Escuela Superior
de Ingeniería y Tecnología
Universidad de La Laguna

Trabajo de Fin de Grado

Grado en Ingeniería Informática

Deepview: Sistema para detección automática y análisis de actividad biológica en imágenes submarinas

Deepview: *System for automatic detection and analysis of biological activity in underwater images*

Miguel Alejandro Martín Reyes

La Laguna, 7 de julio de 2022

D. **Manuel Rodriguez Valido**, con N.I.F. 52840833N profesor Titular de Universidad adscrito al Departamento Ingeniería Industrial de la Universidad de La Laguna, como tutor

D. **María de la Peña Fabiani Bendicho**, con N.I.F. 25143894H profesora Contratada doctora adscrita al Departamento de Ingeniería Industrial Universidad de La Laguna, como cotutora

C E R T I F I C A (N)

Que la presente memoria titulada:

"**Deepview:** Sistema para detección automática y análisis de actividad biológica en imágenes submarinas"

ha sido realizada bajo su dirección por **D. Miguel Alejandro Martín Reyes**, con N.I.F. 51.776.894-F.

Y para que así conste, en cumplimiento de la legislación vigente y a los efectos oportunos firman la presente en La Laguna a 7 de julio de 2022

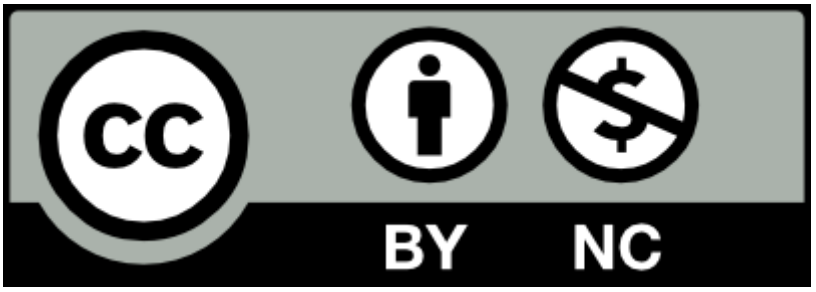
Agradecimientos

A mis padres, que con su dedicación, sacrificio y esfuerzo me han inculcado los valores que me hacen ser la persona que soy hoy.

A todos los docentes que me han acompañado durante mi trayectoria académica, por motivarme siempre a dar lo mejor de mi.

A mis amigos, cuya amistad ha sido siempre fuente de energía y ánimos para mi.

Licencia



Resumen

DEEPCOM es un proyecto con financiación a nivel nacional cuyo principal objetivo es el estudio de vida submarina mediante sensores de imagen y audio sumergidos.

Dichos sensores se encuentran soportados dentro del sistema **Delphis**, creado por los investigadores del proyecto y colaboradores de la Universidad de Auckland (Nueva Zelanda). Este sistema está constituido, a rasgos generales, por:

- Una carcasa resistente a la presión hasta 1000 m de profundidad.
- Un sistema de iluminación LED.
- **Una cámara SONY RX0**.
- Diversos sensores acústicos.

Actualmente se dispone de gran cantidad de imágenes de vídeo y sonar recogidas por este sistema, cuyo procesamiento se ha venido realizando de forma totalmente manual.

El objetivo del presente trabajo es explorar distintos métodos de detección sobre las imágenes submarinas captadas por el sistema antes descrito, para ello se desarrolla un sistema informático que permite automatizar los estudios estadísticos sobre la actividad biológica recogida en dichas bases de datos.

Más concretamente, el software o aplicación desarrollada es capaz de detectar automáticamente las partículas que constituyen gran parte de la biomasa de la *Capa de Dispersión Profunda (DDL* por sus siglas en inglés) captadas por el sensor de imagen del sistema Delphis. De esta forma, se facilita la extracción y análisis de los datos por parte de los miembros que componen el equipo de trabajo del proyecto Deepcom.

Palabras clave: DEEPCOM, procesamiento de imágenes submarinas, capa de dispersión profunda.

Abstract

***DEEPCOM** is a nationally funded project whose main goal is the study of underwater life using submerged image and audio sensors.*

*These sensors are supported within the **Delphis** system, created by the project researchers and collaborators from the University of Auckland (New Zealand). This system is constituted, in general terms, by:*

- *A pressure-resistant casing up to 1000 m deep.*
- *An LED lighting system.*
- *A SONY RX0 camera.*
- *Various acoustic sensors.*

There is currently a large number of video and sonar images collected by this system, the processing of which has been carried out completely manually.

In the present work, different detection methods are explored on the underwater images captured by the system described above. For this purpose, a computer system that allows the statistical analysis of the biological activity collected in said databases to be automated is developed.

More specifically, the software developed is capable of automatically detecting the particles that constitute a large part of the biomass of the Deep Dispersion Layer (DDL) captured by the image sensor of the Delphis system. In this way, the extraction and analysis of the data by the members that make up the Deepcom project work team is facilitated.

***Keywords**: DEEPCOM, underwater image processing, deep scattering layer.*

Índice general

Índice de figuras

Índice de tablas

1. Introducción

1.1. Estado del arte

La adquisición y análisis de imágenes submarinas supone tener en cuenta la forma en que viaja la luz a través del medio acuático. A diferencia de las imágenes capturadas en aire, estas se caracterizan por una visibilidad pobre debida a la atenuación que sufre la luz de forma exponencial a medida que viaja a través del agua. Para resolver este problema, generalmente se puede seguir una de las siguientes aproximaciones [1]:

- **Técnicas de realce**: son agnósticas respecto al proceso de formación de la imagen y no necesitan de conocimiento *a priori* del medio en el que se capturó. En [2] se propone un algoritmo de este tipo para preprocesar imágenes submarinas que reduce las perturbaciones y mejora la calidad de la misma mediante la ejecución de una serie de pasos independientes.

- **Técnicas de restauración:** se emplea un modelo de degradación de la imagen que depende del medio en el que se capturó y la imagen y, por lo tanto, de una gran cantidad de parámetros como los coeficientes de atenuación y difusión.

Por otra parte, para la detección de partículas, puntos luminosos, u objetos en general, en movimiento en una imagen existen diversas técnicas y herramientas cuya eficacia depende del contexto sobre el que se esté trabajando. Tal es el caso de [3], cuya aproximación se basa en el uso de redes neuronales auto-organizativas y una cámara fija; [4], que proponen una solución enfocada en la sustracción de fondo adaptativa, entre otros.

1.2. Antecedentes

Hasta ahora, el procesamiento de las imágenes recogidas por el sistema Delphis se ha venido haciendo de forma manual con la ayuda del software *MotionMeerkat* [5] para detectar organismos de un tamaño suficientemente grande como para realizar un análisis taxonómico. Con este software se adquirieron las imágenes en las que aparecían los organismos en el tiempo de encendido de la cámara de grabación (10 minutos encendido, 10 minutos apagado) [7].

MotionMeerkat funciona mediante la detección de eventos. En este caso, cada uno de estos eventos corresponde a la aparición en la escena de animales suficientemente grandes como para realizar el análisis taxonómico.

Sin embargo, en el caso de partículas en suspensión, el paradigma de detección de eventos no es una solución óptima, ya que la finalidad última no es conocer el instante en el que ha aparecido una partícula determinada sino conocer datos generales, principalmente la cantidad de las mismas. Por esta razón, no es posible llevar a cabo un análisis automático de las partículas que componen la biomasa presente en las imágenes captadas por el sistema Delphis empleando este software.

1.3. Objetivos

El objetivo general de este proyecto es automatizar el análisis de la biomasa presente en las imágenes captadas por el sistema Delphis del proyecto Deepcom.

Como objetivos específicos se pretende:

1. Diseñar un algoritmo que permita el preprocesamiento y procesamiento de las imágenes submarinas para detectar biomasa.

2. Diseñar e implementar un sistema que permita ejecutar el algoritmo de detección y extraer datos estadísticos de los resultados obtenidos.

3. Diseñar e implementar una interfaz de usuario para facilitar el uso del sistema.

1.4. Análisis del problema y definición de requisitos iniciales de la solución.

El material captado por el sistema Delphis sobre el que se trabaja en este proyecto está constituido principalmente por vídeos con las siguientes características:

- **Resolución (en píxeles)**: 1080 de alto por 1920 de ancho.
- **Tipo de archivo**: mp4.

- **Tasa de *frames* por segundo (fps)**: 30.

Este sistema captura las imágenes a una profundidad de hasta 600 metros [7], un aspecto crítico a tener en cuenta debido a las propiedades de transmisión en el agua específicas de la luz, que producen que las imágenes sufran de iluminación no uniforme, bajo contraste, distorsión de los colores, difuminación, etc [2].

Teniendo estos aspectos en cuenta, los requisitos iniciales de la solución incluyen:

- La capacidad para detectar de forma automática las partículas de biomasa presentes en el material visual. Teniendo en cuenta los inconvenientes mencionados en el apartado anterior.
- Almacenamiento de los resultados obtenidos para su posterior análisis.
- Representación gráfica de los resultados obtenidos.

1.5. Diseño inicial del sistema

El diseño inicial de la solución se basa en una arquitectura cliente - servidor (figura 1) que hace uso de un paquete de Python desarrollado específicamente para implementar las funciones de procesamiento de los archivos de vídeo.

La motivación de emplear esta estructura responde a la flexibilidad que aporta separar la interfaz de usuario (*frontend*), el servidor de la aplicación (*backend*) y el *core* de procesamiento, aplicación. Dicha flexibilidad se ve reflejada a la hora de elegir distintas herramientas o *frameworks* para desarrollar cada uno de los distintos componentes, de esta forma, es posible cambiar de *framework* sin necesidad de modificar todo el sistema.

