UNIVERSITAT DE BARCELONA

Treball final de grau

GRAU D'ENGINYERIA INFORMÀTICA

Facultat de Matemàtiques i Informàtica
Universitat de Barcelona

Tratamiento de imágenes acuáticas para el estudio del fondo marino

---

Autor: Joan Benito Martínez

Director: Núria Pujol Vilanova
Realitzat a: Departamento de matemáticas e informática

Barcelona, 22 de juny de 2017

Abstract

The final project of my degree, Treatment of aquatic images for the study of the seafloor, talks about the search of problems that we can have to take aquatic photographies. Also, we can know the possible ways to determine and improve the results obtained.

The images in this project are taken from a sporty camera anchored to a robot ship that is unmanned. This ship tours a programmed way in a region, always two meters from the seafloor. However, the results obtained for the camera aren't professionals, so the main objective is to find the best way to treat the images by a software and try to get valid images to do any study with them. Histogram equalization should be the answer, I have been able to check that adaptative equalization with algorithms CLAHE is a good option.

Furthermore, these images suffer the movement of the ship. The movement is caused by the addition of underwater currrents, the vibration of the engines and the auto correcction in order to follow the way. I have wanted to demostrate this event creating a kind of panoramic. The purpose is to see how far the ship suffers the deviation. I have proposed a way for the deviation calculation. This way, consist in compare centers of the image with the projection and translation that it suffers while the panoramic was done. So that, the calculation give us a value pretty close comparing with the maker. It's approaching to 3% of error rate.

Finally, with the little remaining time, I have introduced a little purpose to detection of vegetation. It's an orientative way in order to search pixels inside a colour range. This method can improve with cascade classifiers or with the characteristics to indentify pattern.

Resumen

El proyecto final de grado **tratamiento de imágenes acuáticas para el estudio del fondo marino**, trata sobre la búsqueda de los problemas que tenemos al hacer fotografiás acuáticas procedentes de una cámara de bajo coste (baja calidad, mala estabilidad de imagen, etcétera) y las posibles formas de hacerlas útiles en estudios científicos sobre el hábitat marino.

Las imágenes que presenta este proyecto son tomadas desde una cámara deportiva anclada a un vehículo autónomo submarino, esta nave recorre un trayecto programado en una zona, siempre a dos metros del fondo marino. Tenemos que, los resultados obtenidos por la cámara no son los deseados, por tanto el objetivo principal es el de encontrar la mejor forma de tratar las imágenes por software y conseguir unas imágenes validas para realizar cualquier estudio con ellas. La ecualización de histograma suele ser la respuesta, he podido verificar que la ecualización adaptativa con algoritmos CLAHE es un verdadero acierto.

Además, estas imágenes sufren el movimiento del vehículo. El movimiento es causado por el sumatorio de las corrientes marinas, la vibración de los motores y

la auto corrección de la nave para seguir el camino. He querido demostrar este suceso creando una especie de panorámica proyectada para ver hasta que punto el vehículo sufre la desviación. Para obtener mayor veracidad, he propuesto una forma de cálculo de desviación, esta consiste en comparar centros de imágenes con la proyección y traslación sufrida mientras se formaba la panorámica, nos da un valor bastante aproximado a lo que nos promete el fabricante. Se acerca al 3% de tasa de error, a lo que seguir el rumbo se refiere.

Por último, con el poco tiempo restante he introducido una pequeña propuesta de detección de vegetación. Es una forma orientativa basada en buscar los píxeles dentro de un rango de color. Este método fácilmente se puede mejorar con clasificadores en cascada o el uso de las características para identificar patrones.

Agradecimientos

Quiero agradecer a Núria, mi tutora del trabajo final de grado, por su apoyo y guia a la hora de hacer este proyecto. También quiero agradecer a mi pareja Anna, por la paciencia que ha demostrado y darme el espacio que he necesitado. Por último y no menos importante, a mi familia, porque siempre están ahí, contigo, incondicionalmente.

Índex

1 Introducción y motivación

El trabajo final de grado que seguidamente se les explica es el de Tratamiento de imágenes acuáticas para estudio del fondo marino. Este, consiste en el filtrado y manipulación de imágenes tomadas bajo el agua para una mejora de contraste y visibilidad.

Mi motivación para realizar este trabajo es el de profundizar en temas de visión artificial, conseguir un mejor entendimiento de estos temas, hechos por encima en el grado. Además, la complejidad adicional en el tratamiento de imágenes acuáticas. Dónde las cámaras tienen un hándicap negativo y es que hay el comportamiento marino hace que los colores que corresponden a diferentes frecuencias quedan absorbidos de forma no uniforme, por lo que tenemos resultados con unos colores no reales.

El código de este proyecto se ha generado enteramente en lenguaje python, versión 2.7, usando la herramienta jupyter notebook de anaconda. Además, las principales librerías incorporadas para poder trabajar y tratar las imágenes han sido OpenCV, Skimage, Numpy y pyplot de matplotlib.

Las imágenes utilizadas en este proyecto son todas cedidas por la **Unidad de Tecnología (UTM) del Consejo Superior de Investigaciones Científicas (CSIC)** , estas imágenes han sido tomadas desde una cámara deportiva estilo 'GoPro', este tipo de cámaras vienen con una funda impermeable para poder hacer tomas acuáticas. La óptica de estás cámaras es una gran angular.

