilizar programas de ordenador en un número de copias superior al autorizado por el fabricante en sus contratos o licencias de uso.

- En empresas y demás organizaciones, al fomentar, consciente o inconscientemente, permitir, obligar o presionar a los empleados a realizar o distribuir copias no autorizadas del programa.

- Al efectuar copias no autorizadas porque alguien lo requiere u obliga a ello. Al ceder o prestar el programa de forma que pueda ser copiado o al copiarlo mientras está en su posesión en calidad de cedido o prestado.

- Al crear, importar, poseer o negociar con artículos destinados a burlar o neutralizar cualquier medio técnico aplicado para proteger el programa de ordenador.

Medidas Judiciales

Si finalmente existe evidencia de delito, las medidas judiciales que pueden adoptarse son:

- Solicitar al Juez un registro sorpresa de las instalaciones del presunto infractor, tanto por la vía civil, como por la penal.

- Solicitar al Juez la adopción urgente de medidas cautelares de protección.

- Exigir indemnizaciones acordes con los daños materiales y morales causados.

- El cierre del centro de actividad del infractor.

- El secuestro de todos aquellos medios destinados a suprimir los dispositivos técnicos que protegen un programa desarrollado y comercializado por un fabricante de programas.

Capítulo 4. Aportaciones a nuestro entorno socio-económico

Desde el año 2008, el Cabildo de Tenerife, a través del área de pesca de la Consejería de Agricultura, Ganadería y Pesca, ya había elaborado una serie de propuestas de acciones destinadas a la conservación de las costas de la isla y su riqueza marina[24]. Entre otras, la prioridad se centró en la erradicación del erizo de lima (*Diadema antillarum*), que afecta al 60% de los ecosistemas marinos rocosos de las costas de Tenerife. Esta actuación, entre otras, fue consensuada en una reunión entre las distintas administraciones insulares y autonómicas, así como las instituciones científicas más importantes del Archipiélago.

Con respecto al erizo de lima, se diseñó un Plan Regional para su erradicación, elaborando un protocolo de actuaciones consensuado que permita la regeneración de los ecosistemas más afectados por la plaga del erizo:

- La puesta a punto de una metodología para la eliminación de los erizos;
- Implicar y concienciar a los buceadores y pescadores deportivos y profesionales en la erradicación de la plaga.
- La realización de campañas de divulgación.
- El establecimiento de acciones de futuro para la continuación en el tiempo de estas labores de contención de la plaga.

Estas líneas de actuación encajan asimismo, dentro del convenio firmado entre la Consejería de Agricultura, Ganadería, Pesca y Alimentación del Gobierno de Canarias y el Cabildo Insular de Tenerife para el apoyo a los recursos pesqueros y el control del Diadema antillarum, para lo que se destinaron 70.282 euros en dos anualidades distribuidas en el 2008 y el 2009.

La presencia del erizo de lima es natural en las islas. La sobrepesca, el calentamiento global de la temperatura del agua marina y la contaminación, entre otras causas, han propiciado su explosión demográfica. Este erizo es un voraz consumidor de algas (base de la cadena trófica de numerosos ecosistemas litorales del Archipiélago); esta acción predadora del erizo produce zonas desprovistas de cualquier tipo de cobertura vegetal y animal que son conocidas como blanquizales, lo que supone pérdidas de biodiversidad, productividad, zonas de refugio y alimentación y zonas de reproducción y cría.

Asímismo, el Cabildo de Gran Canaria, ha coordinado también desde el año 2008 mediante sus Consejerías de Medio Ambiente y Pesca este proyecto, que busca una metodología de reducción y control de los blanquizales a nivel insular, huyendo del desarrollo de pequeñas acciones de erradicación de Diadema antillarum de forma local y aislada, y generando un proyecto continuado de años de duración que buscará protocolos efectivos de actuación, recopilará información científica de los puntos de actuación y fomentará el uso sostenible de los recursos marinos[25].

Este proyecto contó con un presupuesto de 50.741 euros y se ha ejecutado en Sardina del Norte, en la Bahía del Confital y en aguas de la Reserva de la Biosfera de Gran Canaria (concretamente en la Baja de Pasito Blanco).

También, en marzo del año 2010, la Consejería de Pesca del Cabildo de Lanzarote pone en marcha la primera acción de dicho año para el control de las poblaciones del erizo de lima. En esa ocasión fue el litoral del Marina Rubicón, en Playa Blanca, la zona elegida para actuar. Después, se continuó por otras zonas del litoral en La Graciosa y en Punta Mujeres.

El programa de control se inició el domingo 28 de marzo, a partir de las 10:30 de la mañana, con la participación de diversos centros y clubes de buceo y unos 40 voluntarios que se sumaron a la iniciativa. Este programa de actuaciones es posible gracias a un convenio de colaboración entre el Cabildo de Lanzarote y la Viceconsejería de Pesca del Gobierno de Canarias, puesto en marcha en el año 2008 y en el que se trabaja de forma continuada anualmente.

Según afirma la Consejera de Pesca, Nereida Pérez, les satisface poder continuar con este programa de control porque ya empiezan a ver los resultados. Sólo en el caso de Playa Blanca, se detectaba una considerable recuperación del fondo; los erizos se habían reducido en un 90%, las algas habían crecido tanto en porte como en número de pies y la fauna había mostrado un importante incremento con las estrellas de mar, gallos cochino, bocinegros y samas[26].

Así pues, de tener éxito, este proyecto podría aportar una ayuda a la eliminación del problema medioambiental que produce dicho ejemplar de erizo (Diadema antillarum), ya que, como se ha comentado anteriormente, la tarea de identificación y recuento de estos ejemplares se realizan en la actualidad mediante técnicas de conteo manual, bien in-situ con buceadores o bien sobre imágenes de vídeos, con elevado coste en términos de tiempo, capacidad de muestreo y mano de obra.

Este proyecto puede, a su vez, suponer un interesante avance, pues además de simplificar el desarrollo de estas tareas, serviría de ejemplo de transferencia tecnológica al ámbito de las ciencias marinas.

Este sistema se ha diseñado también para poder ser empleado a bordo de un vehículo subacuático que pudiera desarrollar tareas de evaluación de impacto (localización, conteo…) en aquellas zonas donde los buceadores no llegan.

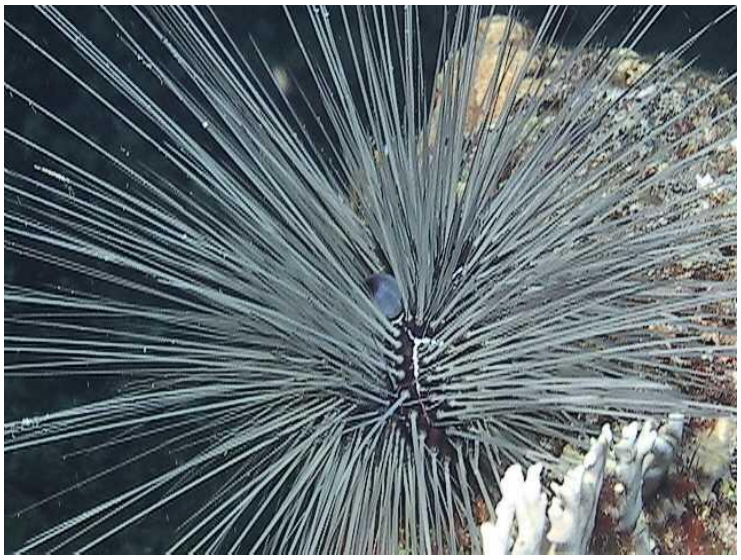

Fig. 4.1. Un ejemplar de erizo de lima (Diadema antillarum).

Fig. 4.2. Blanquizal. Ejemplo de acción predadora del erizo *Diadema antillarum*.

Capítulo 5. Normativa y legislación

A continuación se incluye la legislación vigente que afecta a este Trabajo de Fin de Grado.

5.1. BSD Licenses

OpenCV fue liberado bajo la licencia *BSD*[27][28][29][30]. Las licencias *BSD* son una familia de licencias de software libre[33] permisivas. La licencia original se utilizó para la *Berkeley Software Distribution* (*BSD*), un sistema operativo tipo *Unix*, de donde adoptó su nombre.

Los dueños originales de *BSD* fueron los Regentes de la Universidad de California, porque *BSD* fue escrito por primera vez en dicha Universidad, en Berkeley. La primera versión de la licencia ha sido revisada, y las licencias resultantes son más adecuadamente denominadas *modified BSD licenses*.

Dos variantes de la licencia, la *New BSD License/Modified BSD License* y la *Simplified BSD License/FreeBSD License*, han sido verificadas como compatibles con las licencias de software libre *GPL* por la *Free Software Foundation*, y han sido seleccionadas como las licencias de código abierto por la *Open Source Initiative*, mientras que la original, licencia de 4 cláusulas, no ha sido aceptada como una licencia de código abierto y, aunque la original es considerada una licencia de software libre por la *FSF*, la *FSF* no considera que sea compatible con la *GPL* debido a la cláusula de publicidad.

Al ser licencias de software libre permisivas, estas ponen requisitos mínimos acerca de cómo se puede redistribuir el software. Esto está en contraste con las *copyleft licenses*, que tienen reciprocidad/requisitos de compartir por igual.

El autor, bajo esta licencia, mantiene la protección de los derechos de autor únicamente para la renuncia de garantía y para requerir la adecuada atribución de la autoría en trabajos derivados, pero permite la libre redistribución y modificación, permitiéndose el uso en software comercial. Por este motivo, se ha utilizado en software propietario, como es el caso de *Mac OS X*.

Puede argumentarse que esta licencia asegura “verdadero” software libre, en el sentido de que el usuario tiene libertad ilimitada con respecto al software, y que puede decidir incluso redistribuirlo como no libre. Otras opiniones están orientadas a destacar que este tipo de licencia no contribuye al desarrollo de más software libre.

Las licencias derivadas de la *BSD* se conocen comúmente como "*licencia BSD*". Hoy en día, la licencia BSD típica es la versión de 3 cláusulas.

A continuación se muestra una serie de tablas de cada *licencia BSD*:

BSD License (Original) (de 4 cláusulas)	
Autor	Universidad de California
Versión	N/a
Edición	Dominio público
Fecha de publicación	1989
Compatible con DFSG	Si
Software libre	Si
Aprobada por OSI	No
Compatible GPL	No
Copyleft	No
Utilizable junto con otras licencias	Si

Fig. 5.1. BSD (4 cláusulas)

"New" BSD License (de 3 cláusulas)

Autor	Universidad de California
Versión	N/a
Edición	Dominio público
Fecha de publicación	1990
Software libre	Si
Aprobada por OSI	Si
Compatible GPL	Si
Copyleft	No
Utilizable junto con otras licencias	Si

Fig. 5.2. BSD (3 cláusulas)

"Simplified" BSD License (de 2 cláusulas)	
Autor	El proyectoFreeBSD
Versión	N/a
Edición	Dominio público
Fecha de publicación	?
Software libre	Si
Aprobada por OSI	Si
Compatible GPL	Si
Copyleft	No
Utilizable junto con otras licencias	Si

Fig. 5.3. BSD (2 cláusulas)

5.2. GNU General Public License

La *Licencia Pública General GNU* o más conocida por su nombre en inglés *GNU General Public License* o simplemente sus siglas del inglés *GNU GPL*[31][32], es una licencia creada por la *Free Software Foundation* en 1989 (la primera versión), y está orientada principalmente a proteger la libre distribución, modificación y uso de software. Su propósito es declarar que el software cubierto por esta licencia es software libre y protegerlo de intentos de apropiación que restrinjan esas libertades a los usuarios. Existen varias licencias "hermanas" de la *GPL*, como la *licencia de documentación libre de GNU* (*GFDL*), la *Open Audio License*, para trabajos musicales, etc., y otras menos restrictivas, como la *MGPL*, o la *LGPL* (*Lesser General Publical License*, antes *Library General Publical License*), que permiten el enlace dinámico de aplicaciones libres a aplicaciones no libres.

La licencia *GPL*, al ser un documento que cede ciertos derechos al usuario, asume la forma de un contrato, por lo que usualmente se la denomina *contrato de licencia* o *acuerdo de licencia*.

5.2.1. Software libre

El software libre (en inglés free software, aunque esta denominación también se confunde a veces con "gratis" por la ambigüedad del término en el idioma inglés) es la denominación del software que respeta la libertad de los usuarios sobre su producto adquirido y, por tanto, una vez obtenido, puede ser usado, copiado, estudiado, modificado y redistribuido libremente[34]. Según la *Free Software Foundation*, el software libre se refiere a la libertad de los usuarios para ejecutar, copiar, distribuir, estudiar, modificar el software y distribuirlo modificado.

El software libre suele estar disponible gratuitamente o al precio de costo de la distribución a través de otros medios; sin embargo, no es obligatorio que sea así, por lo tanto, no hay que asociar software libre a "software gratuito" (denominado usualmente freeware), ya que, conservando su carácter de libre, puede ser distribuido comercialmente (lo que se denomina "software comercial"). Análogamente, el "software gratuito" incluye en ocasiones el código fuente; no obstante, este tipo de software no es libre en el mismo sentido que el software libre, a menos que se garanticen los derechos de modificación y redistribución de dichas versiones modificadas del programa.

Tampoco debe confundirse software libre con "software de dominio público". Este último, es aquel software que no requiere de licencia, pues sus derechos de explotación son para toda la humanidad, porque pertenece a todos por igual. Cualquiera puede hacer uso de él, siempre con fines legales y consignando su autoría original. Este software sería aquél cuyo autor lo dona a la humanidad o cuyos derechos de autor han expirado, tras un plazo contado desde la muerte de éste, habitualmente 70 años. Si un autor condiciona su uso bajo una licencia, por muy débil que sea, ya no es del dominio público.

Capítulo 6. Requisitos hardware y software

Para poder llevar a cabo los objetivos del proyecto se han usado, durante el desarrollo de éste, una serie de recursos. Estos recursos se han desglosado en tres grupos: *hardware*, *software* y *otras herramientas*.

6.1. Requisitos hardware

Para la realización de este proyecto ha sido necesario disponer de una máquina de cierto nivel de prestaciones para la parte del entrenamiento, que es computacionalmente intensiva. En concreto, se ha usado un portátil con 4GB de RAM y una tarjeta gráfica de 1 GB.

6.2. Requisitos software

Se ha utilizado el programa *Matlab* para las primeras fases de desarrollo, en las que se realizaban aplicaciones para poder utilizar posteriormente las herramientas necesarias para la generación de los clasificadores, así como en fases posteriores.

También se ha usado en las siguientes fases de desarrollo un editor ligero denominado *Gedit* para utilizar los clasificadores generados. Para compilar los programas escritos en el editor *Gedit*, se ha optado por el *g++*.

Para utilizar las herramientas necesarias para la generación de los clasificadores se ha instalado la librería *OpenCV*, una librería muy utilizada en *Visión por Computador*.

Sistema Operativo

Para las primeras fases de desarrollo, al utilizar *Matlab*, se ha usado el sistema operativo *Windows*. Concretamente, se ha optado por la distribución *Windows 7 Professional* última distribución existente cuando comenzó la realización del proyecto.

Para las demás fases se ha utilizado el Sistema Operativo *Linux*. *Linux* es un poderoso y sumamente versátil Sistema Operativo de 32-64 bits, multiusuario y multitarea. Fue creado en 1991 por Linus Torvalds, siendo entonces un estudiante de la Universidad de Helsinki. Linus se basó sobre *Unix*.

Linux y toda la comunidad que gira alrededor de él hacen que sea una buena plataforma de desarrollo. Por un lado, se dispone de un sistema operativo estable, seguro y en constante evolución. Por otro lado, existen las herramientas gratuitas de todo tipo y que además son de alta calidad.

La distribución utilizada para la realización del proyecto es *Ubuntu 11.04*, última distribución existente cuando se realizó el proyecto.

6.3. Otras herramientas utilizadas

Lenguaje de programación C

C es un lenguaje de programación creado en 1972 por *Dennis M. Ritchie* en los *Laboratorios Bell* como evolución del anterior lenguaje *B*, a su vez basado en *BCPL*.

Al igual que *B*, es un lenguaje orientado a la implementación de Sistemas Operativos, concretamente *Unix*. *C* es apreciado por la eficiencia del código que produce y es el lenguaje de programación más popular para crear software de sistemas, aunque también se utiliza para crear aplicaciones.

Se trata de un lenguaje débilmente tipificado de medio nivel pero con muchas características de bajo nivel. Dispone de las estructuras típicas de los lenguajes de alto nivel pero, a su vez, dispone de construcciones del lenguaje que permiten un control a muy bajo nivel. Los compiladores suelen ofrecer extensiones al lenguaje que posibilitan mezclar código en ensamblador con código *C* o acceder directamente a memoria o dispositivos periféricos.

Uno de los objetivos de diseño del lenguaje *C* es que sólo sean necesarias unas pocas instrucciones en lenguaje máquina para traducir cada elemento del lenguaje, sin que haga falta un soporte intenso en tiempo de ejecución. Es muy posible escribir *C* a bajo nivel de abstracción; de hecho, *C* se usó como intermediario entre diferentes lenguajes.

En parte a causa de ser de relativamente bajo nivel y de tener un modesto conjunto de características, se pueden desarrollar compiladores de *C* fácilmente. En consecuencia, el lenguaje *C* está disponible en un amplio abanico de plataformas (seguramente más que cualquier otro lenguaje). Además, a pesar de su naturaleza de bajo nivel, el lenguaje se desarrolló para incentivar la programación independiente de la máquina. Un programa escrito, cumpliendo los estándares e intentando que sea portátil, puede compilarse en muchos computadores.

Lenguaje de programación C++

C++ (pronunciado *C Plus Plus*) fue diseñado por *Bjarne Stroustrup* y fue el segundo intento de proporcionar orientación a objetos a *C* y es la variante más difundida y aceptada. Esta versión combina la flexibilidad y el acceso de bajo nivel de *C* con las características de programación orientada a objetos como abstracción, encapsulación y ocultación.

Colección de imágenes y vídeos

Se ha utilizado una colección de imágenes y vídeos de calidad media para la generación de los clasificadores y la puesta a prueba de éstos. Debido a falta de recursos y a la falta de colaboración de investigadores e instituciones que, disponiendo de este material, no han querido cederlo, no se ha podido conseguir una colección con la calidad deseada. Por ello, ha sido necesario reunir este material realizando una laboriosa búsqueda de dicho material en Internet, a veces con una calidad, en términos de tamaño de imagen, calidad y resolución, inadecuadas.

Conexión a Internet

Para completar la colección de imágenes y vídeos ha sido necesaria conexión a Internet, así como para obtener cierta información para la generación de la memoria del proyecto y el aprendizaje necesario para llevar a cabo este proyecto.

Capítulo 7. Plan de trabajo y temporización

En este capítulo se presenta el plan de trabajo desglosado en etapas, con una estimación en cada etapa del tiempo de ejecución.

- **Etapa 1. Análisis global. Revisión bibliográfica y análisis previo de Teoría de Clasificadores.**

  El tiempo invertido en esta etapa fue de 25 horas. De ese tiempo, 15 horas se destinaron a leer sobre Teoría de Clasificadores; y las últimas 10 horas se emplearon en la búsqueda de imágenes y vídeos en Internet.

- **Etapa 2. Análisis y diseño de estructura de datos y modelo en Matlab.**

  El tiempo invertido en esta etapa fue de 60 horas. De ese tiempo, alrededor de 20 horas se destinaron al estudio de las aplicaciones auxiliares a implementar. Las otras 40 horas fueron para el diseño de dichas aplicaciones en Matlab.

- **Etapa 3. Implementación en Matlab de aplicaciones auxiliares.**

  El tiempo invertido en esta etapa fue aproximadamente de 70 horas. Este tiempo es estimado, siendo una conjunción entre programación y búsqueda de las funciones necesarias.

- **Etapa 4. Análisis y diseño de estructura de datos y modelo de la librería OpenCV.**

  El tiempo invertido en esta etapa fue aproximadamente de 60 horas. Las horas invertidas son una estimación, y engloban la lectura de documentación y búsqueda de funciones para la generación de los clasificadores.

- **Etapa 5. Implementación en C/C++ del uso de los clasificadores y pruebas.**

  El tiempo invertido en esta etapa fue de 150 horas. De ese tiempo, alrededor de 50 horas se destinaron al diseño e implementación del uso de los clasificadores, junto con la implementación de las pruebas. Las otras 100 horas se usaron para el estudio de las pruebas necesarias para contrastar los resultados obtenidos y estudiarlos detenidamente.

- **Etapa 6. Documentación y defensa. Confección de la documentación del proyecto y preparación de la defensa.**

  El tiempo invertido en esta etapa fue de 50 horas. Este tiempo fue destinado a la elaboración de la memoria del proyecto.

El tiempo de la elaboración del proyecto fue aproximadamente de 400 horas.

Capítulo 8. Análisis

Este capítulo está destinado al análisis, uno de los puntos más importantes antes de comenzar con el desarrollo del software. Concretamente, se identifican los objetivos y las dificultades inherentes para su consecución.

El escenario en el que nos encontramos es el de un sistema embarcado en un vehículo o transportado por un buceador que inspecciona una cierta área del fondo. El sistema, en una versión diferente, podría estar fijado al fondo de forma semipermanente en una zona a estudiar (seguimiento, identificación de patrones de movimiento, etc.). En este escenario nos encontramos las siguientes características:

- La aproximación es gradual, por lo que la resolución de los detalles (mejor calidad) será progresiva. En los casos en los que el sistema está fijado al fondo, la "aproximación gradual" sería de los animales a la cámara.

- Esto ha llevado a proponer un esquema donde existan varios filtros operando en paralelo. Un tipo detectaría zonas de "probables positivos" (zonas oscuras de geometría redondeada) y serviría para marcar erizos (de cualquier tipo) a cierta distancia. A medida que el observador se aproxime a esas "marcas", es posible que los detectores que discriminen entre erizos *Diadema* y "otros" puedan identificar positivos (*Diadema)* a partir de las zonas activadas con el primer detector.

En resumen, resulta posible justificar un esquema donde se usen de forma escalonada diferentes tipos de detectores y donde el sistema opere (en teoría) guiado por "evidencias".

Para llevar a cabo la evaluación de si, mediante el empleo de técnicas de visión por computador, es posible resolver la tarea propuesta mediante la inspección automática de vídeos e imágenes, se han utilizado varios tipos de clasificadores, incluyéndose a continuación una breve explicación de cada uno de ellos y de las herramientas *OpenCV* y *MATLAB*.

8.1. Teoría sobre clasificadores

En este apartado se va da una breve descripción de cada uno de los métodos utilizados para la realización de este proyecto.

8.1.1. Viola-Jones

El sistema de detección de objetos de *Viola-Jones* es el primer sistema de detección de objetos en ofrecer tasas competitivas para la detección de objetos en tiempo real propuesto en 2001 por *Paul Viola* y *Michael Jones*.

A pesar de que pueden ser entrenados para detectar una variedad de clases de objetos, fue motivada principalmente por el problema de la detección de rostros.

Características y evaluación

Las características empleadas por el sistema de detección universal se fundamentan en la suma de los píxeles de la imagen dentro de áreas rectangulares. Como tales, tienen cierto parecido a las funciones base *Haar*[44], que han sido utilizadas con anterioridad en el ámbito de la detección de objetos basado en imágenes. Sin embargo, dado que todas las características utilizadas por *Viola* y *Jones* se basan en más de un área rectangular, son, por lo general, más complejas.

La siguiente figura ilustra los cuatro diferentes tipos de características utilizadas.

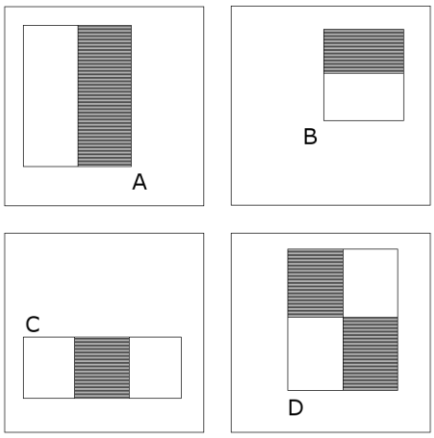

Fig. 8.1. Características “rectángulo”. La suma de los píxeles que se encuentran dentro de los rectángulos blancos se restará de la suma de los píxeles en el rectángulo gris. Las características “de dos rectángulos” son mostradas en (A) y (B). La figura (C) muestra una característica “de tres rectángulos”, y (D) una característica “de cuatro rectángulos”.

El valor de una característica “de dos rectángulos” es la diferencia entre la suma de los píxeles dentro de dos regiones rectangulares. Las regiones tienen el mismo tamaño y forma y están horizontal o verticalmente adyacentes.

La característica “de tres rectángulos” calcula la suma dentro de los dos rectángulos exteriores y se le resta a la suma en el rectángulo central.

Por último, una característica “de cuatro rectángulos” calcula la diferencia entre pares diagonales de los rectángulos.

Como es de esperar, los elementos rectangulares de este tipo son bastante primitivos en comparación con otras alternativas como filtros orientables. Sin embargo, con el uso de una representación de imagen denominada “imagen integral”, los elementos rectangulares se pueden evaluar en tiempo constante, lo que les da una ventaja considerable con respecto a la velocidad de sus “familiares” más sofisticados.

Imagen Integral

Las características “rectángulo” se pueden calcular muy rápidamente usando una representación intermedia de la imagen denominada imagen integral. La imagen integral en el lugar (x,y) contiene la suma de los píxeles situados arriba y a la izquierda de (x,y):

$$ii(x,y) = \sum_{x' \leq x, y' \leq y} i(x',y')$$

Donde $ii(x,y)$ es la imagen integral y la imagen original es $i(x,y)$.

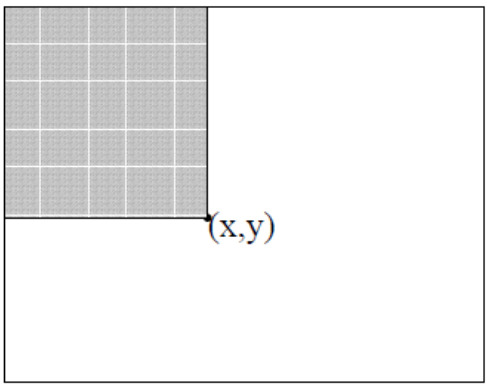

Fig. 8.2. El valor de la imagen integral en el punto (*x*,*y*) es la suma de todos los píxeles situados arriba y a la izquierda.

Arquitectura en cascada

La evaluación de los clasificadores buenos generados por el proceso de aprendizaje se puede hacer rápidamente, pero no es lo suficientemente rápido para ejecutarse en tiempo real. Por este motivo, los clasificadores robustos están dispuestos en una cascada por orden de complejidad, donde cada clasificador sucesivo es entrenado sólo en las muestras seleccionadas que pasan a través de los clasificadores anteriores. Si en cualquier etapa de la cascada un clasificador rechaza la sub-ventana bajo la inspección, no se lleva a cabo ningún tratamiento posterior y se continúa buscando en la siguiente sub-ventana. En la siguiente imagen se muestra un esquema de este proceso.

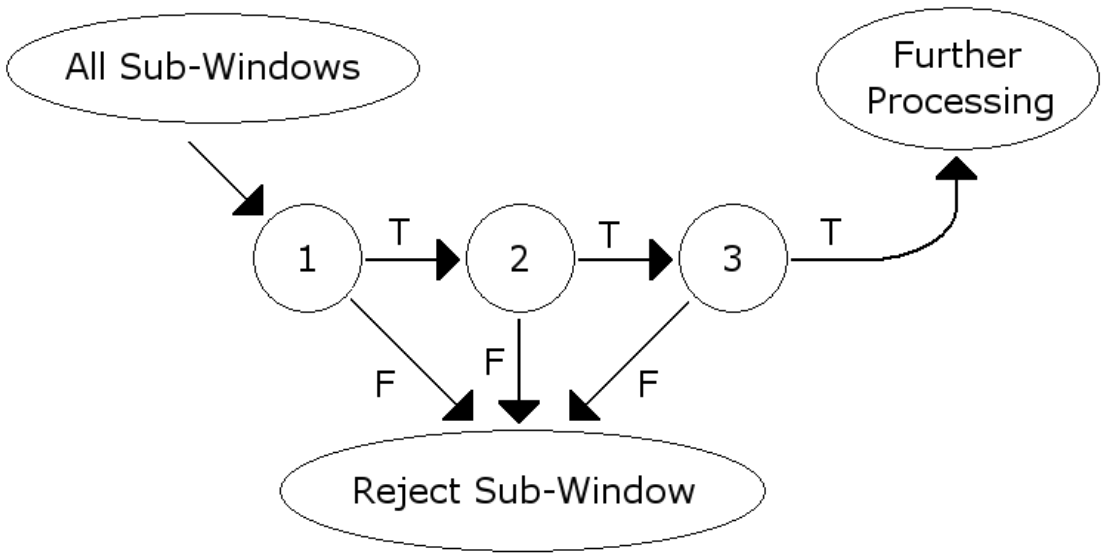

Fig. 8.3. Arquitectura en cascada

La arquitectura en cascada tiene implicaciones interesantes para el rendimiento de los clasificadores individuales. Debido a la activación de cada clasificador, depende totalmente de la conducta de su predecesor. La tasa de falsos positivos para una cascada entera es:

$$F = \prod_{i=1}^{K} f_i.$$

Ec. 8.1. Tasa de falsos positivos de una cascada

Del mismo modo, la tasa de detección es:

$$D = \prod_{i=1}^{K} d_i.$$

Ec. 8.2. Tasa de detección de una cascada

Por lo tanto, para que coincida con las tasas de falsos positivos normalmente conseguidos por los otros detectores, cada clasificador puede llegar a tener resultados sorprendentemente pobres. Por ejemplo, para una cascada de 32 etapas, para lograr una disminución de la tasa de falsos positivos de un 60%, cada clasificador puede llegar a alcanzar una tasa de falsos positivos de alrededor del 65%. Al mismo tiempo, sin embargo, cada clasificador tiene que ser excepcionalmente capaz de alcanzar tasas de detección adecuadas. Por ejemplo, para lograr una tasa de detección de alrededor del 90%, cada clasificador de la mencionada cascada necesita alcanzar una tasa de detección de aproximadamente 99.7%.

8.1.2. Local Binary Patterns

El operador de textura *Local Binary Patterns* (*LBP*) fue introducido por primera vez como una medida complementaria para el contraste de la imagen local. La primera versión operaba con los ocho vecinos de un píxel, utilizando el valor del píxel central como umbral. Un código *LBP* para un "vecindario" ha sido producido multiplicando los valores límite con las ponderaciones asignadas a los píxeles correspondientes y sumando el resultado.

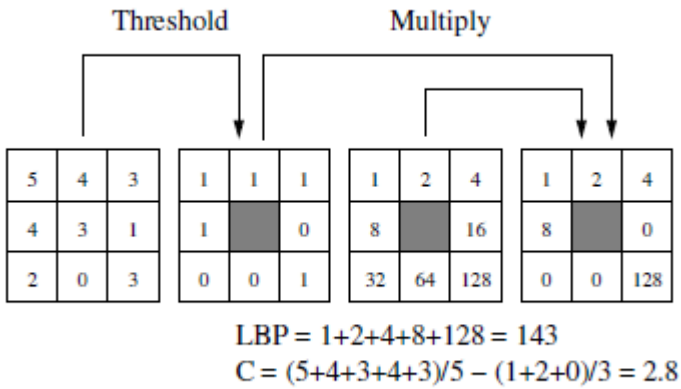

Fig. 8.4. Cómo calcular el código LBP original y una medida de contraste.

Dado que el *LBP* es, por definición, invariante a los cambios mono-tónicos en escala de grises, se complementó con una medida independiente de contraste local. El dibujo anterior muestra cómo se ha obtenido la medida de contraste (C). El promedio de los niveles de gris por debajo del píxel central se resta del de los niveles de gris igual o por encima que el píxel central.

En la forma actual del operador *LBP* de la versión básica, la definición original se extiende a "vecindarios" circulares arbitrarios. Se han desarrollado varias variantes de esta idea básica. La idea básica es, sin embargo, la misma: un código binario que describe el patrón de la textura local es construido mediante el umbral de un "vecindario" por el valor de gris de su centro. El operador está relacionado con muchos métodos de análisis de texturas conocidos.

Estas relaciones se resumen en el siguiente dibujo.

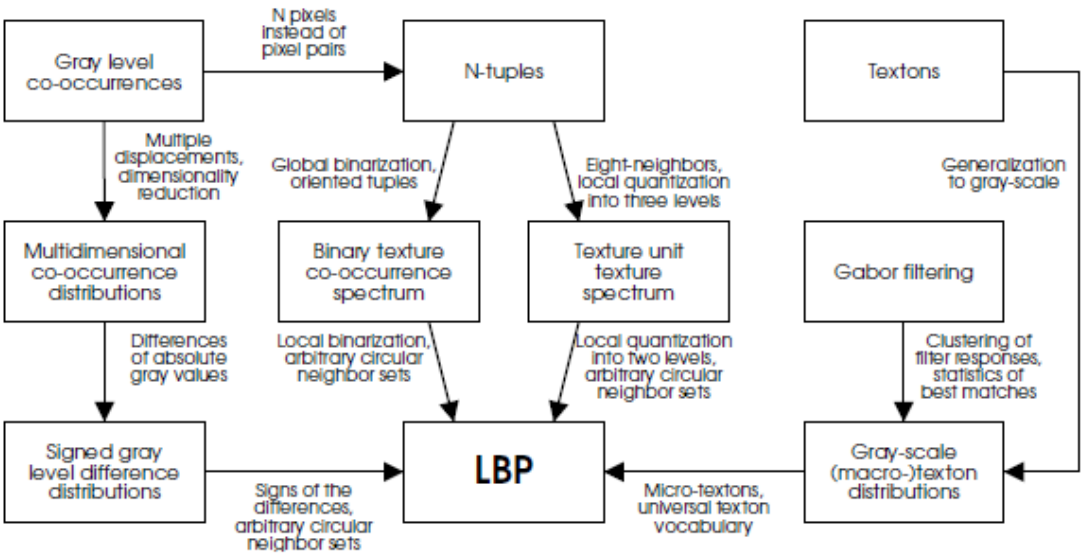

Fig. 8.5. LBP en el campo de los operadores de análisis de texturas.