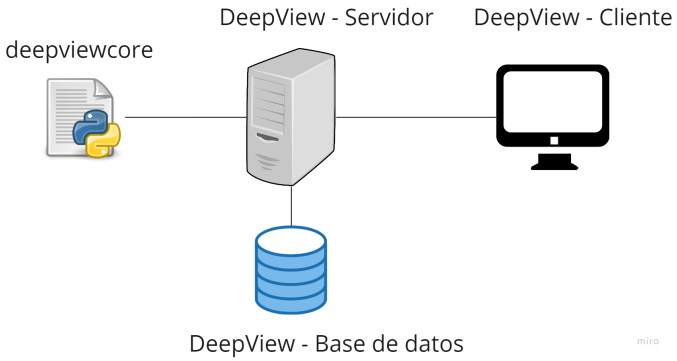


Figura 1. Estructura del sistema

2. Desarrollo e implementación

2.1. Procesamiento de las imágenes

Para el procesamiento de las imágenes se desarrolló un paquete en el lenguaje de programación Python denominado **deepviewcore**, utilizando las funcionalidades ofrecidas por la reconocida biblioteca de procesamiento de imágenes **OpenCV** [8]**.**

2.1.1. Detección de partículas

En este caso de estudio, el problema de la detección de partículas se reduce a identificar los contornos de las mismas en cada una de las imágenes. Entendiendo como contorno la curva que une todos los puntos continuos (a lo largo del perímetro) que tienen el mismo color o intensidad [9].

OpenCV provee una función específica para la detección de contornos, llamada **findContours.** Para mejorar la precisión de la detección, la imagen es binarizada antes de pasársela a la función, de forma que los objetos cuyos contornos se desean detectar (partículas) sean de color blanco y el fondo de color negro, tal y como espera la función.

Así, para binarizar la imagen, el primer paso consistió en pasar la imagen a escala de grises para disminuir la carga computacional del resto de pasos.

Posteriormente, se aplicó un proceso de umbralización variando el valor del umbral entre 27 y 30. Se seleccionó este rango de valores para realizar las pruebas de umbralizado después de analizar los histogramas y rangos de algunas imágenes para así determinar cuál era el valor máximo de gris que tomaba el fondo y separarlo de las partículas. Al realizar este análisis se observó que el valor máximo de gris que tomaba el fondo era de alrededor de **28**. En la figura 2 se pueden observar los resultados del umbralizado para los distintos valores sobre la imagen original en escala de grises.

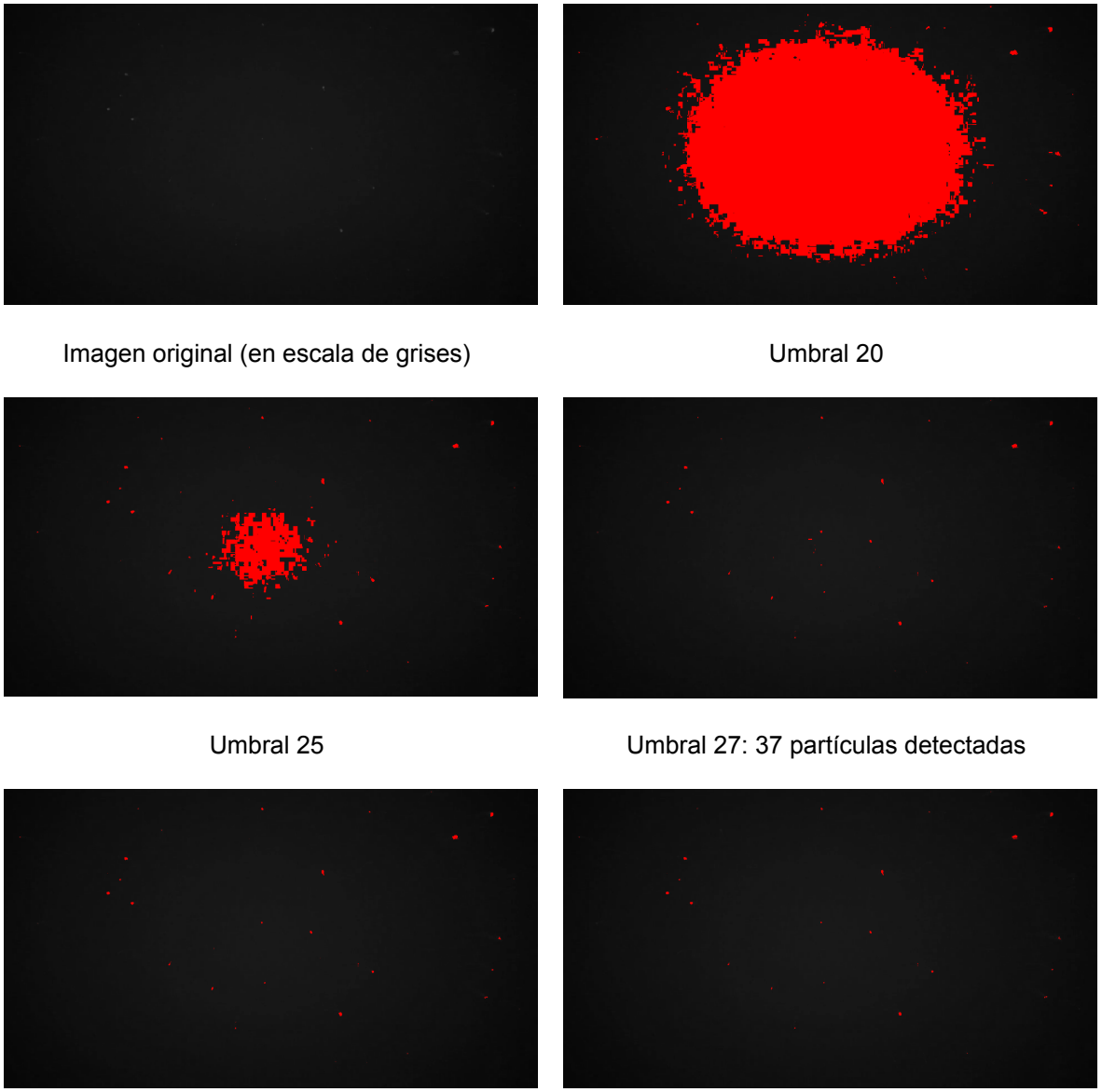


Imagen original (en escala de grises) | Umbral 20

Umbral 25 | Umbral 27: 37 partículas detectadas

Umbral 28: 32 partículas detectadas | Umbral 30: 33 partículas detectadas

Figura 2: Umbralización[1]

Como se puede observar, un valor muy bajo del umbral ocasiona la aparición de falsos positivos debido a la **iluminación no homogénea** ocasionada por el uso de una fuente de luz artificial, concretamente dos focos laterales equipados con 3 luces LED blancas de 1200 lúmenes de potencia [7].

[1] Las partes coloreadas en rojo representan la parte "frontal" o *foreground* de la imagen, es decir, aquellos píxeles con un valor superior al umbral, después de aplicar la binarización.

Imagen original

Imagen en escala de grises y con contraste aumentado

Figura 3: Iluminación no homogénea

Este problema de iluminación (figura 3) ocasiona que el valor de gris que toma el fondo de la imagen pueda variar en función de la potencia de los focos, condiciones del agua, etc, lo que podría invalidar el cálculo de un umbral fijo a partir del estudio del histograma [10].

2.1.2. Homogeneización de la iluminación de fondo

Por la razón descrita en el apartado anterior, se optó por probar una alternativa diferente basada en un filtro de tipo **Top-hat** para homogeneizar la iluminación de fondo antes de aplicar el binarizado de la imagen. El filtro Top-hat se aplica calculando la apertura de la imagen por un objeto estructurante y sustrayendo el resultado de la imagen original. En la figura 4 se observan los resultados obtenidos al aplicar un filtro Top-hat empleando un elemento estructurante rectangular de dimensiones 9 por 9.

Imagen original (en escala de grises) con contraste aumentado

Imagen después de aplicar Top-hat y aumentar el contraste

Figura 4: Imagen con fondo homogéneo binarizada

Una vez el fondo de la imagen ha sido homogeneizado se puede proceder a binarizar la imagen para ejecutar la detección de contornos sobre la misma empleando un umbral de 15[2] (figura 5).

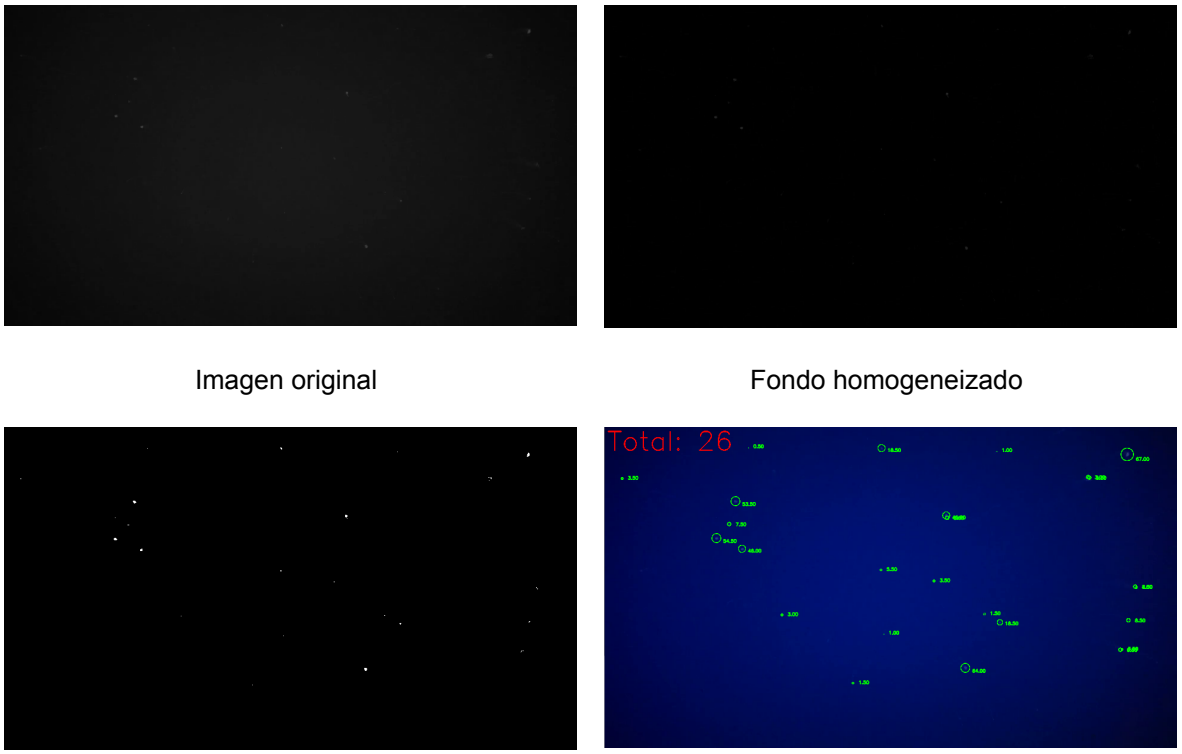


Imagen original | Fondo homogeneizado

Imagen binarizada | Imagen original con contornos detectados

Figura 5. Detección de contornos

2.1.3. Ruido oscuro

La “corriente oscura” es aquella que fluye incluso cuando no hay fotones incidiendo en el sensor de la cámara. Se trata de un fenómeno térmico resultante de la generación espontánea de electrones en el silicio del sensor. La variación en la cantidad de “electrones oscuros” captados durante la exposición en una escena completamente a oscuras es lo que se conoce como ruido oscuro. Este ruido, por lo tanto, es dependiente de la temperatura del sensor [11].