La cámara iba anclada a un vehículo autónomo submarino (AUV), concretamente era un vehículo basado en el Iver2-580 del fabricante Oceanserver, este vehículo de forma tubular recorre un trayecto prefijado por el fondo marino. Está programado para ir navegando a dos metros del fondo marino con una profundidad máxima de hasta cien metros, la velocidad a la que los motores impulsan la nave es de una media de 1.5 nudos o convertido unos 0.8 metros por segundo.

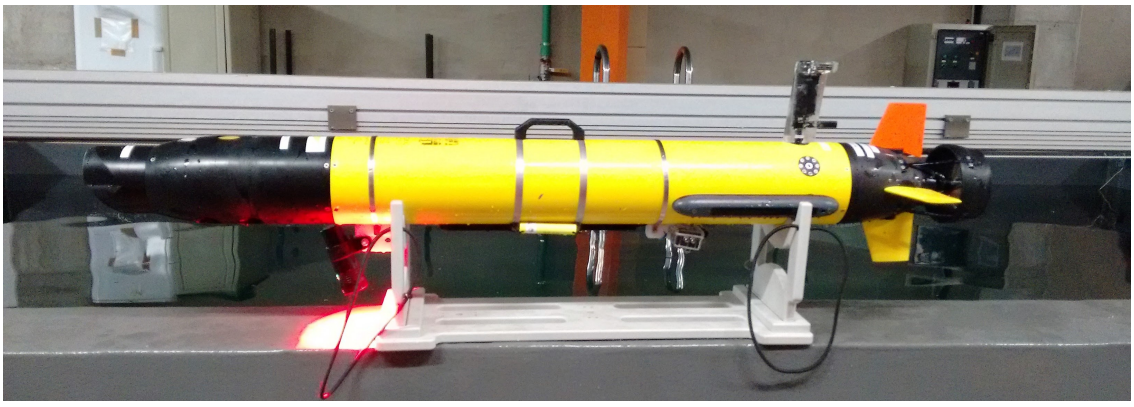


Figura 1: Imagen del vehículo autónomo submarino usado para la expedición

Debo decir que el tema de este trabajo final de grado es fácilmente extensible, una vez vas metiéndote en el tema observas que hay muchas posibilidades y tareas

con las que puedes aportar información para un mayor estudio. Es un ámbito puramente científico y se intenta conseguir que estas imágenes puedan valer para la mayor parte de proyectos e investigaciones.

2 Planificación

2.1 Teórica

El proyecto podría separarse en tres etapas generales. Preparación, desarrollo y documentación. Se estipuló un período de un mes para cada etapa, con revisiones presenciales cada una o dos semanas haciendo meetings en persona. La etapa de preparación es necesaria para saber que buscas y a que te enfrentas, no hay que menospreciarla. La etapa de desarrollo es la etapa en la que se genera código, pruebas y compruebas resultados. La etapa de documentación nos da un tiempo para crear una memoria, acabar de entender los resultados y explicarlos. Una dedicación ideal diaria sería, unas dos horas de lunes a viernes.

2.2 Real

El plan seguido realmente no fue tan optimista, debido a falta de tiempo no se pudo hacer un seguimiento tan ideal. Se usó mucho el mail para intentar mitigar la falta de meetings en persona. La etapa de preparación y desarrollo fue prolongada más tiempo del pensado por algunas dificultades a la hora de entender el tema planteado. Además, pasé bastante tiempo enfocando mal el camino del proyecto. Por tanto, la etapa de documentación ha sido bastante corta y a marchas forzadas. A su vez, es una etapa en la que terminas de afianzar conceptos y tienes que hacer cambios.

3 Objetivos

3.1 Tratamiento y mejora de imágenes acuáticas

El principal objetivo de este trabajo final de grado es el de obtener una mejora en la calidad de imágenes acuáticas.

Este objetivo, que podríamos considerar como primer paso, es primordial ya que necesitamos tener unas imágenes claras y con unos colores reales. Ya que después estas imágenes van a ser usadas para desarrollar proyectos, aplicaciones, funciones de detección y reconocimiento o casi cualquier cosa que se requiera de imágenes acuáticas. Se busca que las imágenes tengan unos colores realistas y un contraste acorde a la composición.

Mi meta era conseguir encontrar el método o manera capaz de hacer la transformación necesaria y hacer de las imágenes tomadas un material útil.

3.2 Construcción de panorámica para demostrar la desviación del vehículo

Este objetivo trata de unir una serie de imágenes para formar una especie de panorámica con la que poder hacer una aproximación de la desviación sufrida por el AUV. El motivo de este objetivo es, debido a que el medio acuático tiene una serie de condiciones adversas que sufre el vehículo cuándo está haciendo un trayecto. Estas condiciones mencionadas van desde las corrientes marinas, la propia vibración de los motores o el reajuste de rumbo que hace su sistema de navegación basado en compás magnético.

Mi motivación es llegar a dar una aproximación tangible, un número a tener en cuenta al hacer un despliegue de un vehículo en una campaña oceanográfica.

4 Desarrollo

4.1 Herramientas

Para el desarrollo del código he usado el lenguaje de programación Python, concretamente la versión 2.7.13. Junto con el pack de librerías y herramientas para Python Anaconda, versión 4.3.0.

Dentro de Python hemos usado diversas librerías, la que más se ha usado, debido a que su especialización en tratamiento de imagen es **OpenCV**, adicionalmente para la ecualización adaptativa **Skimage**. Después para computo de matrices **Numpy**. Y para el dibujo de resultados e histogramas **Matplot**.

El proyecto ha sido desarrollado enteramente en un ordenador portátil HP Envy-13 con un procesador i7-7500U (2 nucleos), 8 giga bytes de memoria RAM y 250 giga bytes de almacenamiento en un disco duro SSD. El sistema operativo dónde ha sido creado y probado es Windows 10 Home.