En el siguiente apartado se comentan otros tipos de clasificadores que se han utilizado, a saber, *Distancia Euclídea*, *SVM* y *PCA*. Se usan estos clasificadores porque suelen dar buenas soluciones en otros problemas como, por ejemplo, en problemas de detección de rostros.

8.1.3. Distancia Euclídea

Las reglas de clasificación basadas en distancia son las siguientes.

Sea *d : E x E -> R* una métrica y sea *y = y(x)* un punto de *E* y teniendo en cuenta el caso:

- **Vecino más próximo (1-NN):** Sea P_c un conjunto de prototipos representante de la clase *c*:

  $$x \in c` \Leftrightarrow \exists p \in Pc` : d(y, p) \leq d(y, p') \; \forall p' \in Pc, 1 \leq c \leq C, c \neq c'$$

  En caso de empate se decide, entre las empatadas, la clase con mayor número de prototipos.

  Y teniendo en cuenta el caso de la *Distancia Euclídea*, dicha distancia será:

$$d(P_1, P_2) = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$$

Ec. 8.3. Distancia entre dos puntos

En resumen, se clasifica *y* en la clase a la que pertenezcan la mayoría de sus *k* vecinos más próximos.

8.1.4. Análisis de Componentes Principales

En estadística, el *Análisis de Componentes Principales* o *ACP* (en inglés *Principal Component Analysis* o *PCA*) es una técnica utilizada para reducir la dimensionalidad de un conjunto de datos. Intuitivamente, la técnica sirve para hallar las causas de la variabilidad de un conjunto de datos y ordenarlas por importancia.

Técnicamente, el *PCA* busca la proyección según la cual los datos queden mejor representados en términos de mínimos cuadrados. Este método se emplea sobre todo en análisis exploratorio de datos y para construir modelos predictivos.

El *PCA* construye una transformación lineal que escoge un nuevo sistema de coordenadas para el conjunto original de datos en el cual la varianza de mayor tamaño del conjunto de datos es capturada en el primer eje, denominado el *Primer Componente Principal*, la segunda varianza más grande es el segundo eje, y así sucesivamente. Para construir esta transformación lineal debe construirse primero la *matriz de covarianza* o *matriz de coeficientes de correlación*. Debido a la simetría de esta matriz, existe una base completa de vectores propios de la misma. La transformación que lleva de las antiguas coordenadas a las coordenadas de la nueva base es, precisamente, la transformación lineal necesaria para reducir la dimensionalidad de datos. Además, las coordenadas en la nueva base dan la composición en factores subyacentes de los datos iniciales.

Una de las ventajas del *PCA* para reducir la dimensionalidad de un grupo de datos es que retiene aquellas características del conjunto de datos que contribuyen más a su varianza, manteniendo un orden de bajo nivel de los componentes principales e ignorando los de alto nivel. El objetivo es que esos componentes de bajo orden a veces contienen el aspecto más importante de esa información.

8.1.5. Máquinas de Vectores de Soporte

Las *máquinas de Vectores Soporte* (en inglés, *Support Vector Machines* o *SVMs*) son un conjunto de algoritmos de aprendizaje supervisado desarrollados por *Vladimir Vapnik* y su equipo en los laboratorios *AT&T*.

Estos métodos están propiamente relacionados con problemas de clasificación y regresión. Dado un conjunto de ejemplos de entrenamiento o muestras, podemos etiquetar las clases y entrenar una *SVM* para construir un modelo que prediga la clase de una nueva muestra. Intuitivamente, una *SVM* es un modelo que representa a los puntos de muestra en el espacio, separando las clases por un espacio lo más amplio posible. Cuando las nuevas muestras se ponen en correspondencia con dicho modelo, en función de su proximidad pueden ser clasificadas a una u otra clase.

Más formalmente, una *SVM* construye un hiperplano o conjunto de hiperplanos en un espacio de dimensionalidad muy alta (o incluso infinita) que puede ser utilizado en problemas de clasificación o regresión. Una buena separación entre las clases permitirá una clasificación correcta.

Dado un conjunto de puntos, subconjunto de un conjunto mayor (espacio), en el que cada uno de ellos pertenece a una de dos posibles categorías, un algoritmo basado en *SVM* construye un modelo capaz de predecir si un punto nuevo pertenece a una categoría o a la otra.

Como en la mayoría de los métodos de clasificación supervisada, los datos de entrada son vistos como un vector p-dimensional. La *SVM* busca un hiperplano que separe de forma óptima a los puntos de una clase de la de otra, que eventualmente han podido ser previamente proyectados a un espacio de dimensionalidad superior. En ese concepto de “separación óptima” es donde reside la característica fundamental de las *SVM*: este tipo de algoritmos buscan el hiperplano que tenga la máxima distancia, o margen, con los puntos que estén más cerca de él mismo. Por eso, también a veces se conoce a las *SVM* como *clasificadores de margen máximo*. De esta forma, los puntos del vector que sean etiquetados con una categoría estarán a un lado del hiperplano y los casos que se encuentren en la otra categoría estarán al otro lado.

En la literatura de las *SVM*, se llama *atributo* a la variable predictora y *característica* a un atributo transformado que es usado para definir el hiperplano. La elección de la representación más adecuada del universo estudiado se realiza mediante un proceso denominado *selección de características*. Al vector formado por los puntos más cercanos al hiperplano se le llama *vector de soporte*.

A continuación se muestra un ejemplo bidimensional. La representación de los datos a clasificar se realiza en el plano x-y. El algoritmo *SVM* trata de encontrar un hiperplano unidimensional (una recta) que una a las variables predictoras y constituya el límite que define si un elemento de entrada pertenece a una categoría o a la otra. Existe un número infinito de posibles hiperplanos que realicen la clasificación pero, ¿cuál es el mejor y cómo lo definimos? La mejor solución es aquella que permita un margen máximo entre los elementos de las dos categorías.

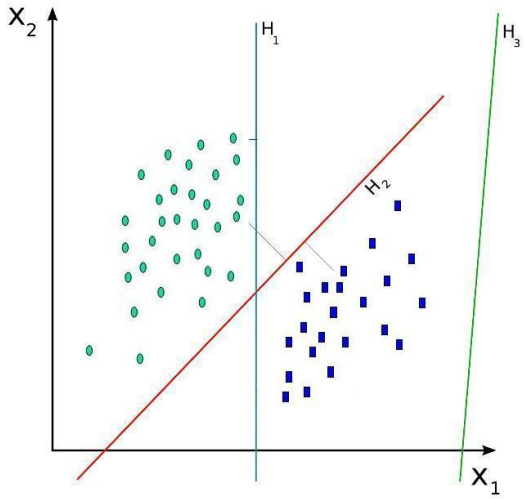

Fig. 8.6. Ejemplo de clasificación SVM.

Idealmente, el modelo basado en *SVM* debería producir un hiperplano que separe completamente los datos del universo estudiado en dos categorías. Sin embargo, una separación perfecta no siempre es posible y, si lo es, el resultado del modelo no puede ser generalizado para otros datos. Esto se conoce como *sobreajuste* (*overfitting*). Con el fin de permitir cierta flexibilidad, las *SVM* manejan un parámetro *C* que controla la compensación entre errores de entrenamiento y los márgenes rígidos, creando así un margen blando (*soft margin*) que permita algunos errores en la clasificación, a la vez que los penaliza.

8.2. OpenCV

OpenCV es una librería de visión por computador de código abierto disponible en http://SourceForge.net/projects/opencvlibrary. Está escrito en C y C++ y corre bajo Linux, Windows y Mac OS X. Hay un desarrollo activo de las interfaces para *Python*, *Ruby*, *Matlab* y otros lenguajes.

OpenCV ha sido diseñado para la eficiencia computacional y con un fuerte enfoque en aplicaciones de tiempo real. Está escrito en C optimizado y puede aprovechar los procesadores multinúcleo.

Uno de los objetivos de *OpenCV* es proporcionar una infraestructura de visión por computador fácil de utilizar, que ayuda a crear rápidamente aplicaciones de visión bastante sofisticadas. La librería *OpenCV* contiene más de 500 funciones que abarcan muchas áreas de la visión, incluyendo el reconocimiento de productos de fábricas, imágenes médicas, seguridad, interfaz de usuario, calibración de cámara, visión estéreo y robótica. Debido a que la visión por computador y el aprendizaje automático aparecen a menudo juntos, *OpenCV* contiene también una completa *Librería de Aprendizaje Automático* o, en inglés, *Machine Learning Library* (*MLL*). Esta librería se centra en el reconocimiento de patrones estadísticos y de agrupamiento. La *MLL* es de gran utilidad para las tareas de visión que se encuentran en el núcleo de la misión de *OpenCV*, pero es lo suficientemente general como para ser utilizada para cualquier problema de aprendizaje automático.

8.3. MATLAB

MATLAB (abreviatura de MAtrix Laboratory) es un software matemático que ofrece un entorno de desarrollo integrado (*IDE*) con un lenguaje de programación propio (lenguaje *M*). Está disponible para las plataformas *Unix*, *Windows* y *Apple Mac OS X*.

Entre sus prestaciones básicas se hallan:

- Manipulación de matrices.
- Representación de datos y funciones.
- Implementación de algoritmos.
- Creación de interfaces de usuario (*GUI*).
- Comunicación con programas en otros lenguajes y con otros dispositivos hardware.

El paquete *MATLAB* dispone de dos herramientas adicionales que expanden sus prestaciones, a saber, *Simulink* (plataforma de simulación multidominio) y *GUIDE* (editor de interfaces de usuario). Además, se pueden ampliar las capacidades de *MATLAB* con los *toolboxes*; y las de *Simulink* con los *blocksets*.

Es un software muy utilizado en universidades y centros de investigación y desarrollo. En los últimos años, ha aumentado el número de prestaciones, como la de programar directamente procesadores digitales de señal o crear código *VHDL* (lenguaje utilizado para describir circuitos digitales).

MATLAB es un programa de cálculo numérico orientado a matrices. Por lo tanto, será más eficiente si se diseñan los algoritmos en términos de matrices y vectores.

En la página oficial http://www.mathworks.es/ podemos encontrar la siguiente tabla, que nos expone ciertos requisitos necesarios para la versión *2011b*.

Operating Systems	Processors	Disk Space	RAM
32-Bit and 64-Bit MATLAB and Simulink Product Families			
Windows 7 or Service Pack 1 Windows Vista Service Pack 2 Windows XP Service Pack 3 Windows XP x64 Edition Service Pack 2 Windows Server 2008 Service Pack 2 or R2 Windows Server 2003 R2 Service Pack 2	Any Intel or AMD x86 processor supporting SSE2 instruction set*	1 GB for MATLAB only, 3–4 GB for a typical installation	1024 MB (At least 2048 MB recommended)

Fig. 8.7. Tabla con los requisites para la versión *2011b* de *MATLAB*

Capítulo 9. Desarrollo

Tras la realización del análisis de necesidades, así como de los requisitos hardware y software, se comienza con el desarrollo del software. Para ello, se realiza el diseño y planteamiento del sistema de detección y se continúa con la implementación de dicho planteamiento.

9.1. Diseño

Como se ha explicado anteriormente, se ha utilizado un esquema donde se usan de forma escalonada diferentes tipos de detectores y donde el sistema opera (en teoría) guiado por "evidencias". Así pues, se ha diferenciado dos tipos de detectores "erizo-probable" y "erizo-diadema", donde el "erizo-probable" se guía más por evidencias y, a medida que nos acercamos a dichas evidencias, el "erizo-diadema" discrimina entre erizos *Diadema* y "otros", pudiendo identificar positivos (*Diadema*) a partir de las zonas activadas con el primer detector.

Así, se podría decir que se tienen dos clasificadores, actuando el segundo como refinador de las detecciones realizadas por el primer clasificador y restringiendo la detección sólo a las muestras que interesan (*Diadema*).

El siguiente gráfico muestra la relación existente entre los distintos clasificadores, sirviendo como resumen de lo expuesto anteriormente.

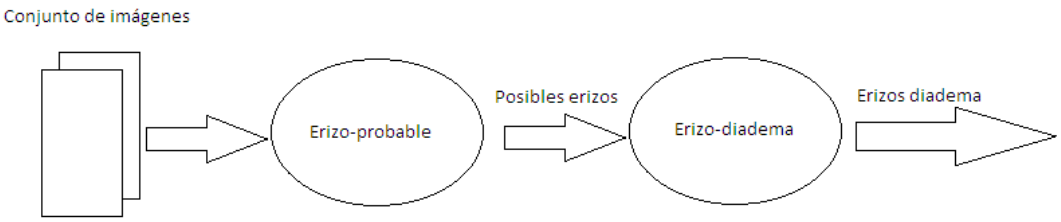

Fig. 9.1. Esquema de los clasificadores

9.2. Implementación

En este apartado se va a explicar cómo se ha implementado cada una de las funciones y los clasificadores. Las funciones correspondientes al etiquetado y el recorte de las muestras, así como los clasificadores correspondientes a *Distancia Euclídea*, *PCA* y *SVM* se han implementado en *MATLAB*. Las funciones que utilizan los detectores *HAAR* y *LBP* se han implementado en *C* y *C++*, utilizando las herramientas de *OpenCV*.

En los siguientes apartados se muestran, en primer lugar el programa utilizado para el etiquetado de las muestras, seguido de los pasos necesarios para la implementación de los detectores *HAAR* y *LBP* (creación del conjunto de entrenamiento, entrenamiento y uso). Por último se explican los programas utilizados para *Distancia Euclídea*, *SVM* y *PCA*. El código de cada una de estas funciones se encuentra en el anexo **Código**.

9.2.1. MarcaErizo

Ha sido necesario implementar una función en *MATLAB* que nos permita "marcar" las muestras positivas que se encuentran en una imagen, ya que después se necesitará pasar esta información a las herramientas de *OpenCV* utilizadas para crear los detectores.

9.2.2. Detectores HAAR

Los detectores implementados usando el método de *Viola-Jones* se han implementado siguiendo los pasos que se detallan a continuación.

Creación del conjunto de datos de entrenamiento

Para cualquier tipo de objeto, es necesario construir dos conjuntos de imágenes independientes, cuando se crea una cascada basada en el método de *Viola-Jones*. En el anexo **Creación del conjunto de datos de entrenamiento** se detallan los pasos a seguir.

1. **Conjunto de muestras positivas:**

   En este conjunto se encuentran las imágenes que contienen muestras del objeto que se desea detectar, es decir, aquel para el que se diseña el clasificador. Este conjunto de imágenes es necesario para que se genere el conjunto de imágenes de entrenamiento en el formato esperado por quien realice efectivamente el cálculo del clasificador.

2. **Conjunto de muestras negativas:**

   De forma análoga a las muestras positivas, usamos un conjunto de imágenes sin presencia del patrón a buscar.

Entrenamiento

Una vez obtenidos y anotados los conjuntos de muestras de imágenes positivas y negativas, podemos proceder al entrenamiento. Para ello necesitamos hacer uso del comando **haartraining**, que dispone de las siguientes opciones:

- *data*: Localización del clasificador resultante.
- *vec*: Muestras positivas.
- *bg*: Fichero indicando las imágenes negativas.
- *npos* y *nneg*: Muestras positivas y negativas a usar.
- *nstages*: Etapas a calcular.
- *mode*: ALL usa las features rotadas añadidas por *Lienhart*.
- *w* y *h*: Indican el ancho y alto de las muestras.
- *sym*: Si se usa asume que el patrón es simétrico.
- *nsplit*: Número de árboles por etapa.
- *mem*: Memoria disponible en MB para el cálculo.
- *minhitrate*: Mínima tasa de acierto en positivas.
- *maxfalsealarm*: Máxima tasa de error en negativas.

Para el detector *erizo-probable* se han realizado tres derivados, variando el número de etapas, para contrastar los resultados y elegir el clasificador que más se adecuara a cada situación. Los parámetros usados han sido los siguientes:

1. **20 etapas:**

   haartraining -data ./data_HAAR_20_20x20/clasif –vec ./data_HAAR_20_20x20/samples.vec –bg ./negatives/train/train.txt –npos 10000 –nneg 20000 –nstages 20 –maxtreesplits 0 –mem 500 –mode ALL –w 20 –h 20

2. **18 etapas:**

   haartraining -data ./data_HAAR_18_20x20/clasif –vec ./data_HAAR_18_20x20/samples.vec –bg ./negatives/train/train.txt –npos 10000 –nneg 20000 –nstages 18 –maxtreesplits 0 –mem 500 –mode ALL –w 20 –h 20

3. **16 etapas:**

   haartraining -data ./data_HAAR_16_20x20/clasif –vec ./data_HAAR_16_20x20/samples.vec –bg ./negatives/train/train.txt –npos 10000 –nneg 20000 –nstages 16 –maxtreesplits 0 –mem 500 –mode ALL –w 20 –h 20

Para el detector *erizo-diadema* se han usado los siguientes parámetros:

haartraining -data ./data_HAAR_20_70x70/clasif –vec ./data_HAAR_20_70x70/samples.vec –bg ./negatives/train/train.txt –npos 10000 –nneg 20000 –nstages 20 –maxtreesplits 0 –mem 500 –mode ALL –w 70 –h 70

Test

Para probar la bondad del clasificador, se hace uso del comando **performance**:

performance –data <localización del clasificador> -info <muestras de test>

Como resumen, se muestra una gráfica conteniendo los distintos detectores implementados con el método de *Viola-Jones*.

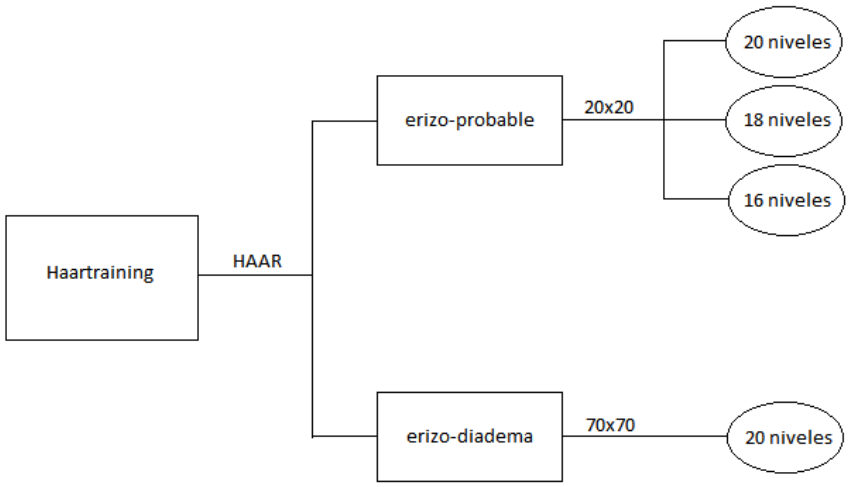

Fig. 9.2. Detectores HAAR

Uso

Para el uso de los detectores implementados mediante el método de *Viola-Jones*, se han implementado los siguientes programas.

detecta_erizo_probable_haar

Como su nombre indica, este programa se ha realizado para el uso de los detectores *erizo-probable*.

Este programa está escrito en *C*. Toma una imagen y muestra dicha imagen con las detecciones de los erizos marcadas mediante cuadrados de color rojo. Además, crea un fichero donde guarda las coordenadas de cada erizo detectado.

detecta_erizo_probable_haar_guarda

Como su nombre indica, este programa se ha realizado para el uso de los detectores *erizo-probable*.

Este programa está escrito en *C*. Toma una imagen y muestra dicha imagen con las detecciones de los erizos marcadas mediante cuadrados de color rojo. Además, guarda también dicha imagen con las detecciones.

detecta_erizo_diadema

Como su nombre indica, este programa se ha realizado para el uso del detector *erizo-diadema*.

Este programa está escrito en *C*. Toma una imagen y muestra dicha imagen con las detecciones de los erizos marcadas mediante cuadrados de color rojo. Además, crea un fichero donde guarda las coordenadas de cada erizo detectado.

Se han utilizado dos variantes de este detector, dándole los valores *1.1* y *1.2* al parámetro **scale_factor** de la función **cvHaarDetectObjects**, que indica el factor por el cual se escala la ventana de búsqueda en los análisis posteriores (1.1 significa una ventana de aumento del 10%).

detecta_erizo_diadema_guarda

Como su nombre indica, este programa se ha realizado para el uso del detector *erizo-diadema*.

Este programa está escrito en *C*. Toma una imagen y muestra dicha imagen con las detecciones de los erizos marcadas mediante cuadrados de color rojo. Además, guarda también dicha imagen con las detecciones.

Se han utilizado dos variantes de este detector, dándole los valores *1.1* y *1.2* al parámetro **scale_factor** de la función **cvHaarDetectObjects**.

Video_diadema

Como su nombre indica, este programa se ha realizado para el uso del detector *erizo-diadema*.

Este programa está escrito en *C++*. Toma un vídeo y guarda cada frame con las detecciones, para, posteriormente, unirlas para formar un vídeo, si se desea, con alguna aplicación como, por ejemplo, *Avidemux*[45].

9.2.3. Detectores LBP

Para implementar los detectores usando el método *LBP* se han seguido los pasos que se detallan a continuación.

Creación del conjunto de datos de entrenamiento

Para la creación del conjunto de datos de entrenamiento se siguen los mismos pasos seguidos para los detectores *Viola-Jones*. (Ver anexo **Creación del conjunto de datos de entrenamiento**).

Entrenamiento

Una vez obtenidos y anotados los conjuntos de muestras de imágenes positivas y negativas, podemos proceder al entrenamiento. Para ello necesitamos hacer uso del comando **opencv_traincascade**, que dispone de las siguientes opciones:

- *data*: Localización del clasificador resultante.
- *vec*: Muestras positivas.
- *bg*: Fichero indicando las imágenes negativas.
- *numPos* y *numNeg*: Muestras positivas y negativas a usar.
- *numStages*: Etapas a calcular.
- *mode*: ALL usa las features rotadas añadidas por *Lienhart*.
- *w* y *h*: Indican el ancho y alto de las muestras.
- *featureType*: Indica el método a utilizar.

Para el detector *erizo-probable* se han realizado tres derivados, con los siguientes parámetros:

1. 20 etapas:

```
opencv_traincascade -data ./data_LBP_20_20x20/clasif -vec ./data_LBP/samples.vec -bg ./negatives/train/train.txt -numPos 10000 -numNeg 20000 -numStages 20 -featureType LBP -w 20 -h 20 -mode ALL
```

2. 18 etapas:

```
opencv_traincascade -data ./data_LBP_18_20x20/clasif -vec ./data_LBP/samples.vec -bg ./negatives/train/train.txt -numPos 10000 -numNeg 20000 -numStages 18 -featureType LBP -w 20 -h 20 -mode ALL
```

3. 16 etapas:

```
opencv_traincascade -data ./data_LBP_16_20x20/clasif -vec ./data_LBP/samples.vec -bg ./negatives/train/train.txt -numPos 10000 -numNeg 20000 -numStages 16 -featureType LBP -w 20 -h 20 -mode ALL
```

Para el detector erizo-diadema se han usado los siguientes parámetros:

```
opencv_traincascade -data ./data_LBP_20_70x70/clasif -vec ./data_LBP/samples.vec -bg ./negatives/train/train.txt -numPos 10000 -numNeg 20000 -numStages 20 -featureType LBP -w 70 -h 70 -mode ALL
```

A continuación, se muestra una gráfica resumen que contiene los distintos detectores implementados con el método de *LBP*.

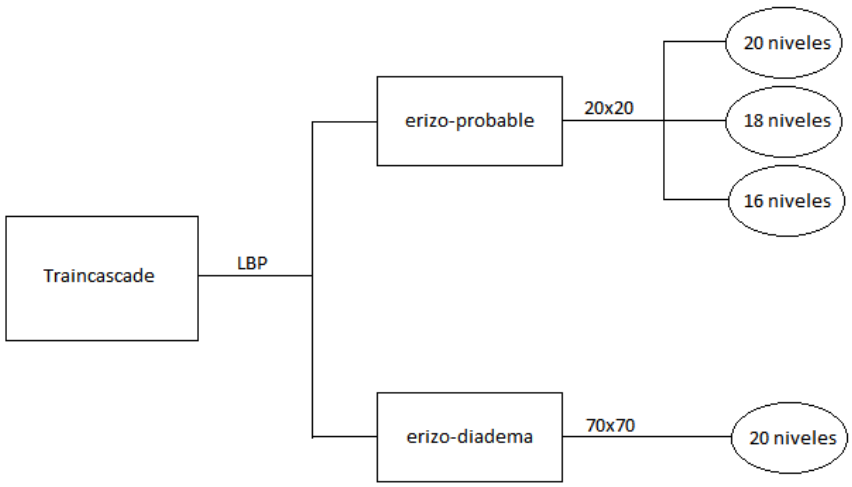

Fig. 9.3. Detectores LBP

A continuación se muestra una gráfica resumen de los detectores obtenidos con el método de *Viola-Jones* y *LBP*.

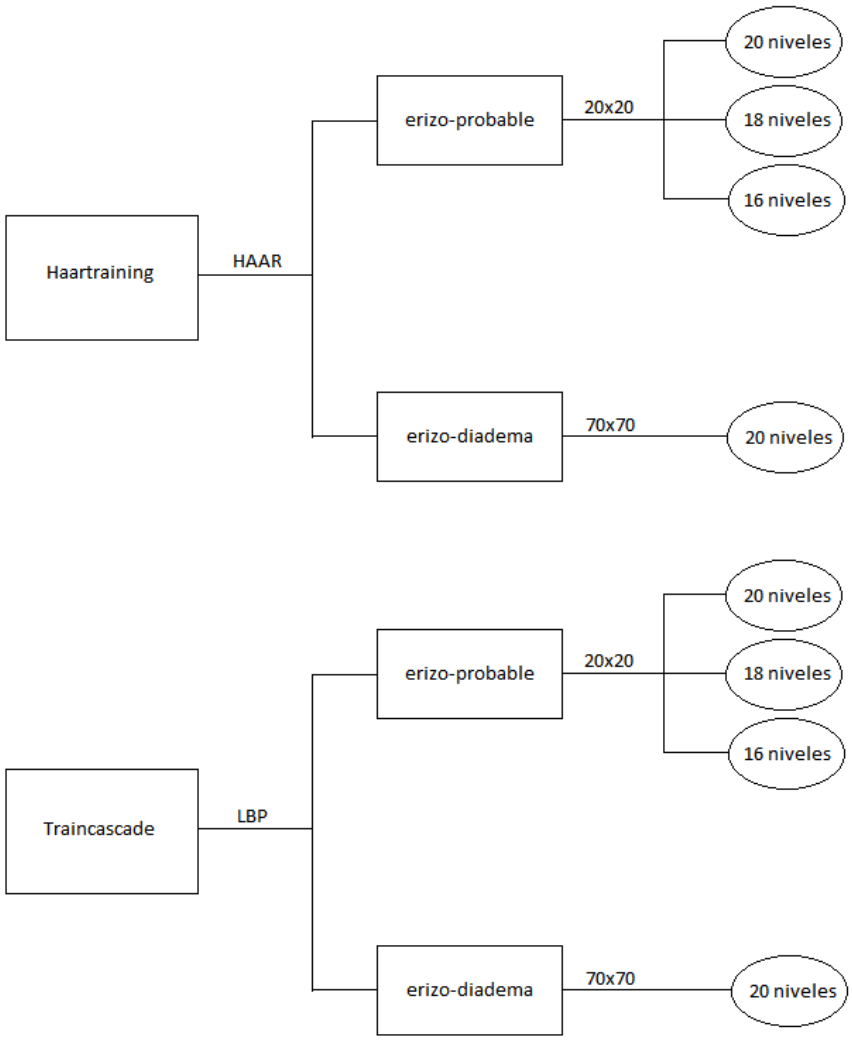

Fig. 9.4. Detectores HAAR y LBP

Uso

Para el uso de los detectores implementados mediante el método *LBP*, se han implementado los siguientes programas.

detecta_erizo_probable_lbp

Como su nombre indica, este programa se ha realizado para el uso de los detectores *erizo-probable*.

Este programa está escrito en *C++*. Toma una imagen y muestra dicha imagen con las detecciones de los erizos marcadas mediante círculos de colores escogidos aleatoriamente. Además, crea un fichero donde guarda las coordenadas de cada erizo detectado.

detecta_erizo_probable_lbp_guarda

Como su nombre indica, este programa se ha realizado para el uso de los detectores *erizo-probable*.

Este programa está escrito en *C++*. Toma una imagen y muestra dicha imagen con las detecciones de los erizos marcadas mediante círculos. Además, guarda la imagen con las detecciones.

detecta_erizo_diadema_lbp

Como su nombre indica, este programa se ha realizado para el uso de los detectores *erizo-diadema*.

Este programa está escrito en *C++*. Toma una imagen y muestra dicha imagen con las detecciones de los erizos marcadas mediante círculos. Además, crea un fichero donde guarda las coordenadas de cada erizo detectado.

detecta_erizo_diadema_lbp_guarda

Como su nombre indica, este programa se ha realizado para el uso de los detectores *erizo-diadema*.

Este programa está escrito en *C++*. Toma una imagen y muestra dicha imagen con las detecciones de los erizos marcadas mediante círculos. Además, guarda la imagen con las detecciones.

Video_probable

Como su nombre indica, este programa se ha realizado para el uso de los detectores *erizo-probable*.

Este programa está escrito en *C++*. Toma un vídeo y guarda los frames con las detecciones de los erizos marcadas mediante círculos. Además, crea un fichero donde guarda las coordenadas de cada erizo detectado.

9.2.4. SVM, PCA y Distancia Euclídea

Todas las funciones utilizadas para estos clasificadores han sido implementadas en *MATLAB* y se han seguido los pasos que se detallan a continuación.

Creación del conjunto de datos de entrenamiento y test

GeneraMuestrasRecortadasEntrenamiento

Ha sido necesario implementar una función que nos permita “extraer” las muestras positivas que se encuentran en una imagen utilizada para el entrenamiento del clasificador, ya que después se necesitarán estas muestras para los detectores *SVM*, *PCA* y de *Distancia Euclídea*.

Las muestras extraídas han de ser todas del mismo tamaño, por lo que ha sido necesario también redimensionar dichas muestras antes de guardarlas a disco.

GeneraMuestrasRecortadasTests

Ha sido necesario implementar una función que nos permita "extraer" las muestras positivas que se encuentran en una imagen utilizada para las pruebas del clasificador.

Las muestras extraídas han de ser todas del mismo tamaño, por lo que ha sido necesario también redimensionar dichas muestras antes de guardarlas a disco.

Entrenamiento y test

Una vez obtenidos y anotados los conjuntos de muestras de imágenes positivas, negativas y de test, podemos proceder al entrenamiento y test.

DemoPCASVM

Esta es la función principal de los detectores *PCA*, *SVM* y de *Distancia Euclídea*.

LeeMuestras

Esta función es llamada por la función **DemoPCASVM** y, a partir de la ruta de las imágenes, las carga componiendo una matriz donde cada columna es una imagen.

PruebaRendimiento

Esta función es llamada por la función **DemoPCASVM** y es la que se encarga de llamar a las funciones necesarias para realizar todos los cálculos, mostrando también el resultado de las detecciones.

GeneraPCA

Esta función es llamada por la función **PruebaRendimiento** y, a partir de la ruta de las imágenes de entrenamiento, crea la matriz donde cada columna es una imagen de entreno y calcula su PCA, así como la proyección de las imágenes de entreno.

A continuación se muestra un gráfico resumen del clasificador *SVM*.

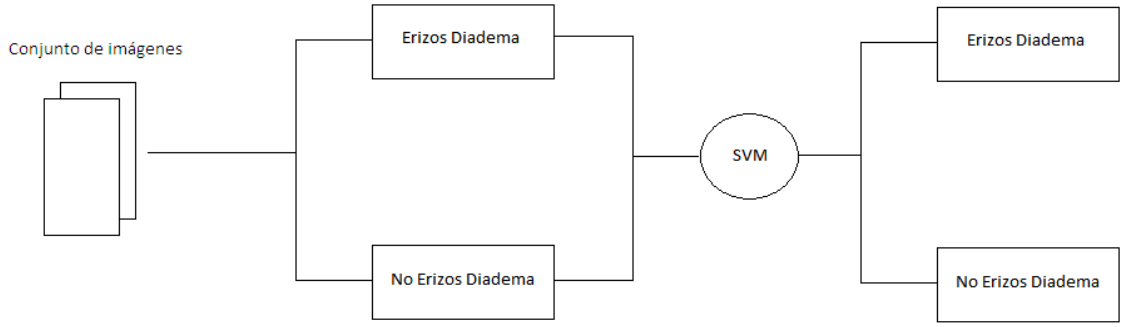

Fig. 9.5. Clasificador SVM

Con respecto a las dos categorías de imágenes, en la categoría “Erizos Diadema”, se encontrarían las muestras positivas y la categoría “No Erizos Diadema”, englobaría a las muestras negativas, tanto erizos de otro tipo, como “no erizos”.

10. Resultados y experimentos

En este capítulo se exponen los experimentos realizados y un resumen de los resultados obtenidos.

10.1. Viola-Jones y LBP

En este apartado se muestran los resultados obtenidos con los métodos de *Viola-Jones* y *LBP*.

10.1.1. Imágenes

Se muestran, a nivel ilustrativo, algunos de los resultados obtenidos en imágenes, con cada detector.

Erizo-probable

A continuación se muestran los resultados obtenidos con el detector *erizo-probable*. En algunos de los resultados se ha considerado necesario mostrar antes la imagen original, ya que al reducir su tamaño no se sabe si la mancha oscura es un erizo o una oquedad.