[2] El valor del umbral ahora puede ser significativamente más bajo que en los apartados anteriores debido a la homogeneización del fondo.

Generalmente, mientras mayor sea la temperatura del sensor, mayor ruido oscuro habrá, debido a que los electrones de valencia se excitan y saltan a la banda de conducción [11].

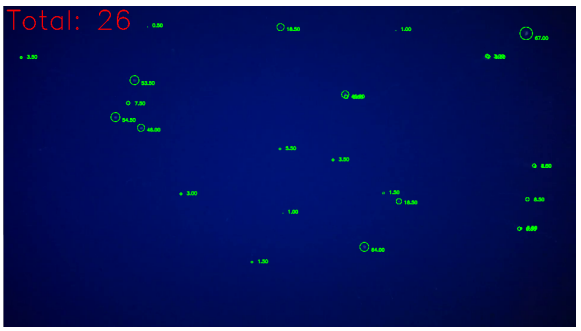
En la figura 6 se observa el ruido oscuro captado en un período de total oscuridad de la cámara y cómo afecta a la detección de partículas

Ruido oscuro en escala de grises

Imagen original en escala de grises

Imagen original con ruido oscuro sustraído

Contornos detectados sobre imagen con ruido sustraído



Contornos detectados sobre imagen sin ruido sustraído

Figura 6: Ruido oscuro

Como se puede observar, el ruido oscuro parece no influir de forma notable sobre la detección de las partículas, tal y como se esperaba debido a que el agua del océano atlántico actúa como un refrigerante de la cápsula dentro de la cual funciona la cámara.

2.1.4. Operaciones de filtrado

Como resultado de las pruebas descritas anteriormente se desarrolló un flujo de procesamiento en el que para cada uno de los frames del vídeo se aplica una serie de operaciones de filtrado independientes y encadenadas, esto es, la salida de una operación es la entrada de la siguiente (figura 7).

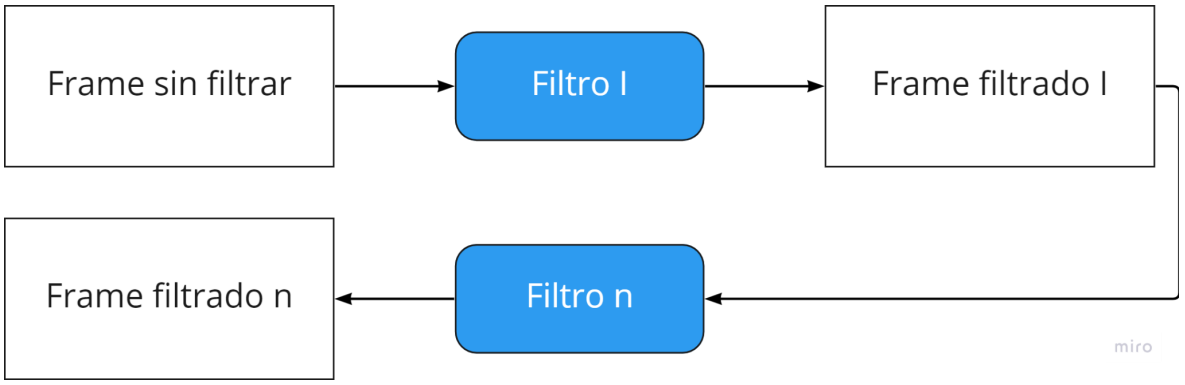


Figura 7. Esquema de filtrado general

Además, estas operaciones de filtrado se dividen en dos fases: pre-procesamiento y procesamiento; con la finalidad de diferenciar la fase de preparación de la imagen y la fase de separación del fondo y frente de la imagen. Esta arquitectura *pipeline* propicia también la modularidad del *software* y su mantenimiento, puesto que facilita la adición de nuevos filtros o el cambio en el orden de aplicación de los mismos.

En la fase de preparación, o **preprocesamiento** (figura 8)**,** se convierte la imagen original, a color, en una imagen en escala de grises, empleando el método de luminosidad o *método ponderado* [12], para disminuir la carga computacional;

posteriormente, se homogeneiza el fondo de la misma mediante la aplicación del filtro **top-hat** [13] y por último se sustrae el ruido oscuro de la imagen resultante[3].

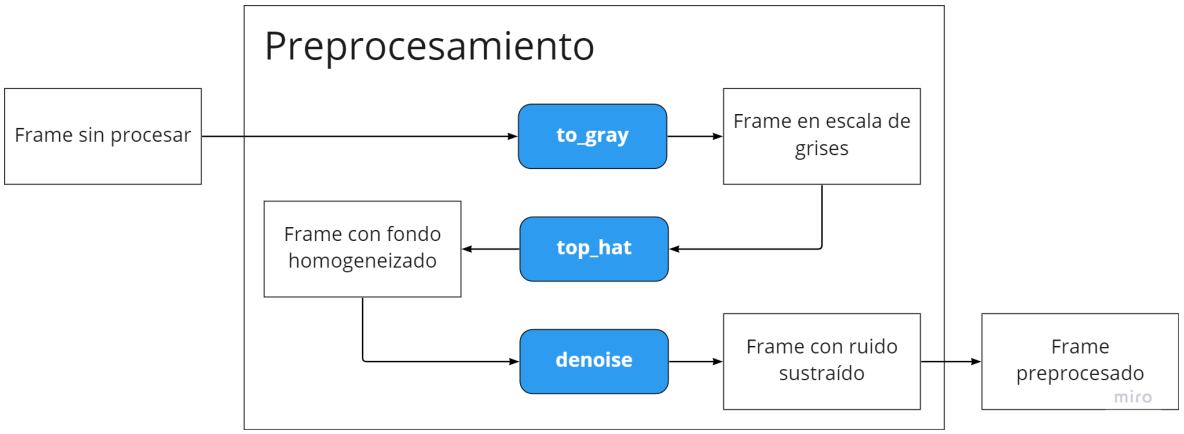


Figura 8. Fase de preprocesamiento

Por otro lado, la fase de procesamiento de la imagen (figura 9) consta de una única operación de filtrado: la binarización simple [14], que recibe como entrada el frame preprocesado y produce como resultado una imagen en la que las partículas (frente de la imagen) tienen un nivel de gris 255 y el fondo un nivel 0.

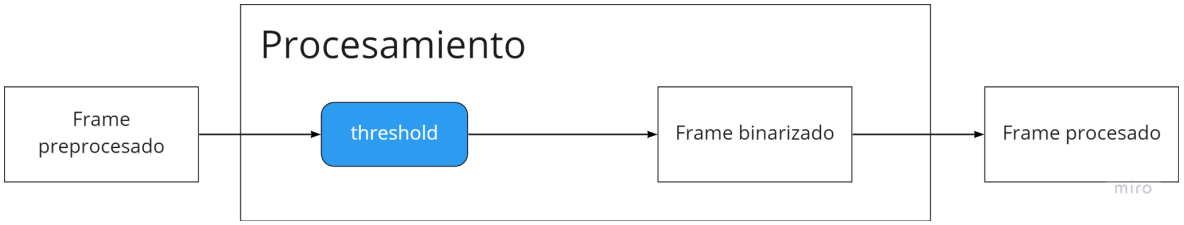


Figura 9. Fase procesamiento

Este frame binarizado es la entrada de la fase de detección de partículas, ejecutada aplicando la función **findContours** [15], mencionada anteriormente, y que arroja como resultado los contornos de las partículas detectadas.

[3] Dado que no se apreciaron cambios sustanciales en el procesamiento de las imágenes debidos al ruido, se optó por “desactivar” el filtrado del ruido para aumentar la velocidad de procesamiento.

2.1.5. Conclusiones del procesamiento

Como fruto de las tareas de investigación y diseño realizadas durante esta fase del proyecto, se desarrolló el paquete de Python **deepviewcore,** que fue publicado en el índice de paquetes oficial de Python (PyPi). De esta forma, es accesible para su uso público a través del siguiente enlace.

Este paquete fue desarrollado siguiendo el paradigma de POO (programación orientada a objetos) y consta de una clase principal llamada **Video.** Esta clase se encarga de abrir un archivo de vídeo, cuya ruta se pasa a través del constructor, y procesarlo (figura 10) aplicando los pasos de filtrado descritos anteriormente.

```
from deepviewcore.Video import Video

video_path = "C:/videos/plancton_3.mp4"
video = Video(video_path)
video.process()
```

Figura 10. Uso de la clase Video

Los resultados del procesamiento se guardan en la instancia de la clase como un vector de objetos de la forma descrita en la figura 11, donde:

- ***circle*** representa el círculo de menor radio [16] que encierra a la partícula, siendo la tupla ***(x, y)*** y ***r*** el centro y radio del círculo respectivamente.
- ***area*** representa el área en píxeles [17] que ocupa la partícula detectada, siendo ***a*** el valor numérico de dicha área.

```
{
  circle: [[x, y], r],
  area: a,
}
```

Figura 11. Formato de partícula detectada

Asimismo, entre los argumentos que recibe la función ***process()*** se encuentra definido el argumento opcional denominado ***action*****.** Este argumento es una función que se ejecuta cada **n** frames durante el procesamiento de los vídeos, siendo **n** el valor que toma el atributo de instancia ***frame_interval*** definido en la clase **Video**.