4.2 Primer objetivo. Tratamiento de imágenes acuáticas

Para empezar, el agua es un medio más denso que el aire. Esto significa que los elementos como la luz, partículas, etc, tienen un comportamiento diferente que en aire. Para comprender un poco estos problemas focalizados en fotografía tuve que documentarme sobre problemas de luz, color... Para empezar, hubo que documentarse sobre los problemas que existen en las imágenes tomadas bajo el agua. Los cambios físicos son perceptibles desde el minuto uno. En el agua tenemos condiciones completamente diferentes en cuestiones de reflexión y absorción de rayos. Esto implica una serie de problemas, voy a explicar los principales y que más nos importan para llegar a nuestro objetivo:

- **Pérdida de luz**: este problema se puede dividir en dos principales motivos, primero la superficie del agua, que actúa a modo espejo y refleja una buena cantidad de rayos de luz solar de vuelta al aire. El otro principal motivo es por la densidad del agua, que absorbe los rayos mucho más rápido que el aire, por lo que conforme aumente la profundidad la cantidad de luz decae de manera drástica. Además, otro problema de su densidad es que sostiene mucha materia en suspensión por lo que la trayectoria de la luz se ve obstaculizada.

- **Pérdida de color**: el agua afecta al color, esta al ser de un fuerte cian o azul verde va absorbiendo los diferentes colores a diferentes velocidades, o mejor dicho, profundidades. Tenemos que el color rojo es absorbido sobre unos 5 o 10 metros. El color verde consigue ser visible sobre unos 30 metros. Y, por último, el color azul que consigue predominar junto con el negro hasta algo más de 60 metros.

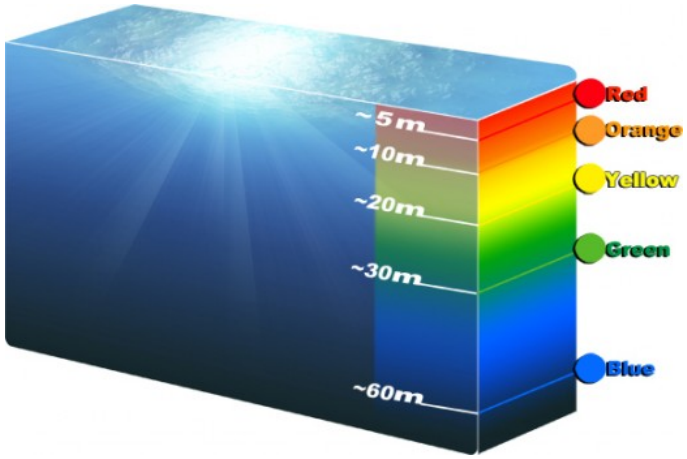


Figura 2: Imagen que muestra hasta que profundidad llegan los colores

- **Pérdida de contraste**: otro gran problema es la perdida de contraste, este problema es derivado directo de la perdida de luz. Al no tener una buena iluminación perdemos visibilidad, por lo que a su vez perdemos distancia de visión. También sufrimos cuando tenemos rocas con pequeñas brechas o partes más profundas, todo esto nos lleva a tener colores muy extenuados, perdemos contraste.

Después de saber los problemas que conllevan las fotografías acuáticas, empecé a mirar idiomas y/o librerías que me ayudaran al tratamiento de las imágenes. Python fue el idioma elegido, por su sencillez a la hora de empezar a obtener resultados y fácil manejo, además dispone de su versión de la librería OpenCV, la cual nos facilita el trabajo con las imágenes y nos proporciona una comunidad en la red bastante significativa para el tema que estamos tratando.

El primer paso es cargar la imagen, usando la función cv2.imread de OpenCV, ya que nos transforma la imagen en una matriz con la información y sus tres canales. La función cv2.imread pasa la imagen leída al modelo de color BGR, por lo que hacía una conversión a RGB con la función cv2.cvtColor.

Una vez tenemos la imagen cargada tenemos diversas funciones con diferentes desarrollos a la hora de tratar las imágenes, he dejado en mi versión final del script preparado para ejecutar todas ellas, son funcionales y con sus características finales en la imagen. En el apartado de pruebas añado los diferentes resultados.

4.2.1 Tratamiento por ecualización de histograma

Antes de seguir con la explicación, haré un pequeño repaso del histograma. El histograma es básicamente una gráfica que te muestra la cantidad de píxeles (eje de las y) que hay por cada nivel de gris, o color (eje de las x).

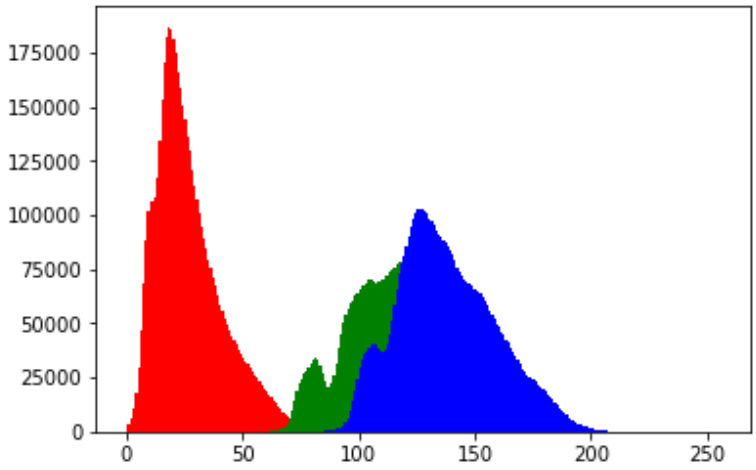


Figura 3: Histograma de los canales RGB de una imagen

El histograma es útil para hacer un análisis con el que poder apreciar contraste e intensidad. Si alteramos el histograma de una imagen esto se traduce en cambios en la imagen. Un problema frecuente y que se puede detectar en el histograma es, cuando el rango de niveles de un canal se encuentra concentrado en una zona, decimos que la imagen tiene poco contraste.