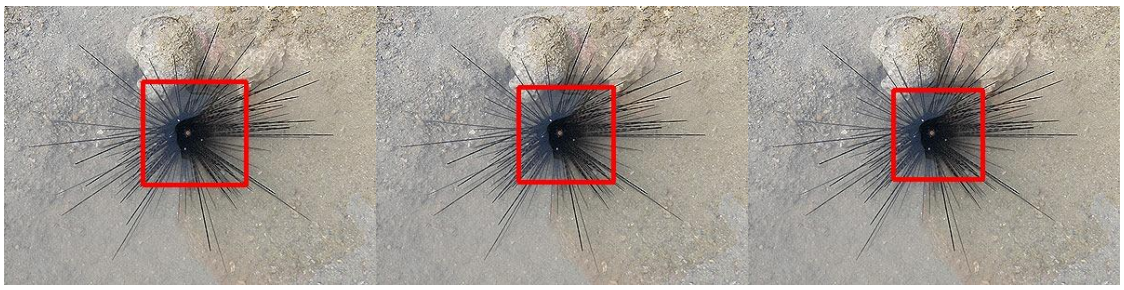

Imagen 1. Resultado HAAR (16 niveles) Imagen 1. Resultado HAAR (18 niveles) Imagen 1. Resultado HAAR (20 niveles)

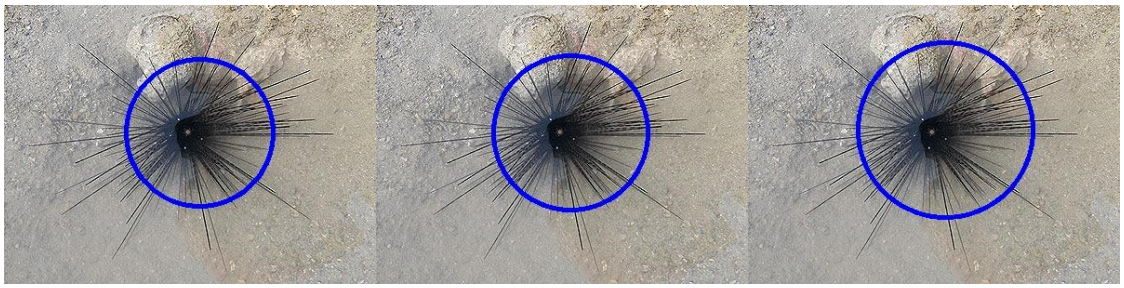

Imagen 1. Resultado LBP (16 niveles) Imagen 1. Resultado LBP (18 niveles) Imagen 1. Resultado LBP (20 niveles)

Fig. 10.1 Resultados Imagen 1

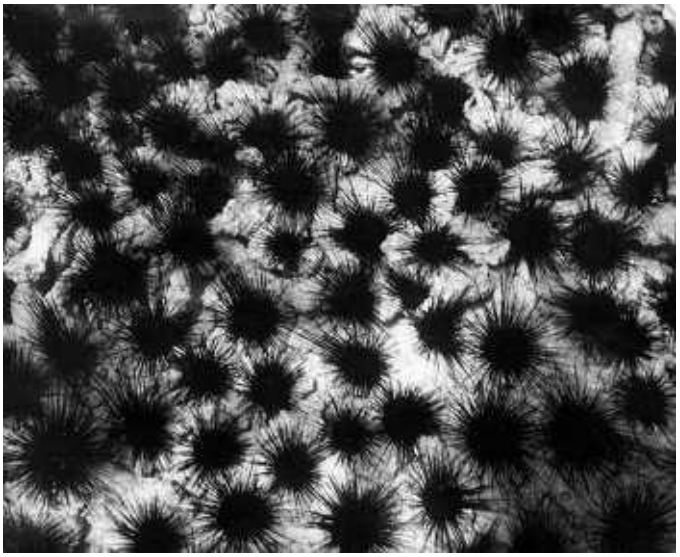

Imagen 2

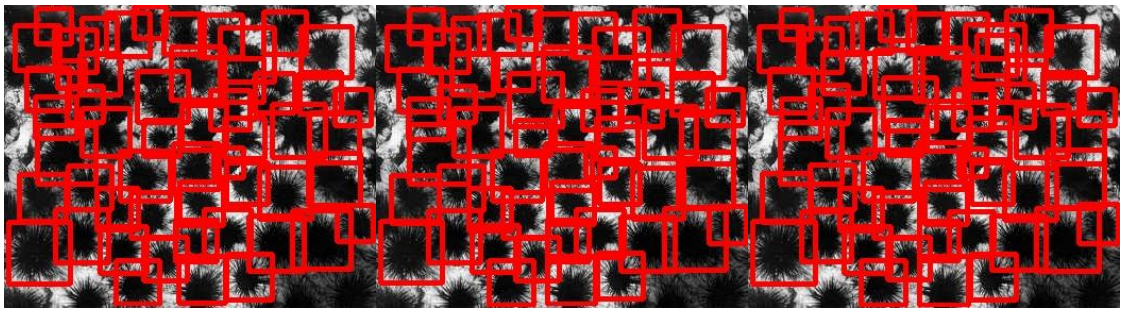

Imagen 2. Resultado HAAR (16 niveles) Imagen 2. Resultado HAAR (18 niveles) Imagen 2. Resultado HAAR (20 niveles)

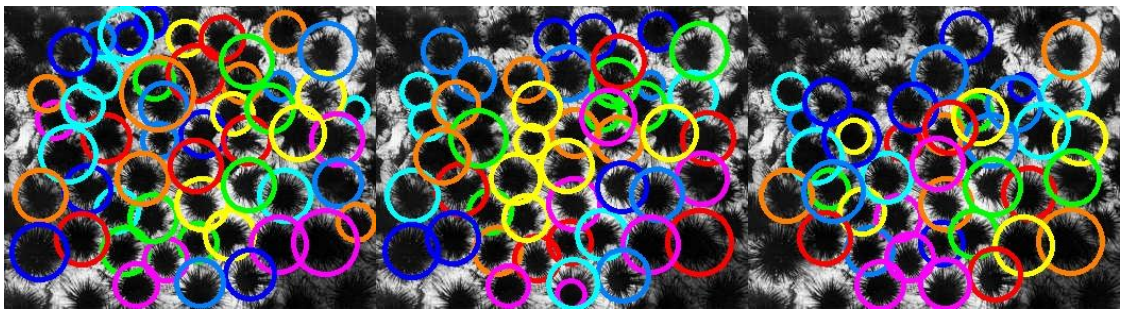

Imagen 2. Resultado LBP (16 niveles) Imagen 2. Resultado LBP (18 niveles) Imagen 2. Resultado LBP (20 niveles) (Los colores de los círculos son aleatorios)

Fig. 10.2. Resultados Imagen 2

Imagen 3. Resultado HAAR (16 niveles) Imagen 3. Resultado HAAR (18 niveles) Imagen 3. Resultado HAAR (20 niveles)

Imagen 3. Resultado LBP (16 niveles) Imagen 3. Resultado LBP (18 niveles) Imagen 3. Resultado LBP (20 niveles)

Fig 10.3. Resultados Imagen 3

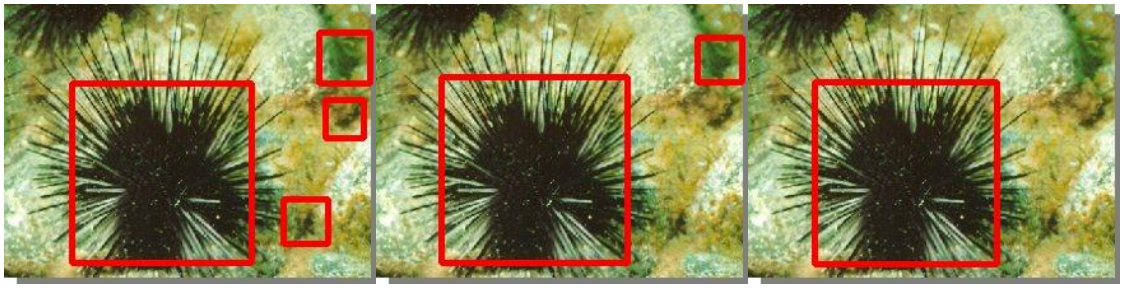

Imagen 4. Resultado HAAR (16 niveles) Imagen 4. Resultado HAAR (18 niveles) Imagen 4. Resultado HAAR (20 niveles)

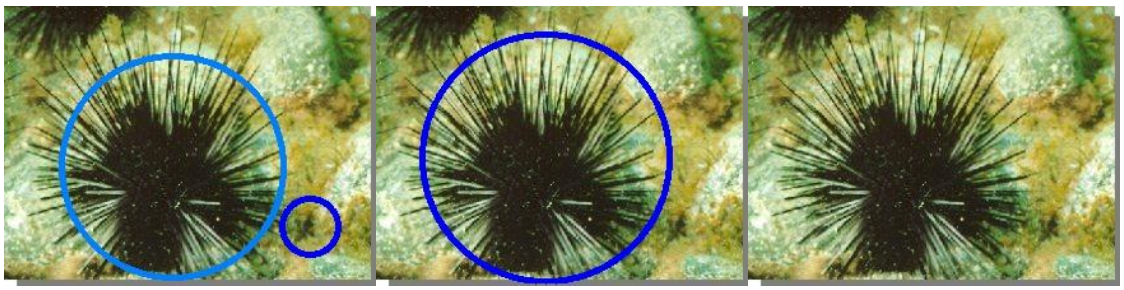

Imagen 4. Resultado LBP (16 niveles) Imagen 4. Resultado LBP (18 niveles) Imagen 4. Resultado LBP (20 niveles)

Fig. 10.4. Resultados Imagen 4

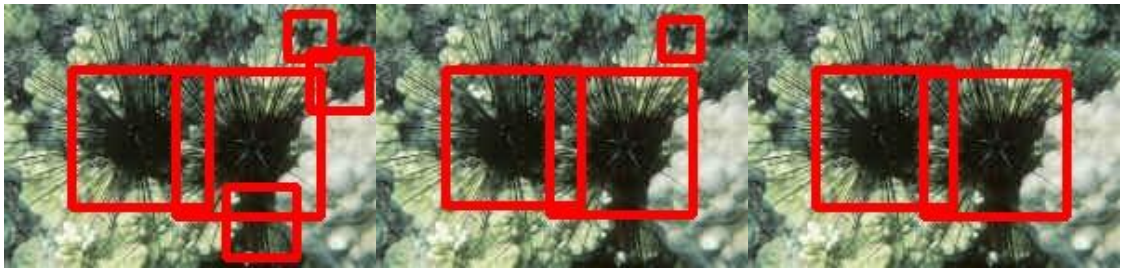

Imagen 5. Resultado HAAR (16 niveles) Imagen 5. Resultado HAAR (18 niveles) Imagen 5. Resultado HAAR (20 niveles)

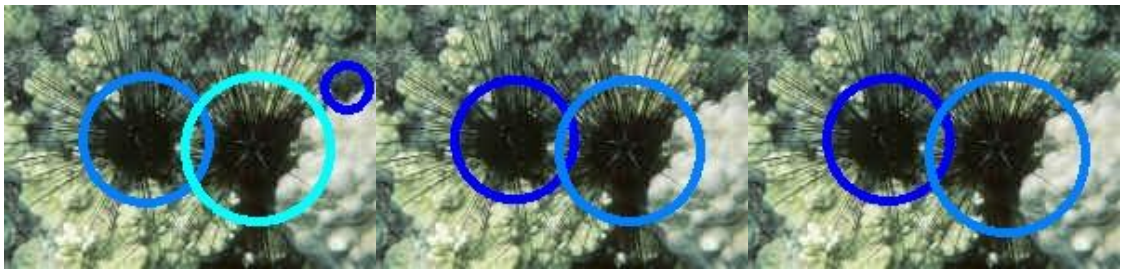

Imagen 5. Resultado LBP (16 niveles) Imagen 5. Resultado LBP (18 niveles) Imagen 5. Resultado LBP (20 niveles)

Fig. 10.5. Resultados Imagen 5

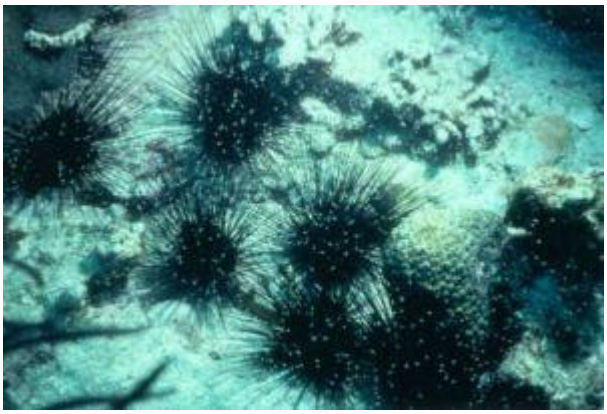

Imagen 6

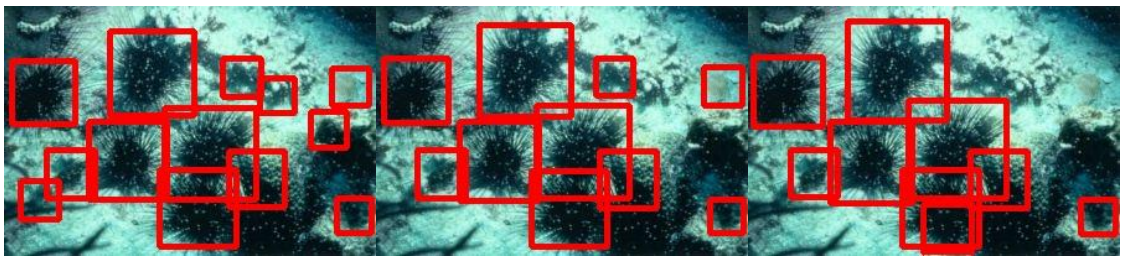

Imagen 6. Resultado HAAR (16 niveles) Imagen 6. Resultado HAAR (18 niveles) Imagen 6. Resultado HAAR (20 niveles)

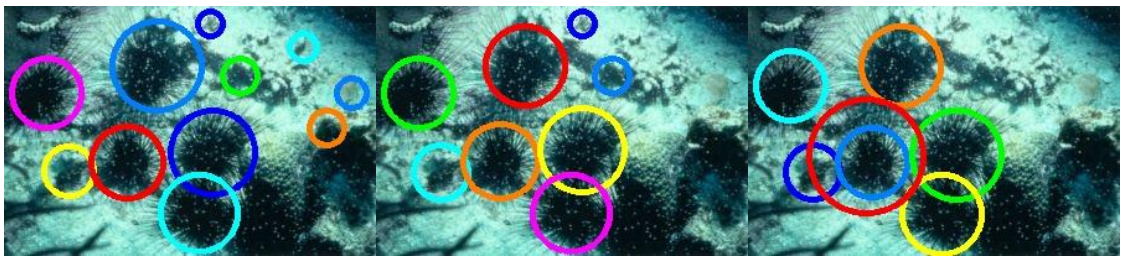

Imagen 6. Resultado LBP (16 niveles) Imagen 6. Resultado LBP (18 niveles) Imagen 6. Resultado LBP (20 niveles)

Fig 10.6. Resultados Imagen 6

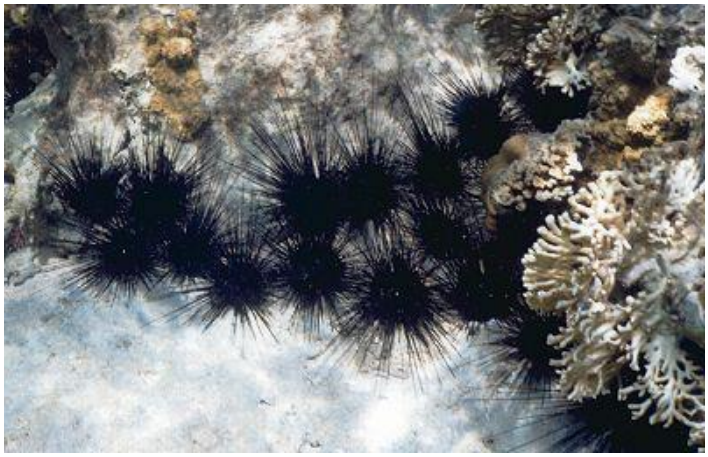

Imagen 7

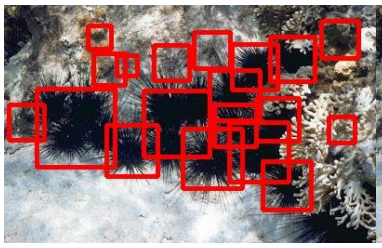

Imagen 7. Resultado HAAR (16 niveles)

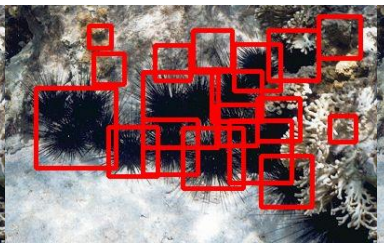

Imagen 7. Resultado HAAR (18 niveles)

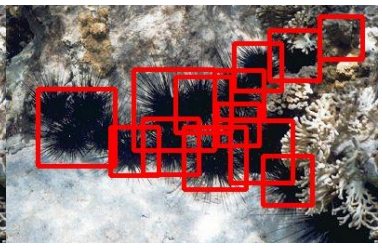

Imagen 7. Resultado HAAR (20 niveles)

Imagen 7. Resultado LBP (16 niveles)

Imagen 7. Resultado LBP (18 niveles)

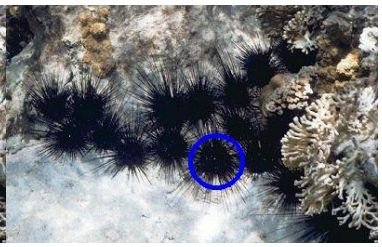

Imagen 7. Resultado LBP (20 niveles)

Fig. 10.7. Resultados Imagen 7

Imagen 8

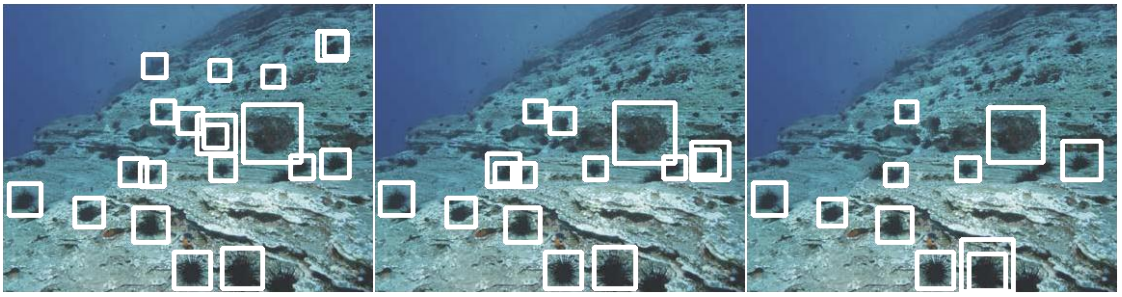

Imagen 8. Resultado HAAR (16 niveles) Imagen 8. Resultado HAAR (18 niveles) Imagen 8. Resultado HAAR (20 niveles)

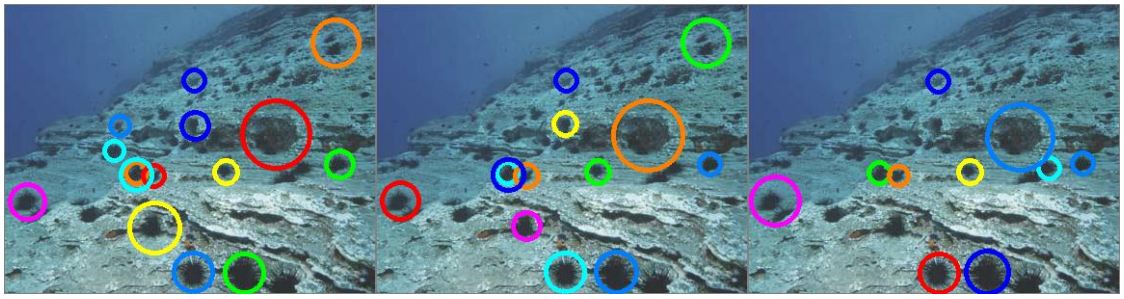

Imagen 8. Resultado LBP (16 niveles) Imagen 8. Resultado LBP (18 niveles) Imagen 8. Resultado LBP (20 niveles)

Fig. 10.8. Resultados Imagen 8

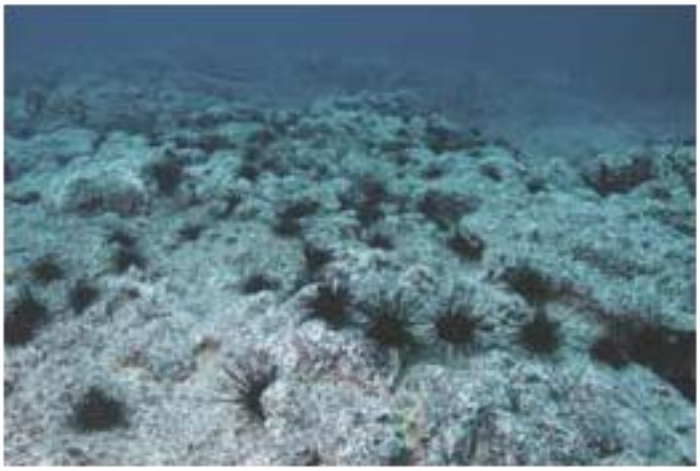

Imagen 9

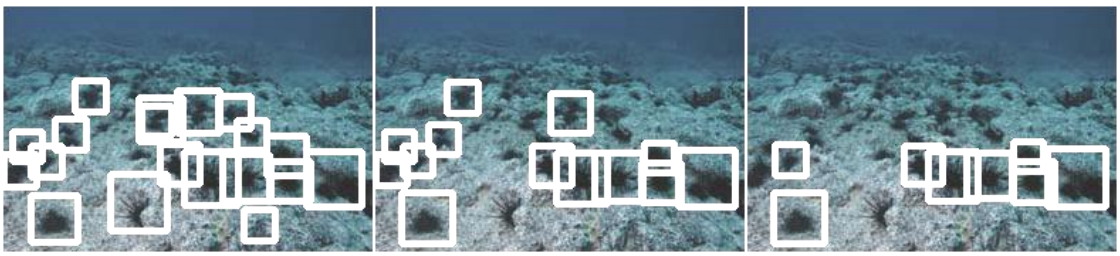

Imagen 9. Resultado HAAR (16 niveles) Imagen 9. Resultado HAAR (18 niveles) Imagen 9. Resultado HAAR (20 niveles)

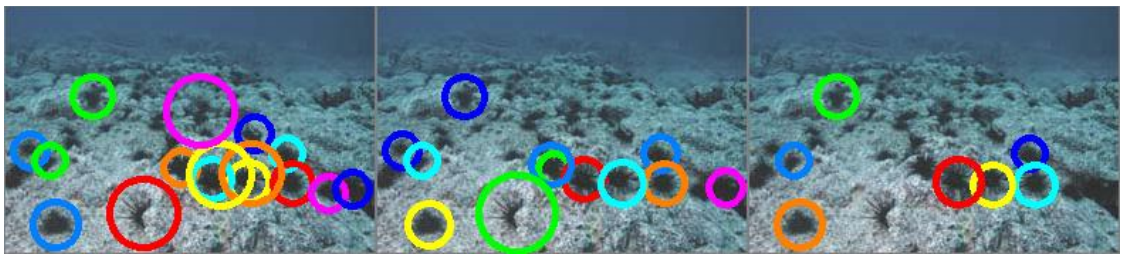

Imagen 9. Resultado LBP (16 niveles) Imagen 9. Resultado LBP (18 niveles) Imagen 9. Resultado LBP (20 niveles)

Fig. 10.9. Resultados Imagen 9

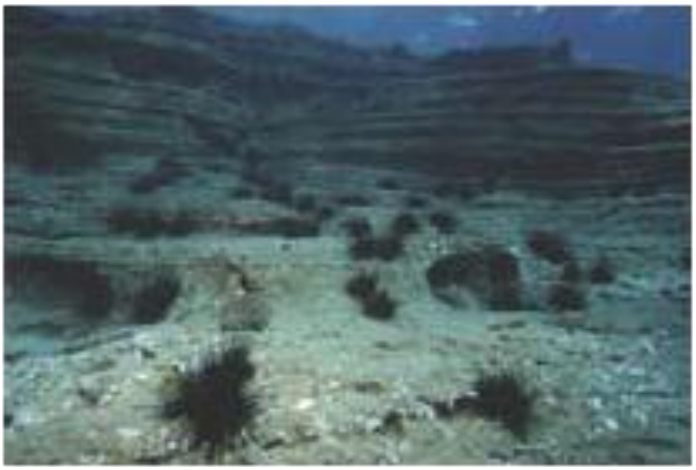

Imagen 10

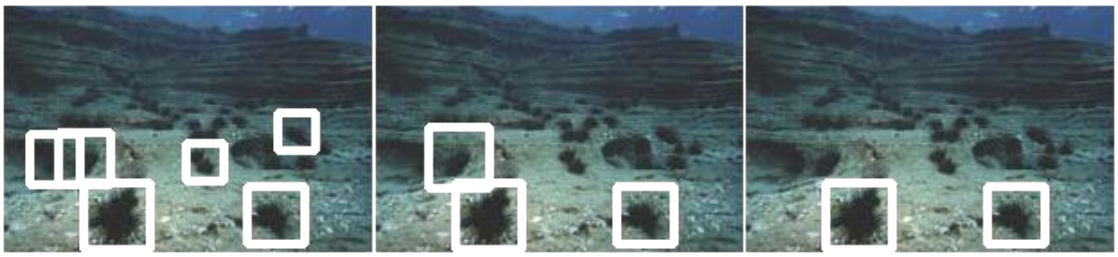

Imagen 10. Resultado HAAR (16 niveles) Imagen 10. Resultado HAAR (18 niveles) Imagen 10. Resultado HAAR (20 niveles)

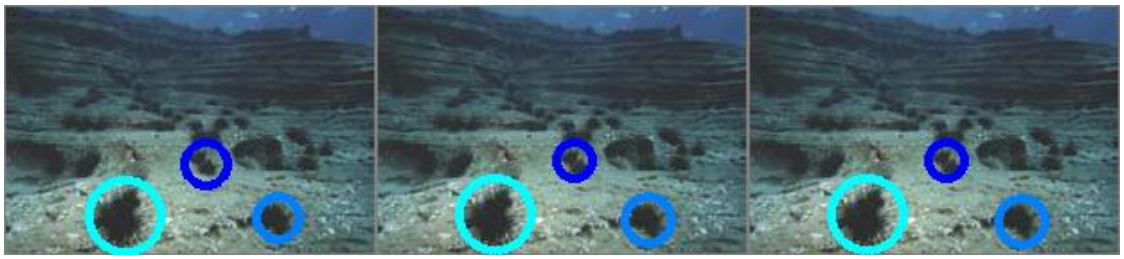

Imagen 10. Resultado LBP (16 niveles) Imagen 10. Resultado LBP (18 niveles) Imagen 10. Resultado LBP (20 niveles)

Fig. 10.10. Resultados Imagen 10

A continuación se muestran unas tablas resumen de los resultados. En dichas tablas se introducen algunos términos que se explican a continuación.

Dado un estimador para una variable estadística discreta binaria se definen dos valores asociados importantes[43]:

- **Sensibilidad**: Indica la capacidad del estimador para dar como casos positivos los casos realmente positivos. Se define como:

$$Sensibilidad = \frac{TP}{TP+FN}$$

Ec. 10.1. Sensibilidad de un estimador

Donde *TP* es *Verdaderos Positivos* (*True Positives)* y *FN* es *Falsos Negativos* (*False Negatives*).

- **Especificidad**: Indica la capacidad del estimador para dar como casos negativos los casos realmente negativos. Se define como:

$$Especificidad = \frac{TN}{TN + FP}$$

Ec. 10.2. Especificidad de un estimador

Donde *TN* es *Verdaderos Negativos* (*True Negatives*) y *FP* es *Falsos Positivos* (*False Positives*)

Foto	Tamaño	Haar (20 niveles)				LBP (20 niveles)			
		TPR	FPR	FNR	Tiempo	TPR	FPR	FNR	Tiempo
1	400x300	100%(1/1)	0%(0/1)	0%(0/1)	86,8921 ms	100% (1/1)	0% (0/1)	0% (0/1)	60,0825 ms
2	350x285	83,33% (55/66)	0%(0/66)	16,67% (11/66)	439,516 ms	63,63% (42/66)	0% (0/66)	36,36% (24/66)	88,6748 ms
3	225x257	100%(1/1)	100% (1/1)	0%(0/1)	96,3096 ms	100% (1/1)	0% (0/1)	0% (0/1)	28,4455 ms
4	292x220	100%(1/1)	0%(0/1)	0%(0/1)	111,897 ms	0%(0/1)	0% (0/1)	100% (1/1)	33,8243 ms
5	200x142	100%(2/2)	0%(0/2)	0%(0/2)	66,1027 ms	100% (2/2)	0% (0/2)	0% (0/2)	15,4394 ms
6	300x202	83,33%(5/6)	50%(3/6)	16,67% (1/6)	146,503 ms	100% (6/6)	0% (0/6)	0% (0/6)	37,1619 ms
7	400x256	75% (12/16)	6,25% (1/16)	25% (4/16)	229,176 ms	6,25% (1/16)	0% (0/16)	93,75% (15/16)	63,0466 ms
8	386x301	69,23% (9/13)	7,69% (1/13)	30,77% (4/13)	226,412 ms	69,23% (9/13)	7,69% (1/13)	30,77% (4/13)	65,4437 ms
9	262x175	50% (8/16)	0%(0/16)	50% (8/16)	93,4916 ms	43,75% (7/16)	0% (0/16)	56,25% (9/16)	28,0773 ms
10	207x139	12,5% (2/16)	0%(0/16)	87,5% (14/16)	55,321 ms	18,75% (3/16)	0% (0/16)	81,25% (13/16)	22,3283 ms
TOTAL		69,5652% (96/138)	4,3478% (6/138)	30,4348% (42/138)		52,1739% (72/138)	0,7246% (1/138)	47,8261% (66/138)	

Fig. 10.11. Tabla resultados HAAR y LBP (20 niveles)

Foto	Tamaño	Haar (18 niveles)				LBP (18 niveles)			
		TPR	FPR	FNR	Tiempo	TPR	FPR	FNR	Tiempo
1	400x300	100% (1/1)	0% (0/1)	0% (0/1)	85,2772 ms	100% (1/1)	0% (0/1)	0% (0/1)	55,5968 ms
2	350x285	81,82% (54/66)	0% (0/66)	18,18% (12/66)	470,165 ms	72,73% (48/66)	0% (0/66)	27,27% (18/66)	90,9871 ms
3	225x257	100% (1/1)	100% (1/1)	0% (0/1)	97,971 ms	100% (1/1)	0% (0/1)	0% (0/1)	38,1902 ms
4	292x220	100% (1/1)	100% (1/1)	0% (0/1)	113,011 ms	100% (1/1)	0% (0/1)	0% (0/1)	37,7351 ms
5	200x142	100% (2/2)	50% (1/2)	0% (0/2)	65,9397 ms	100% (2/2)	0% (0/2)	0% (0/2)	18,3459 ms
6	300x202	83,33% (5/6)	83,33% (5/6)	16,67% (1/6)	151,747 ms	83,33% (5/6)	50% (3/6)	16,67% (1/6)	36,597 ms
7	400x256	56,25% (9/16)	56,25% (9/16)	43,75% (7/16)	243,921 ms	37,5% (6/16)	6,25% (1/16)	62,5% (10/16)	66,9546 ms
8	386x301	84,62% (11/13)	15,38% (2/13)	15,38% (2/13)	236,645 ms	69,23% (9/13)	23,08% (3/13)	30,77% (4/13)	80,4091 ms
9	262x175	81,25% (13/16)	0% (0/16)	18,75% (3/16)	91,0349 ms	68,75% (11/16)	0% (0/16)	31,25 (5/16)	40,3449 ms
10	207x139	12,5% (2/16)	6,25% (1/16)	87,5% (14/16)	58,1537 ms	18,75% (3/16)	0% (0/16)	81,25% (13/16)	13,7172 ms
TOTAL		71,7391% (99/138)	14,4928% (20/138)	28,2609% (39/138)		63,0435% (87/138)	5,0725%(7/138)	36,9565% (51/138)	

Fig. 10.12. Tabla resultados HAAR y LBP (18 niveles)

Foto	Tamaño	HAAR (16 niveles)				LBP (16 niveles)			
		TPR	FPR	FNR	Tiempo	TPR	FPR	FNR	Tiempo
1	400x300	100% (1/1)	0% (0/1)	0% (0/1)	80,9269 ms	100% (1/1)	0% (0/1)	0% (0/1)	58,6068 ms
2	350x285	80,30% (53/66)	0% (0/66)	19,70% (13/66)	538,655 ms	80,30% (53/66)	0% (0/66)	19,70% (13/66)	107,336 ms
3	225x257	100% (1/1)	100% (1/1)	0% (0/1)	104,511 ms	100% (1/1)	150% (2/1)	0% (0/1)	36,2113 ms
4	292x220	100% (1/1)	300% (3/1)	0% (0/1)	122,007 ms	100% (1/1)	100% (1/1)	0% (0/1)	35,024 ms
5	200x142	100% (2/2)	150% (3/2)	0% (0/2)	64,1843 ms	100% (2/2)	50% (1/2)	0% (0/2)	16,2802 ms
6	300x202	83,33% (5/6)	133,33% (8/6)	16,67% (1/6)	157,386 ms	83,33% (5/6)	100% (6/6)	16,67% (1/6)	39,7706 ms
7	400x256	50% (8/16)	68,75% (11/16)	50% (8/16)	246,197 ms	62,5% (10/16)	25% (4/16)	37,5% (6/16)	65,7991 ms
8	386x301	92,3077% (12/13)	61,54% (8/13)	7,6923% (1/13)	233,363 ms	92,31% (12/13)	15,38% (2/13)	7,69% (1/13)	39,7706 ms
9	262x175	100% (16/16)	12,5% (2/16)	0% (0/16)	90,2858 ms	81,25% (13/16)	6,25% (1/16)	18,75% (3/16)	27,8921 ms
10	207x139	12,5% (2/16)	25% (4/16)	87,5% (14/16)	49,2703 ms	18,75% (3/16)	0% (0/16)	81,25% (13/16)	20,7251 ms
TOTAL		73,1884% (101/138)	28,9855 % (40/138)	26,8116% (37/138)		73,1884% (101/138)	12,3188% (17/138)	26,8116% (37/138)	

Fig. 10.13. Tabla resultados HAAR y LBP (16 niveles)

En el siguiente gráfico se muestran las curvas *ROC* correspondientes a los distintos clasificadores *erizo-probable*. En este gráfico se representa la razón de verdaderos positivos frente a la razón de falsos positivos.