La utilidad de este argumento se ve reflejada a la hora de manejar los resultados del procesamiento para, por ejemplo, guardarlos en una base de datos a medida que los frames van siendo procesados.

Esta función también recibe un argumento opcional denominado **options,** que define los parámetros de los filtros a emplear, en la figura 12 se describe la forma de este objeto y los valores que toma por defecto.

```
{
  "preprocess": {
    "top_hat": {
      "filterSize": (9, 9)
    },
    "denoise": None,
  },

  "process": {
    "threhold": {
      "thresh": 20,
      "maxValue": 255,
    }
  },
}
```

Figura 12. Forma y parámetros por defecto del argumento ***options***.

2.2. **Backend**: Servidor y API

Entre las principales responsabilidades del *backend* de la aplicación se pueden enumerar las siguientes:

- Alojar los archivos multimedia (vídeos).
- Manejar el procesamiento de estos archivos.
- Comunicarse con la base de datos donde se almacenarán los resultados.
- Exponer la API (*Application Programming Interface*) que permita a los clientes comunicarse con el servidor.

2.2.1. Framework utilizado

Dado que para el desarrollo del paquete **deepviewcore** se empleó Python como lenguaje de programación, para la implementación del servidor se optó por el mismo lenguaje. De esta forma se facilita la integración entre ambos componentes del proyecto.

Más específicamente, se empleó el conocido *framework* Django [18] por tratarse de una herramienta de alto nivel que facilita el desarrollo rápido de sitios web de forma mantenible y segura. Además, al igual que Python, es gratuito y de código abierto, lo que propicia su continua mejora mediante aportes de la comunidad, así como la redacción de documentación extensa y mantenimiento de foros sobre su funcionamiento.

2.2.2. Base de datos

En cuanto al sistema gestor de base de datos, se optó por emplear **MongoDB**[4] en su versión *Community Edition 5.0*.

La motivación de esta elección responde a la facilidad de configuración de la herramienta, y a que, debido a que almacena los datos como documentos en formato JSON, el mapeo entre los datos almacenados y los objetos nativos del programa se realiza de forma sencilla y natural. Además, siguiendo la línea del resto de herramientas utilizadas en este proyecto, la versión *Community Edition* es gratuita y de código abierto.

[4] https://www.mongodb.com/. Accedido 06/07, 2022.

Para integrar Django con MongoDB se utilizó la librería *djongo*[5], que permita mapear objetos nativos de Python a documentos de MongoDB (*Object Document Mapping*) y realiza comprobaciones de integridad y validaciones para asegurar que solo datos "limpios" son almacenados en la base de datos. Además, con esta librería, no es necesario modificar el ORM (*Object-Relational Mapper*) de Django, la forma que tiene el *framework* de manipular la base de datos programáticamente.

La estructura de la base de datos utilizada por la aplicación consta de dos modelos[6] (*VideoModel* y *ProcessingParametersModel*), que a su vez, son mapeados a dos colecciones de MongoDB: *deepcom_videomodel* y *deepcom_processingparametersmodel*.

La documentos almacenados en la colección *deepcom_videomodel* contienen los datos referentes a un archivo multimedia cuyo procesamiento ha, por lo menos, iniciado, esto es, una vez que se inicia el procesamiento se genera un documento con la forma descrita en la figura 13. Donde *_id* es el identificador del documento; *created_at* es la fecha de creación del documento; *video_path* es la ruta del archivo multimedia relativa a la ubicación del directorio raíz del servidor; *status* es el estado actual del vídeo; *by_second* es un vector de números enteros donde cada uno de sus valores corresponde al número de partículas detectadas en ese segundo y *spent_time* corresponde a los segundos transcurridos entre la creación del documento y la finalización del procesamiento.

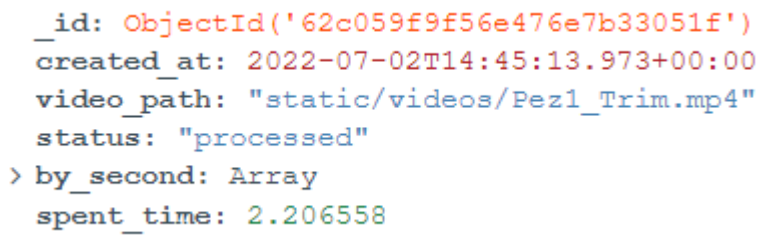


Figura 13. Colección *deepcom_videomodel*

[5] https://www.djongomapper.com/. Accedido 06/07, 2022.
[6] Objetos nativos de Python utilizados para acceder y manejar los datos.

Por otro lado, los documentos almacenados en la colección *deepcom_processingparametersmodel* contienen los datos referentes a los parámetros de procesamiento guardados para un vídeo en específico (figura 14).

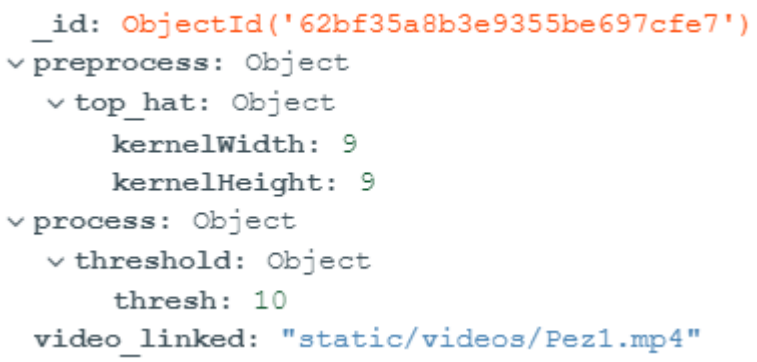


Figura 14. Colección *deepcom_processingparametersmodel*

La estructura de estos documentos consta de un campo *_id*, que es el identificador del documento, y dos objetos principales llamados *preprocess* y *process*. Estos objetos almacenan los parámetros para los filtros aplicados en las fases de preprocesado y procesado respectivamente, siendo el nombre del filtro la "clave"[7] de dichos parámetros.

Actualmente, los parámetros que pueden ser guardados, y por tanto modificados por el usuario, son los referentes al filtro de binarizado (*threshold*) y al filtro de homogeneización del fondo (*top_hat*). El proceso para determinar los parámetros a utilizar para el procesamiento de un vídeo es el descrito en la figura 15.

[7] En el sentido clave - valor de la sintaxis de un documento JSON.

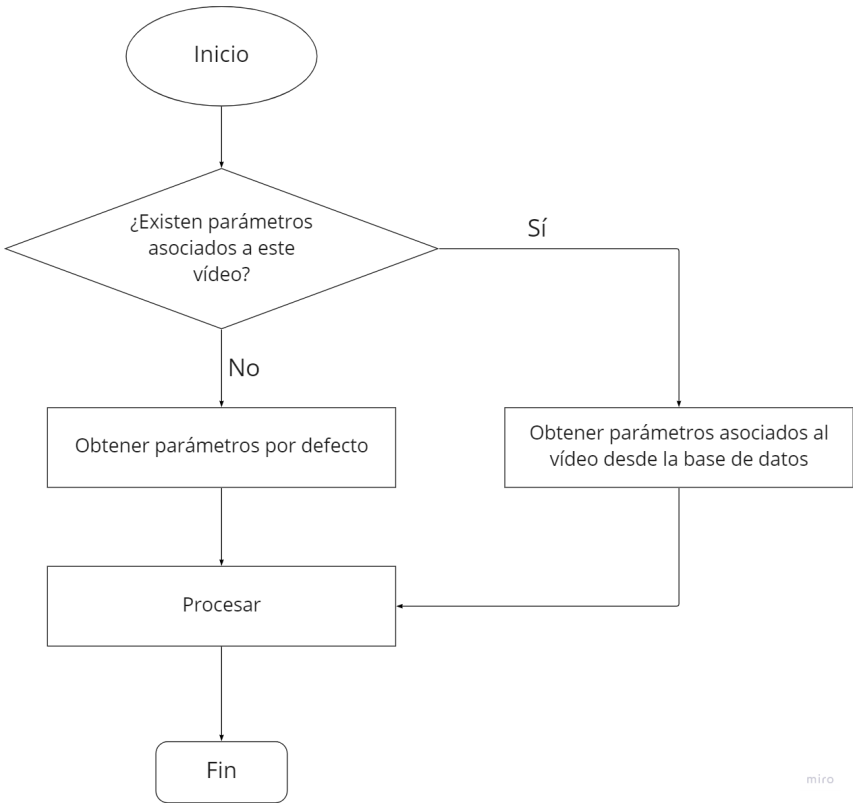


Figura 15. Diagrama de flujo para obtención de los parámetros de procesamiento

2.2.3. Diseño de la API

Para la comunicación entre el servidor y el cliente de la aplicación se diseñó una API (*Application Programming Interface*) REST (*Representational State Transfer*) [19] basada en el protocolo de la capa de aplicación HTTP que permite el intercambio de datos mediante objetos JSON.

Para posibilitar el acceso del cliente al estado actual de los recursos en el servidor y su manejo esta API expone los siguientes *endpoints*:

2.2.3.1. video/

Este *endpoint* acepta peticiones, o *requests*, que implementan los métodos **GET** y **POST.**

Por una parte, el formato de las peticiones **GET** incluye la presencia de un parámetro de cadena de consulta, o *query string parameter*, denominado ***action*** y

que identifica la acción a llevar a cabo por el servidor. Las posibles acciones incluyen:

- ***check-status****:* comprueba el estado de un vídeo cuyo nombre es pasado como otro parámetro de cadena de consulta denominado ***video_name***.

- ***list-available***: recupera la lista de archivos multimedia (vídeos) disponibles para ser procesados por el servidor. En concreto, devuelve una lista de objetos que incluyen el nombre, tamaño en megabytes, duración en segundos, *frames* por segundo y estado, de cada uno de los vídeos disponibles.