En el caso de las imágenes acuáticas es justamente este, el poco contraste en las imágenes. Tenemos un método de solventar este problema y tiene que ver con el histograma. La ecualización del histograma.

La ecualización por histograma es una especie de estiramiento del histograma. La idea es reescalar valores de píxeles para llenar todo el rango del histograma.

Esta metodología trata de hacer una aproximación dispersando los picos del histograma de la imagen y dejando intactas las partes bajas del histograma, esto se consigue con una función de transferencia que tiene una alta inclinación siempre que el histograma presenta un pico y una baja inclinación con el resto del histograma.

Esta forma de mejorar el contraste es francamente útil cuando tenemos imágenes con fondos y primeros planos que sean ambos brillantes o ambos oscuros.

Por el contrario, este método es indiscriminado, por lo que podría en algunas situaciones, aumentar el ruido que se encuentra en el contraste de fondo y por lo tanto disminuir la señal utilizable. Un problema que he sufrido al realizar pruebas, es que en imágenes con baja profundidad de color no genera muy poca mejora de contraste, incluso en ocasiones el resultado es menos satisfactorio que la imagen original. Además, suele producir efectos no realistas en las fotografías.

La implementación de la ecualización del histograma se basa es la función de

distribución acumulativa (CDF). Esta expresa la probabilidad acumulada asociada con una distribución. Con ella se determina la probabilidad de que un valor sea menor que otro valor, mayor que él o que este entre dos valores dados.

Al final del proceso de ecualización añado un paso adicional para ayudar a mejorar el contraste, tiene que ver con mejorar el brillo y la saturación. Se realiza el estiramiento (ecualización) del histograma de los canales S (saturación) y V (valor, o lo que es lo mismo brillo) de la imagen pasada de RGB (ya ecualizada) a HSV.

A la hora de implementarlo en el código he seguido los siguientes pasos:

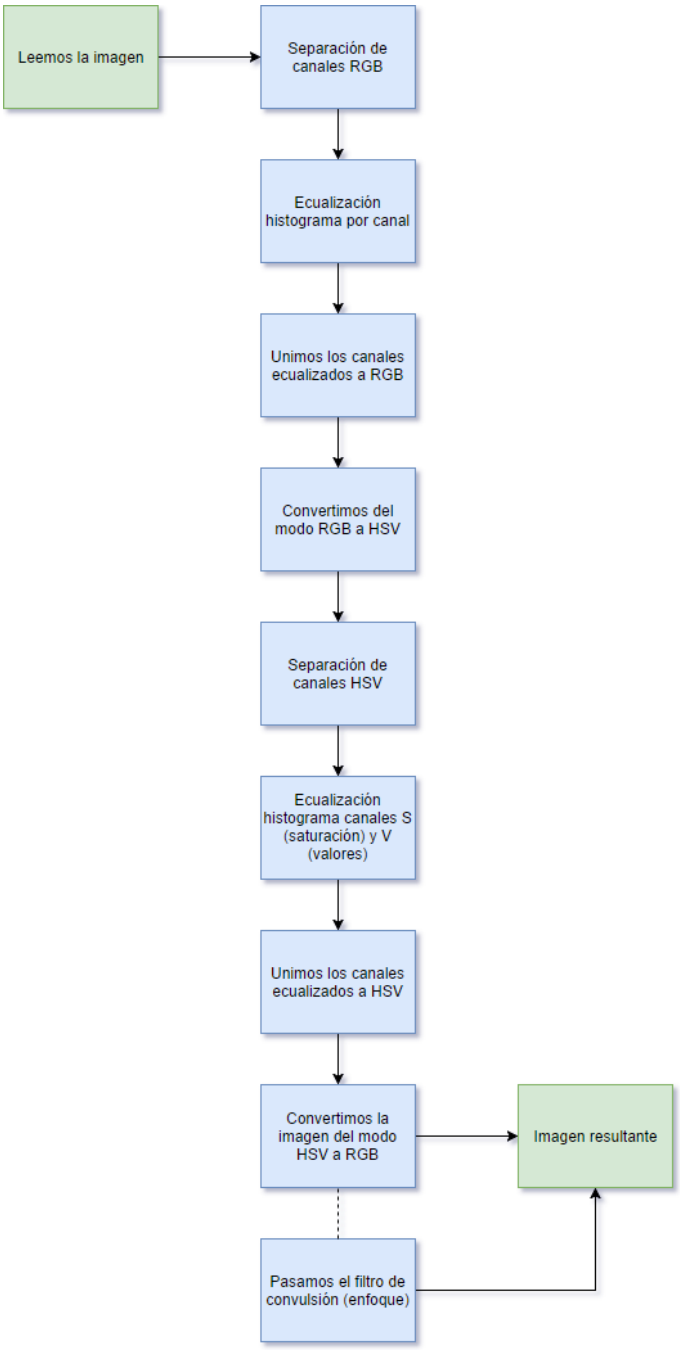


Figura 4: Diagrama de flujo con los pasos de la ecualización ordinaria

- Se separan los canales de la imagen con la función cv2.split, obteniendo los valores de cada color (r, g, b).