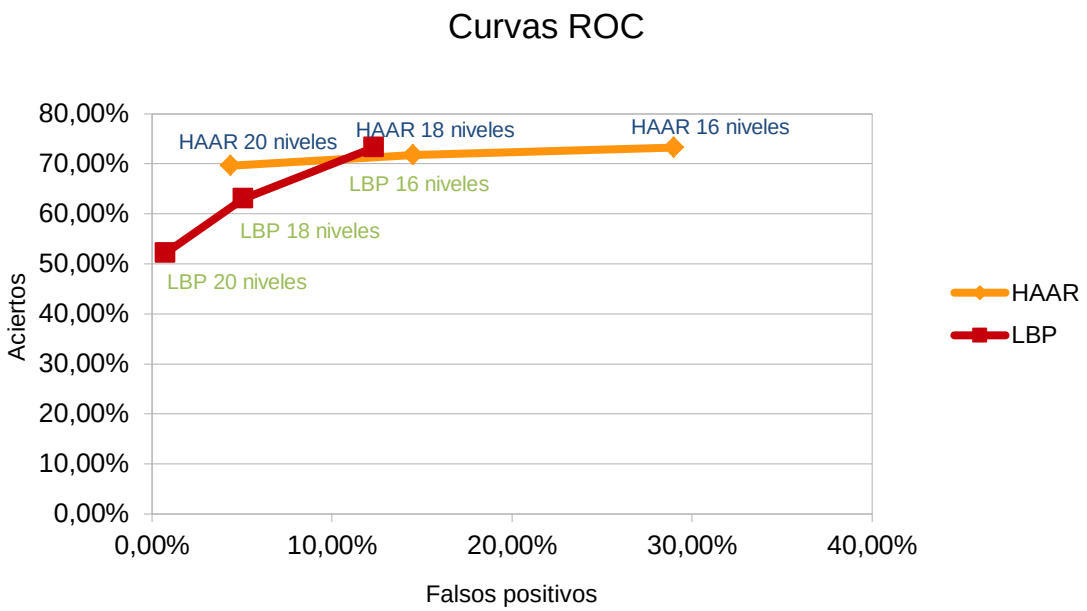

Fig. 10.14. Curvas ROC HAAR y LBP

Atendiendo a esta gráfica se puede ver que los clasificadores *HAAR* obtienen mayor tasa de aciertos para 20 y 18 niveles, pero el *LBP* obtiene menor tasa de falsos positivos. Además, si se miran las tablas anteriores, se observa que los clasificadores *LBP* son más rápidos, por lo que se ha seleccionado el clasificador *LBP* con mayor tasa de aciertos (el de 16 niveles) para la detección en vídeos, ya que, además, se requiere una detección rápida.

Erizo-diadema

A continuación se muestran los resultados obtenidos con el detector *erizo-diadema*.

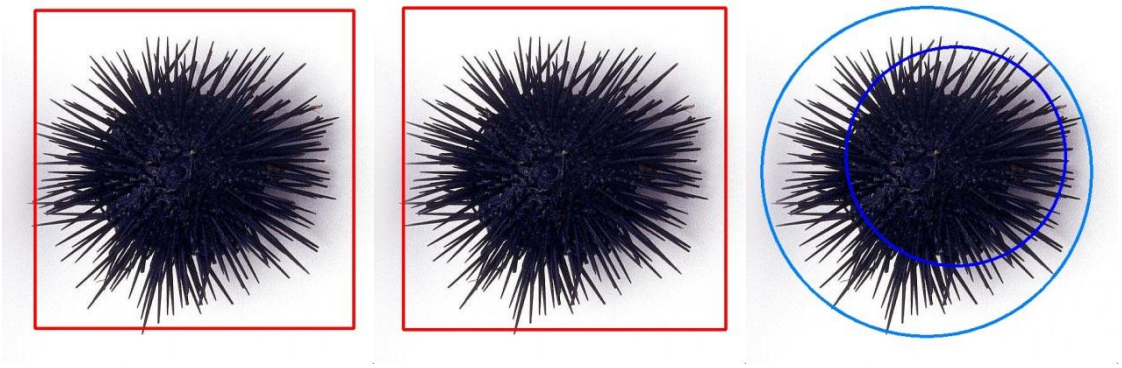

Imagen 11. Resultado HAAR (**scale_factor** 1.1) Imagen 11. Resultado HAAR (**scale_factor** 1.2) Imagen 11. Resultado LBP

Fig. 10.15. Resultados Imagen 11

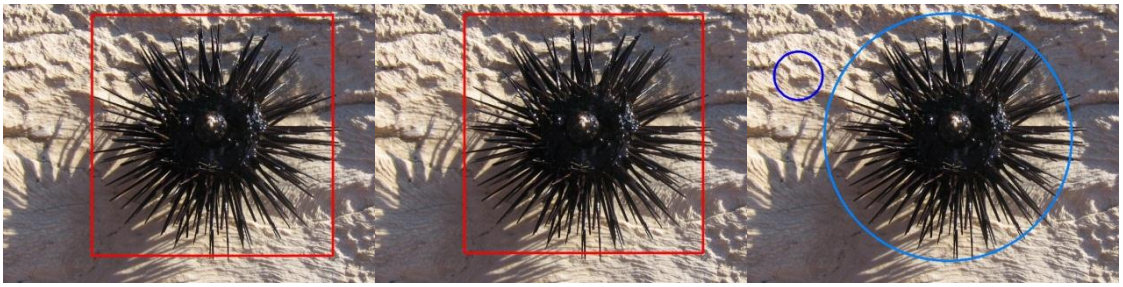

Imagen 12. Resultado HAAR (**scale_factor** 1.1) Imagen 12. Resultado HAAR (**scale_factor** 1.2) Imagen 12. Resultado LBP

Fig. 10.16. Resultados Imagen 12

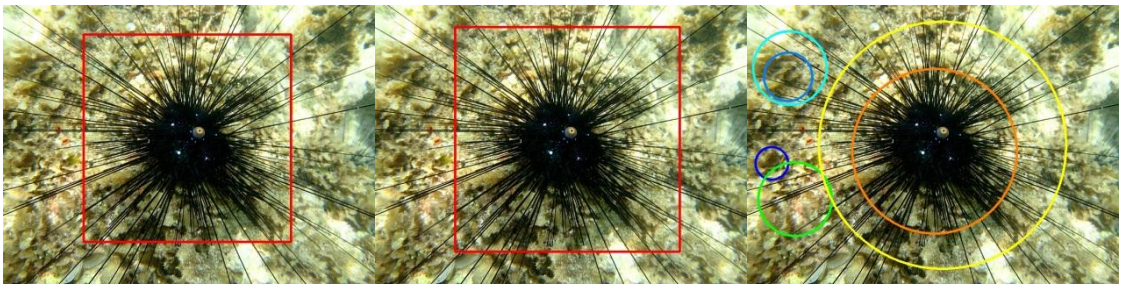

Imagen 13. Resultado HAAR (**scale_factor** 1.1) Imagen 13. Resultado HAAR (**scale_factor** 1.2) Imagen 13. Resultado LBP

Fig. 10.17. Resultados Imagen 13

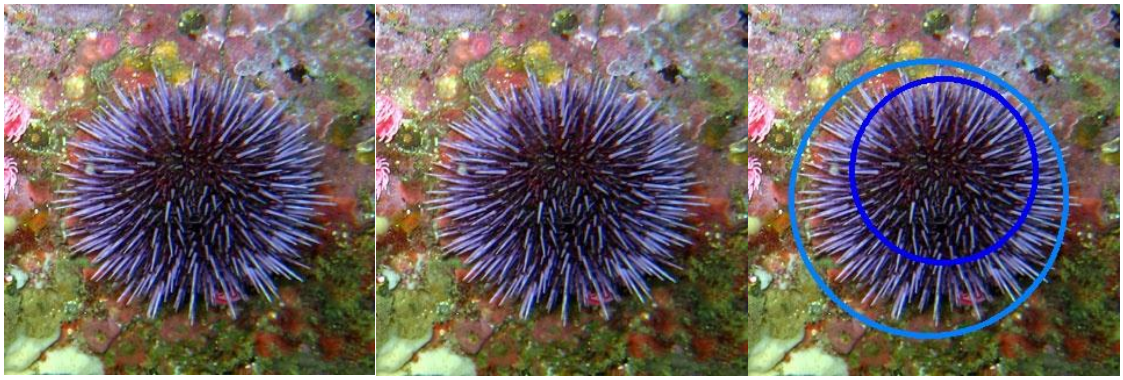

Imagen 14. Resultado HAAR (**scale_factor** 1.1) Imagen 14. Resultado HAAR (**scale_factor** 1.2) Imagen 14. Resultado LBP

Fig. 10.18. Resultados Imagen 14

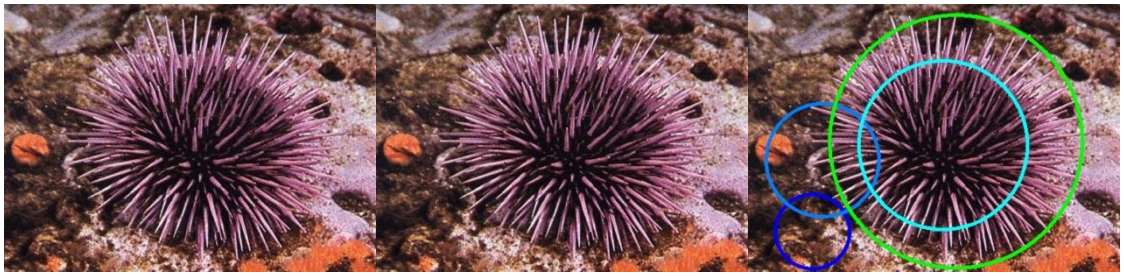

Imagen 15. Resultado HAAR (**scale_factor** 1.1) Imagen 15. Resultado HAAR (**scale_factor** 1.2) Imagen 15. Resultado LBP

Fig. 10.19. Resultados Imagen 15

Imagen 16. Resultado HAAR (**scale_factor** 1.1) Imagen 16. Resultado HAAR (**scale_factor** 1.2) Imagen 16. Resultado LBP

Fig. 10.20. Resultados Imagen 16

Imagen 17. Resultado HAAR (**scale_factor** 1.1) Imagen 17. Resultado HAAR (**scale_factor** 1.2) Imagen 17. Resultado LBP

Fig. 10.21. Resultados Imagen 17

Imagen 18. Resultado HAAR (**scale_factor** 1.1) Imagen 18. Resultado HAAR (**scale_factor** 1.2) Imagen 18. Resultado LBP

Fig. 10.22. Resultados Imagen 18

Imagen 19. Resultado HAAR (**scale_factor** 1.1) Imagen 19. Resultado HAAR (**scale_factor** 1.2) Imagen 19. Resultado LBP

Fig. 10.23. Resultados Imagen 19

Imagen 20. Resultado HAAR (**scale_factor** 1.1) Imagen 20. Resultado HAAR (**scale_factor** 1.2) Imagen 20. Resultado LBP

Fig. 10.24. Resultados Imagen 20

Imagen 21. Resultado HAAR (**scale_factor** 1.1) Imagen 21. Resultado HAAR (**scale_factor** 1.2) Imagen 21. Resultado LBP

Fig. 10.25. Resultados Imagen 21

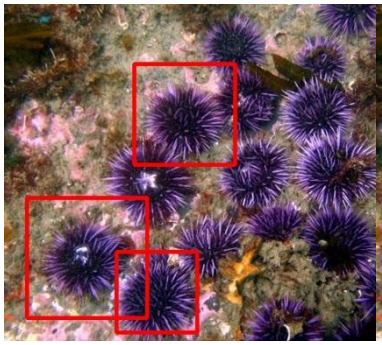
Imagen 22. Resultado HAAR (**scale_factor** 1.1)

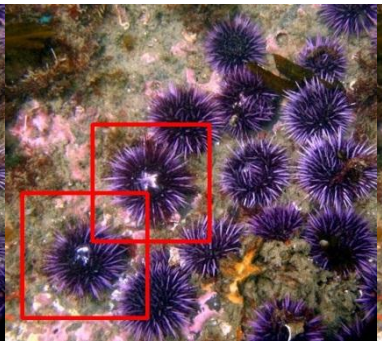
Imagen 22. Resultado HAAR (**scale_factor** 1.2)

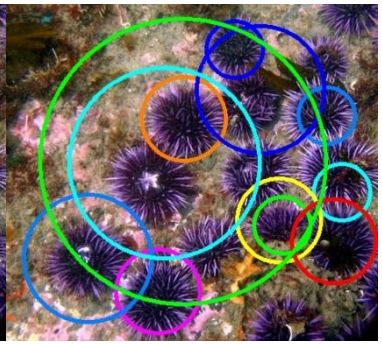
Imagen 22. Resultado LBP

Fig. 10.26. Resultados Imagen 22

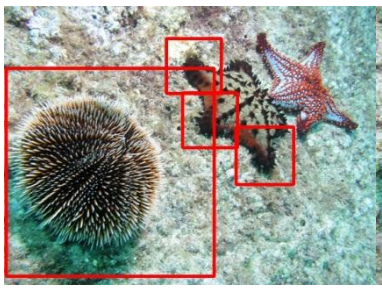
Imagen 23. Resultado HAAR (**scale_factor** 1.1)

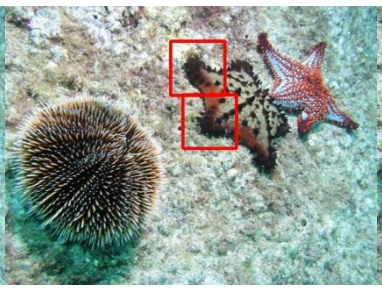
Imagen 23. Resultado HAAR (**scale_factor** 1.2)

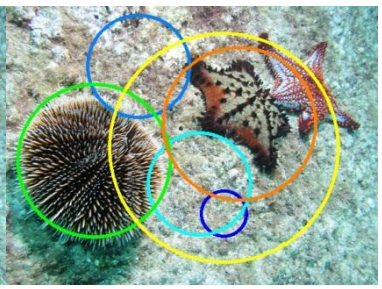
Imagen 23. Resultado LBP

Fig. 10.27. Resultados Imagen 23

A continuación se muestra una tabla resumen de los resultados de los detectores *HAAR*, obviando el *LBP* debido a que se puede ver a simple vista que los resultados no son muy favorables. Viendo los resultados se comprueba que los detectores *HAAR* son más restrictivos que el *LBP*, que apenas diferencia entre ambos tipos de erizos. Además, entre los dos detectores *HAAR*, el que tiene el parámetro **scale_factor** a 1.2 se comporta de manera más discriminatoria.

Foto	Tamaño	Haar (scale_factor = 1.1)				Haar (scale_factor = 1.2)			
		TPR	FPR	FNR	Tiempo	TPR	FPR	FNR	Tiempo
11	600x590	100% (1/1)	0%(0/1)	0%(0/1)	138,583 ms	100%(1/1)	0%(0/1)	0%(0/1)	88,5994 ms
12	800x600	100% (1/1)	0%(0/1)	0%(0/1)	244,038 ms	100%(1/1)	0%(0/1)	0%(0/1)	143,443 ms
13	800x600	100% (1/1)	0%(0/1)	0%(0/1)	270,537 ms	100%(1/1)	0%(0/1)	0%(0/1)	155,625 ms
14	320x320	100% (0/0)	0%(0/0)	0%(0/0)	27,6753 ms	100%(0/0)	0%(0/0)	0%(0/0)	18,1597 ms
15	521x375	100% (0/0)	0%(0/0)	0%(0/0)	76,6739 ms	100%(0/0)	0%(0/0)	0%(0/0)	48,4057 ms
16	410x308	100% (0/0)	0%(0/0)	0%(0/0)	35,8091 ms	100%(0/0)	0%(0/0)	0%(0/0)	21,6064 ms
17	468x315	100% (0/0)	0%(0/0)	0%(0/0)	39,2948 ms	100%(0/0)	0%(0/0)	0%(0/0)	31,4171 ms
18	432x324	100% (0/0)	0%(0/0)	0%(0/0)	43,0873 ms	100%(0/0)	0%(0/0)	0%(0/0)	26,996 ms
19	1000x717	100% (0/0)	0%(0/0)	0%(0/0)	424,153 ms	100%(0/0)	0%(0/0)	0%(0/0)	243,437 ms
20	600x547	100% (0/0)	0%(0/0)	0%(0/0)	151,379 ms	100%(0/0)	0%(0/0)	0%(0/0)	95,0447 ms
21	612x486	100% (0/0)	2/0	0%(0/0)	143,914 ms	100%(0/0)	0%(0/0)	0%(0/0)	84,1898 ms
22	461x418	100% (0/0)	3/0	0%(0/0)	86,6634 ms	100%(0/0)	2/0	0%(0/0)	54,0036 ms
23	576x432	100% (0/0)	4/0	0%(0/0)	35,8091 ms	100%(0/0)	2/0	0%(0/0)	67,1734 ms
TOTAL		100% (3/3)	300% (9/3)	0% (0/3)		100% (3/3)	133,33% (4/3)	0% (0/3)	

Fig. 10.28. Tabla Resultados HAAR (*erizo-diadema*)

Además de las pruebas anteriores, se han realizado otras pruebas que se han encontrado interesantes debido a que se encuentran a una distancia un poco lejana, son de un tamaño pequeño, se encuentran partes de erizos y además algunos erizos se superponen con otros, siendo más compleja la detección que en los casos anteriores. Estas pruebas se han considerado que ponen al límite al detector *erizo-diadema* y se usan para estudiar su comportamiento frente a estas situaciones. A continuación los resultados obtenidos.

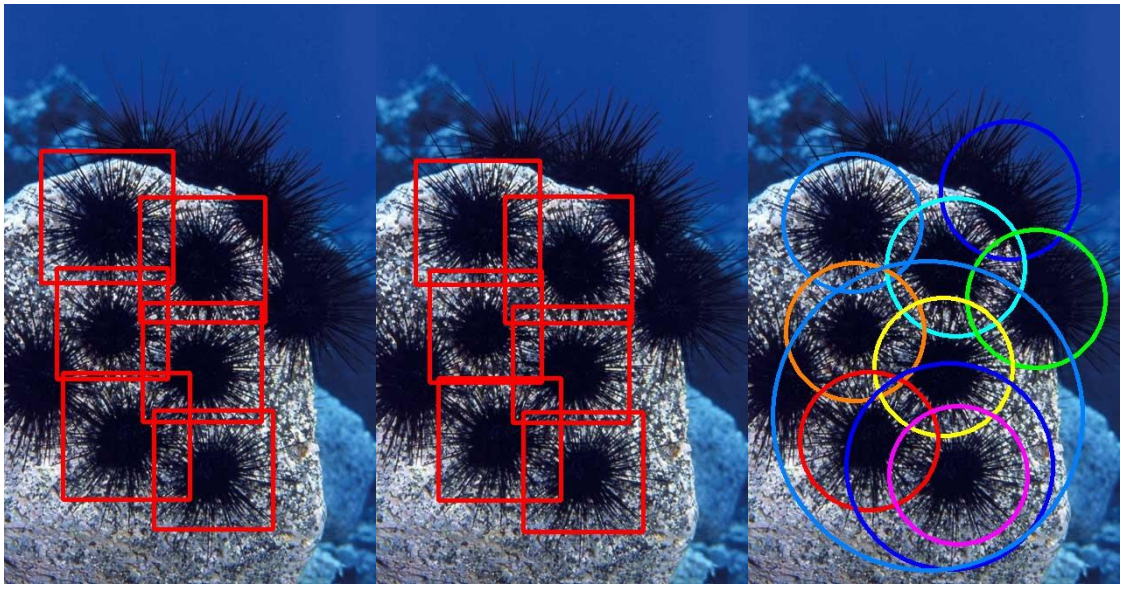

Imagen 24. Resultado HAAR (**scale_factor** 1.1) Imagen 24. Resultado HAAR (**scale_factor** 1.2) Imagen 24. Resultado LBP

Fig. 10.29. Resultados Imagen 24

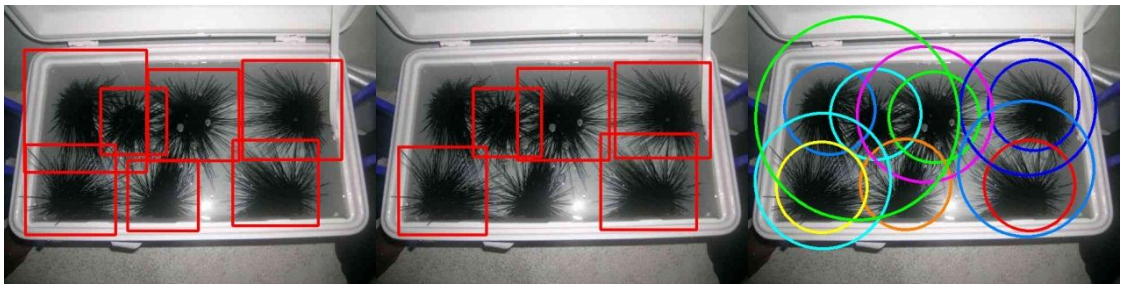

Imagen 25. Resultado HAAR (**scale_factor** 1.1) Imagen 25. Resultado HAAR (**scale_factor** 1.2) Imagen 25. Resultado LBP

Fig. 10.30. Resultados Imagen 25

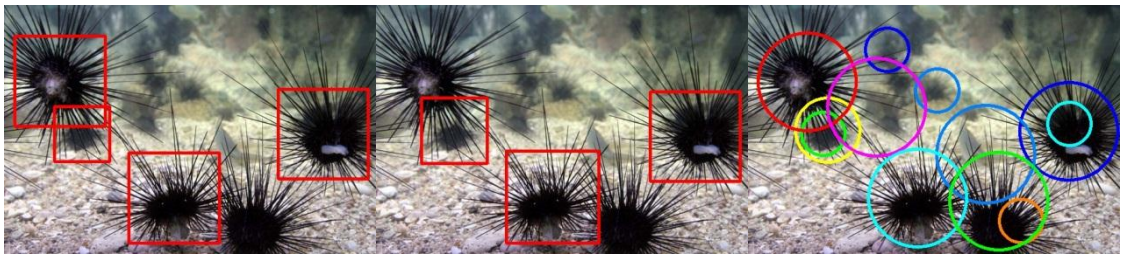

Imagen 26. Resultado HAAR (**scale_factor** 1.1) Imagen 26. Resultado HAAR (**scale_factor** 1.2) Imagen 26. Resultado LBP

Fig. 10.31. Resultados Imagen 26

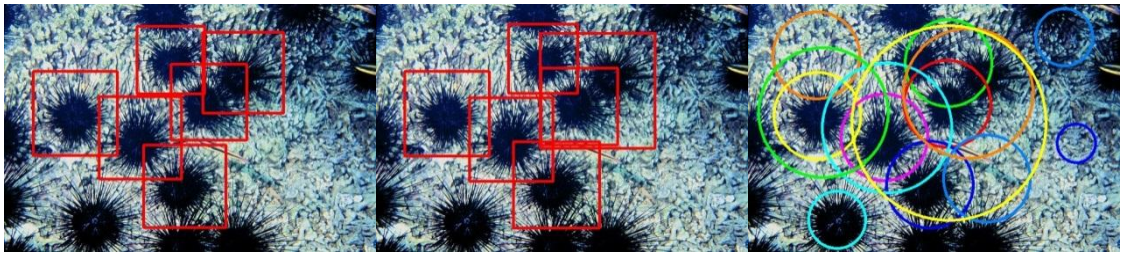

Imagen 27. Resultado HAAR (**scale_factor** 1.1) Imagen 27. Resultado HAAR (**scale_factor** 1.2) Imagen 27. Resultado LBP

Fig. 10.32. Resultados Imagen 27

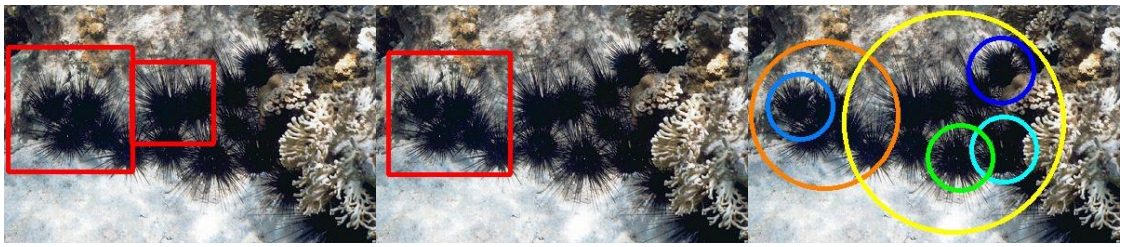

Imagen 28. Resultado HAAR (**scale_factor** 1.1) Imagen 28. Resultado HAAR (**scale_factor** 1.2) Imagen 28. Resultado LBP

Fig. 10.33. Resultados Imagen 28

10.1.2. Vídeos

A continuación se muestran algunos de los frames resultantes del uso en vídeos.

erizo-probable

A continuación se muestran los resultados obtenidos con el detector *erizo-probable*.

Vídeo 1

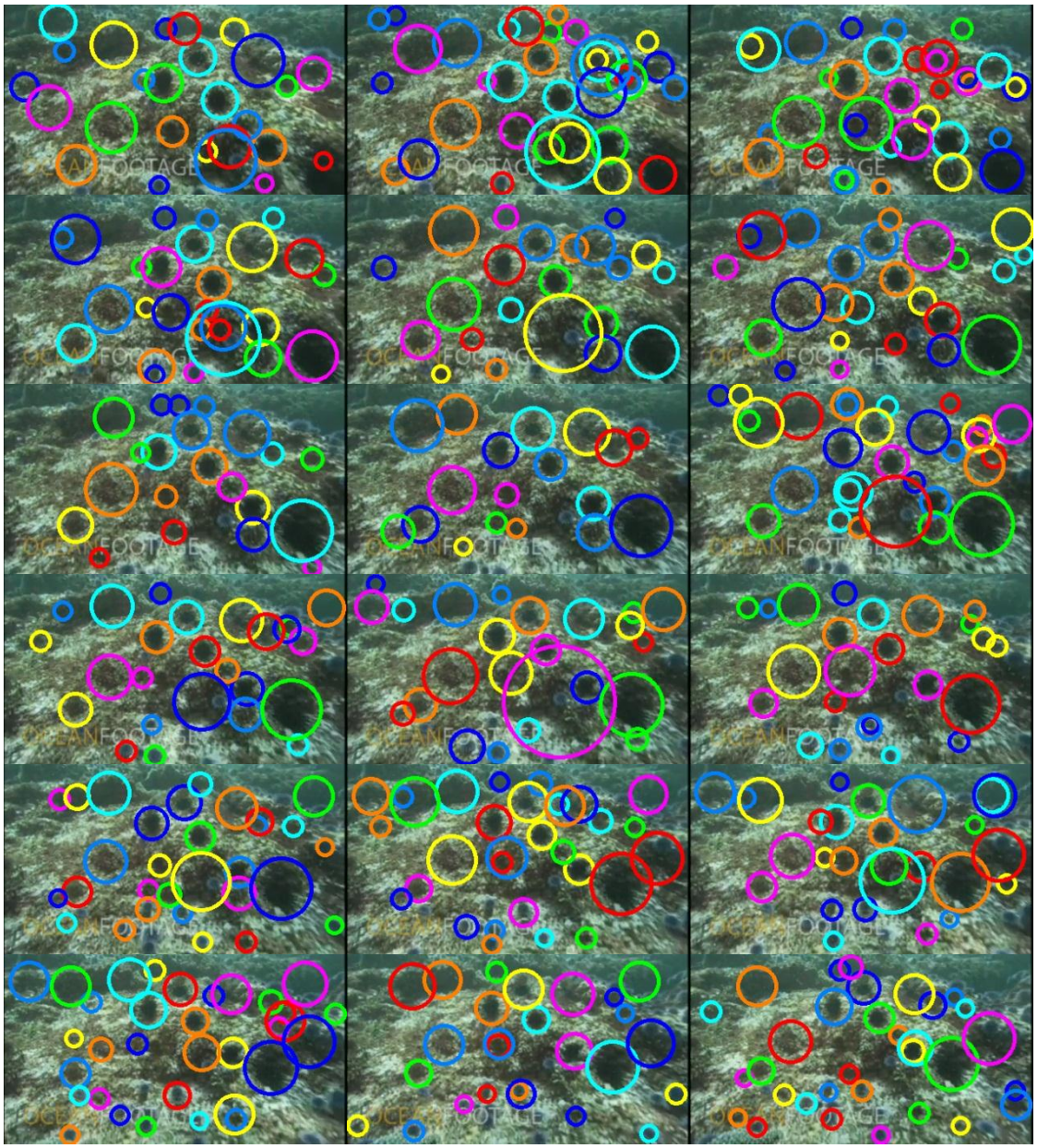

Fig. 10.34. Resultado Vídeo 1 (*LBP erizo-probable*)

Vídeo 2

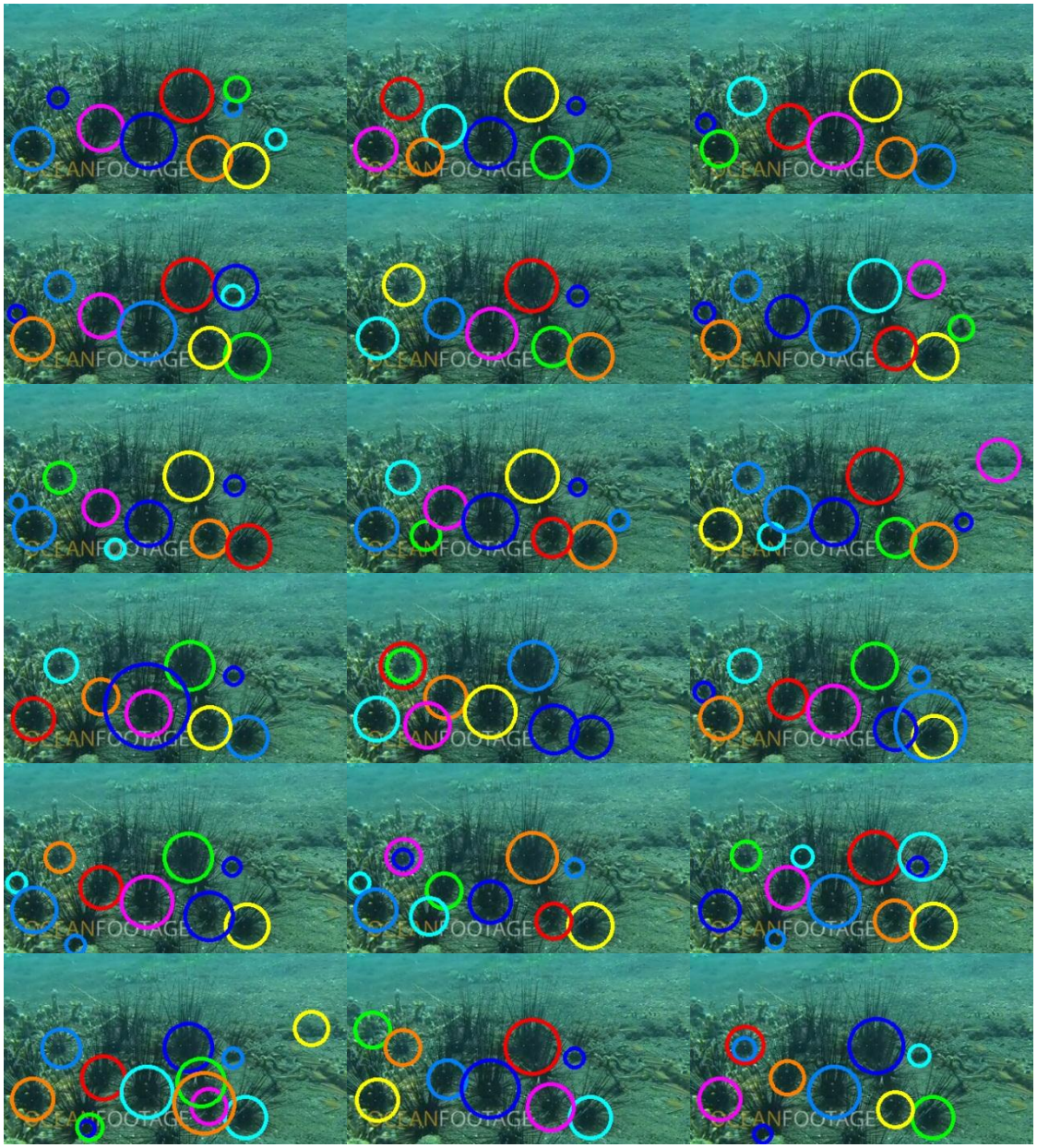

Fig. 10.35. Resultado Vídeo 2 (*LBP erizo-probable*)

Vídeo 3

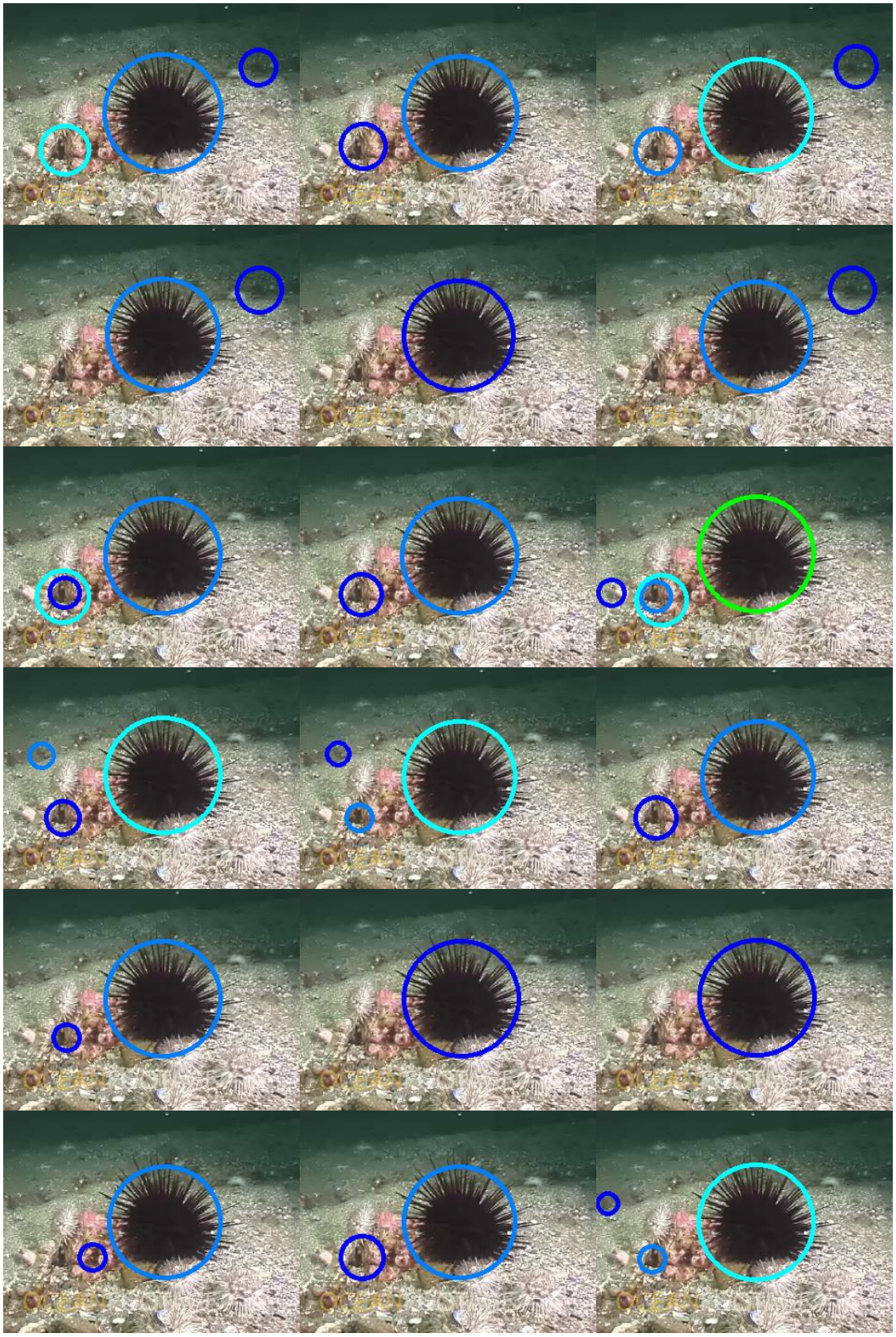

Fig. 10.36. Resultado Vídeo 3 (*LBP erizo-probable*)