- ***get-data***: recupera los datos correspondientes al procesamiento de un vídeo en específico, cuyo nombre es pasado como otro parámetro de cadena de consulta denominado ***video_name***. Los datos devueltos siguen un formato de lista de números donde cada valor corresponde al número de partículas detectadas en una unidad de tiempo. Esta unidad de tiempo puede ser especificada mediante el parámetro de cadena de consulta opcional denominado ***unit***, que puede ser *seconds, minutes* u *hours*, su valor por defecto es *minutes.*

Por otro lado, el formato de las peticiones **POST** incluye la presencia de un parámetro denominado ***action*** y un parámetro denominado ***payload*** presentes en el cuerpo, o *body*, de la petición. De manera similar al método GET, el parámetro *action* especifica la acción a realizar, siendo estas las acciones disponibles:

- ***stop:*** Detiene el procesamiento de un vídeo. El nombre del vídeo se específica mediante el parámetro *payload*.

- ***process***: Inicia el procesamiento de un vídeo. El nombre del vídeo se específica mediante el parámetro *payload*.

- ***process_frame***: Inicia el procesamiento de un frame y devuelve una lista de partículas detectadas, cada una de las cuales contiene su posición en el frame, radio y área en píxeles. En *payload* se pasan los parámetros necesarios para procesar el frame: *video_name,* nombre del vídeo; *params*, parámetros de procesamiento; *frame_index* índice del frame a procesar.

2.2.3.2. parameters/

Este *endpoint* acepta peticiones, o *requests*, que implementan los métodos **GET** y **POST.**

Por una parte, el formato de las peticiones **GET** incluye la presencia de un parámetro de cadena de consulta, o *query string parameter*, denominado ***action*** y que identifica la acción a llevar a cabo por el servidor. Las posibles acciones incluyen:

- ***get-parameters-for-video***: Recupera los parámetros de procesamiento asociados a un vídeo cuyo nombre se pasa como parámetro de cadena de consulta denominado ***video_name***. Si no se encuentran parámetros asociados a ese vídeo, se devuelven los parámetros por defecto.

Por otro lado, el formato de las peticiones **POST** incluye la presencia de un parámetro denominado ***action*** y un parámetro denominado ***payload*** presentes en el cuerpo, o *body*, de la petición. De manera similar al método GET, el parámetro *action* especifica la acción a realizar, siendo estas las acciones disponibles:

- ***add:*** Añade o actualiza los parámetros de procesamiento asociados a un vídeo. El parámetro *payload* debe contener los parámetros necesarios para ejecutar la acción: *video_name*, nombre del vídeo; *parameters* parámetros a añadir.

  Si el vídeo especificado no tiene parámetros asociados, se guardan los nuevos, en caso de que sí tenga parámetros asociados, estos son actualizados.

2.2.4. Procesamiento

2.2.4.1. Hilos

El procesamiento de los vídeos se realiza empleando hilos concurrentes utilizando la librería de Python *threading*[8], donde, para cada vídeo que está siendo procesado, existe un hilo independiente.

El uso de hilos posibilita la rápida respuesta al cliente una vez que este ha solicitado el inicio del procesamiento de un vídeo. Además, dado que los hilos comparten la

[8] https://docs.python.org/3/library/threading.html. Accedido 06/07, 2022.

memoria del hilo principal, se facilita el manejo de los mismos desde el código del servidor.

De esta forma, la clase *VideoService*, encargada de proporcionar servicios relacionados con los vídeos, define un atributo de tipo diccionario denominado *processes*, donde se almacenan las instancias de la clase *Video* definida en *deepviewcore* que están realizando los procesamientos de los vídeos. Además, define un atributo denominado *lock*, que es una primitiva de sincronización que permite sincronizar los hilos de procesamiento. Una vez el procesamiento ha finalizado, la instancia es eliminada del objeto *processes.*

Antes de iniciar el procesamiento, se define una función *saveData*, que será la encargada de guardar los datos procesados en la base de datos. Posteriormente, se le asigna el valor 2010 al atributo *frame_interval*, descrito en la sección 2.1.5, del objeto *Video*. De esta forma, la función *saveData* es ejecutada en intervalos de 2010 frames, o 67 segundos de vídeo.

2.2.4.2. Cálculo de los datos

La función *saveData* mencionada en el apartado anterior se encarga de convertir los datos obtenidos del procesamiento (lista de partículas con ubicación y área) en una lista de números enteros que representan la cantidad de partículas detectadas en un segundo.

Este número se corresponde a la **moda** de los valores correspondientes al número de partículas detectadas en grupos de 30 *frames* (figura 16), dado que esta es la tasa de *frames* por segundo. Se optó por emplear esta medida para paliar posibles falsos positivos del sistema en *frames* muy específicos.

Para el cálculo de la moda se emplearon las librerías *scipy*[9] y *numpy*[10] por proveer estructuras de datos y métodos optimizados para cálculos estadísticos.

2.2.5. Fork de Django: contenido estático

Para facilitar el uso de la aplicación, se decidió añadir la posibilidad de visualizar los vídeos disponibles mediante un elemento ***<video>*** html en el cliente. Para ello, es necesario "servir" dicho vídeo desde el servidor como contenido estático. Django ofrece esta característica por defecto.

[9] https://scipy.org/. Accedido 06/07, 2022.
[10] https://numpy.org/. Accedido 06/07, 2022.

Sin embargo, una de sus limitaciones es que dicho servicio no soporta peticiones parciales, o *range requests*, por lo que se pierde la capacidad de posicionarse en instantes concretos del archivo multimedia al ser este servido de forma secuencial y fija.

Esta simple característica resulta muy importante para el correcto uso de la aplicación, ya que permite al usuario final situarse en un instante del vídeo donde, mediante el análisis de los datos procesados, ha identificado eventos de interés.

Por esta razón, se procedió a realizar una investigación de posibles alternativas o adaptaciones a la versión de Django utilizada (3.1.12) para añadir el soporte de peticiones parciales.

Como fruto de dicha investigación se realizó una "bifurcación", o *fork*, de la versión de Django utilizada, alojada en la cuenta oficial de github[11], y se le añadió el soporte de peticiones parciales utilizando el código desarrollado por el usuario de github satchamo[12] en su bifurcación del proyecto django[13].

En resumen, se combinó el trabajo realizado por el usuario satchamo con la versión de django que se había estado usando en el desarrollo de este trabajo hasta el momento, adaptando los ficheros del código fuente del paquete.

El resultado de dicha adaptación fue alojado en un repositorio de la cuenta personal de Github[14] del autor de este trabajo, de forma que pueda ser instalado desde el mismo haciendo uso de la herramienta *git* (figura 16).

```
[packages]
djongo = "*"
pymongo = "==3.12.1"
scipy = "*"
deepviewcore = "*"
django = {git = "https://github.com/miguel-martinr/django.git"}
django-cors-headers = "*"
```

Figura 16. Dependencias de DeepView

[11] https://github.com/django/django/releases/tag/3.1.12. Accedido 06/07, 2022.
[12] Matt Johnson, https://github.com/satchamo. Accedido 06/07, 2022.
[13] *Fork* django https://github.com/satchamo/django/tree/httprange. Accedido 06/07, 2022.
[14] https://github.com/miguel-martinr/django. Accedido 06/07, 2022.

2.3. Frontend: Cliente e interfaz de usuario

El *frontend* de la aplicación es el código cliente que hace de interfaz entre el servidor y el usuario final.

2.3.1. Herramientas utilizadas

La librería principal utilizada para el desarrollo de este elemento de la aplicación fue ***ReactJS***[15], una librería de JavaScript para construir interfaces de usuario del tipo SPA (*Single Page Application*). En este tipo de interfaces, a diferencia de los sitios web tradicionales, el renderizado de los componentes visuales se realiza íntegramente en el cliente (agente), por ello, en la petición inicial del cliente al servidor, se descargan todos los ficheros necesarios para poder ejecutar dicho renderizado (Html, JavaScript y CSS).

Además, como lenguaje de programación se utilizó **TypeScript**[16], un superconjunto del lenguaje JavaScript que añade soporte de tipos tales como comprobación en tiempo de transpilación e inferencia de tipos en tiempo de desarrollo.

Asimismo, se empleó **Redux Toolkit**[17] para el manejo del estado de la aplicación. Se trata de una librería que facilita el manejo del estado global de la aplicación mediante la arquitectura Flux[18], una alternativa al patrón tradicional Modelo - Vista - Controlador. También es importante recalcar que el uso de esta librería trae como ventaja el uso de su principal herramienta de depurado (*Redux dev tools*), una extensión del navegador *Chrome* que permite visualizar el estado de la aplicación en cualquier momento.

Cabe mencionar que, en las etapas iniciales de diseño, se utilizó también el conocido software de diseño **Figma**[19] para maquetar el aspecto visual de las páginas de la aplicación. Figma es una moderna herramienta colaborativa de diseño basada en la nube y con planes de carácter gratuito.

---

[15] https://reactjs.org/. Accedido 06/07, 2022.
[16] https://www.typescriptlang.org/. Accedido 06/07, 2022.
[17] https://redux-toolkit.js.org/. Accedido 06/07, 2022.
[18] https://facebook.github.io/flux/. Accedido 06/07, 2022.
[19] https://www.figma.com/. Accedido 06/07, 2022.

En cuanto a los estilos de los elementos que conforman la interfaz gráfica, se empleó ***Bootstrap***[20], una popular librería de CSS que permite construir aplicaciones responsivas.

2.3.2. Estructuración de las páginas

Dado que se trata de una SPA, la aplicación no consta de “páginas” tradicionales, entendiendo estas como un documento plantilla, normalmente html, que es generado en el servidor y enviado al cliente. Por el contrario, las “páginas” de una SPA en React son componentes creados por el desarrollador y que la librería se encarga de manejar y renderizar con los datos obtenidos de las peticiones asíncronas al servidor.

2.3.2.1. Inicio

Este componente (figura 17) representa la página principal de la aplicación, y está formado únicamente por un botón **Vídeos** que redirige al usuario a la **Página de vídeos.** La motivación de incluir esta página responde a posibles extensiones futuras de la aplicación que involucren el procesado no solo de archivos de vídeo sino también de audio o de cualquier otro tipo.

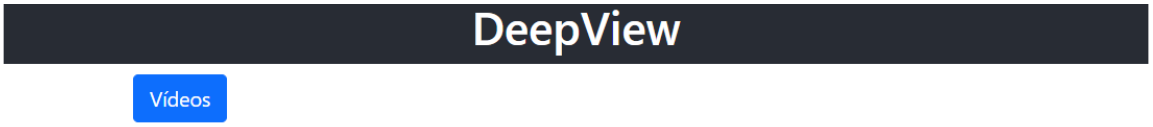


Figura 17. Página de inicio

[20] https://getbootstrap.com/docs/5.0/getting-started/introduction/ . Accedido 06/07, 2022.