- Realizamos la ecualización de histograma por cada canal, usando la función cv2.equalizeHist.

- Unimos los canales ecualizados al modo de color RGB, con la función cv2.merge.

- Ahora la imagen es convertida del modo de color RGB a HSV, con la función cv2.cvtColor. Ya que vamos a tocar la luminosidad y la saturación.

- Se vuelve a hacer el separado de canales, esta vez obteniendo el matiz (H), la saturación (S) y el valor (V) que vendría siendo el brillo.

- Y, se ecualiza el histograma de la saturación y del valor (brillo).

- Hacemos la unión de los canales HSV.

- La imagen vuelve a pasarse al modo de color RGB para su ilustración y guardado.

4.2.2 Tratamiento por ecualización adaptativa

La ecualización adaptativa difiere de la ecualización ordinaria en que básicamente la ecualización de histogramas se realiza de diferentes regiones de la imagen. Estos histogramas los utiliza para redistribuir los valores de luminosidad de la imagen. Por tanto, es adecuado para mejorar el contraste local y la mejora de las definiciones de bordes en cada región de una imagen. Solo aplica la ecualización en el canal V (valor) del modo de color HSV, por lo que no tenemos una traslación de picos dando lugar a colores quemados o poco fieles a la realidad.

La adaptación de la ecualización del histograma (AHE) genera un nuevo valor en cada píxel con una función de transformación derivada de los valores vecinos, la forma más simple es que el píxel se transforma en base al histograma de un cuadro que lo rodea. La función de transformación usada para la derivación es la misma que en la ecualización ordinaria, la función de distribución acumulativa.

La ecualización adaptativa que he utilizado se basa en el uso del algoritmo CLAHE. Este es un algoritmo de mejora de contraste local. Al tratarse de una mejora local, podemos obtener mejoras en las partes más oscuras y profundas, además de tener una colores más fieles a la realidad.

El uso de la variante con el algoritmo CLAHE se justifica para evitar la tendencia a sobre amplificar el ruido en regiones ya homogeneizadas. Esto es causado porque las zonas homogéneas tienen un fuerte pico en el histograma, por lo que la función de transformación asignará un rango estrecho de valores de píxeles a todo el rango de la imagen, por lo que aparecerá más ruido.

En cuanto el problema que tenemos con los píxeles situados en el borde la imagen, que no disponen de los suficientes píxeles vecinos, extendemos la imagen a ambos lados de los cuales no existe imagen haciendo espejo de las respectivas partes contrarias. Por ejemplo, si nos situamos en la esquina superior izquierda de una imagen, los píxeles que harán espejo son los situados abajo del píxel a calcular para extender la parte de arriba del píxel, y los píxeles situados a la derecha de él para obtener a su izquierda.

Para realizar la ecualización adaptativa del histograma, he usado la librería Skimage tiene el método exposure.equalize_adapthist el cual convierte la imagen al modo de color HSV, corre el algoritmo CLAHE sobre el canal V y devuelve la imagen a RGB. He dejado la máscara (kernel) por defecto, es 1/8 de la imagen de anchura y 1/8 de la imagen de altura. Aunque el contraste es mayor al tener máscaras más pequeñas, pero se necesita más tiempo de computación. Para evitar la sobreamplificación de ruido que la ecualización adaptativa provoca en el histograma, CLAHE limita la amplificación por el recorte del histograma a un valor predefinido antes de calcular la CDF. Esto limita la pendiente de la CDF y por lo tanto de la función de transformación. La función nos permite seleccionar el recorte (clip_limit) que sufrirá el histograma antes de entrar en la función de distribución (CDF), a mayor valor tendremos un mayor contraste. He podido comprobar que un exceso de contraste puede llegar a ser contraproducente para el resultado final de la imagen.

A la hora de implementar el código he seguido los siguientes pasos:

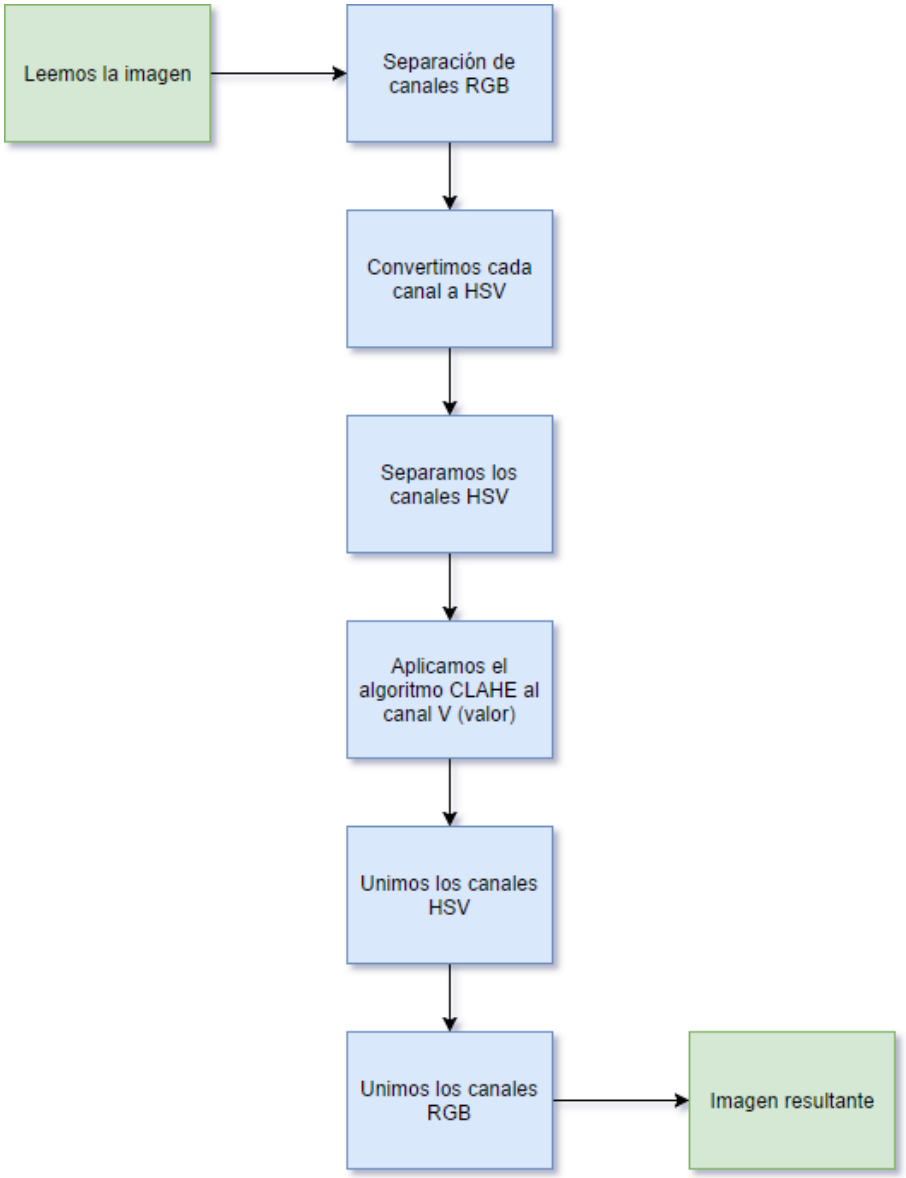


Figura 5: Diagrama de flujo con los pasos de la ecualización adaptativa

- Separar los canales de color de la imagen RGB con cv2.split.
- Realizar por cada canal la función exposure.equalize_adapthist, de la librería Skimage.
- Hacer un merge de los tres canales ya ecualizados.

La función exposure.equalize_adapthist realiza tres pasos:

- Convierte la imagen al modo de color HSV, en nuestro caso convierte la imagen por cada canal de color RGB.

- Aplica el algoritmo CLAHE sobre el canal V (valor).
- Convierte el canal de nuevo al modo de color RGB.

4.3 Segundo objetivo. Panorámica

El objetivo de la panorámica es el de poder ver la trayectoria real que sigue el vehículo a la hora de tomar imágenes del fondo marino. Este va sufriendo un desplazamiento involuntario por culpa de las corrientes marinas. El propio vehículo no es capaz de calcular este error ya que el sensor gps no recibe señal debajo del agua. Aunque el fabricante nos proporciona un dato sobre el error del sistema de navegación, este está basado en un compás magnético y nos asegura que el error esta por debajo del 3

Para poder realizar una unión en las imágenes y así crear una panorámica, se ha utilizado lo que se conoce en el campo de la visión artificial como s**image stitching**. Para ello hay una serie de pasos que expongo en el siguiente diagrama de flujo.

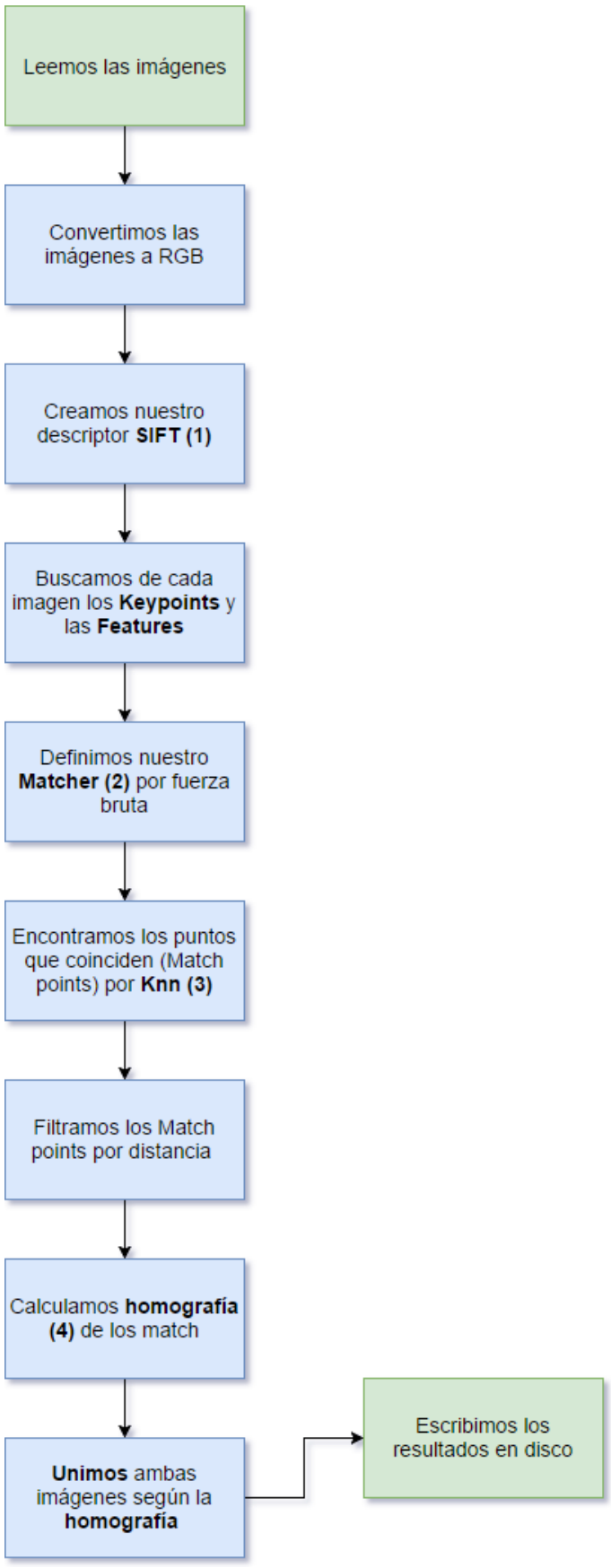


Figura 6: Diagrama de flujo sobre la creación de la panorámica

Explicación de algunos puntos del método usado.