Vídeo 4

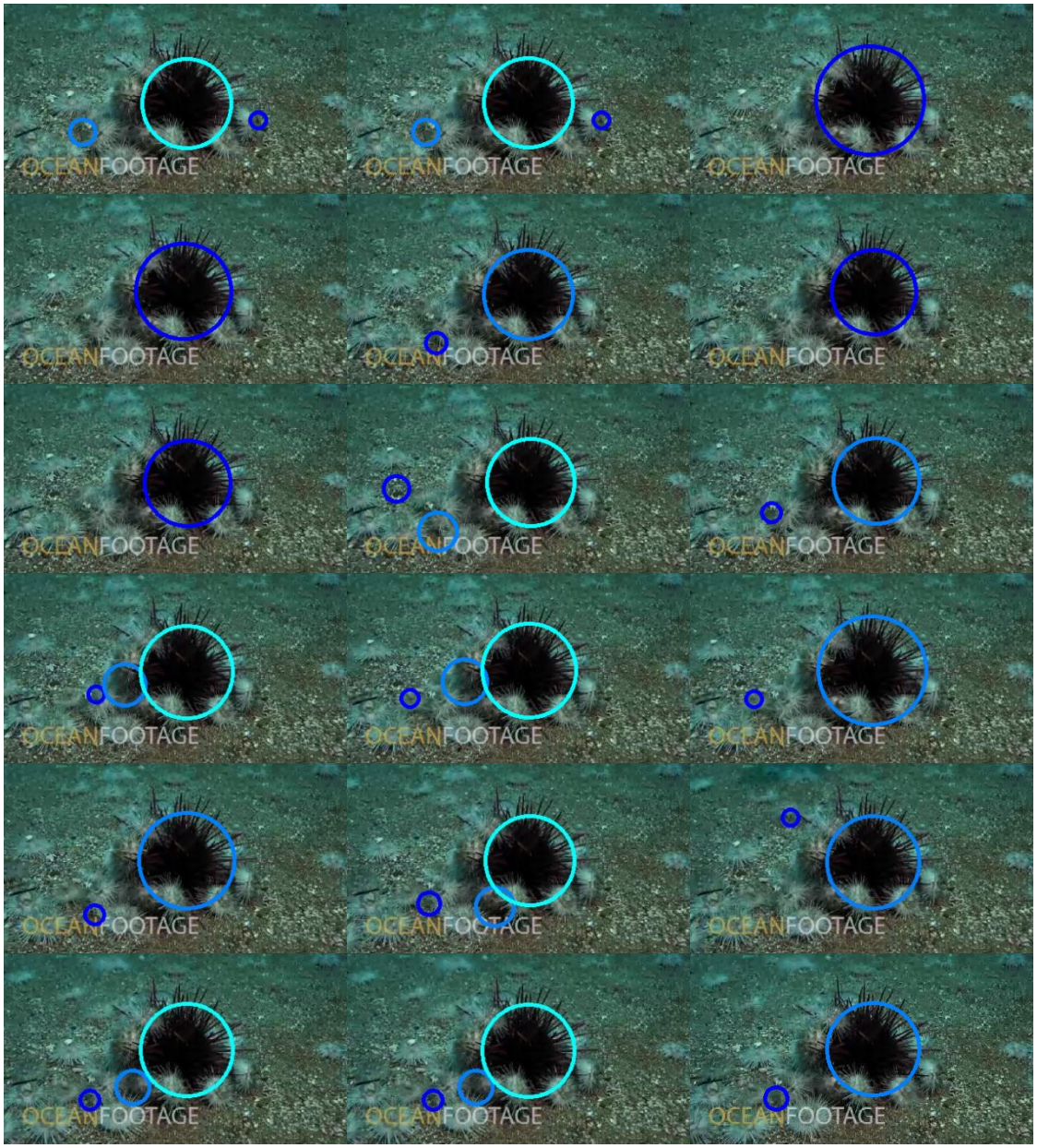

Fig. 10.37. Resultado Vídeo 4 (*LBP erizo-probable*)

Los resultados muestran que el detector *erizo-probable* en efecto marca las zonas que se parecen mucho a un erizo *Diadema*, comportándose de una manera óptima teniendo en cuenta, además, la mala calidad de algunos de los vídeos usados.

erizo-diadema

Además, se ha experimentado también con el detector *erizo-diadema* (*HAAR*), sólo como curiosidad, ya que dicho detector tarda mucho tiempo para ser usado sobre vídeos.

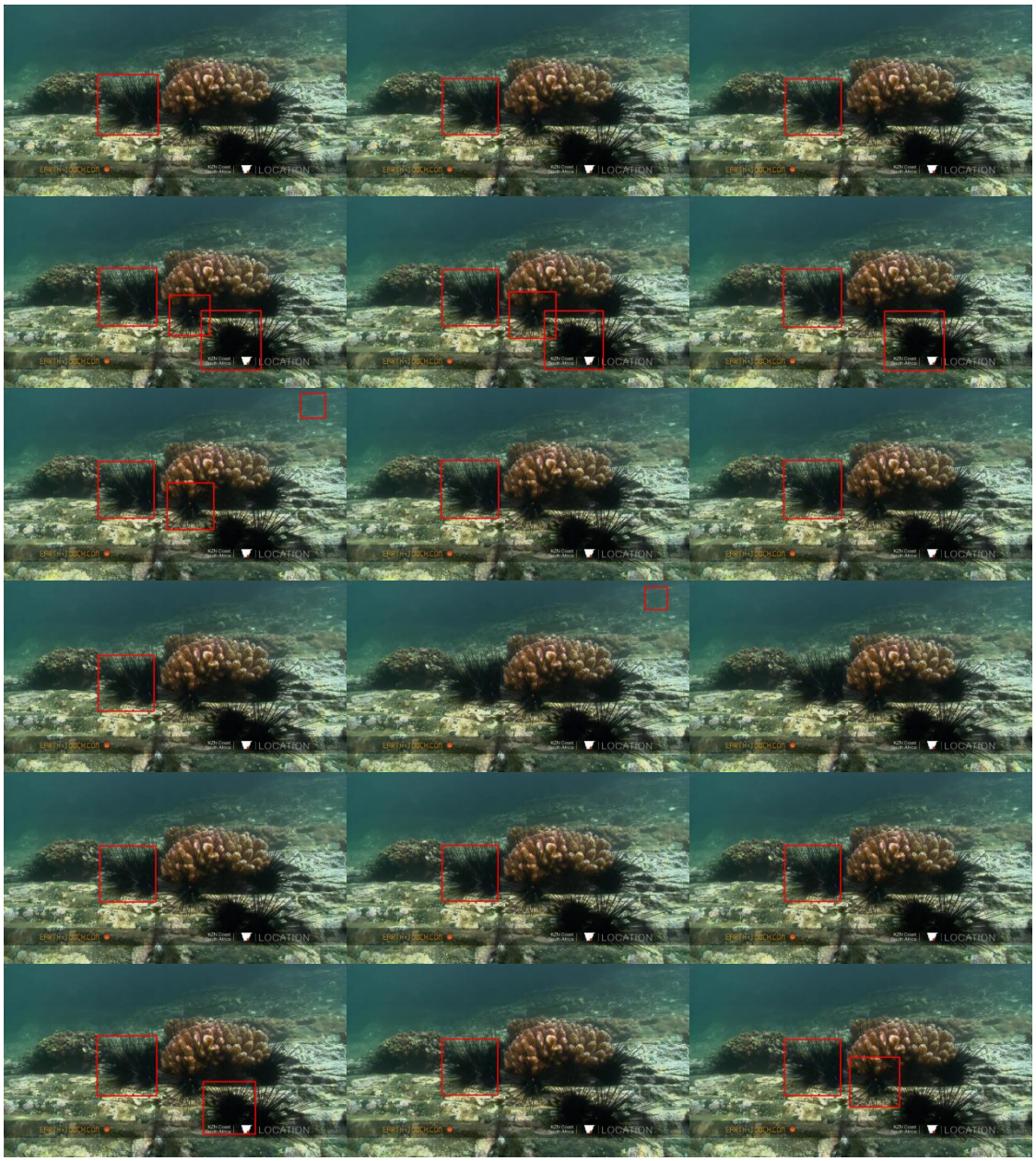

Fig. 10.38. Resultado Vídeo 5 (*HAAR erizo-diadema*)

El detector *erizo-diadema* se comprueba que no se puede utilizar en sistemas tiempo real, ya que al utilizarlo sobre el vídeo el coste computacional es muy caro, traduciéndose esto en un tiempo enorme para poder ser usado en este tipo de sistemas.

10.2. SVM, PCA y Distancia Euclídea

A continuación se muestran los resultados obtenidos con los métodos de *Distancia Euclídea*, *SVM* y *PCA*.

Primero se utilizó el conjunto de imágenes de entrenamiento como el conjunto de test, dando los siguientes resultados.

Método	Aciertos	Fallos
Píxeles + Distancia Euclídea	98.2704% (625/636)	1.7296% (11/636)
PCA + Distancia Euclídea	100% (636/636)	0% (0/636)
PCA + SVM	77.3585% (492/636)	22.6415% (144/636)

Fig. 10.39. Tabla Resultados *SVM*, *PCA* y *Distancia Euclídea* (Conjunto entrenamiento)

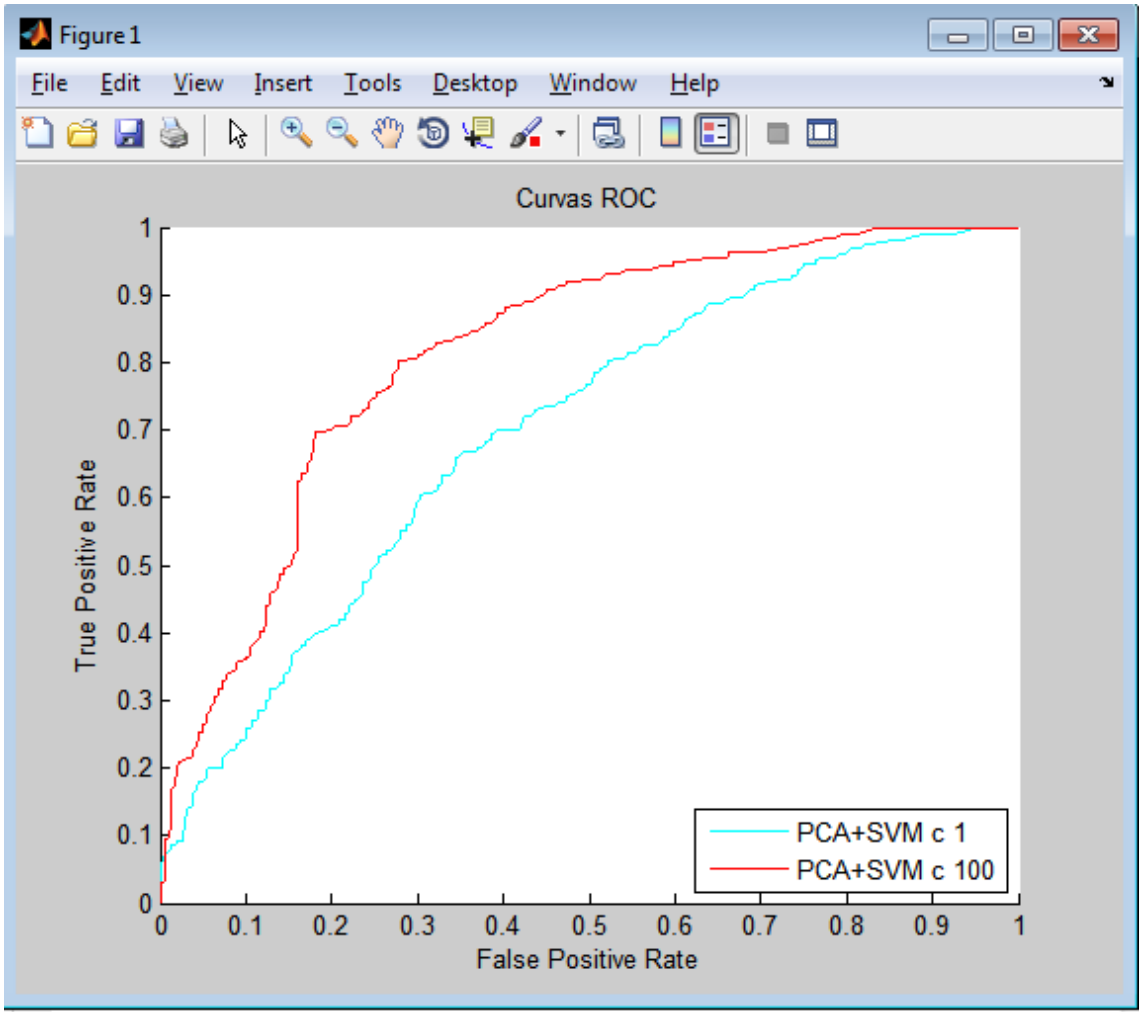

Fig. 10.40. Curvas ROC PCA+SVM (Conjunto entrenamiento)

A continuación se muestran los resultados obtenidos para el segundo conjunto de imágenes utilizadas.

Método	Aciertos	Fallos
Píxeles + Distancia Euclídea	48.8372% (21/43)	51.1628% (22/43)
PCA + Distancia Euclídea	74.4186% (32/43)	25.5814% (11/43)
PCA + SVM	65.1163% (28/43)	34.8837% (15/43)

Fig. 10.41. Tabla Resultados *SVM*, *PCA* y *Distancia Euclídea* (Segundo conjunto de imágenes)

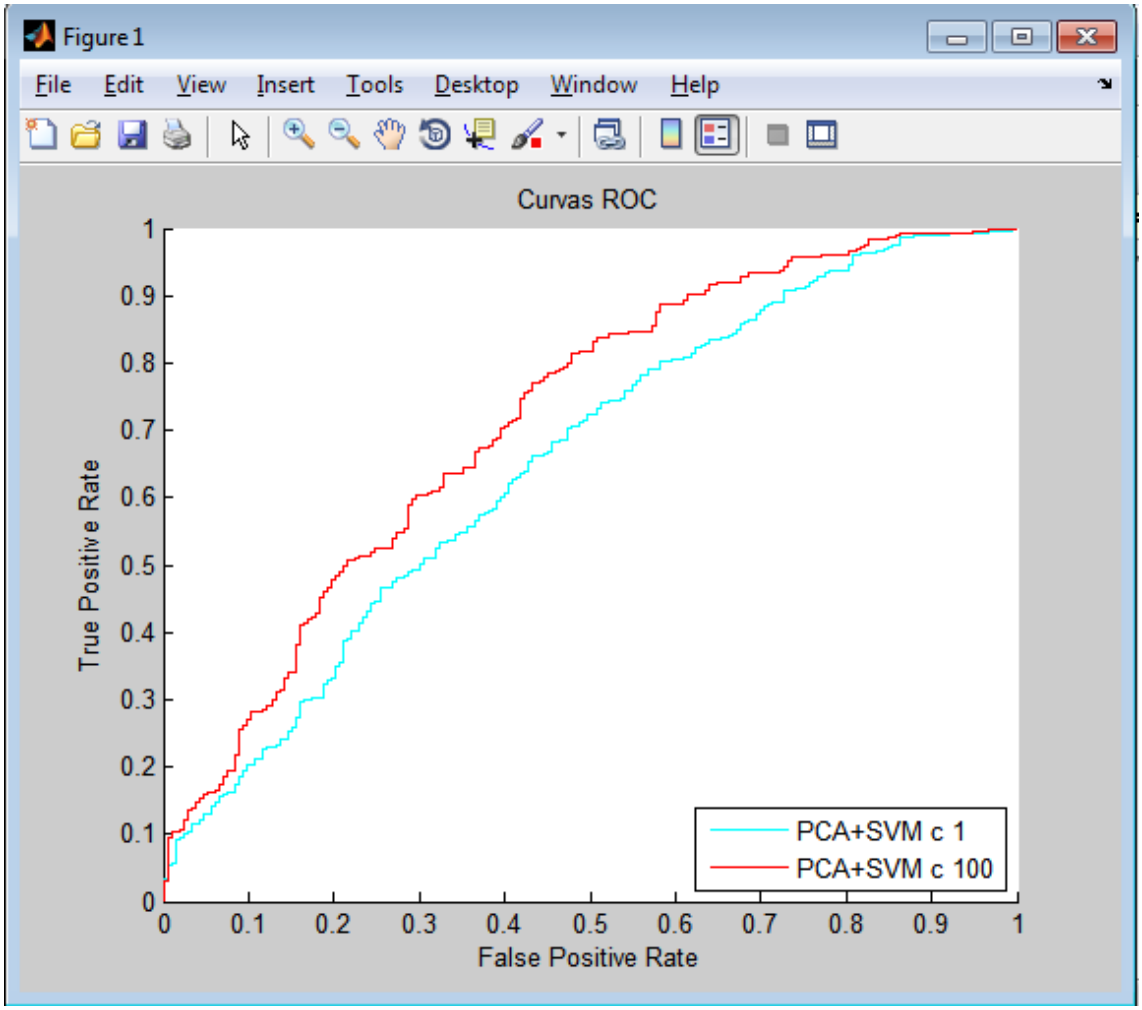

Fig. 10.42. Curvas ROC PCA+SVM (Segundo conjunto de imágenes)

Los siguientes resultados corresponden al tercer conjunto de imágenes utilizado.

Método	Aciertos	Fallos
Píxeles + Distancia Euclídea	64.1026% (25/39)	35.8974% (14/39)
PCA + Distancia Euclídea	66.6667% (26/39)	33.3333% (13/39)
PCA + SVM	61.5385% (24/39)	38.4615% (15/39)

Fig. 10.43. Tabla Resultados *SVM*, *PCA* y *Distancia Euclídea* (Tercer conjunto de imágenes)

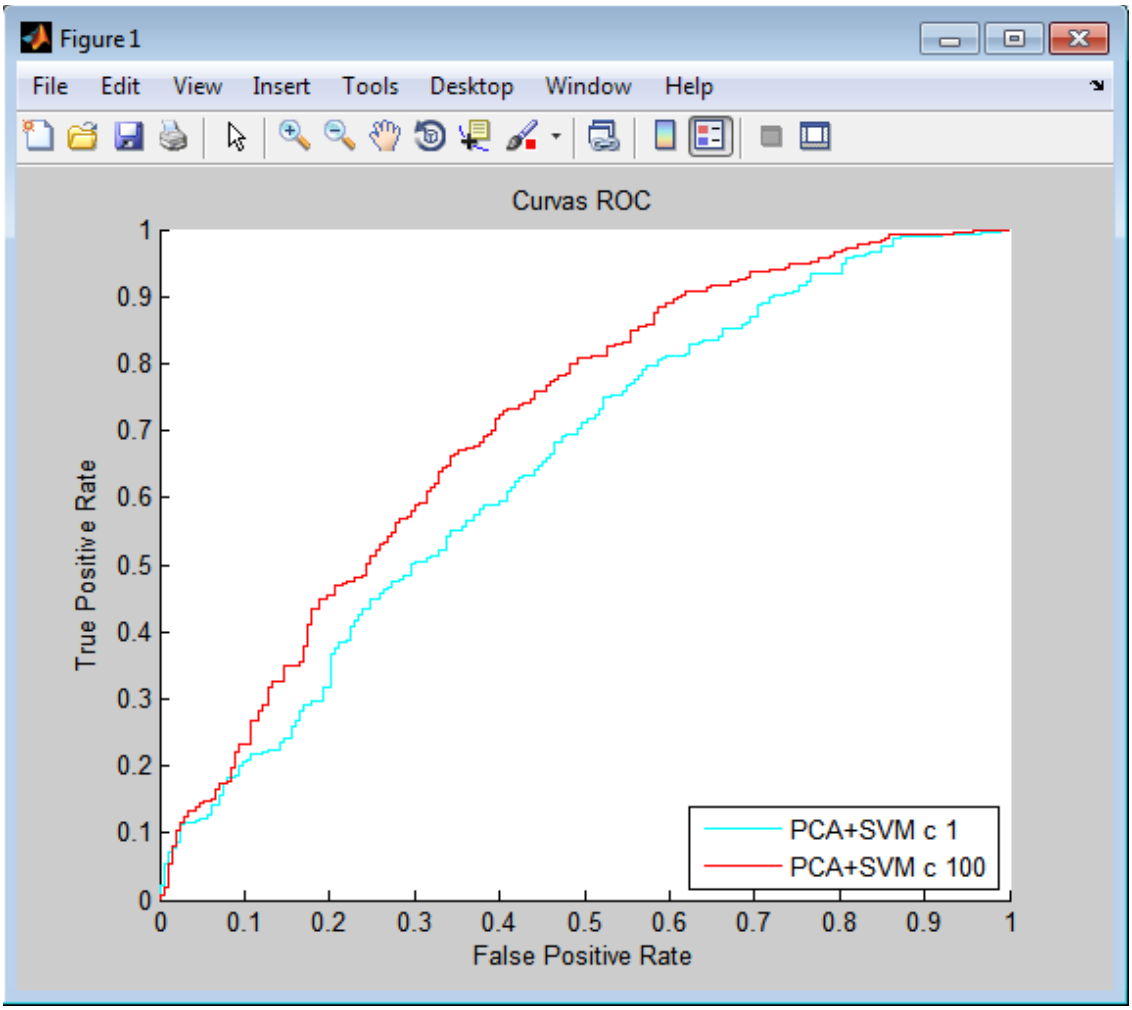

Fig. 10.44. Curvas ROC PCA+SVM (Tercer conjunto de imágenes)

Se obtienen mejores resultados utilizando los métodos *PCA+Distancia Euclídea*, como se puede comprobar observando las tablas. De hecho, para el conjunto de entrenamiento, se logra un resultado de un 100%.

Se contempla además, que utilizando *PCA+SVM* no se obtienen tan buenos resultados. Comprobando las "etiquetas" que daba este clasificador a las muestras (a qué clase decía que pertenecían, *Diadema* o *No-Diadema*) se ha observado que, de hecho, la mayoría de las veces el clasificador etiqueta a las muestras como *Diadema*, comportándose como un clasificador "tonto". Debido a falta de tiempo, no se ha podido profundizar en la solución a este problema, lo que podría ser una futura ampliación de este Trabajo de Fin de Grado.

También se percibe, en las gráficas de las curvas ROC, que *PCA+SVM* obtiene mejores resultados con el parámetro *c* tomando el valor 100, que cuando está igualado a 1.

Capítulo 11. Conclusiones

El desarrollo de este Trabajo de Fin de Grado, que tenía como objetivo explorar la robustez con la que se pueden clasificar visualmente diferentes tipos de erizos a partir tanto de imágenes estáticas como de secuencias de vídeo para evaluar si, mediante el empleo de técnicas de visión por computador, es posible resolver estas tareas de clasificación mediante la inspección automática de dichos vídeos e imágenes, ha llevado a diversas conclusiones como:

- Para poder desarrollar estos detectores de manera correcta, hay que llevar a cabo un estudio bastante profundo de los resultados obtenidos para ir variando los parámetros que se crean convenientes a la hora de modificarlo, teniendo especial cuidado, dado que el mínimo cambio puede dar resultados muy diferentes.

- También se ha descubierto que las condiciones ópticas de operación a larga distancia (detector *erizo-probable*) dificultan de forma importante la detección. Por ello, es preferible tener una alta tasa de falsos positivos, descartables después a corta distancia, que correr el riesgo de perder esos positivos.

- Observando los resultados obtenidos se demuestra que dichos métodos de clasificación pueden llegar a ser realmente útiles en el campo de estudio, siendo necesario, no obstante, el refinamiento de los detectores resultantes si se quiere un resultado verdaderamente óptimo.

En definitiva, se llega a la conclusión de que utilizando los detectores *HAAR* se consiguen mejores resultados, pero a un coste computacional bastante alto. Sin embargo, los detectores *LBP* tienen un coste computacional bastante más bajo. Por ello en el caso de sistemas de tiempo real, se podrían utilizar los detecores *LBP*, a pesar de que no obtenga mejores resultados, pero sí "en el momento", y después refinar dichos resultados con los detectores *HAAR erizo-diadema*, ya que en este caso sí sería importante tener resultados óptimos.

También se ha visto la importancia de los tamaños de ventana en los detectores *HAAR*, consiguiendo mejores resultados si se aplica una ventana de aumento del 20% frente a aplicar una ventana de aumento del 10%.

11.1. Propuestas de mejoras futuras

Como posibles mejoras futuras se proponen:

- Intentar mejorar los detectores mediante un estudio más profundo de los resultados obtenidos y el planteamiento de cambios que pudieran llevar a cabo dichas mejoras y/o empleando un conjunto de entrenamiento y test más amplio y de mejor calidad.

- Posibilidad de detectar también otro tipo de erizos o animales marinos que puedan resultar de interés en estudios de relación o asociación entre especies.

- Estudio de otras técnicas de visión por computador y posible combinación con las estudiadas en este trabajo.

- Aplicar este proyecto en algún campo de estudio. Por ejemplo, para registrar las trayectorias de un conjunto muy amplio de animales que se mueven lentamente sin necesidad de marcarlos. Para ello, bastaría con situar una cámara con dos *grados de libertad*[22] en la escena a estudiar conectada a un sistema empotrado que realizaría el *tracking*[23] y permitiría almacenar imágenes a una cierta frecuencia.

Capítulo 12. Manual de usuario

En este capítulo se va a explicar cómo utilizar el software empleado a lo largo de este Trabajo de Fin de Grado.

12.1. MarcaErizo

Para usar este programa hay que hacer los siguientes cambios en el código:

```
imgsfolder='<RUTA_DONDE_SE_ENCUENTRA_LA_IMAGEN>';
filestosearch=sprintf('%s<NOMBRE_DE_LA_IMAGEN>',imgsfolder);
```

Al ejecutarlo en *MATLAB*, nos preguntará el número de erizos que se desean marcar, se le introduce dicho dato y se marcan los erizos, empezando por la esquina superior izquierda del contenedor, a continuación la esquina superior derecha y, por último, la esquina inferior derecha del contenedor de dicho erizo.

A continuación se muestran unas cuantas capturas que explican dicho procedimiento.