2.3.2.2. Vídeos

En esta página el usuario puede visualizar los archivos multimedia disponibles para ser procesados por el servidor, así como acceder a los detalles de los mismos, iniciar y detener el procesamiento y también comprobar su estado.

Como se puede apreciar en la figura 18, el estado de los vídeos puede ser uno de *No procesado*, *Procesando*, *Procesado* o *Detenido.*

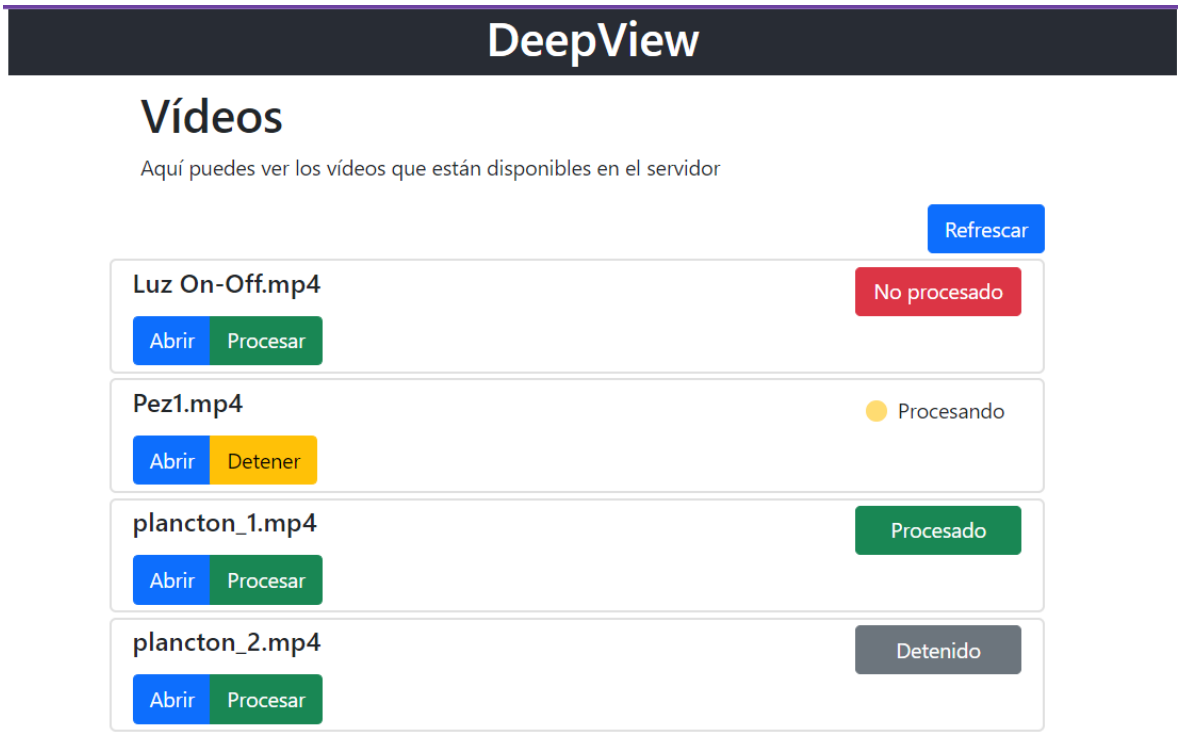


Figura 18. Página de vídeos

Los datos de esta página son recuperados del servidor haciendo una petición al *endpoint* ***GET video/ action=list-available*.** El estado de los vídeos puede ser actualizado mediante el botón *Refrescar* y, en el caso de los vídeos en estado *Procesando* su estado es actualizado automáticamente cada 20 segundos.

2.3.2.3. Página de Evaluación / Análisis

Esta página es específica para cada vídeo y permite al usuario realizar las operaciones principales ofrecidas por la aplicación: análisis de los datos, ajuste de parámetros, visualización del vídeo, etc…

Esta página consta de dos modos de trabajo:

- **Modo Evaluación**: en este modo el usuario es capaz de asegurarse de la fiabilidad aproximada con la que el procesamiento está siendo llevado a cabo por el servidor. De esta forma, el usuario puede visualizar cómo se están detectando las partículas en cada *frame* del vídeo, así como ajustar y guardar los parámetros de procesamiento en respuesta (figura 19).

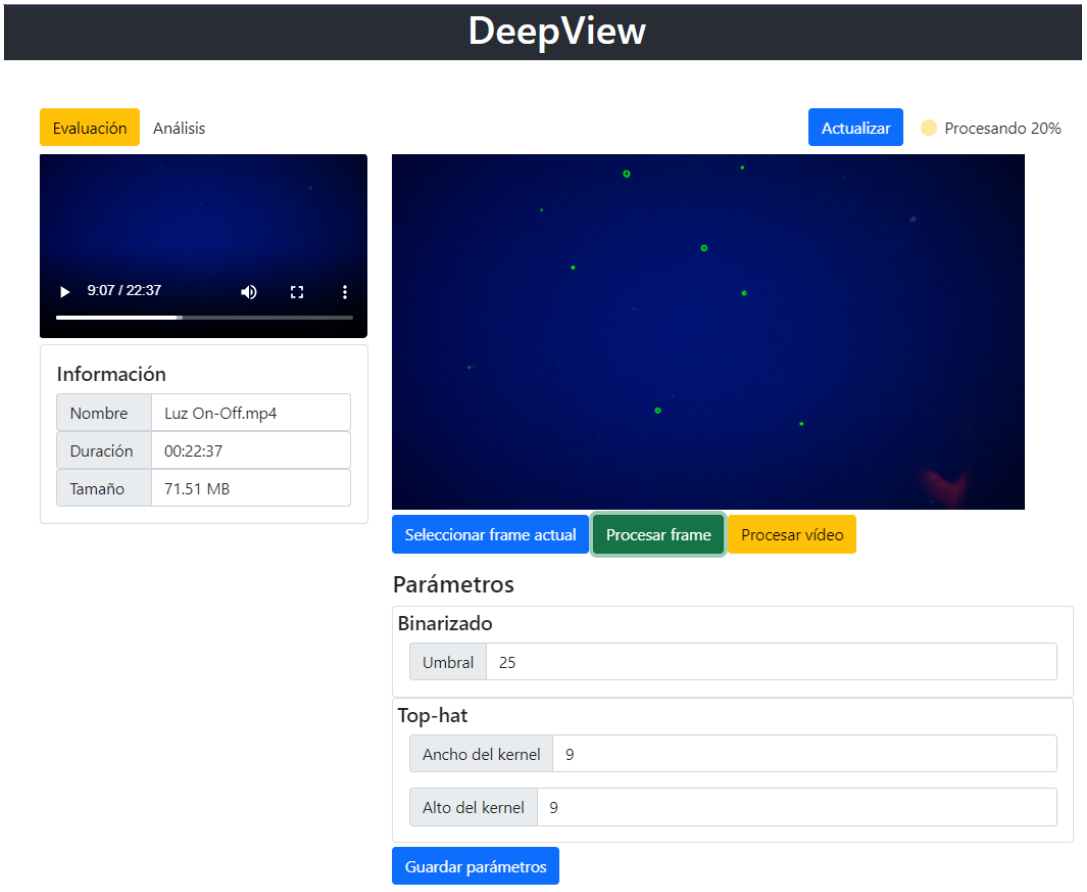


Figura 19. Modo evaluación

- **Modo Análisis:** en este modo el usuario es capaz de analizar gráficamente los resultados obtenidos tras el procesamiento del vídeo en cuestión. Consta de una gráfica que representa la densidad temporal de partículas en diferentes escalas (figura 20).

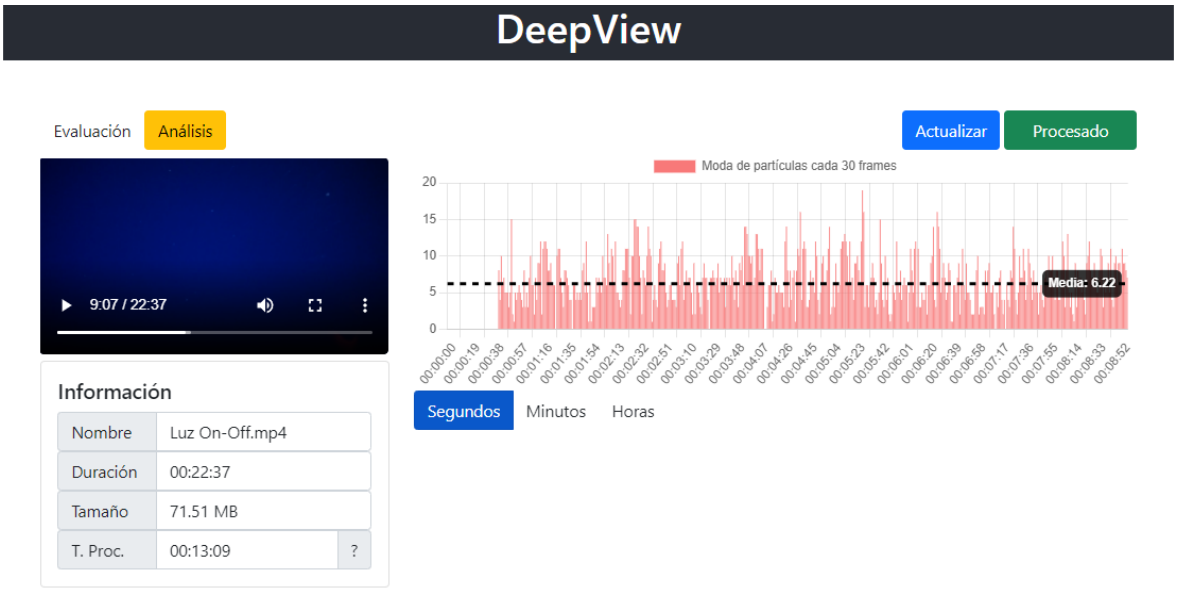


Figura 20: Modo análisis

2.3.3. Test de usabilidad

La usabilidad es un indicador que mide qué tan bien puede un usuario específico desenvolverse con una herramienta para alcanzar su objetivo [21]. Más específicamente, la usabilidad permite medir tres aspectos principales[22]:

- **Eficacia:** usuarios que han conseguido de manera correcta alcanzar su objetivo.
- **Eficiencia:** tiempo empleado para que el usuario cumpla su objetivo.
- **Satisfacción:** qué tan satisfecho se siente un usuario al utilizar la herramienta.