1. **SIFT** Algoritmo muy usual en el campo de la visión artificial, usado para extraer características relevantes de una imagen. Estás características pueden usarse en diferentes ámbitos, en nuestro caso será utilizado para la unión de imágenes para poder formar una panorámica. Cuando hablamos de características relevantes hablamos de puntos de interés, estos son puntos que guardan información del entorno (por ejemplo, bordes). Además, suelen ser invariantes a la transformaciones locales, globales y a las rotaciones. SIFT es el primer algoritmo que además de ser invariante a las transformaciones y rotaciones también lo es al escalado. Partes principales:

   - **Detección de extremos en la escala-espacio** En esta etapa se compara cada punto de escala-tiempo con los valores de sus vecinos en la misma escala y en las escalas anteriores y posteriores. Esto se hace con la diferencia gaussiana con distintos tamaños de región y así buscar máximos locales.

   - **Localización de puntos de interés** La localización de los puntos de interés se afina a nivel sub-pixel usando la expansión de la serie de Taylor de la escala-tiempo. Si el valor del punto de interés encontrado es menor que cierto umbral el punto es descartado. Además, SIFT utiliza una matriz hessiana (matriz cuadrada n x n de las segundas derivadas parciales) para calcular las curvaturas principales y así descartar los bordes que no son esquinas.

   - **Asignación de orientaciones** A cada punto de interés se le asigna una orientación para garantizar que al rotar la imagen lo podamos perder. Se calcula haciendo la magnitud y dirección del gradiente de los puntos vecinos, se hace un histograma y el mayor pico indica la orientación.

   - **Descriptor de puntos de interés** Se crean descriptores del punto, cada uno es de 16 x 16 del vecindario del punto. Estos se dividen en bloques de 4 x 4 y en cada ellos se crea histograma de orientaciones. Después se hace una concatenación en un vector de valores de los bloques de cada histograma.

   - **Correspondencia de puntos de interés** La correspondencia entre los puntos de interés de dos imágenes se obtiene a través de una búsqueda del punto más próximo en el espacio de los descriptores de puntos de interés.

2. **Matcher fuerza bruta** El matcher de fuerza bruta es la forma más simple de hacer un match. Toma el descriptor de una de las características del primer set y lo compara con el resto del segundo set ejerciendo un cierto cálculo de la distancia. El más cercano se devuelve y ya tenemos el match.

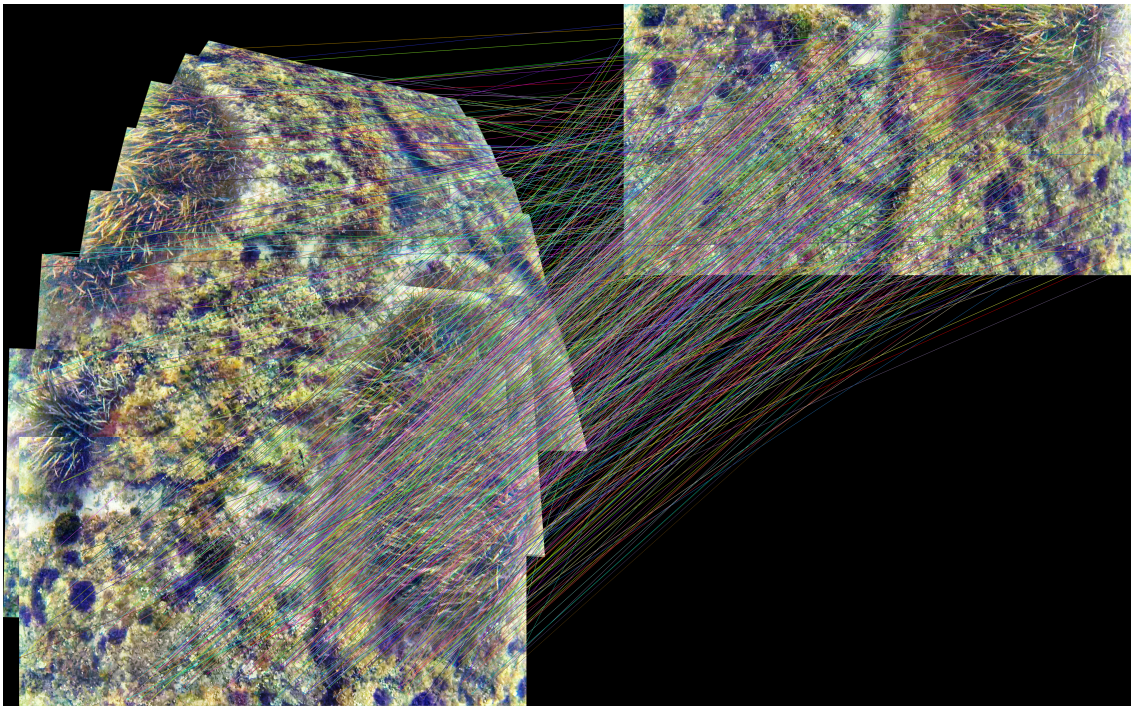


Figura 7: Imagen que muestra los puntos de match de una imagen con el resto de la panorámica