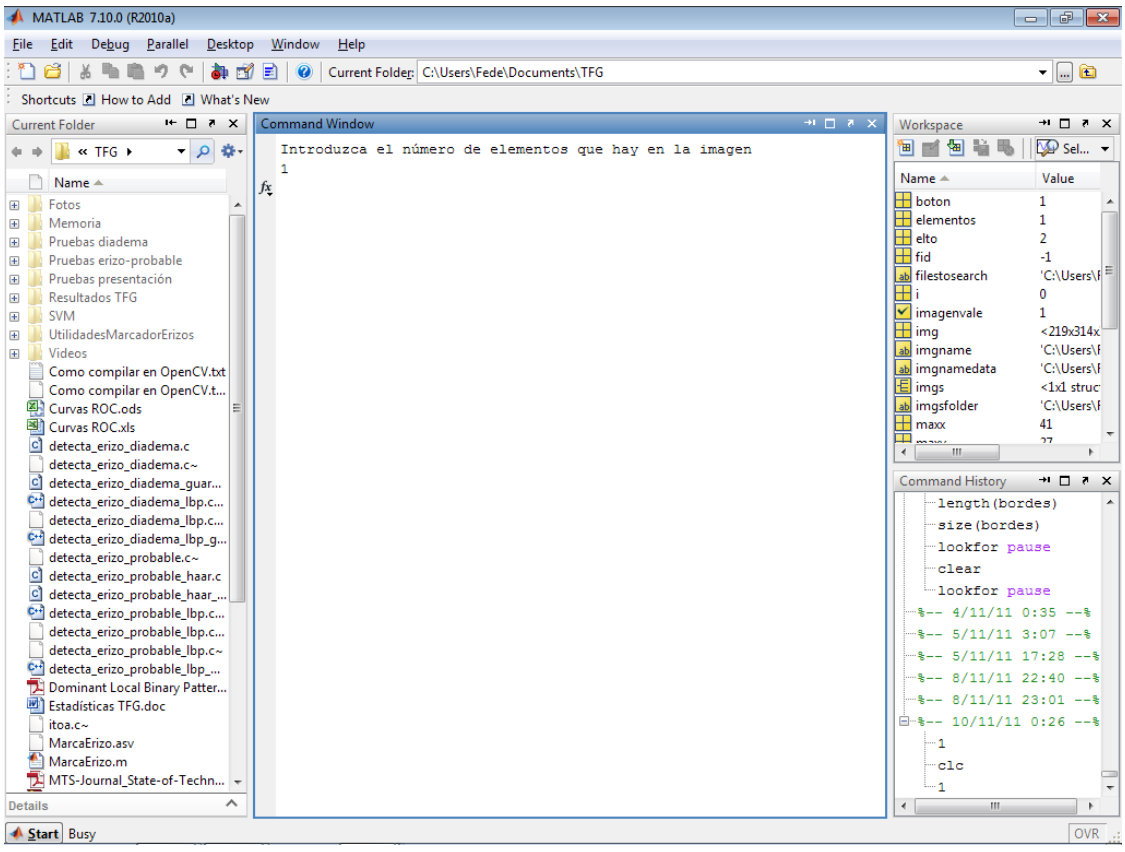

Fig. 12.1. Paso 1 MarcaErizo

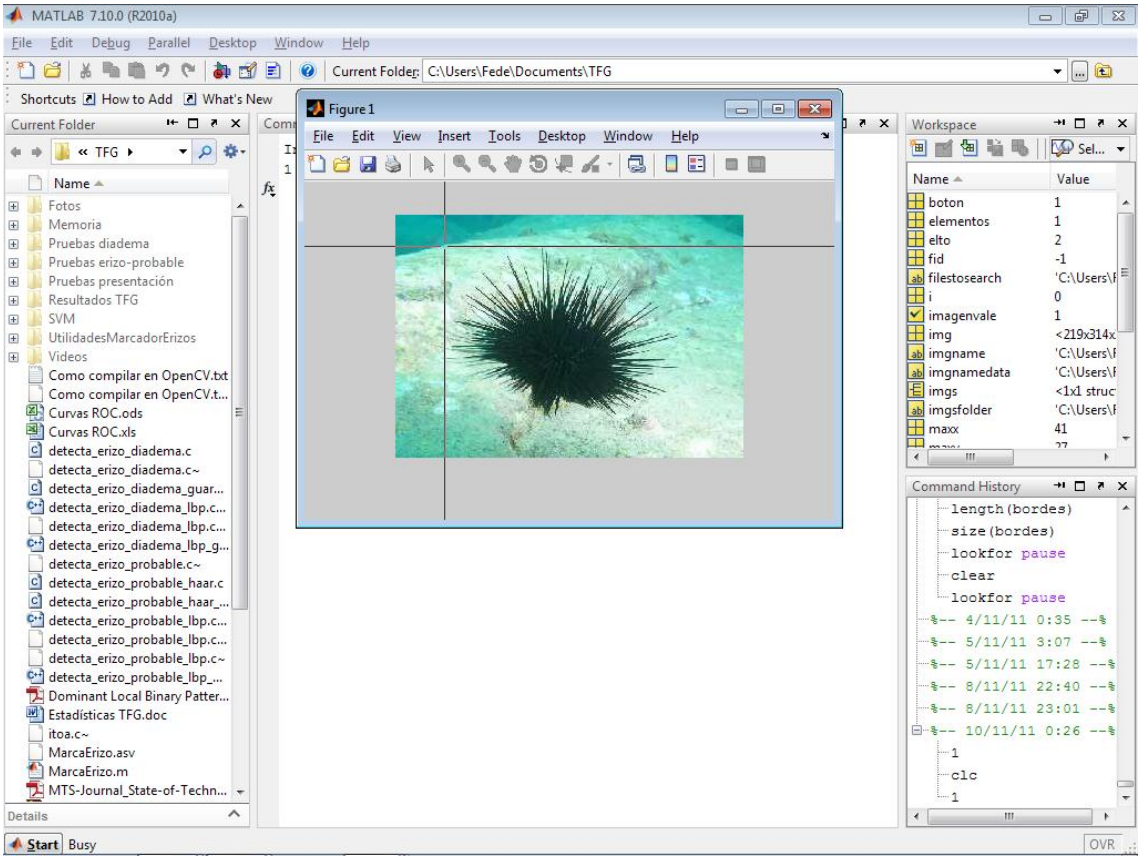

Fig. 12.2. Paso 2 MarcaErizo

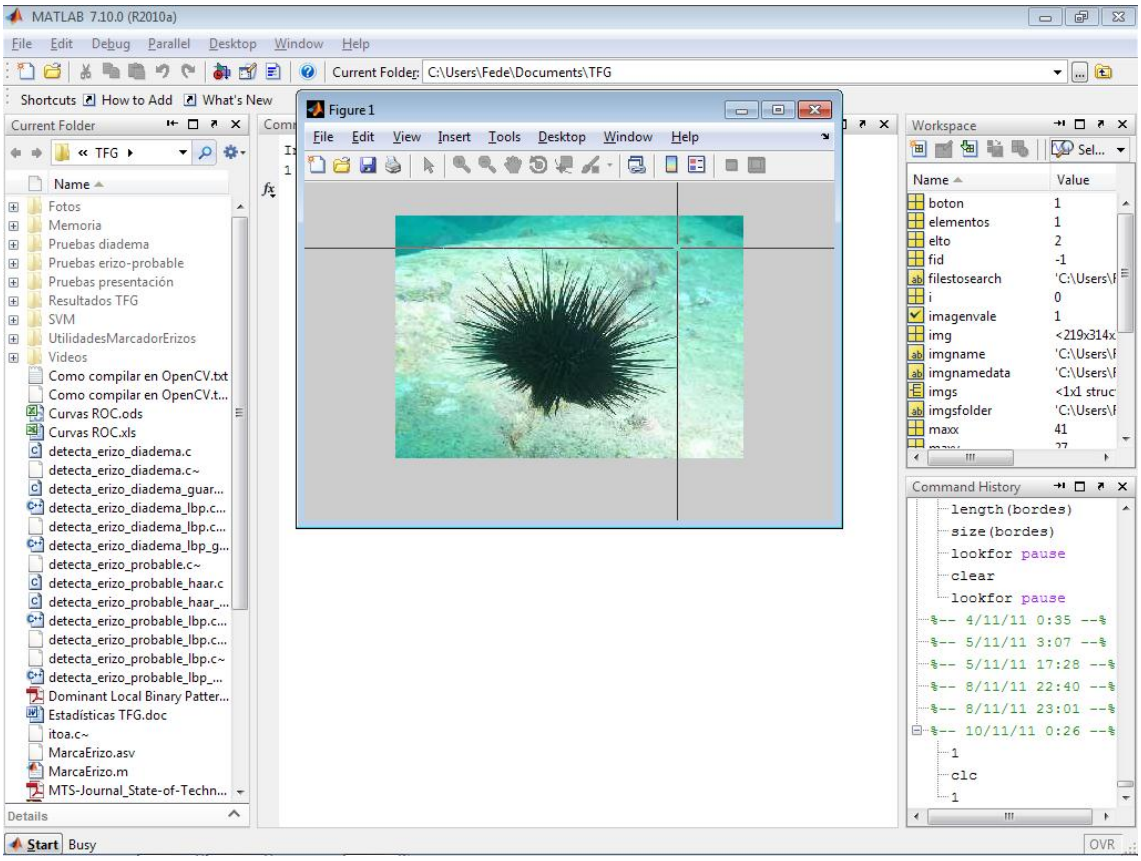

Fig. 12.3. Paso 2 MarcaErizo

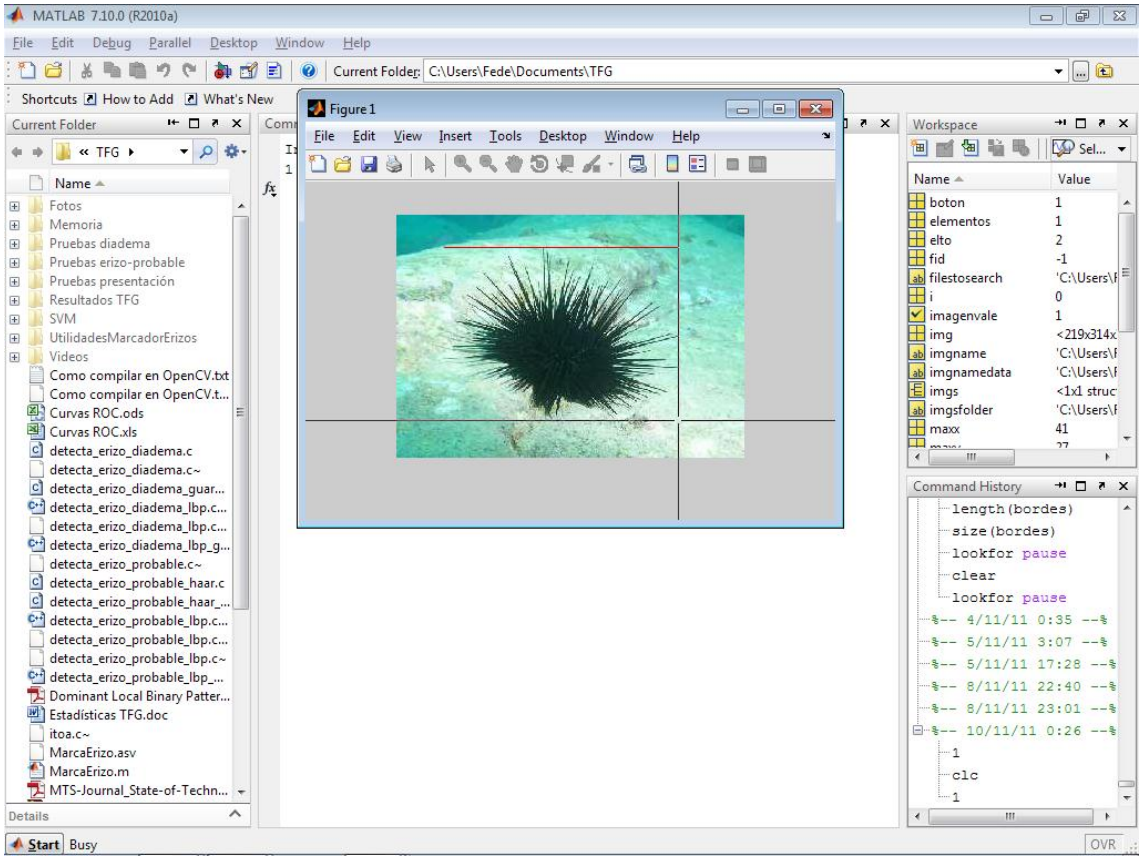

Fig. 12.4. Paso 2 MarcaErizo

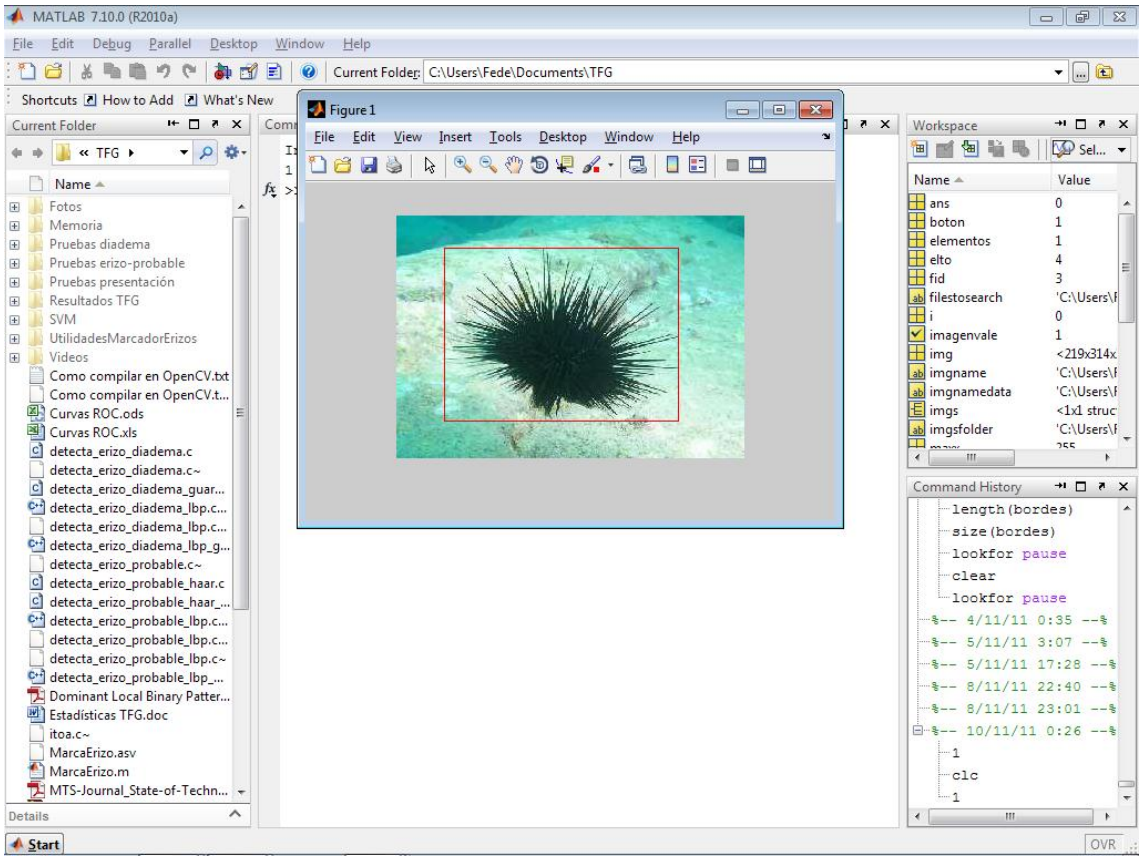

Fig. 12.5. Paso 2 MarcaErizo

12.2. detecta_erizo_probable, detecta_erizo_diadema, Video

Para usar cualquiera de las variantes de estos detectores hay que realizar el siguiente cambios:

```
const char* cascade_name =
"<RUTA_CLASIFICADOR>/<CARPETA_CLASIFICADOR(P.E.data_HAAR_20_70x70)>/clasif.xml";
```

Después de compilarlo se ejecutan utilizando una terminal, para ello, hay que utilizar el siguiente comando:

…/<NOMBRE_DETECTOR> <IMAGEN O VIDEO>

Las variantes que guardan la imagen la guardan con el siguiente formato:

<NOMBRE_IMAGEN>_resultado

Los detectores que utilizan vídeos guardan los frames con el siguiente formato:

<NOMBRE_VIDEO>_resultado_<NUMERO_DE_FRAME>.png

12.3. GeneraMuestrasRecortadas

Para usar este programa hay que hacer los siguientes cambios en el código:

```
fid=fopen('<RUTA_DE_LA_IMAGEN><NOMBRE_DE_LA_IMAGEN>.txt','r');
```

Si se desea cambiar el tamaño de la imagen de 100x100 a <*A*>x<*A*>, se tendría que cambiar la siguiente línea:

```
TAM=<A>;
```

Las imágenes generadas serán de tamaño <*A*>x<*A*> y tendrán el siguiente formato:

- **Entrenamiento:**

  <NOMBRE_DE_LA_IMAGEN>_scaled_<NUMERO_DE_MUESTRA>

- **Test:**

  <NOMBRE_DE_LA_IMAGEN>_scaled_<IDENTIFICADOR><NUMERO_DE_MUESTRA>

  Cambiando en el código:

  ```
  thisimgnamescaled=sprintf('%s<IDENTIFICADOR>_%i.png',
  imgnamescaled,i);
  ```

  Si se desea guardar con algún identificador.

12.4. DemoPCASVM

Esta función hay que ejecutarla en *MATLAB* de la siguiente manera:

```
DemoPCASVM(<RUTA_DE_CONJUNTO_DE_ENTRENAMIENTO>,
<RUTA_DE_CONJUNTO_DE_TEST>, <FICHERO_PCA_RESULTANTE>)
```

Capítulo 13. Instalación de OpenCV

El sitio principal de *OpenCV* se encuentra en *SourceForge* en http://sourceforge.net/projects/opencvlibrary/ y la página de "Wiki" de *OpenCV está en* http://opencvlibrary.sourceforge.net/. Para Linux, la última distribución que se usó fue la 2.3.1, de la página http://sourceforge.net/projects/opencvlibrary/files/opencv-unix/2.3.1/. Seleccionando el archivo OpenCV-2.3.1a.tar.bz2. Sin embargo, la versión más actualizada está siempre en el servidor CVS (*Concurrent Versions System*) en SourceForge.

Una vez descargadas las librerías, se deben instalar. Para obtener instrucciones detalladas de instalación de Linux o Mac OS, se puede consultar el archivo de texto llamado *INSTALL* que se encuentra en el directorio *.../opencv/*. Este archivo también describe cómo compilar y ejecutar las rutinas de prueba de *OpenCV*. El fichero enumera los programas adicionales que se necesitan para convertirse en un desarrollador de *OpenCV*, como *autoconf*, *automake*, *libtool* y *swig*.

En la versión de *OpenCV* para Linux no están incluidos los binarios precompilados, debido a la gran variedad de versiones de *gcc* y *glibc* en las diferentes distribuciones (*SuSE*, *Debian*, *Ubuntu*, etc.) Si la que se utiliza distribución no ofrece *OpenCV*, será necesario compilarlo a partir de los fuentes, como se detalla en el archivo *.../opencv/INSTALL*.

Para compilar las librerías y las demos, se necesita *GTK+2.x* o superior, incluyendo las cabeceras. También son necesarios *pkgconfig*, *libpng*, *zlib*, *libjpeg*, *libtiff* y *libjasper*. También son necesarias *libavcodec* y las demás librerías *libav* (incluidas las cabeceras) de *ffmpeg*.

ffmpeg se puede descargar de http://ffmpeg.mplayerhq.hu/download.html. El programa *ffmpeg* tiene una licencia *LGPL* (*Lesser General Public License*). Para usarla con software de tipo no *GPL* (como *OpenCV*), hay que compilarlo y usar una librería de *ffmpeg* compartida:

```
$> ./configure --enable-shared
$> make
$> sudo make install
```

Para compilar OpenCV una vez descargado:

```
$> ./configure
$> make
$> sudo make install
$> sudo ldconfig
```

Una vez completa la instalación, la ruta de instalación por defecto es */usr/local/lib/* y */usr/local/include/opencv/*. Por lo tanto, es necesario añadir */usr/local/lib* a */etc/ld.so.conf* (y ejecutar *ldconfig* después) o añadirla a la variable de entorno *LD_LIBRARY_PATH*; después ya se ha terminado.

Alternativamente, se puede añadir *<ruta_de_instalación>/bin* y *<ruta_de_instalación>/bin/linux32*, uno por línea a */etc/ld.so.conf* y luego ejecutar *ldconfig* como *root* (o usando *sudo*).

Eso es todo, si se quieren más detalles, se puede ver *…/opencv/INSTALL*.

Obtener el último OpenCV mediante CVS

OpenCV está en constante desarrollo y los errores se arreglan a menudo rápidamente cuando los informes de errores contienen descripciones precisas y el código que muestra el error. Sin embargo, las versiones oficiales de *OpenCV* se lanzan sólo una o dos veces al año. Si se está desarrollando seriamente un proyecto o producto, es probable que se deseen correcciones de código y actualizaciones tan pronto como estén disponibles. Para ello, se tendrá que acceder al *Sistema de Versiones Concurrentes* de *OpenCV* (*Concurrent Versions System* o *CVS*) en *SourceForge*.

En Linux se pueden usar los siguientes comandos:

```
cvs -d:pserver:anonymous@opencvlibrary.cvs.sourceforge.net:/cvsroot/opencvlibrary
login
```

Cuando se pregunte por una clave, se presiona la tecla *ENTER*. A continuación se usa:

```
cvs -z3 -d:pserver:anonymous@opencvlibrary.cvs.sourceforge.net:/cvsroot/opencvlibrary
co -P opencv
```

Capítulo 14. Creación del conjunto de datos de entrenamiento

Para cualquier tipo de objeto, es necesario construir dos conjuntos de imágenes independientes, cuando se crea una cascada basada en el método de *Viola-Jones*.[18]

1. **Conjunto de muestras positivas:**

   En este conjunto se encuentran las imágenes que contienen muestras del objeto que se desea detectar, es decir, aquel para el que se diseña el clasificador. Este conjunto de imágenes es necesario para que el comando **createsamples** genere el conjunto de imágenes de entrenamiento en el formato esperado por el comando **haartraining** que, como se indicará más adelante, será quien realice efectivamente el cálculo del clasificador. Entre sus parámetros se pueden destacar:

   - *info*: permite especificar el fichero que lista las imágenes que contienen el patrón de interés.
   - *vec*: especifica el nombre del fichero de muestras resultante.
   - *w* y *h*: ancho y alto, respectivamente, de las muestras positivas a generar.
   - *num*: número de muestras a generar.

   Las imágenes positivas proporcionadas no necesariamente deben ser todas del mismo tamaño, pero se sugiere que estén bastante alineadas; será **createsamples** quien las escale en base a los parámetros w y h proporcionados.

   createsamples permite además distorsionar las muestras que tenemos para obtener un número mayor de muestras positivas a partir de un grupo inicial de imágenes más reducido.

   En los siguientes casos, **createsamples** toma la lista de muestras positivas contenidas en el fichero de texto *pos.txt*, usándose como fondo las imágenes contenidas en *./negatives/train/train.txt*, generando 10.000 imágenes que se almacenan en *samples.vec*.

   Para el detector *erizo-probable*, se han usado los siguientes parámetros:

```
createsamples -info pos.txt -num 10000 -bg ./negatives/train/train.txt -vec data_HAAR_20x20/samples.vec -w 20 -h 20
```

En este caso, extrae las ventanas conteniendo al objeto y las transforma al tamaño 20x20.

Para el detector erizo-diadema, se han usado los siguientes parámetros:

```
createsamples -info pos.txt -num 10000 -bg ./negatives/train/train.txt -vec data_HAAR_70x70/samples.vec -w 70 -h 70
```

En este caso, extrae las ventanas conteniendo al objeto y las transforma al tamaño 70x70.

Cada fila del fichero de muestras positivas, pos.txt en este caso, localiza una imagen que contiene una o varias muestras del objeto.

Tras la ruta de la imagen, se indica el número de muestras positivas del patrón de interés contenidas en la imagen y el contenedor de cada una en el formato: Coordenadas *x* e *y* de la esquina superior izquierda, seguidas del ancho y alto del contenedor. A continuación se muestra una parte del fichero de texto *pos.txt*.

```
./diadema/cerca/02.jpg 1 26 19 329 270
./diadema/cerca/12.jpg 2 117 1 272 294 186 89 225 319
./diadema/cerca/06.jpg 1 1 1 639 479
./diadema/cerca/16.jpg 1 2 37 423 465
./diadema/cerca/09.jpg 1 0 1 652 466
./diadema/cerca/10.jpg 1 50 280 315 272
./diadema/cerca/15.jpg 1 69 32 90 99
./diadema/cerca/14.jpg 1 78 1 162 158
./diadema/cerca/01.png 2 250 108 198 259 178 168 80 83
./diadema/cerca/05.JPG 1 142 269 305 289
```

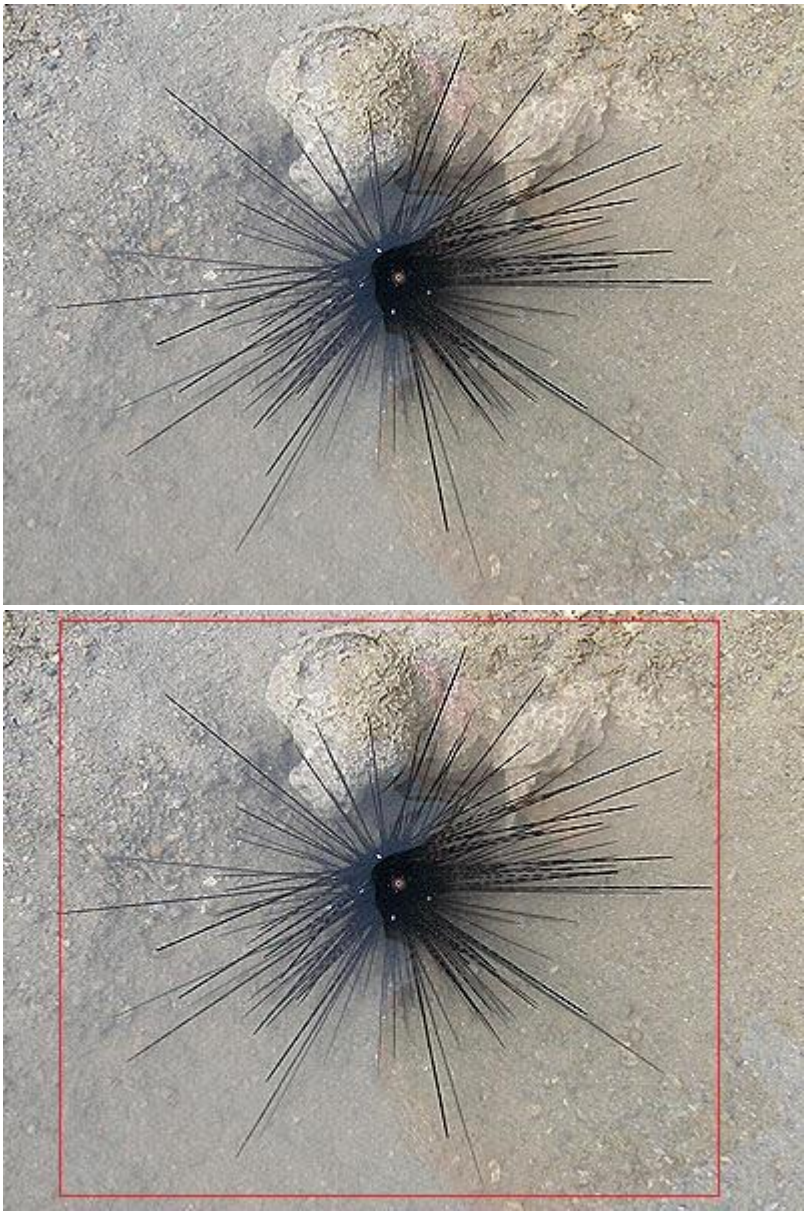

Fig. 14.1. Ejemplo de imagen cuya línea en el fichero de muestras positivas es *./diadema/cerca/02.jpg 1 26 19 329 270*.
El contenedor del objeto está mostrado en la imagen inferior.

Tras crear el fichero de muestras, en este caso *samples.vec*, y ejecutar **createsamples**, es posible visualizar las muestras positivas creadas una vez realizada la transformación con la opción **vec** del comando **createsamples**.

```
createsamples -vec data_HAAR_20x20/samples.vec -w 20 -h 20
% Visualiza el fichero .vec de muestras positivas
```

2. Conjunto de muestras negativas:

De forma análoga a las muestras positivas, usamos un fichero de texto para indicar la localización de cada muestra negativa.

Cada fila del fichero de texto localiza una imagen sin presencia del patrón a buscar, es decir, contiene la ruta de la imagen. Se sugiere utilizar imágenes grandes estilo fondo de escritorio para poder obtener un dominio mayor de falsos positivos. A continuación se muestra una parte del fichero de texto *train.txt*.

./samples/000001.jpg

./samples/000290a.jpg

./samples/005140.jpg

./samples/006140_1.jpg

./samples/007180_2.jpg

./samples/008150.jpg

./samples/01-underwater.jpg

./samples/010_siena.jpg

./samples/018_pisa.jpg

./samples/01d_sonnenspiegel.JPG

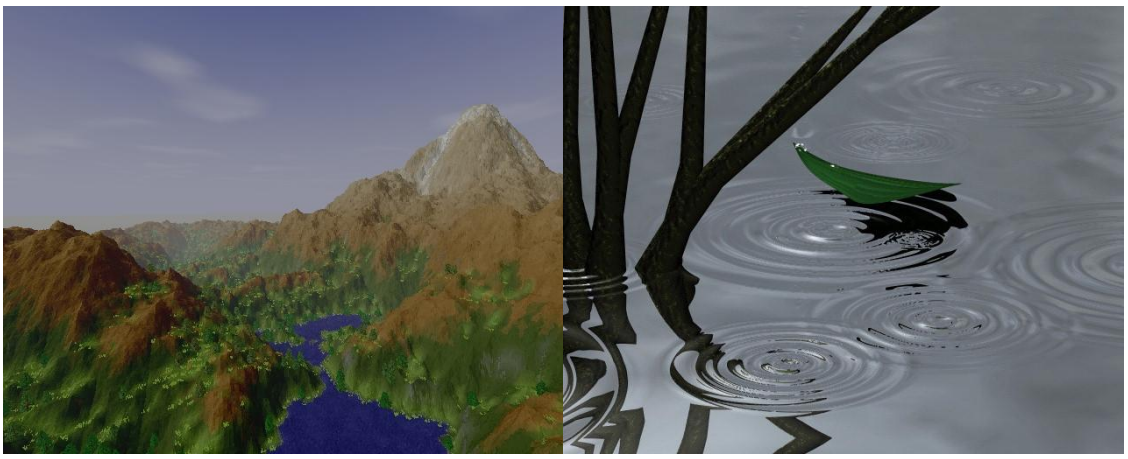

Fig. 14.2. Ejemplos de imágenes del conjunto de muestras negativas empleadas.

Capítulo 15. Código

En este anexo se muestra el código empleado para la implementación de cada uno de los módulos utilizados.

15.1. MarcaErizo

```
clear;

%Utilidad para anotar muestras de un patrón

%Carpeta de localización de las imágenes con muestras

imgsfolder='C:\Users\Fede\Documents\TFG\Fotos\otros_erizos\';

%Obtenemos los nombres de las imágenes en el directorio
filestosearch=sprintf('%surchins.jpg',imgsfolder);
imgs=dir(filestosearch);

%Número de puntos a  marcar por elemento
neltos=3;

%Número de elementos (erizos) en la imagen
%elementos=1;

%Inicializamos el número de imágenes
nimgs=0;

%Para todas las imágenes disponibles
for nimg=1:size(imgs,1)

    %Nombre de la imagen
    imgname=imgs(nimg).name;

    %Incremento el número de imágenes
    nimgs=nimgs+1;

    %¿Hay fichero de datos de esta imagen?
    imgnamedata=sprintf('%s%s_erizo.txt', imgsfolder,
imgs(nimg).name);

%imgnamedata=sprintf('%s%s_erizo.txt',imgsfolder,imgs(nimg).name(1:17)
);

    fid = fopen(imgnamedata,'r');

    if fid==-1 %No hay fichero, anotamos imgnamedata

        %Tomamos imagen
        %Cargamos la imagen de disco imagen
        imgname=sprintf('%s%s',imgsfolder,imgs(nimg).name);
        img = imread(imgname);
```

```
            %Muestra imagen
            imshow(uint8(img));

            %Pregunta el número de elementos que hay en la imagen
            elementos=input('Introduzca el número de elementos que hay
en la imagen\n');

            imagenvale=true;
            for i=0:elementos-1
                elto=1;
               while elto <= neltos && imagenvale==true
                    %Botón izquierdo para elementos buenos, y derecho
para
                    %aquellos no buenos
                    [x y boton]=ginput(1);

                    %Según el botón del ratón seleccionado
                    if boton == 3           %Botón derecho
                        imagenvale=false;
                    else
                        if boton == 1       %Botón izquierdo
                            hold on;

                            %Muestra imagen
                            imshow(uint8(img));

                            %Lo guardo
                            pointx(elto+i*neltos)=round(x(1));
                            pointy(elto+i*neltos)=round(y(1));

                            %Pinto el contenedor hasta ahora
                            minx=pointx(1+i*neltos);
                            maxx=pointx(1+i*neltos);
                            miny=pointy(1+i*neltos);
                            maxy=pointy(1+i*neltos);
                            for nep=2:elto
                                if pointx(nep+i*neltos)>maxx
                                    maxx=pointx(nep+i*neltos);
                                end
                                 if pointy(nep+i*neltos)>maxy
                                    maxy=pointy(nep+i*neltos);
                                 end
                                 if pointx(nep+i*neltos)<minx
                                    minx=pointx(nep+i*neltos);
                                 end
                                 if pointy(nep+i*neltos)<miny
                                    miny=pointy(nep+i*neltos);
                                end
                            end

                            %Pinto
                            plot([minx maxx maxx minx minx],[miny miny
maxy maxy miny],'r-');

                            elto=elto+1;

                            hold off;
```

```
                    end
                end

                %pause(1);
            end
        end

        %Si hay un erizo en la imagen escribe los datos a disco
        if imagenvale==true
            %Escribimos datos
            fid = fopen(imgnamedata,'w');

            pos=strfind(imgname,' ');
            while pos~=0
                imgname(pos)='_';
                pos=strfind(imgname,' ');
            end

           fprintf(fid,'%s %d ',imgname, elementos);

           for i=0:elementos-1

              %Coordenadas x e y de la esquina superior izquierda

              fprintf(fid, '%d %d ', pointx(1+i*neltos),
pointy(1+i*neltos));

              %Ancho y alto del contenedor

              fprintf(fid, '%d %d ', pointx(2+i*neltos)-
pointx(1+i*neltos), pointy(3+i*neltos)-pointy(1+i*neltos));
           end
            %for elto=1:neltos
            %    fprintf(fid,'%d %d ',pointx(elto),pointy(elto));
            %end
            fclose(fid);
        else
            fid = fopen(imgnamedata,'w');
            fprintf(fid,'-1 -1 -1 -1 -1 -1 -1 -1\n');
            fclose(fid);

        end

    else %Hay fichero lo cierra
         fclose(fid);
    end
end
```

15.2. Detectores HAAR

15.2.1. detecta_erizo_probable_haar

```
// Include header files
#include "cv.h"
#include "highgui.h"

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <assert.h>
#include <math.h>
#include <float.h>
#include <limits.h>
#include <time.h>
#include <ctype.h>
#include <fcntl.h>
#include <unistd.h>
#include <sys/types.h>


// Create a string that contains the exact cascade name
const char* cascade_name =
"/home/fede/Escritorio/TFG/Fotos/data_HAAR_16_20x20/clasif.xml";
FILE *f;


// Function prototype for detecting and drawing an object from an image
void detect_and_draw( IplImage* image );

// Main function, defines the entry point for the program.
int main( int argc, char** argv )
{
   char *filename;

   if(argc != 2) fprintf(stderr, "Falta el nombre de la foto\n");

   filename=strdup(argv[1]);

   printf("%s\n",filename);

   // Salida con las coordenadas
   char salida[strlen(filename)+strlen("_coordenadas_haar_16")+2];
   strcpy(salida, "./");
   strcat(salida, filename);
   strcat(salida, "_coordenadas_haar_16.txt");

   f=fopen(salida,"w");

    // Create a sample image
    IplImage *img = cvLoadImage(filename, -1);

    // Call the function to detect and draw the urchin positions
    detect_and_draw(img);

    // Wait for user input before quitting the program
    cvWaitKey(0);
```

```
        // Release the image
        cvReleaseImage(&img);

        // Destroy the window previously created with filename: "result"
        cvDestroyWindow("result");

        // return 0 to indicate successfull execution of the program
        return 0;
    }

    // Function to detect and draw any faces that is present in an image
    void detect_and_draw( IplImage* img )
    {

        // Create memory for calculations
        static CvMemStorage* storage = 0;

        // Create a new Haar classifier
        static CvHaarClassifierCascade* cascade = 0;

        int scale = 1;
        double t = 0;

        // Create a new image based on the input image
        IplImage* temp = cvCreateImage( cvSize(img->width/scale,img-
>height/scale), 8, 3 );

        // Create two points to represent the urchin locations
        CvPoint pt1, pt2;
        int i;

        // Load the HaarClassifierCascade
        cascade = (CvHaarClassifierCascade*)cvLoad( cascade_name, 0, 0, 0 );

        // Check whether the cascade has loaded successfully. Else report and
error and quit
        if( !cascade )
        {
            fprintf( stderr, "ERROR: Could not load classifier cascade\n" );
            return;
        }

        // Allocate the memory storage
        storage = cvCreateMemStorage(0);

        // Create a new named window with title: result
        cvNamedWindow( "result", 1 );

        // Clear the memory storage which was used before
        cvClearMemStorage( storage );

        // Find whether the cascade is loaded, to find the urchins. If yes, then:
        if( cascade )
        {
              t = (double)cvGetTickCount();
            // There can be more than one urchin in an image. So create a
growable sequence of urchins.
            // Detect the objects and store them in the sequence
            CvSeq* urchins = cvHaarDetectObjects( img, cascade, storage,
                                                  1.1, 2, CV_HAAR_DO_CANNY_PRUNING,
// Más rápido, menos detecciones
```

```
                                                    cvSize(20, 20), cvSize(100, 100)
);
        t = (double)cvGetTickCount() - t;
        printf( "detection time = %g ms\n", t/((double)cvGetTickFrequency()*1000.)
);
             // Loop the number of urchin found.
             for( i = 0; i < (urchins ? urchins->total : 0); i++ )
             {
                // Create a new rectangle for drawing the urchin
                 CvRect* r = (CvRect*)cvGetSeqElem( urchins, i );

                 // Find the dimensions of the urchin,and scale it if necessary
                 pt1.x = r->x*scale;
                 pt2.x = (r->x+r->width)*scale;
                 pt1.y = r->y*scale;
                 pt2.y = (r->y+r->height)*scale;

                 fprintf(f,"%d %d %d %d\n",pt1.x,pt1.y,pt2.x,pt2.y); // Se guardan
las coordenadas de los erizos

                 // Draw the rectangle in the input image
                 cvRectangle( img, pt1, pt2, CV_RGB(255,0,0), 3, 8, 0 );
             }
         }

         fclose(f);

         // Show the image in the window named "result"
         cvShowImage( "result", img );

         // Release the temp image created.
         cvReleaseImage( &temp );
    }
```