[21] https://www.interaction-design.org/literature/topics/usability . Accedido 06/07, 2022.
[22] https://blog.sinapsis.agency/definiciones-ux-usabilidad/ . Accedido 06/07, 2022.

En el caso de este trabajo, los usuarios finales de la aplicación son principalmente los miembros del equipo de biólogos que trabajan en el proyecto Deepcom.

2.3.3.1. Diseño

Para medir la usabilidad de la interfaz de usuario se diseñaron dos cuestionarios, el primero de ellos (apéndice 1) consta de preguntas específicas acerca de la aplicación y su contexto, mientras que el segundo (apéndice 2) se trata del *Sistema de Escalas de Usabilidad* [20], un método desarrollado por John Brooke en 1986 que permite evaluar de forma general los aspectos descritos en la sección anterior.

2.3.3.2. Resultados

En este *test* participaron tres miembros del equipo de trabajo de Deepcom, todos ellos pertenecientes al ámbito de la Biología. A pesar del limitado tamaño de la muestra, este *test* sirvió para conocer la opinión de los usuarios finales de la aplicación así como para recoger recomendaciones y sugerencias.

Como resultado, se concluyó que la aplicación obtuvo una gran aceptación dentro del grupo de usuarios finales, como se ve reflejado en la tabla 1 y en la tabla 2.

Aspectos específicos		Valoración general del usuario	
Métrica	**Media de las respuestas**	**Métrica**	**Media de las respuestas**
Eficacia	13 / 15	Contenido útil	9 / 9
Eficiencia	13 / 15	Comodidad	9 / 9
Satisfacción	15 / 15	Satisfacción	7 / 9
		Facilidad	9 / 9

Tabla 1. Resultados del test de usabilidad específico para Deepview

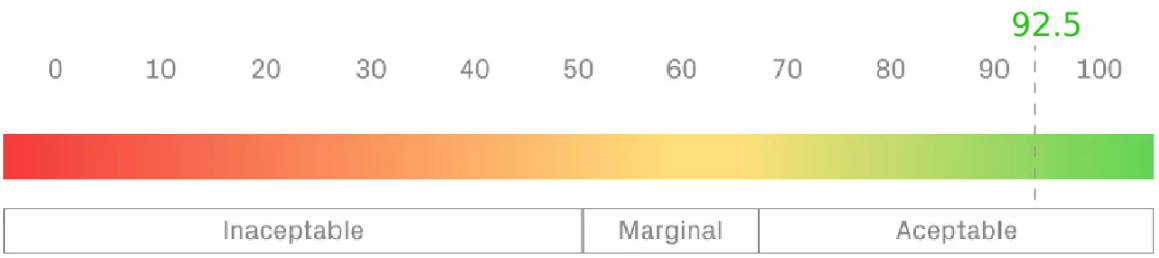


Tabla 2. Resultados del test de Sistema de Escalas de Usabilidad

3. Resultados

3.1. Producto final

Como producto final de este trabajo se desarrolló un sistema informático, basado en la arquitectura cliente - servidor, capaz de procesar vídeos capturados en los trabajos de campo realizados por el equipo del proyecto Deepcom. El procesamiento llevado a cabo por el sistema detecta las partículas en cada uno de los frames del archivo multimedia, sometiendo dicha imagen a una serie de filtros cuyos pasos están específicamente diseñados para este caso en particular.

Este sistema está compuesto, a grandes rasgos, por tres elementos principales:

- **deepviewcore:** un paquete de Python desarrollado para este caso particular y que implementa las funciones de procesamiento sobre los archivos de vídeo. Este paquete se encuentra publicado en el índice de paquetes oficial de Python[23]

- **Servidor o *backend*:** se trata de un servidor web desarrollado con una bifurcación específica de django adaptada para este caso en concreto. Se encuentra publicada en un repositorio personal del autor de este trabajo[24]

- **Cliente o *frontend***: el código cliente encargado de hacer de interfaz entre el usuario y el servidor de la aplicación. Es de tipo SPA y se desarrolló haciendo uso de la librería *React*. Se encuentra publicado en un repositorio personal del autor de este trabajo[25]

La versión final de este sistema está pensada para ser desplegada en un entorno local, esto es, el servidor y el cliente se ejecutan en la misma máquina. Sin embargo, gracias a la arquitectura utilizada, su modificación para ser desplegada en un servidor y usada en remoto en el futuro no resultaría compleja.

3.2. Manual de de usuario

Asimismo, teniendo en cuenta las disciplinas de los usuarios a los que está destinada la aplicación, mayoritariamente Biología marina, se elaboró un manual de usuario que, al término de la redacción de este documento, cuenta con una

[23] deepviewcore: https://pypi.org/project/deepviewcore/. Accedido 06/07, 2022.
[24] Servidor o *backend*: https://github.com/miguel-martinr/DeepView Accedido 06/07, 2022.
[25] Cliente o *frontend*: https://github.com/miguel-martinr/DeepView_frontend Accedido 06/07, 2022.

explicación detallada y serie de pasos para la instalación de la herramienta, así como con una descripción en detalle de los elementos de la interfaz gráfica y de su funcionamiento.

En la sección de pasos de instalación de este manual se detallan los siguientes pasos:

1. Descarga e instalación de la base de datos.
2. Descarga e instalación de Python.
3. Descarga e instalación de *pipenv*, el gestor de dependencias utilizado por DeepView.
4. Descarga e instalación de *git*, un gestor de versiones necesario para instalar la versión específica de Django modificada para este proyecto.
5. Descarga e instalación del sistema DeepView.

Cabe destacar que este manual de usuario se trata de un "documento vivo", esto es, el formato en el que está guardado es editable y extensible para así realizar mejoras continuas del mismo. Además, utiliza series de imágenes en formato *.gif* a modo de ejemplo para ilustrar los pasos de una forma amena y amigable.

Este documento se encuentra publicado en el directorio compartido en la nube, a través de *Google Drive*, de este trabajo[26]

4. Líneas futuras

A continuación, se describen las líneas futuras a seguir en este proyecto. Algunos de estos apartados se encuentran sugerencias propuestas directamente por los usuarios finales de la aplicación, obtenidas como resultado del breve estudio de usabilidad llevado a cabo; otros son el resultado de ideas surgidas durante el desarrollo del trabajo y que no fueron previstas en la fase inicial.

[26] Manual de usuario: https://docs.google.com/document/d/1VawEY0MafwXodMvFPfJLTGfBE83ELT0gTTtfZYZkLBs/edit?usp=sharing Accedido 06/07, 2022.

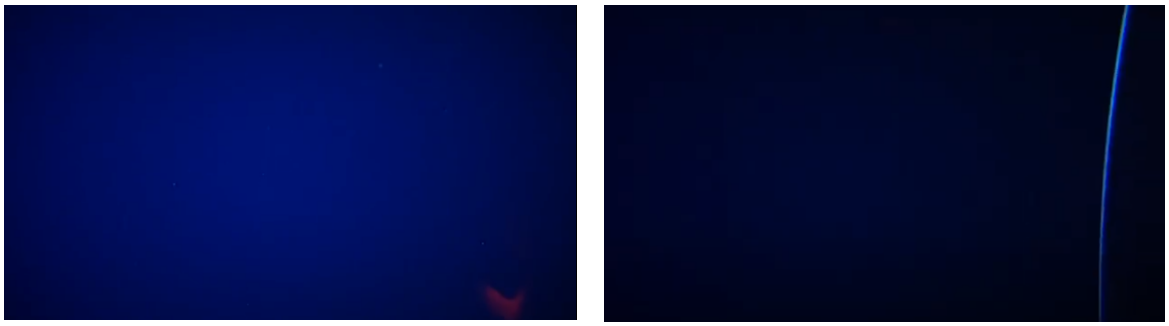
4.1. Estudio de eficacia y fiabilidad

Se propone realizar un estudio estadístico para determinar la fiabilidad del software y calcular el error aproximado que puede cometerse a la hora de detectar las partículas.

4.2. Detección de partículas

4.2.1. Sustracción dinámica del fondo de la imagen.

Uno de los inconvenientes encontrados en el procesamiento de la imagen fue la **presencia de estructuras fijas** no previstas en algunas de las grabaciones realizadas, ocasionada normalmente por reflejos de indicadores led de la propia cámara en el cristal de la cápsula o por los cables que la sujetan (figura 21). En general, estas estructuras pueden causar la detección de falsos positivos.



Reflejo rojo en la esquina inferior derecha

Cabo en el extremo derecho

Figura 21: Estructuras fijas en las imágenes

Para solucionar este problema se propone la aplicación de una sustracción dinámica del fondo que permita tener en cuenta las estructuras fijas como parte del fondo de la imagen.

4.2.2. *Tracking* de partículas

Actualmente, los datos obtenidos se corresponden con la masa de partículas presentes en cada uno de los frames por separado, por lo que se propone investigar posibles formas de rastrear partículas a través de los distintos frames, de manera

que los datos obtenidos se correspondan al número de partículas que aparecieron en escena.

4.2.3. Mejora de la detección: organismos grandes con puntos.

En algunas ocasiones, aparecen en escena organismos que presentan diversos puntos luminosos y que son detectados como partículas individuales, tal es el caso de los Sifonóforos (figura 22). Se propone mejorar la detección para tener en cuenta estos casos.

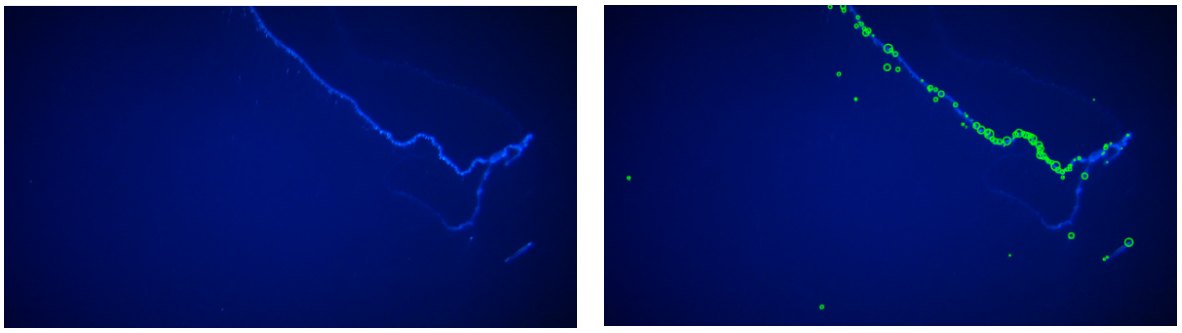


Imagen original Partículas detectadas

Figura 22. Sifonóforo

4.2.4. Detección de eventos

Una de las características que poseía el anterior software empleado para el procesamiento es la detección de eventos, entendiendo estos como la aparición de organismos de tamaño suficientemente grande como para realizar un análisis taxonómico, por ejemplo peces.

Se propone añadir esta funcionalidad al sistema DeepView, teniendo en cuenta las sugerencias siguientes:

- Obtener los instantes exactos en los que inicia y acaba el evento (tramos) y que, además, estos datos sean descargables en formato tabla, preferiblemente en un formato compatible con *Excel*.
- Obtener también como salida el fragmento del vídeo en el cual ocurre el evento para su posterior clasificación taxonómica.

4.3. Experiencia de usuario

En cuanto a la experiencia de usuario, se recogieron las siguientes sugerencias, presentadas en su mayoría por los usuarios finales de la aplicación:

- Abrir la herramienta en el navegador automáticamente una vez se inicia el servidor.

- Permitir que el usuario defina los tramos del vídeo que deben ser procesados. Esto es útil, sobre todo, para no procesar los momentos durante los que se prepara la cápsula, así como su inmersión y emersión.

- Añadir un botón a la *Página de Evaluación / Análisis* que permita volver a la *Página de vídeos* sin necesidad de hacer una nueva petición al servidor.

- Añadir la posibilidad de descarga de los resultados del procesamiento en formato tabla, preferiblemente en un formato compatible con *Excel.*

- Permitir que el usuario pueda descargar la gráfica de densidad temporal.

- Añadir un botón para cerrar todo el sistema.

4.4. Facilitar comparaciones

Actualmente, para cada vídeo presente en el servidor, solo puede haber un conjunto de datos asociados a su procesamiento, esto es, para comparar los resultados del procesamiento de un mismo vídeo con diferentes parámetros, es necesario crear una copia de ese vídeo y añadirla al servidor con un nombre diferente.

Se propone la posibilidad de asociar distintos datos de procesamiento para un mismo vídeo, de forma que se facilite la comparación de los resultados para diferentes parámetros.

5. Presupuesto

La tabla 3 presenta las horas empleadas en el desarrollo de las distintas fases del trabajo, así como sus costos correspondientes y períodos temporales en los que fueron distribuidas.

Actividad	Inicio	Final	Duración (h)	Coste unitario (€)	Total (€)	Febrero	Marzo	Abril	Mayo	Junio	Julio
Análisis del problema y definición de requisitos iniciales de la solución.	10/02/2022	24/02/2022	36	12.5	450						
Estudio de diferentes alternativas	25/02/2022	11/03/2022	48	12.5	600						
Diseño e implementación de la solución	12/03/2022	30/06/22	210	12.5	2625						
Elaboración de la memoria del trabajo	10/02/2022	06/07/22	90	12.5	1125						
TOTAL:			384		4800						

Tabla 3. Presupuesto general

6. Conclusiones

Como conclusión de este trabajo se ha desarrollado una aplicación web completa capaz de procesar imágenes submarinas que mejora el anterior método de procesamiento utilizado.

Esta aplicación hace uso de los resultados obtenidos después de analizar distintos métodos y técnicas para la mejora de las imágenes sobre las que realizar el procesamiento.

Además, se realizó un breve estudio de usabilidad para medir la eficacia, eficiencia y satisfacción de los usuarios finales de la aplicación. Este estudio demostró una gran aceptación y satisfacción por parte de los usuarios para con la aplicación, al tiempo que sirvió para recabar sugerencias y mejoras a implementar en la misma.

En resumen, con este trabajo se cumplieron la mayoría de los objetivos planteados en la fase de planificación, a excepción de la elaboración de un análisis estadístico de fiabilidad del software, y se establecieron las líneas de acción a seguir para continuar con el desarrollo y mejora de la herramienta .

7. Summary and conclusions

As a conclusion to this work, a complete web application capable of processing underwater images, that improves the previous processing method used, has been developed.

This application makes use of the results obtained after analyzing different methods and techniques to improve the images on which to perform the processing.

In addition, a brief usability study was carried out to measure the effectiveness, efficiency and satisfaction of the end users of the application. This study showed a great acceptance and satisfaction on the part of the users towards the application, while it served to collect suggestions and improvements to be implemented in it.

In summary, with this work most of the objectives set in the planning phase were fulfilled, with the exception of the elaboration of a statistical analysis of software reliability, and the lines of action to be followed were established to continue with the development and improvement of the tool.

8. Apéndices

8.1. Test de usabilidad específico para usuarios de DeepView

EFICACIA DE LA APLICACIÓN.		1	2	3	4	5
A1	¿Puedes identificar fácilmente qué representa la gráfica del modo "Análisis"?					
A2	¿Consideras que los datos presentes en las gráficas son útiles?					
A3	¿Consideras que es fácil calibrar los parámetros de procesamiento?					
EFICIENCIA DE LA APLICACIÓN.		1	2	3	4	5
B1	¿La aplicación es capaz de **procesar** los vídeos y mostrar los resultados en un tiempo aceptable?					
B2	¿Las **gráficas** son lo suficientemente flexibles para explorar los reultados?					
B3	¿La aplicación es **estable**? Es decir, ¿se visualiza correctamente?					
SATISFACCIÓN DEL USUARIO.		1	2	3	4	5
C1	Los contenidos de la interfaz siguen un orden coherente.					
C2	La aplicación es fácil de utilizar					
C3	La aplicación mejora el método de procesamiento previo.					

8.2. Sistema de escalas de usabilidad

	Total desacuerdo	Desacuerdo	Ni acuerdo ni desacuerdo	Acuerdo	Total acuerdo
Me gustará utilizar este sistema con frecuencia					
Encontré el sistema nnecesariamente complejo					
El sistema es fácil de usar					
Necesité el apoyo de un técnico para poder usar este sistema					
Las diversas funciones en el sistema están bien integradas					
Hay muchas incosistencias en el sistema					
La mayoría de la gente puede aprender a utilizar el sistema rápidamente					
El sistema es engorroso de usar					
Me sentí muy seguro con el sistema					
Tuve que aprender muchas cosas antes de poder utilizar el sistema					

9. Bibliografía

1. Schettini, R., Corchs, S. *Underwater Image Processing: State of the Art of Restoration and Image Enhancement Methods*. EURASIP J. Adv. Signal Process. 2010, 746052 (2010). Artículo.

2. Bazeille S, Quidu I, Jaulin L, Malkasse JP: *Automatic underwater image pre-processing. Proceedings of the Caracterisation* du Milieu Marin (CMM '06), 2006. Artículo .

   Bazeille S: Vision sous-marine monoculaire pour la reconnaissance d'objets, Ph.D. thesis. Université de Bretagne Occidentale; 2008. Artículo

3. L. Maddalena and A. Petrosino, "*A Self-Organizing Approach to Background Subtraction for Visual Surveillance Applications*," in IEEE *Transactions on Image Processing*, vol. 17, no. 7, pp. 1168-1177, Julio 2008, doi: 10.1109/TIP.2008.924285. Artículo

4. D. M. Rashed, M. S. Sayed and M. I. Abdalla, *"Improved moving object detection algorithm based on adaptive background subtraction," 2013 Second International Japan-Egypt Conference on Electronics, Communications and Computers* (JEC-ECC), 2013, pp. 29-33, doi: 10.1109/JEC-ECC.2013.6766380. Artículo

5. Weinstein B. Motion meerkat. http://benweinstein.weebly.com Sitio web. Accedido

   05/06, 2022.

6. Salhaoui, M.; Molina-Molina, J.C.; Guerrero-González, A.; Arioua, M.; Ortiz, F.J. *Autonomous Underwater Monitoring System for Detecting Life on the Seabed by Means of Computer Vision Cloud Services*. Remote Sens. 2020, 12, 1981. Artículo

7. Sobrino Martínez, Águeda.”*Analysis of images of deep water cameras for studies of mesopelagic communities*”. Repositorio institucional de la Universidad de La Laguna. Julio 2021. Repositorio.

8. Bradski G. *The OpenCV library*. *Dr. Dobb's Journal of Software Tools*. 2008.

9. OpenCV. *OpenCV documentation* Sitio web. Accedido 28/06/, 2022.

10. Vélez Serrano J. Visión por computador. Madrid: Dykinson; 2003:112.

11. ThorLabs, Camera Noise and Temperature Tutorial Sitio web. Accedido 29/06, 2022.

12. OpenCV. *Color conversions* Sitio web. Accedido 03/07, 2022.
13. OpenCV. *Morphological operations* Sitio web. Accedido 03/07, 2022.
14. OpenCV. *Simple thresholding.* Sitio web. Accedido 03/07/2022.
15. OpenCV. *Contour approximation methods.* Sitio web. Accedido 03/07, 2022.
16. OpenCV. *Minimum enclosing circle.* Sitio web. Accedido 03/07, 2022.
17. OpenCV. *Contour area.* Sitio web. Accedido 04/07,2022.
18. Django *Software Foundation*, 2019. Django, Sitio web. Accedido 04/07, 2022.
19. Thomas Fielding. *Architectural Styles and the Design of Network-based Software Architectures*, 2000. Disertación. Accedido 05/07, 2022.
20. Brooke, John. (1995). SUS: A quick and dirty usability scale. Usability Eval. Ind.. 189.