15.2.2. detecta_erizo_probable_haar_guarda

```
// Include header files
#include "cv.h"
#include "highgui.h"

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <assert.h>
#include <math.h>
#include <float.h>
#include <limits.h>
#include <time.h>
#include <ctype.h>
#include <fcntl.h>
#include <unistd.h>
#include <sys/types.h>


// Create a string that contains the exact cascade name
const char* cascade_name =
"/home/fede/Escritorio/TFG/Fotos/data_HAAR_20_20x20/clasif.xml";

// Function prototype for detecting and drawing an object from an image
void detect_and_draw(IplImage* image, char *imgname);

// Main function, defines the entry point for the program.
int main( int argc, char** argv )
{
   char *filename;

   if(argc != 2) fprintf(stderr, "Falta el nombre de la foto\n");

   filename=strdup(argv[1]);

   char imgname[strlen(argv[1]) + strlen("_resultado")];

   strcpy(imgname, argv[1]);
   strcat(imgname, "_resultado");

    // Create a sample image
    IplImage *img = cvLoadImage(filename, -1);

    // Call the function to detect and draw the urchin positions
    detect_and_draw(img, imgname);

    // Wait for user input before quitting the program
    cvWaitKey(0);

    // Release the image
    cvReleaseImage(&img);

    // Destroy the window previously created with filename: "result"
    cvDestroyWindow("result");

    // return 0 to indicate successfull execution of the program
    return 0;
}

// Function to detect and draw any faces that is present in an image
```

```
    void detect_and_draw(IplImage* img, char *imgname)
    {

        // Create memory for calculations
        static CvMemStorage* storage = 0;

        // Create a new Haar classifier
        static CvHaarClassifierCascade* cascade = 0;

        int scale = 1;
        double t = 0;

        // Create a new image based on the input image
        IplImage* temp = cvCreateImage( cvSize(img->width/scale,img-
>height/scale), 8, 3 );

        // Create two points to represent the urchin locations
        CvPoint pt1, pt2;
        int i;

        // Load the HaarClassifierCascade
        cascade = (CvHaarClassifierCascade*)cvLoad( cascade_name, 0, 0, 0 );

        // Check whether the cascade has loaded successfully. Else report and
error and quit
        if( !cascade )
        {
            fprintf( stderr, "ERROR: Could not load classifier cascade\n" );
            return;
        }

        // Allocate the memory storage
        storage = cvCreateMemStorage(0);

        // Create a new named window with title: result
        cvNamedWindow( "result", 1 );

        // Clear the memory storage which was used before
        cvClearMemStorage( storage );

        // Find whether the cascade is loaded, to find the urchins. If yes, then:
        if( cascade )
        {
              t = (double)cvGetTickCount();
            // There can be more than one urchin in an image. So create a
growable sequence of urchins.
            // Detect the objects and store them in the sequence
            CvSeq* urchins = cvHaarDetectObjects( img, cascade, storage,
                                                  1.1, 2, CV_HAAR_DO_CANNY_PRUNING,
// Más rápido, menos detecciones
                                                  cvSize(20, 20), cvSize(100, 100)
);
       t = (double)cvGetTickCount() - t;
       printf( "detection time = %g ms\n", t/((double)cvGetTickFrequency()*1000.)
);
            // Loop the number of urchin found.
            for( i = 0; i < (urchins ? urchins->total : 0); i++ )
            {
               // Create a new rectangle for drawing the urchin
                CvRect* r = (CvRect*)cvGetSeqElem( urchins, i );

                // Find the dimensions of the urchin,and scale it if necessary
```

```
            pt1.x = r->x*scale;
            pt2.x = (r->x+r->width)*scale;
            pt1.y = r->y*scale;
            pt2.y = (r->y+r->height)*scale;

            // Draw the rectangle in the input image
            cvRectangle( img, pt1, pt2, CV_RGB(255,0,0), 3, 8, 0 );
        }
    }

    // Show the image in the window named "result"
    cvShowImage( "result", img );

   cvSaveImage(imgname, img);

    // Release the temp image created.
    cvReleaseImage( &temp );
}
```

15.2.3. detecta_erizo_diadema

```
    // Include header files
    #include "cv.h"
    #include "highgui.h"

    #include <stdio.h>
    #include <stdlib.h>
    #include <string.h>
    #include <assert.h>
    #include <math.h>
    #include <float.h>
    #include <limits.h>
    #include <time.h>
    #include <ctype.h>


    // Create a string that contains the exact cascade name
    const char* cascade_name =
"/home/fede/Escritorio/TFG/Fotos/data_HAAR_20_70x70/clasif.xml";
    FILE *f;

    // Function prototype for detecting and drawing an object from an image
    void detect_and_draw( IplImage* image );

    // Main function, defines the entry point for the program.
    int main( int argc, char** argv )
    {
        char *filename;

        if(argc != 2) fprintf(stderr, "Falta el nombre de la foto\n");

        filename=strdup(argv[1]);

        printf("%s\n",filename);

        // Salida con las coordenadas
        char salida[strlen(filename)+strlen("_coordenadas_haar_1.2")+2];
        strcpy(salida, "./");
        strcat(salida, filename);
        strcat(salida, "_coordenadas_haar_1.2.txt");

        f=fopen(salida,"w");

        // Create a sample image
        IplImage *img = cvLoadImage(filename, -1);

        // Call the function to detect and draw the urchin positions
        detect_and_draw(img);

        // Wait for user input before quitting the program
        cvWaitKey(0);

        // Release the image
        cvReleaseImage(&img);

        // Destroy the window previously created with filename: "result"
        cvDestroyWindow("result");

        // return 0 to indicate successfull execution of the program
        return 0;
```

```
    }

    // Function to detect and draw any urchins that is present in an image
    void detect_and_draw( IplImage* img )
    {

       // Create memory for calculations
       static CvMemStorage* storage = 0;

       // Create a new Haar classifier
       static CvHaarClassifierCascade* cascade = 0;

       int scale = 1;
       double t = 0;

       // Create a new image based on the input image
       IplImage* temp = cvCreateImage( cvSize(img->width/scale,img-
>height/scale), 8, 3 );

       // Create two points to represent the urchin locations
       CvPoint pt1, pt2;
       int i;

       // Load the HaarClassifierCascade
       cascade = (CvHaarClassifierCascade*)cvLoad( cascade_name, 0, 0, 0 );

       // Check whether the cascade has loaded successfully. Else report and
error and quit
       if( !cascade )
       {
          fprintf( stderr, "ERROR: Could not load classifier cascade\n" );
          return;
       }

       // Allocate the memory storage
       storage = cvCreateMemStorage(0);

       // Create a new named window with title: result
       cvNamedWindow( "result", 1 );

       // Clear the memory storage which was used before
       cvClearMemStorage( storage );

       // Find whether the cascade is loaded, to find the urchins. If yes, then:
       if( cascade )
       {
          t = (double)cvGetTickCount();
          // There can be more than one urchin in an image. So create a growable
sequence of urchins.
          // Detect the objects and store them in the sequence
          CvSeq* urchins = cvHaarDetectObjects(img, cascade, storage,
                                               1.2, 2, CV_HAAR_DO_CANNY_PRUNING,
                                               cvSize(70, 70), cvSize(1024,
768));
          t = (double)cvGetTickCount() - t;
          printf( "detection time = %g ms\n",
t/((double)cvGetTickFrequency()*1000.) );
          // Loop the number of urchin found.
          for( i = 0; i < (urchins ? urchins->total : 0); i++ )
          {
             // Create a new rectangle for drawing the urchin
             CvRect* r = (CvRect*)cvGetSeqElem( urchins, i );
```

```
            // Find the dimensions of the urchin and scale it, if necessary
            pt1.x = r->x*scale;
            pt2.x = (r->x+r->width)*scale;
            pt1.y = r->y*scale;
            pt2.y = (r->y+r->height)*scale;

            fprintf(f,"%d %d %d %d\n",pt1.x,pt1.y,pt2.x,pt2.y);

            // Draw the rectangle in the input image
            cvRectangle( img, pt1, pt2, CV_RGB(255,0,0), 3, 8, 0 );
        }
    }

    fclose(f);

    // Show the image in the window named "result"
    cvShowImage( "result", img );

    // Release the temp image created.
    cvReleaseImage( &temp );
}
```

15.2.4. detecta_erizo_diadema_guarda

```
// Include header files
#include "cv.h"
#include "highgui.h"

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <assert.h>
#include <math.h>
#include <float.h>
#include <limits.h>
#include <time.h>
#include <ctype.h>


// Create a string that contains the exact cascade name
const char* cascade_name =
"/home/fede/Escritorio/TFG/Fotos/data_HAAR_20_70x70/clasif.xml";

// Function prototype for detecting and drawing an object from an image
void detect_and_draw(IplImage* image, char *imgname);

// Main function, defines the entry point for the program.
int main( int argc, char** argv )
{
   char *filename;

   if(argc != 2) fprintf(stderr, "Falta el nombre de la foto\n");

   filename=strdup(argv[1]);

   char imgname[strlen(argv[1]) + strlen("_resultado")];

   strcpy(imgname, argv[1]);
   strcat(imgname, "_resultado");

   // Create a sample image
   IplImage *img = cvLoadImage(filename, -1);

   // Call the function to detect and draw the urchin positions
   detect_and_draw(img, imgname);

   // Wait for user input before quitting the program
   cvWaitKey(0);

   // Release the image
   cvReleaseImage(&img);

   // Destroy the window previously created with filename: "result"
   cvDestroyWindow("result");

   // return 0 to indicate successfull execution of the program
   return 0;
}

// Function to detect and draw any urchins that is present in an image
void detect_and_draw( IplImage* img, char *imgname )
{
```

```
        // Create memory for calculations
        static CvMemStorage* storage = 0;

        // Create a new Haar classifier
        static CvHaarClassifierCascade* cascade = 0;

        int scale = 1;
        double t = 0;

        // Create a new image based on the input image
        IplImage* temp = cvCreateImage( cvSize(img->width/scale,img-
>height/scale), 8, 3 );

        // Create two points to represent the urchin locations
        CvPoint pt1, pt2;
        int i;

        // Load the HaarClassifierCascade
        cascade = (CvHaarClassifierCascade*)cvLoad( cascade_name, 0, 0, 0 );

        // Check whether the cascade has loaded successfully. Else report and
error and quit
        if( !cascade )
        {
           fprintf( stderr, "ERROR: Could not load classifier cascade\n" );
           return;
        }

        // Allocate the memory storage
        storage = cvCreateMemStorage(0);

        // Create a new named window with title: result
        cvNamedWindow( "result", 1 );

        // Clear the memory storage which was used before
        cvClearMemStorage( storage );

        // Find whether the cascade is loaded, to find the urchins. If yes, then:
        if( cascade )
        {
           t = (double)cvGetTickCount();
           // There can be more than one urchin in an image. So create a growable
sequence of urchins.
           // Detect the objects and store them in the sequence
           CvSeq* urchins = cvHaarDetectObjects(img, cascade, storage,
                                                1.1, 2, CV_HAAR_DO_CANNY_PRUNING,
                                                cvSize(70, 70), cvSize(1024,
768));
           t = (double)cvGetTickCount() - t;
           printf( "detection time = %g ms\n",
t/((double)cvGetTickFrequency()*1000.) );
           // Loop the number of urchin found.
           for( i = 0; i < (urchins ? urchins->total : 0); i++ )
           {
              // Create a new rectangle for drawing the urchin
              CvRect* r = (CvRect*)cvGetSeqElem( urchins, i );

              // Find the dimensions of the urchin,and scale it if necessary
              pt1.x = r->x*scale;
              pt2.x = (r->x+r->width)*scale;
              pt1.y = r->y*scale;
              pt2.y = (r->y+r->height)*scale;
```

```
            // Draw the rectangle in the input image
            cvRectangle( img, pt1, pt2, CV_RGB(255,0,0), 3, 8, 0 );
        }
    }

    // Show the image in the window named "result"
    cvShowImage( "result", img );

    cvSaveImage(imgname, img);

    // Release the temp image created.
    cvReleaseImage( &temp );
}
```

15.2.5. Video_diadema

```
#include <cv.h>
#include <highgui.h>
#include "opencv2/objdetect/objdetect.hpp"
#include "opencv2/highgui/highgui.hpp"
#include "opencv2/imgproc/imgproc.hpp"
#include <iostream>
#include <stdio.h>

using namespace std;
using namespace cv;

const char* cascade_name =
"/home/fede/Escritorio/TFG/Fotos/data_HAAR_20_70x70/clasif.xml";

void detectAndDraw( IplImage* image,
                    char * imgname);

int main(int argc, char *argv[])
{
   if(argc != 2) fprintf(stderr, "Falta la ruta del video\n");

   CvCapture* capture = 0;
   char filename[strlen(argv[1]) + strlen("_resultado_")];

   strcpy(filename, argv[1]);
   strcat(filename, "_resultado_");

   // Se abre el video
   capture = cvCreateFileCapture(argv[1]);

   // Si no se puede, se muestra un mensaje por pantalla y se termina la
aplicación
   if(!capture)
   {
     cout << "ERROR -> No se ha podido abrir el video" << endl;
     return -1;
   }

   // Comienza la lectura del vídeo
   IplImage *frame = cvQueryFrame(capture);
   //double fps = cvGetCaptureProperty(capture, CV_CAP_PROP_FPS), f =
cvGetCaptureProperty(capture, CV_CAP_PROP_FOURCC);
   //CvSize size = cvSize((int)cvGetCaptureProperty(capture,
CV_CAP_PROP_FRAME_WIDTH),
   //                     (int)cvGetCaptureProperty(capture,
CV_CAP_PROP_FRAME_HEIGHT));
   //char *fourcc = (char *) (&f);
   //CvVideoWriter *writer = cvCreateVideoWriter(argv[2], f, fps, size);
   int i=0;
   char num[5], imgname[strlen(filename)+9];
   while((frame = cvQueryFrame(capture)) != NULL)
   {
      i++;
      strcpy(imgname, filename);
      /*cvShowImage("result", bgr_frame);
      cvWaitKey(0);*/
      //cvWriteFrame(writer, bgr_frame);
      // Se guardan los frames obtenidos
```

```
            sprintf(num, "%05d", i); // Número de frame con 5 cifras para poder
formar el vídeo sin problemas
            strcat(imgname, num);
            strcat(imgname, ".png");
            detectAndDraw(frame, imgname);
            //cvSaveImage(imgname, bgr_frame);
         }
         cvReleaseImage(&frame);
         cvReleaseCapture(&capture);
         return 0;
      }

      void detectAndDraw(IplImage *img,
                     char *imgname)
      {
         // Create memory for calculations
         static CvMemStorage* storage = 0;

         // Create a new Haar classifier
         static CvHaarClassifierCascade* cascade = 0;

         int scale = 1;
         double t = 0;

         // Create a new image based on the input image
         IplImage* temp = cvCreateImage( cvSize(img->width/scale,img-
>height/scale), 8, 3 );

         // Create two points to represent the urchin locations
         CvPoint pt1, pt2;
         int i;

         // Load the HaarClassifierCascade
         cascade = (CvHaarClassifierCascade*)cvLoad( cascade_name, 0, 0, 0 );

         // Check whether the cascade has loaded successfully. Else report and
error and quit
         if( !cascade )
         {
            fprintf( stderr, "ERROR: Could not load classifier cascade\n" );
            return;
         }

         // Allocate the memory storage
         storage = cvCreateMemStorage(0);

         // Create a new named window with title: result
         cvNamedWindow( "result", 1 );

         // Clear the memory storage which was used before
         cvClearMemStorage( storage );

         // Find whether the cascade is loaded, to find the urchins. If yes, then:
         if( cascade )
         {
            t = (double)cvGetTickCount();
            // There can be more than one urchin in an image. So create a growable
sequence of urchins.
            // Detect the objects and store them in the sequence
            CvSeq* urchins = cvHaarDetectObjects(img, cascade, storage,
                                                 1.2, 2, CV_HAAR_DO_CANNY_PRUNING,
```

```
                                                cvSize(70, 70), cvSize(1024,
768));
          t = (double)cvGetTickCount() - t;
          printf( "detection time = %g ms\n",
t/((double)cvGetTickFrequency()*1000.) );
          // Loop the number of urchins found.
          for( i = 0; i < (urchins ? urchins->total : 0); i++ )
          {
             // Create a new rectangle for drawing the urchin
             CvRect* r = (CvRect*)cvGetSeqElem( urchins, i );

             // Find the dimensions of the urchin and scale it, if necessary
             pt1.x = r->x*scale;
             pt2.x = (r->x+r->width)*scale;
             pt1.y = r->y*scale;
             pt2.y = (r->y+r->height)*scale;

             // Draw the rectangle in the input image
             cvRectangle( img, pt1, pt2, CV_RGB(255,0,0), 3, 8, 0 );
          }
       }

       cvSaveImage(imgname, img);
    }
```

15.3. Detectores LBP

15.3.1. detecta_erizo_probable_lbp

```
#include "opencv2/objdetect/objdetect.hpp"
#include "opencv2/highgui/highgui.hpp"
#include "opencv2/imgproc/imgproc.hpp"

#include <iostream>
#include <stdio.h>

using namespace std;
using namespace cv;

void detectAndDraw( Mat& img,
                    CascadeClassifier& cascade,
                    double scale);

String cascadeName =
"/home/fede/Escritorio/TFG/Fotos/data_LBP_16_20x20/clasif/cascade.xml";
FILE *file;

int main( int argc, const char** argv )
{
    CvCapture* capture = 0;
    Mat frame, frameCopy, image;
    const String scaleOpt = "--scale=";
    size_t scaleOptLen = scaleOpt.length();
    const String cascadeOpt = "--cascade=";
    size_t cascadeOptLen = cascadeOpt.length();
    String inputName;

    CascadeClassifier cascade;
    double scale = 1;

    if( !cascade.load( cascadeName ) )
    {
        cerr << "ERROR: Could not load classifier cascade" << endl;
        cerr << "Usage: facedetect [--cascade=<cascade_path>]\n"
            "   [--scale[=<image scale>\n"
            "   [filename|camera_index]\n" << endl ;
        return -1;
    }

    // Salida con las coordenadas
   char salida[strlen(argv[1])+strlen("_coordenadas_lbp_16")+2];
   strcpy(salida, "./");
   strcat(salida, argv[1]);
   strcat(salida, "_coordenadas_lbp_16");

   file=fopen(salida,"w");

    image = imread( argv[1], 1 );
    if(image.empty()) cout << "Couldn't read" << argv[1] << endl;

    cvNamedWindow( "result", 1 );

    if( capture )
    {
```

```
            cout << "In capture ..." << endl;
        for(;;)
        {
            IplImage* iplImg = cvQueryFrame( capture );
            frame = iplImg;
            if( frame.empty() )
                break;
            if( iplImg->origin == IPL_ORIGIN_TL )
                frame.copyTo( frameCopy );
            else
                flip( frame, frameCopy, 0 );

            detectAndDraw( frameCopy, cascade, scale );

            if( waitKey( 10 ) >= 0 )
                goto _cleanup_;
        }

        waitKey(0);
_cleanup_:
        cvReleaseCapture( &capture );
    }
    else
    {
          cout << "In image read" << endl;
        if( !image.empty() )
        {
            detectAndDraw( image, cascade, scale );
            waitKey(0);
        }
        else if( !inputName.empty() )
        {
            /* assume it is a text file containing the
            list of the image filenames to be processed - one per line */
            FILE* f = fopen( inputName.c_str(), "rt" );
            if( f )
            {
                char buf[1000+1];
                while( fgets( buf, 1000, f ) )
                {
                    int len = (int)strlen(buf), c;
                    while( len > 0 && isspace(buf[len-1]) )
                        len--;
                    buf[len] = '\0';
                    cout << "file " << buf << endl;
                    image = imread( buf, 1 );
                    if( !image.empty() )
                    {
                        detectAndDraw( image, cascade, scale );
                        c = waitKey(0);
                        if( c == 27 || c == 'q' || c == 'Q' )
                            break;
                    }
                    else
                    {
                        cerr << "Aw snap, couldn't read image " << buf <<
endl;

                    }
                }
                fclose(f);
            }
        }
```

```
        }

        cvDestroyWindow("result");

        return 0;
    }

    void detectAndDraw( Mat& img,
                        CascadeClassifier& cascade,
                        double scale)
    {
        int i = 0;
        double t = 0;
        vector<Rect> urchins;
        const static Scalar colors[] =  { CV_RGB(0,0,255),
            CV_RGB(0,128,255),
            CV_RGB(0,255,255),
            CV_RGB(0,255,0),
            CV_RGB(255,128,0),
            CV_RGB(255,255,0),
            CV_RGB(255,0,0),
            CV_RGB(255,0,255)} ;
        Mat gray, smallImg( cvRound (img.rows/scale), cvRound(img.cols/scale),
CV_8UC1 );

        cvtColor( img, gray, CV_BGR2GRAY );
        resize( gray, smallImg, smallImg.size(), 0, 0, INTER_LINEAR );
        equalizeHist( smallImg, smallImg );

        t = (double)cvGetTickCount();
        cascade.detectMultiScale( smallImg, urchins,
            1.1, 2, //0
            //|CV_HAAR_FIND_BIGGEST_OBJECT
            //|CV_HAAR_DO_ROUGH_SEARCH
            CV_HAAR_DO_CANNY_PRUNING //|CV_HAAR_SCALE_IMAGE
            ,
            Size(20,20) ); //Size(30, 30) );
        t = (double)cvGetTickCount() - t;
        printf( "detection time = %g ms\n",
t/((double)cvGetTickFrequency()*1000.) );
        for( vector<Rect>::const_iterator r = urchins.begin(); r !=
urchins.end(); r++, i++ )
        {
            Mat smallImgROI;
            Point center;
            Scalar color = colors[i%8];
            int radius;
            fprintf(file,"%d %d %d %d\n",r->x,r->y,r->x+r->width,r->y+r->height);
            center.x = cvRound((r->x + r->width*0.5)*scale);
            center.y = cvRound((r->y + r->height*0.5)*scale);
            radius = cvRound((r->width + r->height)*0.25*scale);
            circle( img, center, radius, color, 3, 8, 0 );
            smallImgROI = smallImg(*r);
        }
        fclose(file);
        cv::imshow( "result", img );
    }
```

15.3.2. detecta_erizo_probable_lbp_guarda

```
#include "opencv2/objdetect/objdetect.hpp"
#include "opencv2/highgui/highgui.hpp"
#include "opencv2/imgproc/imgproc.hpp"

#include <iostream>
#include <stdio.h>

using namespace std;
using namespace cv;

void detectAndDraw( Mat& img,
                    CascadeClassifier& cascade,
                    double scale,
                char *imgname);

String cascadeName =
"/home/fede/Escritorio/TFG/Fotos/data_LBP_16_20x20/clasif/cascade.xml";

int main( int argc, const char** argv )
{
    CvCapture* capture = 0;
    Mat frame, frameCopy, image;
    const String scaleOpt = "--scale=";
    size_t scaleOptLen = scaleOpt.length();
    const String cascadeOpt = "--cascade=";
    size_t cascadeOptLen = cascadeOpt.length();
    String inputName;

    CascadeClassifier cascade;
    double scale = 1;

    if( !cascade.load( cascadeName ) )
    {
        cerr << "ERROR: Could not load classifier cascade" << endl;
        cerr << "Usage: facedetect [--cascade=<cascade_path>]\n"
            "   [--scale[=<image scale>\n"
            "   [filename|camera_index]\n" << endl ;
        return -1;
    }

   char imgname[strlen(argv[1]) + strlen("_resultado")];

   strcpy(imgname, argv[1]);
   strcat(imgname, "_resultado");

    image = imread( argv[1], 1 );
    if(image.empty()) cout << "Couldn't read" << argv[1] << endl;

    cvNamedWindow( "result", 1 );

    if( !image.empty() )
    {
       detectAndDraw(image, cascade, scale, imgname);
       waitKey(0);
    }

    cvDestroyWindow("result");

    return 0;
```

```
    }

    void detectAndDraw( Mat& img,
                        CascadeClassifier& cascade,
                        double scale,
                    char *imgname)
    {
        int i = 0;
        double t = 0;
        vector<Rect> urchins;
        const static Scalar colors[] =  { CV_RGB(0,0,255),
            CV_RGB(0,128,255),
            CV_RGB(0,255,255),
            CV_RGB(0,255,0),
            CV_RGB(255,128,0),
            CV_RGB(255,255,0),
            CV_RGB(255,0,0),
            CV_RGB(255,0,255)} ;
        Mat gray, smallImg( cvRound (img.rows/scale), cvRound(img.cols/scale),
CV_8UC1 );

        cvtColor( img, gray, CV_BGR2GRAY );
        resize( gray, smallImg, smallImg.size(), 0, 0, INTER_LINEAR );
        equalizeHist( smallImg, smallImg );

        t = (double)cvGetTickCount();
        cascade.detectMultiScale( smallImg, urchins,
            1.1, 2, //0
            //|CV_HAAR_FIND_BIGGEST_OBJECT
            //|CV_HAAR_DO_ROUGH_SEARCH
            CV_HAAR_DO_CANNY_PRUNING //|CV_HAAR_SCALE_IMAGE
            ,
            Size(20,20) ); //Size(30, 30) );
        t = (double)cvGetTickCount() - t;
        printf( "detection time = %g ms\n",
t/((double)cvGetTickFrequency()*1000.) );
        for( vector<Rect>::const_iterator r = urchins.begin(); r !=
urchins.end(); r++, i++ )
        {
            Mat smallImgROI;
            Point center;
            Scalar color = colors[i%8];
            int radius;
            center.x = cvRound((r->x + r->width*0.5)*scale);
            center.y = cvRound((r->y + r->height*0.5)*scale);
            radius = cvRound((r->width + r->height)*0.25*scale);
            circle( img, center, radius, color, 3, 8, 0 );
            smallImgROI = smallImg(*r);
        }
        cv::imshow( "result", img );
        IplImage *image;
        image = &IplImage(img);
        cvSaveImage(imgname, image);
}
```

15.3.3. detecta_erizo_diadema_lbp

```
#include "opencv2/objdetect/objdetect.hpp"
#include "opencv2/highgui/highgui.hpp"
#include "opencv2/imgproc/imgproc.hpp"

#include <iostream>
#include <stdio.h>

using namespace std;
using namespace cv;

void detectAndDraw( Mat& img,
                    CascadeClassifier& cascade,
                    double scale);

String cascadeName =
"/home/fede/Escritorio/TFG/Fotos/data_LBP_20_70x70/clasif/cascade.xml";
FILE *file;

int main( int argc, const char** argv )
{
   CvCapture* capture = 0;
   Mat frame, frameCopy, image;
   const String scaleOpt = "--scale=";
   size_t scaleOptLen = scaleOpt.length();
   const String cascadeOpt = "--cascade=";
   size_t cascadeOptLen = cascadeOpt.length();
   String inputName;

   CascadeClassifier cascade;
   double scale = 1;

   if( !cascade.load( cascadeName ) )
   {
      cerr << "ERROR: Could not load classifier cascade" << endl;
      cerr << "Usage: facedetect [--cascade=<cascade_path>]\n"
          "   [--scale[=<image scale>\n"
          "   [filename|camera_index]\n" << endl ;
      return -1;
   }

   // Salida con las coordenadas
   char salida[strlen(argv[1])+strlen("_coordenadas_lbp")+2];
   strcpy(salida, "./");
   strcat(salida, argv[1]);
   strcat(salida, "_coordenadas_lbp.txt");

   file=fopen(salida,"w");

   image = imread( argv[1], 1 );
   if(image.empty()) cout << "Couldn't read" << argv[1] << endl;

   cvNamedWindow( "result", 1 );

   if( capture )
   {
          cout << "In capture ..." << endl;
       for(;;)
       {
           IplImage* iplImg = cvQueryFrame( capture );
```

```
                frame = iplImg;
                if( frame.empty() )
                    break;
                if( iplImg->origin == IPL_ORIGIN_TL )
                    frame.copyTo( frameCopy );
                else
                    flip( frame, frameCopy, 0 );

                detectAndDraw( frameCopy, cascade, scale );

                if( waitKey( 10 ) >= 0 )
                    goto _cleanup_;
            }

            waitKey(0);
    _cleanup_:
            cvReleaseCapture( &capture );
        }
        else
        {
              cout << "In image read" << endl;
            if( !image.empty() )
            {
                detectAndDraw( image, cascade, scale );
                waitKey(0);
            }
            else if( !inputName.empty() )
            {
                /* assume it is a text file containing the
                list of the image filenames to be processed - one per line */
                FILE* f = fopen( inputName.c_str(), "rt" );
                if( f )
                {
                    char buf[1000+1];
                    while( fgets( buf, 1000, f ) )
                    {
                        int len = (int)strlen(buf), c;
                        while( len > 0 && isspace(buf[len-1]) )
                            len--;
                        buf[len] = '\0';
                        cout << "file " << buf << endl;
                        image = imread( buf, 1 );
                        if( !image.empty() )
                        {
                            detectAndDraw( image, cascade, scale );
                            c = waitKey(0);
                            if( c == 27 || c == 'q' || c == 'Q' )
                                break;
                        }
                        else
                        {
                            cerr << "Aw snap, couldn't read image " << buf <<
endl;
                        }
                    }
                    fclose(f);
                }
            }
        }

        cvDestroyWindow("result");
```

```
        return 0;
    }

    void detectAndDraw( Mat& img,
                        CascadeClassifier& cascade,
                        double scale)
    {
        int i = 0;
        double t = 0;
        vector<Rect> urchins;
        const static Scalar colors[] =  { CV_RGB(0,0,255),
            CV_RGB(0,128,255),
            CV_RGB(0,255,255),
            CV_RGB(0,255,0),
            CV_RGB(255,128,0),
            CV_RGB(255,255,0),
            CV_RGB(255,0,0),
            CV_RGB(255,0,255)} ;
        Mat gray, smallImg( cvRound (img.rows/scale), cvRound(img.cols/scale),
CV_8UC1 );

        cvtColor( img, gray, CV_BGR2GRAY );
        resize( gray, smallImg, smallImg.size(), 0, 0, INTER_LINEAR );
        equalizeHist( smallImg, smallImg );

        t = (double)cvGetTickCount();
        cascade.detectMultiScale( smallImg, urchins,
            1.5, 2, 0
            //|CV_HAAR_FIND_BIGGEST_OBJECT
            //|CV_HAAR_DO_ROUGH_SEARCH
            //|CV_HAAR_SCALE_IMAGE
       //|CV_HAAR_DO_CANNY_PRUNING
            ,
            Size(70, 70) );
        t = (double)cvGetTickCount() - t;
        printf( "detection time = %g ms\n",
t/((double)cvGetTickFrequency()*1000.) );
        for( vector<Rect>::const_iterator r = urchins.begin(); r !=
urchins.end(); r++, i++ )
        {
            Mat smallImgROI;
            Point center;
            Scalar color = colors[i%8];
            int radius;
            fprintf(file,"%d %d %d %d\n",r->x,r->y,r->x+r->width,r->y+r->height);
            center.x = cvRound((r->x + r->width*0.5)*scale);
            center.y = cvRound((r->y + r->height*0.5)*scale);
            radius = cvRound((r->width + r->height)*0.25*scale);
            circle( img, center, radius, color, 3, 8, 0 );
            smallImgROI = smallImg(*r);
        }
        fclose(file);
        cv::imshow( "result", img );
    }
```

15.3.4. detecta_erizo_diadema_lbp_guarda

```
#include "opencv2/objdetect/objdetect.hpp"
#include "opencv2/highgui/highgui.hpp"
#include "opencv2/imgproc/imgproc.hpp"

#include <iostream>
#include <stdio.h>

using namespace std;
using namespace cv;

void detectAndDraw( Mat& img,
                    CascadeClassifier& cascade,
                    double scale,
                char *imgname);

String cascadeName =
"/home/fede/Escritorio/TFG/Fotos/data_LBP_20_70x70/clasif/cascade.xml";

int main( int argc, const char** argv )
{
   CvCapture* capture = 0;
   Mat frame, frameCopy, image;
   const String scaleOpt = "--scale=";
   size_t scaleOptLen = scaleOpt.length();
   const String cascadeOpt = "--cascade=";
   size_t cascadeOptLen = cascadeOpt.length();
   String inputName;

   CascadeClassifier cascade;
   double scale = 1;

   if( !cascade.load( cascadeName ) )
   {
      cerr << "ERROR: Could not load classifier cascade" << endl;
      cerr << "Usage: facedetect [--cascade=<cascade_path>]\n"
          "   [--scale[=<image scale>\n"
          "   [filename|camera_index]\n" << endl ;
      return -1;
   }

   char imgname[strlen(argv[1]) + strlen("_resultado")];

   strcpy(imgname, argv[1]);
   strcat(imgname, "_resultado");

   image = imread( argv[1], 1 );
   if(image.empty()) cout << "Couldn't read" << argv[1] << endl;

   cvNamedWindow( "result", 1 );

   if( !image.empty() )
   {
      detectAndDraw(image, cascade, scale, imgname);
      waitKey(0);
   }

    cvDestroyWindow("result");

    return 0;
```

```
    }

    void detectAndDraw( Mat& img,
                        CascadeClassifier& cascade,
                        double scale,
                    char *imgname)
    {
        int i = 0;
        double t = 0;
        vector<Rect> urchins;
        const static Scalar colors[] =  { CV_RGB(0,0,255),
            CV_RGB(0,128,255),
            CV_RGB(0,255,255),
            CV_RGB(0,255,0),
            CV_RGB(255,128,0),
            CV_RGB(255,255,0),
            CV_RGB(255,0,0),
            CV_RGB(255,0,255)} ;
        Mat gray, smallImg( cvRound (img.rows/scale), cvRound(img.cols/scale),
CV_8UC1 );

        cvtColor( img, gray, CV_BGR2GRAY );
        resize( gray, smallImg, smallImg.size(), 0, 0, INTER_LINEAR );
        equalizeHist( smallImg, smallImg );

        t = (double)cvGetTickCount();
        cascade.detectMultiScale( smallImg, urchins,
            1.5, 2, 0
            //|CV_HAAR_FIND_BIGGEST_OBJECT
            //|CV_HAAR_DO_ROUGH_SEARCH
            //|CV_HAAR_SCALE_IMAGE
       //|CV_HAAR_DO_CANNY_PRUNING
            ,
            Size(70, 70) );
        t = (double)cvGetTickCount() - t;
        printf( "detection time = %g ms\n",
t/((double)cvGetTickFrequency()*1000.) );
        for( vector<Rect>::const_iterator r = urchins.begin(); r !=
urchins.end(); r++, i++ )
        {
            Mat smallImgROI;
            Point center;
            Scalar color = colors[i%8];
            int radius;
            center.x = cvRound((r->x + r->width*0.5)*scale);
            center.y = cvRound((r->y + r->height*0.5)*scale);
            radius = cvRound((r->width + r->height)*0.25*scale);
            circle( img, center, radius, color, 3, 8, 0 );
            smallImgROI = smallImg(*r);
        }
        cv::imshow( "result", img );
        IplImage *image;
        image = &IplImage(img);
        cvSaveImage(imgname, image);
    }
```

15.3.5. Video_probable

```
    #include <cv.h>
    #include <highgui.h>
    #include "opencv2/objdetect/objdetect.hpp"
    #include "opencv2/highgui/highgui.hpp"
    #include "opencv2/imgproc/imgproc.hpp"
    #include <iostream>
    #include <stdio.h>

    using namespace std;
    using namespace cv;

    void detectAndDraw( Mat& img,
                        CascadeClassifier& cascade,
                        double scale,
                    char * imgname);

    String cascadeName =
"/home/fede/Escritorio/TFG/Fotos/data_LBP_16_20x20/clasif.xml";

    int main(int argc, char *argv[])
    {
        CvCapture* capture = 0;
        Mat frame2, frameCopy, image;
        const String scaleOpt = "--scale=";
        size_t scaleOptLen = scaleOpt.length();
        const String cascadeOpt = "--cascade=";
        size_t cascadeOptLen = cascadeOpt.length();
        String inputName;

        CascadeClassifier cascade;
        double scale = 1;

        if( !cascade.load( cascadeName ) )
        {
            cerr << "ERROR: Could not load classifier cascade" << endl;
            /*cerr << "Usage: facedetect [--cascade=<cascade_path>]\n"
                "   [--scale[=<image scale>\n"
                "   [filename|camera_index]\n" << endl ;*/
            return -1;
        }

       if(argc != 2) fprintf(stderr, "Falta la ruta del video\n");

       char filename[strlen(argv[1]) + strlen("_resultado_")];

       strcpy(filename, argv[1]);
       strcat(filename, "_resultado_");

       // Se abre el video
       capture = cvCreateFileCapture(argv[1]);

       // Si no se puede se muestra un mensaje por pantalla y se termina la
aplicación
       if(!capture)
       {
         cout << "ERROR -> No se ha podido abrir el video" << endl;
         return -1;
       }
```

```
      // Comienza la lectura del video
      IplImage *frame = cvQueryFrame(capture);
      //double fps = cvGetCaptureProperty(capture, CV_CAP_PROP_FPS), f =
cvGetCaptureProperty(capture, CV_CAP_PROP_FOURCC);
      //CvSize size = cvSize((int)cvGetCaptureProperty(capture,
CV_CAP_PROP_FRAME_WIDTH),
      //                     (int)cvGetCaptureProperty(capture,
CV_CAP_PROP_FRAME_HEIGHT));
      //char *fourcc = (char *) (&f);
      //CvVideoWriter *writer = cvCreateVideoWriter(argv[2], f, fps, size);
      int i=0;
      char num[5], imgname[strlen(filename)+9];
      while((frame = cvQueryFrame(capture)) != NULL)
      {
         i++;
         frame2 = frame;
         frame2.copyTo( frameCopy );
         strcpy(imgname, filename);
         /*cvShowImage("result", bgr_frame);
         cvWaitKey(0);*/
         //cvWriteFrame(writer, bgr_frame);
         // Se guardan los frames obtenidos
         sprintf(num, "%05d", i); // Numero de frame con 5 cifras para poder
formar el video sin problemas
         strcat(imgname, num);
         strcat(imgname, ".png");
         detectAndDraw(frameCopy, cascade, scale, imgname);
         //cvSaveImage(imgname, bgr_frame);
      }
      cvReleaseImage(&frame);
      cvReleaseCapture(&capture);
      return 0;
   }

   void detectAndDraw(Mat& img,
                      CascadeClassifier& cascade,
                      double scale,
                  char *imgname)
   {
       int i = 0;
       vector<Rect> urchins;
       const static Scalar colors[] =  { CV_RGB(0,0,255),
           CV_RGB(0,128,255),
           CV_RGB(0,255,255),
           CV_RGB(0,255,0),
           CV_RGB(255,128,0),
           CV_RGB(255,255,0),
           CV_RGB(255,0,0),
           CV_RGB(255,0,255)} ;
       Mat gray, smallImg( cvRound (img.rows/scale), cvRound(img.cols/scale),
CV_8UC1 );

       cvtColor( img, gray, CV_BGR2GRAY );
       resize( gray, smallImg, smallImg.size(), 0, 0, INTER_LINEAR );
       equalizeHist( smallImg, smallImg );

       cascade.detectMultiScale( smallImg, urchins,
           1.1, 2, //0
           //|CV_HAAR_FIND_BIGGEST_OBJECT
           //|CV_HAAR_DO_ROUGH_SEARCH
           CV_HAAR_DO_CANNY_PRUNING //|CV_HAAR_SCALE_IMAGE
           ,
```

```
            Size(20,20) ); //Size(30, 30) );
        for( vector<Rect>::const_iterator r = urchins.begin(); r != 
urchins.end(); r++, i++ )
        {
            Mat smallImgROI;
            Point center;
            Scalar color = colors[i%8];
            int radius;
            center.x = cvRound((r->x + r->width*0.5)*scale);
            center.y = cvRound((r->y + r->height*0.5)*scale);
            radius = cvRound((r->width + r->height)*0.25*scale);
            circle( img, center, radius, color, 3, 8, 0 );
            smallImgROI = smallImg(*r);
        }
        IplImage *image;
        image = &IplImage(img);
        cvSaveImage(imgname, image);
    }
```

15.4. SVM, PCA y Distancia Euclídea

15.4.1. GeneraMuestrasRecortadasEntrenamiento

```
    TAM=100;
    fid=fopen('C:\\Users\\Fede\\Documents\\TFG\\Fotos\\diadema\\parte\\
21.txt','r');

    if fid>-1
        while ~feof(fid)
            imgname=fscanf(fid,'%s ',1);

            %Lectura del nombre de la imagen
            imgnamescaled = sprintf('%s_scaled_',imgname(1:end-4));

            %Número de muestras positivas
            nsamples=fscanf(fid,'%d ',1);

            %Contenedor
            for i=1:nsamples
                x(i)=fscanf(fid,'%d ',1);
                y(i)=fscanf(fid,'%d ',1);
                w(i)=fscanf(fid,'%d ',1);
                h(i)=fscanf(fid,'%d ',1);

                %Fuerzo contenedor cuadrado
                menor=w(i);
                if h(i)<menor
                    menor=h(i);
                end

                x(i)=(x(i)+w(i)*0.5)-menor/2;
                y(i)=(y(i)+h(i)*0.5)-menor/2;

                w(i)=menor;
                h(i)=menor;
            end

            %Recorte y escalado de la zona de la imagen con la muestra
            img = imread(imgname);

            for i=1:nsamples
                img2 = imcrop(img,[x(i) y(i) w(i) h(i)]);

                ratio = TAM/(w(i)+1.1);

                img3 = imresize(img2,ratio,'bilinear');

                thisimgnamescaled =
sprintf('%s%i.png',imgnamescaled,i);

                imwrite(img3,thisimgnamescaled,'png');
            end
        end
        fclose(fid);
    end
```

15.4.2. GeneraMuestrasRecortadasTests

```
   TAM=100;

   fid=fopen('C:\\Users\\Fede\\Documents\\TFG\\Pruebas
diadema\\12.jpg_coordenadas_lbp.txt','r');

   if fid>-1
       i=1;
       while ~feof(fid)

           imgname='C:\\Users\\Fede\\Documents\\TFG\\Pruebas
diadema\\12.jpg';

           %Lectura del nombre de la imagen
           imgnamescaled = sprintf('%s_scaled_',imgname(1:end-4));

           x(i)=fscanf(fid,'%d ',1);
           y(i)=fscanf(fid,'%d ',1);
           x2(i)=fscanf(fid,'%d ',1);
           y2(i)=fscanf(fid,'%d ',1);

           %Fuerzo contenedor cuadrado
           w(i)=x2(i)-x(i);
           h(i)=y2(i)-y(i);
           menor=w(i);
           if h(i)<menor
               menor=h(i);
           end

           x(i)=(x(i)+w(i)*0.5)-menor/2;
           y(i)=(y(i)+h(i)*0.5)-menor/2;

           w(i)=menor;
           h(i)=menor;

           %Recorte y escalado de la zona de la imagen con la muestra
           img = imread(imgname);

           img2 = imcrop(img,[x(i) y(i) w(i) h(i)]);

           ratio = TAM/(w(i)+1.1);

           img3 = imresize(img2,ratio,'bilinear');

           thisimgnamescaled =
sprintf('%slbp_%i.png',imgnamescaled,i);

           imwrite(img3,thisimgnamescaled,'png');

           i=i+1;
       end
       fclose(fid);
   end
```

15.4.3. DemoPCASVM

```
   % Ejemplo de clasificación con PCA y SVM
   % a partir de los datos proporcionados por una base de imágenes
   %
   % Modesto Castrillón, Julio 2011

   function res = DemoPCASVM(TrainPath, TestPath, PCAFile)

   %Cargamos el conjunto de ENTRENAMIENTO desde la ruta TrainPath
   % (ojo todas las imágenes deben tener el mismo tamaño

   %X es la matriz con las muestras leídas, conteniendo un dato en
cada
   %columna
   %clasedeimagenTrain la clase de cada muestra en formato cadena
   %nclasedeimagenTrain la clase asignada de forma numérica (por orden
   %alfabético)
   %nTrain el número de muestras
   %ifil icol las dimensiones de las muestras asumiento que son
imágenes
   [X, clasedeimagen, clasedeimagennumerica, nombreficheroTrain,
nImgs, ifil, icol] = LeeMuestras(TrainPath);

   disp('Imágenes de entreno cargadas');


       %Ajusto conjunto de entrenamiento
       XTrain=X;
       claseimagenTrain=clasedeimagen;
       clasedeimagennumericaTrain=clasedeimagennumerica;
       nTrain=nImgs;

       %Cargamos el conjunto de TEST (ojo todas las imágenes deben
tener
       %el mismo tamaño y coincidir con el tamaño de entreno
       [Xtest, clasedeimagenTest, clasedeimagennumericaTest,
nombreficheroTest, nTest, ifil2, icol2] = LeeMuestras(TestPath);


       %Si coinciden las dimensiones de las imágenes de entreno y
test, realiza el test
       if icol==icol2 && ifil == ifil2

PruebaRendimiento(XTrain,clasedeimagennumericaTrain,Xtest,clasedeimage
nnumericaTest,ifil,icol,PCAFile, nombreficheroTrain,
nombreficheroTest);
       else
           fprintf('No coinciden los tamaños de las imágenes de
entreno y test\n');
       end


   res=1;
```

15.4.4. LeeMuestras

```
   function [X,clasedeimagen, nclasedeimagen,
nombrefichero,Samples_Number, ifil, icol] = LeeMuestras(DatabasePath)
   % Obtiene para un conjunto de entrenamiento su PCA
   %
   % Descripción: A partir de la ruta de las imágenes las carga
componiendo
   % una matriz donde cada columna es una imagen
   %
   % Argumentos:    DatabasePath           - Localizacióon de la
simágenes
   %
   % Returns:       X                      - Matriz 2D conteniendo
todas las
   %                imágenes de entrenamiento (cada una de tamaño
M*N). Para P
   %                imágenes de entrenamiento tendrá P columnas de
M*Nc
   %                clasedeimagen          - Clase asociada a la
imagen
   %                nclasedeimagen         - Clase numérica asociada a
la imagen
   %                Samples_Number         - Número de muestras
   %                ifil, icol             - Dimensiones de las
imágenes



   %%%%%%%%%%%%%%%%%%%%%%%% Busca imágenes disponibles en la carpeta
indicada
   %%%%%%%%%%%%%%%%%%%%%%%% de entrenamiento
   Classes = dir(DatabasePath); %identificamos las carpetas en el
directorio pasado
   Samples_Number = 0;   %Inicialmente no tenemos ninguna imagen de
entrenamiento

   X = [];              %Matriz de imágenes en forma vectorial

   nclases=0;

   %Asumimos que en la carpeta suministrada habrá una carpeta
   %por clase y dentro de cada carpeta habrá imágenes de igual
dimensión todas
   %ellas
   for j = 1:size(Classes,1)

       %Si parece una carpeta asumimos que indica una clase
       if
not(strcmp(Classes(j).name,'.')|strcmp(Classes(j).name,'..')|strcmp(Cl
asses(j).name,'Thumbs.db'))

           %Compone ruta de la carpeta que debe contener los ficheros
de la
           %correspondiente clase
           clases = strcat(DatabasePath,'\',Classes(j).name);

           nclases=nclases+1;%una nueva clase
```

```
            %%Obtiene los ficheros en dicha carpeta, se espera que sean
            %%imágenes, por ello primero recoloca el path acutal
            Files = dir(clases);

            %Recorre todos los ficheros encontrados en la carpeta
            for i = 1:size(Files,1)

                %Incrementa el número de imágenes disponibles excepto
si fuera . ó .. ó Thumbs.db añadiendo el
                %vector de la imagen correspondiente a la matriz T
                if
not(strcmp(Files(i).name,'.')|strcmp(Files(i).name,'..')|strcmp(Files(
i).name,'Thumbs.db'))

                    %Incrementa el número de imágenes hasta el momento
                    Samples_Number = Samples_Number + 1; % Número de
imagenes en el conjunto de entrenamiento

                    %Compone ruta del fichero de imagen a leer
                    str = strcat(clases,'\',Files(i).name);

                    %Leemos la imagen y se convierte a tonos de grises
                    img = imread(str);
                    img = rgb2gray(img);
                    %Obtenemos las dimensiones de la imagen
                    [ifil icol] = size(img);

                    %EN ESTE PUNTO SE PODRÍA APLICAR ALGÚN TIPO DE
                    %PROCESAMIENTO SOBRE LA IMAGEN COMO POR EJHEMPLO
LBP O
                    %GABOR


                    %FIN TRANSFORMACIONES POSIBLES

                    %Transformamos la imagen 2D a un vector 1D que nos
será
                    %útil para calcular PCA
                    temp = reshape(img,ifil*icol,1);  % muestra por
columna

                    %Añadimos a la matriz de imágenes 1D por el final
                    X = [X temp];  %Cada columna tiene un dato, es
decir una imagen

                    %temp = reshape(img,1,ifil*icol);  % muestra por
fila (para
                    %cálculo por SVD)

                     %Añadimos a la matriz de imágenes 1D por el final
                    %X = [X ; temp];  %Cada fila tiene un dato, es
decir una imagen



                    %Almacenamos la clase de cada muestra (asumimos que
se
```

```
                    %indica por la carpeta en la que está)
clasedeimagen(Samples_Number)=cellstr(sprintf('%s',Classes(j).name));
                    nombrefichero(Samples_Number)=cellstr(str);
                    nclasedeimagen(Samples_Number)=nclases;
                end
            end
        end
    end
```

15.4.5. PruebaRendimiento

```
   function [ratio_grises,ratio_pca,ratio_pcasvm] =
PruebaRendimiento(Xtrain,clasedeimagenTrain,Xtest,clasedeimagenTest,if
il,icol,PCAFile,nombreficheroTrain, nombreficheroTest)
   %
   %
   % Descripción: A partir de la ruta de las imágenes de
entrenamiento, crea
   % la matriz donde cada columna es una imagen de entreno y calcula
su PCA, así
   % como la proyección de las imágenes de entreno.
   %
   % Argumentos:    Xtrain                - Conjunto de imágenes
usadas de entreno
   %                clasedeimagenTrain    - Etiquetas imágenes de
entreno en
   %                formato numérico
   %                Xtest                 - Conjunto de imágenes
usadas para test
   %                clasedeimagenTest     - Etiquetas imágenes de test
en
   %                formato numérico
   %                ifil                  - Filas de las imágenes de
entrada
   %                icol                  - Columnas de las imágenes
de entrada
   %
   % Returns:       ratio_grises          - Ratio de aciertos obtenido
con
   %                                        1NN en espacio de grises
   %                ratio_pca             - Ratio de aciertos obtenido
con
   %                                        1NN en espacio proyectado
PCA
   %                ratio_pcasvm          - Ratio de aciertos obtenido
con
   %                                        un clasificador SVM en el
espacio
   %                                        PCA
   %




   %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

   MuestraPCA=0;%Si es mayor que 0 se muestran las caras principales
   AplicaSVM=1;%Si es mayor que 0 se aplica el test svm
   AvisaFallos=1;

   %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
   %CALCULAMOS PCA
   %Carga de fichero con datos PCA si existe si existe
    if exist(PCAFile,'file')
        load(PCAFile);
    else%En caso de no existir lo calcula
```

```
        %Generamos la matriz con las imágenes de entrenamiento, la
imagen promedio y el PCA para los datos de entreno
        [media, Eigenfaces, A, AP] = GeneraPCA(Xtrain);

        %Salvamos el cómputo de componentes principales, para evitar
        %recalcularlo varias veces durante las pruebas si no fuera
necesario
        save(PCAFile,'media','Eigenfaces','A','AP');

        if MuestraPCA>0
            %Visualizamos la media
            %Recoloca la media de 1D a 2D antes de vidualizarla
            media2D=reshape(media,ifil,icol);
            %Convertimos a uint8
            imshow(uint8(media2D));

            pause(0.2);

            %Visualizamos las caras principales una a una
            [dim ncaras]=size(Eigenfaces);

            for i=1:ncaras
                %Recoloca la media de 1D a 2D antes de visualizarla
                eigen2D=reshape(Eigenfaces(:,i),ifil,icol);
                %Convertimos a uint8
                imshow(uint8(eigen2D));
                pause(0.2);
            end
        end
     end


    %Obtenemos el número de  muestras de cada conjunto
    nTest=size(Xtest,2);
    nTrain=size(Xtrain,2);

    %Proyección de las imágenes de entreno tras restarles la imagen
promedio
    for j=1:nTrain
        Diferencia = double(Xtrain(:,j))-media;
        Xtrain_proyectada(:,j)=Eigenfaces'*Diferencia;
    end



    %Hacemos el test para cada muestra en el conjunto de test
    %Inicializamos aciertos y fallos
    Aciertos=0;
    Fallos=0;
    AciertosPCA=0;
    FallosPCA=0;


    %Con cada muestra de test
    for j = 1:nTest
```

```
        %CLASIFICACIÓN

        %%%%%%%%%%%%%%%%%%%%
        %Haciendo uso de la DISTANCIA EUCLIDEA EN EL ESPACIO DE LOS
PIXELES
        % Obtiene la distancia euclídea con respecto a cada imagen de
entreno.
        % Tomaremos como de identificación aquella que proporcione
menor distancia
        %Pueden usarse otras  medidas de distancia

        Dist = [];
        for i = 1 : nTrain
            temp = ( norm( double(Xtest(:,j) - Xtrain(:,i)) ) )^2;
            Dist = [Dist temp];
        end

        %Obtiene el índice del mínimo
        [min_dist, min_index] = min(Dist);

        %Compara la etiqueta del mínimo con la actual, incrementando
aciertos
        %si coinciden
        if clasedeimagenTest(j)==clasedeimagenTrain(min_index)
            Aciertos=Aciertos+1;
        else
            Fallos=Fallos+1;
            if AvisaFallos>0
                fprintf('Falla con una imagen marcada como %0d y
clasificada como %0d (Distancia euclídea en
píxeles)\n',clasedeimagenTest(j),clasedeimagenTrain(min_index));
            end
        end

        %%%%%%%%%%%%%%%%%%%%
        %Haciendo uso de la DISTANCIA EUCLIDEA EN ESPACIO PCA
        %%%%%%Proyectamos al espacio PCA la imagen de test
        %Proyectamos tras restarle la media, y poniendo en forma de
        %columna para permitir el producto
        Diferencia = double(Xtest(:,j))-media;
        ImagenTestProyectada=(Eigenfaces'*Diferencia);

        % Obtiene la distancia euclídea de l aimagen de test proyectada
con respecto a cada imagen de entreno.
        % Tomaremos como de identificación aquella que proporcione
menor distancia
        %Pueden usarse otras  medidas de distancia
          Dist = [];
        for i = 1 : nTrain
            temp = ( norm( ImagenTestProyectada -
Xtrain_proyectada(:,i) ) )^2;
            Dist = [Dist temp];
        end

        %Obtiene el índice del mínimo
        [min_dist , min_index] = min(Dist);

        %Compara la etiqueta del mínimo con la actual, incrementando
aciertos
```

```
        %si coinciden
        if clasedeimagenTest(j)==clasedeimagenTrain(min_index)
            AciertosPCA=AciertosPCA+1;
        else
            FallosPCA=FallosPCA+1;
            if AvisaFallos>0
                fprintf('Falla con una imagen marcada como %0d y
clasificada como %0d (Distancia euclídea en
PCA)\n',clasedeimagenTest(j),clasedeimagenTrain(min_index));
            end
        end


    end

    %Sumario para las variantes basadas en la distancia euclídea
    fprintf('\nEsquema basado en píxeles+euclídea (1-NN):\n\tAciertos
%d, fallos %d  %.1f%%\n',Aciertos,Fallos, Aciertos/nTest*100);
    fprintf('\nEsquema PCA+euclídea (1-NN):\n\tAciertos %d, fallos %d
%.1f%%\n',AciertosPCA,FallosPCA, AciertosPCA/nTest*100);

    %Haciendo uso de la REPRESENTACIÓN PCA PERO CLASIFICANDO CON SVM
    %NORMALIZACIÓN NECESARIA PARA SVM
    %Cada COLUMNA de la matriz AP se corresponde con las
características en el
    %espacio PCA calculado de las muestras de entrenamiento.
    %Para entrenar el SVM es necesario normalizar las características
al
    %intervalo (-1,1). Por esta razón obtenemos el mínimo y máximo para
cada
    %característica, es decir, para cada fila.
    if AplicaSVM>0
        %inicializo la  matriz de test
        Xtest_proyectada_normalizada = [];

        [nfeatures nmuestras]=size(Xtrain_proyectada);

        for i=1:nfeatures
            minc(i)=min(Xtrain_proyectada(:,i));
            maxc(i)=max(Xtrain_proyectada(:,i));
        end

        %Posteriormente normalizamos las distintas carcaterísticas de
las imágenes de entrenamiento
        %haciendo uso de los valores mínimo y máximo, almacenando los
valores normalizados en APn
        for j=1:nmuestras
            for i=1:nfeatures

Xtrain_proyectada_normalizada(i,j)=(Xtrain_proyectada(i,j)-
minc(i))/(maxc(i)-minc(i));
            end
        end

        %Hacemos lo propio con las muestras de test
        for j = 1:nTest
            %Proyectamos la imagen de test tras restarle la media, y
poniendo en forma de
            %columna para permitir el producto
```

```
            Diferencia = double(Xtest(:,j))-media;
            ImagenTestProyectada=(Eigenfaces'*Diferencia);

             %%%%%%%%%%%%%%%%%%%%%%
            %Normalizamos
            %De igual forma que al entrenar, la representación
proyectada debe
            %normalizarse al intervalo (0,1) para realizar la
clasificación con el
            %modelo SVM
            %Cada muestra (imagen) ocupa una columna)
            for i=1:nfeatures

Xtest_proyectada_normalizada(i,j)=(ImagenTestProyectada(i)-
minc(i))/(maxc(i)-minc(i));
            end

        end

        %Proporcionamos las etiquetas y los valores de cada muestra
para entrenar
        %el clasificador SVM
        %Las etiquetas deben ser numéricas, por ello realicé la
conversión en la
        %lectura de muestras
        modelo100 = svmtrain(clasedeimagenTrain',
Xtrain_proyectada_normalizada', '-c 100 -g 0.07 -t 0');

        %De golpe realizamos los cálculos para svm con el modelo
entrenado
        fprintf('\nResultados basado en PCA+SVM:\n\t');
        [eti100,acc100,prob100] =
svmpredict(clasedeimagenTest',Xtest_proyectada_normalizada',modelo100)
;

        length(find(eti100(1:22)>1))
        length(find(eti100(22:end)==1))

         modelo1 = svmtrain(clasedeimagenTrain',
Xtrain_proyectada_normalizada', '-c 1 -g 0.07 -t 0');

        %De golpe realizamos los cálculos para svm con el modelo
entrenado
        fprintf('\nResultados basado en PCA+SVM:\n\t');
        [eti1,acc1,prob1] =
svmpredict(clasedeimagenTest',Xtest_proyectada_normalizada',modelo1);

        length(find(eti1(1:22)>1))
        length(find(eti1(22:end)==1))

    end

    %Una variante que permite visualizar la curva ROC es hacer uso de
plotroc
    if AplicaSVM>0
        fprintf('\nCurvas ROC:\n');
```

```
        %Fuerzo que se muestren todas las curvas en la misma gráfica
        hold on;
        colores=hsv(6); %Genero seis colores

        %Realizo el experimento sobre el conjunto completo de entrada
        clasedeimagen = [clasedeimagenTrain clasedeimagenTest];
        X_proyectada_normalizada= [Xtrain_proyectada_normalizada
Xtest_proyectada_normalizada];

        %Este comando requiere un problema con dos clases cuyas
etiquetas deben ser 1 y -1
        %Para este ejemplo, asumo que las etiquetas son de dos clases:
con
        %etiquetas 1 y 2. Recogemos los índices de lo que no son 1 y
los
        %sobreescribimos con -1
        idnouno=find(clasedeimagen~=1); %localizamos las etieuatas
distintas de 1
        clasedeimagen(idnouno)=-1;% las sustituyo por -1

        %El parámetro v indica el número de carpetas (k-fold)
        plotroc(clasedeimagen', X_proyectada_normalizada', '-c 1 -g
0.07 -v 3',colores(4,:));

        plotroc(clasedeimagen', X_proyectada_normalizada', '-c 100 -g
0.07 -v 3',colores(1,:));

        legend('PCA+SVM c 1', 'PCA+SVM c 100','Location','SouthEast');
        title('Curvas ROC');
        hold off;

        %el comando saveas permite salvar la imagen a disco
    end
```

15.4.6. GeneraPCA

```
function [media, Eigenfaces, A, AP] = GeneraPCA(T)
% Obtiene para un conjunto de entrenamiento su PCA
%
% Descripción: A partir de la ruta de las imágenes de
entrenamiento, crea
% la matriz donde cada columna es una imagen de entreno y calcula
su PCA, así
% como la proyección de las imágenes de entreno.
%
% Argumentos:     T                       - Matriz 2D conteniendo
todas las
%                 imágenes de entrenamiento (cada una de tamaño
M*N). Para P
%                 imágenes de entrenamiento tendrá P columnas de M*N
%
% Returns:        m                       - (M*Nx1) Media de la base
de entrenamiento
%                 Eigenfaces              - (M*Nx(P-1)) Vectores
principales de la matriz de covarianza del conjunto de entrenamiento
%                 A                       - (M*NxP) Matriz de
vectores de imágenes centradas
%                 AP                      - Matriz de vectores de
imágenes de
%                 entrada proyectadas en este espacio PCA%

%    Adaptado de:
%    'Eigenface' Face Recognition System
%    Written by: Amir Hossein Omidvarnia
%    Email: aomidvar@ece.ut.ac.ir



%%%%%%%%%%%%%%%%%%%%%%%% Calculamos la imagen media a  partir de
las
%%%%%%%%%%%%%%%%%%%%%%%% imágenes de entrenamiento, cada imagen una
columna
media = mean(T,2);

%El número de muestras de entrenamiento es el número de columnas
N_Entrenamiento = size(T,2);

%%%%%%%%%%%%%%%%%%%%%%%% Calculamos la diferencia de cada imagen
con
%%%%%%%%%%%%%%%%%%%%%%%% respecto a la media
A = [];  %Inicializamos la matriz
for i = 1 : N_Entrenamiento
    temp = double(T(:,i)) - media; % Cálculo de la imagen de
diferencias (cada imagen es una columna)
    A = [A temp]; % Se van adjuntando al final de A como nueva
columna
end



%%%%%%%%%%%%%%%%%%%%%%%% Snapshot method of Eigenface methos
% El álgebra dice que para una matriz PxQ, el número máximo
% de valores principales no nulos posibles es min(P-1,Q-1).
```

```
   % Dado que el número de imágenes de entrenamiento (P) es
normalmente menor
   % que el de píxeles (M*N), el máximo de valores principales no
nulos es
   % P-1. Si calculamos los valores principales de A'*A (matriz de
dimensión PxP)
   % en lugar de los de A*A' (dimensión M*NxM*N), será más sencillo el
cálculo.

   L = A'*A;
   [V D] = eig(L); % Los elemenos de la diagonal de D son los valores
principales tanto de L=A'*A como de C=A*A'.

   %%%%%%%%%%%%%%%%%%%%%%%% Sorting and eliminating eigenvalues
   % Se ordenan los valores principales de L, eliminando lo que estén
po rdebajo de cierto umbral. Por este motivo
   % de vectores principales puede sr menor que (P-1).

   L_eig_vec = [];
   for i = 1 : size(V,2)
       if( D(i,i)>1 )
           L_eig_vec = [L_eig_vec V(:,i)];
       end
   end

   %%%%%%%%%%%%%%%%%%%%%%%% Calcula los vectores principales de C
   % Los vectores principales de la matriz C (llamadas también caras
principales)
   % se obtienen cobinando A con los vectores principales de L.
   Eigenfaces = A * L_eig_vec;

   %%%%%%%%%%%%%%%%%%%%%%%% Proyecta las imágenes de entrada en este
espacio
   %%%%%%%%%%%%%%%%%%%%%%%% PCA
   AP = [];  %Inicializa la matriz
   for i = 1 : N_Entrenamiento
       temp = Eigenfaces'*A(:,i); % Proyecta cada imagen (en A se
almacenan sin la media) en en espacio PCA
       AP = [AP temp];
   end
```

Capítulo 16. Bibliografía

[1] Barbara R. Jasny, Beverly A. Purnell, **The Glorious Sea Urchin**, Science, no. 938, November 2006.

[2] Sabrina Clemente, José Carlos Hernández, Alberto Brito, **An external tagging technique for the long-spined sea urchin *Diadema* aff. *antillarum***, Journal of the Marine Biological Association of the United Kingdom, no. 87, p. 777-779, 2007.

[3] Fernando Tuya, José Antonio Martín, Ángel Luque, **A novel technique for tagging the long-spined sea urchin Diadema antillarum**, Sarsia, 2003.

[4] Marc Sentís Hernández, **Relaciones interespecíficas entre *Diadema antillarum* (Echinodermata: Diadematidae) y otros invertebrados a lo largo del ciclo lunar**, Anales Universitarios de Etología, no. 2, p. 62-70, 2008.

[5] S. C. C. Steiner, S. M. Williams, **The density and size distribution of *Diadema antillarum* in Dominica(Lesser Antilles): 2001–2004**, Marine Biology, no. 149, p. 1071-1078, 2006.

[6] M. Chiappone, D. W. Swanson, S. L. Miller, S. G. Smith, **Large-scale surveys on the Florida Reef Tract indicate poor recovery of the long-spined sea urchin *Diadema antillarum***, Coral Reefs, no. 21, p. 155-159, 2002.

[7] Clément P. Dumont, John H. Himmelman, Michael P. Russell, **Daily movement of the sea urchin Strongylocentrotus droebachiensis in different subtidal habitats in eastern Canada**, Marine Ecology Progress Series, v. 317, p. 87-99, 2006.

[8] Jean-Sébastien Lauzon-Guay, Robert E. Scheibling, Myriam A. Barbeau, **Movement patterns in the green sea urchin, *Strongylocentrotus droebachiensis*,** Journal of the Marine Biological Association of the United Kingdom, no. 86, p. 167-174, 2006.

[9] F. Tuya, J. A. Martín, A. Luque, **Patterns of nocturnal movement of the long-spined sea urchin *Diadema antillarum* (Philippi) in Gran Canaria (the Canary Islands, central East Atlantic Ocean)**, Helgol, Mar Res, no. 58, p. 26-31, 2004.

[10] José Antonio Martín-García, Ángel Luque Escalona, **Capacidad de retorno de *Diadema antillarum* (Echinodermata: Echinoidea)**, Anales Universitarios de Etología, no. 2, p. 125-131, 2008.

[11] P. Viola, M. J. Jones, **Robust Real-time Object Detection**, Second International Workshop on Statistical and Computational Theories of Vision, 2001.

[12] P. Viola, M. J. Jones, **Robust Real-time Face Detection**, International Journal of Computer Vision, Volume 57(2), Mayo 2004.

[13] M. Pietikäinen, **Image analysis with local binary patterns**, SCIA 2005 Proceedings, Lecture Notes in Computer Science 3540, Springer, p. 115-118, 2005.

[14] S. Liao, W.K. Law, A. Chung, **Dominant local binary patterns for texture classification**, IEEE transactions on image processing, v. 18, no. 5, Mayo 2009, p. 1107-1118.

[15] Jonathon Shlens, **A Tutorial on Principal Component Analysis**, v. 3.01, Abril 2009.

[16] Fernando De la Torre, Michael J. Black, **Robust Principal Component Analysis for Computer Vision**, Int. Conf. on Computer Vision, Julio 2001.

[17] Gary Bradski, Adrian Kaebler, **Learning OpenCV: Computer Vision with the OpenCV Library**.

[18] Modesto Fernando Castrillón Santana, **Tutorial Viola-Jones**, Octubre 2009.

Referencias web

[19] http://en.wikipedia.org/wiki/Sea_urchin

[20] http://es.wikipedia.org/wiki/Tagging

[21] http://en.wikipedia.org/wiki/Homing_(biology)

[22] http://es.wikipedia.org/wiki/Grados_de_libertad_(mec%C3%A1nica)

[23] http://en.wikipedia.org/wiki/Video_tracking

[24] http://www.laopinion.es/sociedad/2008/08/31/mitad-litoral-isla-danado erizos/167603.html

[25] http://portal.grancanaria.com/portal/home.eriz

[26] http://www.marcanario.com/sobre/diadema-antillarum

[27] http://www.gnu.org/licenses/license-list.html#ModifiedBSD

[28] http://www.gnu.org/licenses/license-list.html#FreeBSD

[29] http://www.opensource.org/licenses/bsd-license.php

[30] http://www.gnu.org/licenses/license-list.html#OriginalBSD

[31] http://www.ubuntu.com/project/about-ubuntu/licensing

[32] http://www.gnu.org/copyleft/gpl.html

[33] http://www.gnu.org/philosophy/free-sw.html

[34] https://www.linux.com/

[35] http://cm.bell-labs.com/cm/cs/who/dmr/chist.html

[36] http://www.cplusplus.com/info/description/

[37] http://web.iti.upv.es/~evidal/students/app/tema4/t4app2p.pdf

[38] http://www.dtreg.com/svm.htm

[39] http://www.mathworks.es/products/matlab/index.html

[40] http://opencv.willowgarage.com/documentation/

[41] http://opencv.willowgarage.com/documentation/cpp/

[42] http://opencv.jp/opencv-1.0.0_org/docs/ref/opencvref_highgui.htm

[43] http://en.wikipedia.org/wiki/Sensitivity_and_specificity

[44] http://en.wikipedia.org/wiki/Haar-like_features

[45] http://avidemux.sourceforge.net/

[46] http://es.wikipedia.org/wiki/Echinodermata

[47] http://es.wikipedia.org/wiki/Echinoidea

[48] http://www.wordreference.com/definicion/ambulacral

[49] http://es.wikipedia.org/wiki/Gastropoda

[50] http://es.wikipedia.org/wiki/Dioico

[51] http://es.wikipedia.org/wiki/G%C3%B3nada

[52] http://es.wikipedia.org/wiki/Gameto

[53] http://www.wordreference.com/definicion/fototactismo

[54] http://www.pnas.org/search?fulltext=urchin&submit=yes