- **Matcher FLANN** Dentro del código he realizado una pequeña variante a la hora de hacer el match de los puntos característicos. En vez de utilizar un matcher de fuerza bruta, he decido dar a elegir entre el de fuerza bruta o el de FLANN (Fast Library for Approximate Nearest Neighbors). Este contiene una colección de algoritmos optimizados para la búsqueda de vecinos en caracteristicas de grandes dimensiones. Hay una ventaja clara, el tiempo de computación, FLANN busca realizar la tarea de encontrar matches en mucho menos tiempo que la búsqueda por fuerza bruta.

3. **Knn** El Algoritmo Knn, o k vecinos más cercanos, es usado como método de clasificación de elementos basado en un entrenamiento mediante ejemplos cercanos en el espacio de los elementos. En Knn la función se aproxima solo localmente y todo el cómputo es diferido a la clasificación.

4. **Homografía** Es la transformación proyectiva que determina una correspondencia entre dos figuras geométricas planas, por lo que cada uno de los puntos y rectas de una figura corresponden con un punto y una recta.

4.3.1 Cálculo de desviación

Para demostrar aún más la desviación que sufre la nave he querido hacer un cálculo aproximado, para poder hablar con algo más de propiedad.

Debido que tenemos una panorámica con proyección, he marcado la primera imagen que va a formar la panorámica con un punto de color azul en el centro de la foto situado en el 0 de las y, o lo que es lo mismo arriba de todo.

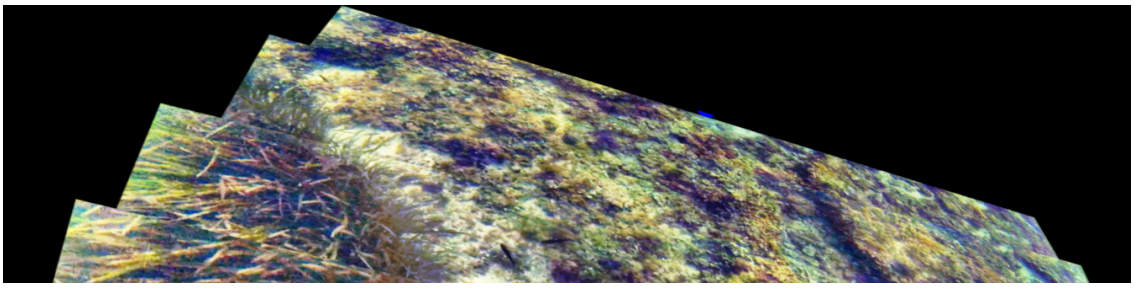


Figura 8: Punto azul situado en la parte superior de la panorámica, centro x de la primera imagen.

Este punto azul tiene sentido porque lo que queremos es añadir un punto de refencia, desde dónde podemos empezar a calcular la desviación. El punto lo podemos encontrar haciendo una máscara del color del que se ha pintado (0,0,255), después tenemos en cuenta las coordenadas del único punto. Ya que es un punto puramente azul, no hayamos falsos positivos porque el resto de píxeles azules de la imagen disponen de más valores en el resto de canales. Gracias a esto, independientemente de la cantidad de imágenes que vayamos a unir detrás o de las diferentes traslaciones que vaya sufriendo la panorámica es posible seguir encontrándolo ya que el punto sufre la proyección y traslación como el resto de puntos. Se busca que conservamos en la propia imagen panorámica resultante información de dónde empezó el centro de nuestra panorámica.

He pintado un segundo punto, esta vez de color rojo, para marcar el centro (de las abscisas también) pero situado en la parte baja de la última imagen que hemos añadido a la panorámica.

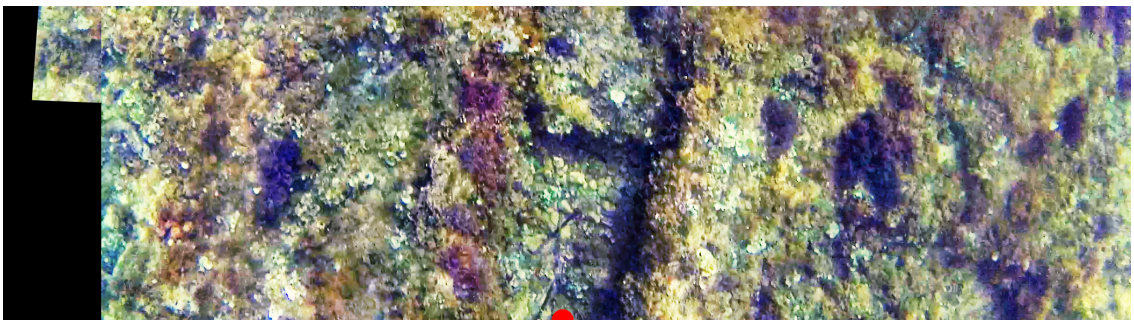


Figura 9: Punto rojo situado en la parte inferior de la panorámica, centro de las abscisas de la última imagen.

Este método aproximado, se basa en calcular la distancia euclidiana del punto rojo y el punto azul pero únicamente su valor de x. Por lo que es como si bajásemos el punto azul hasta abajo de todo de la panorámica. El punto azul lo hemos mantenido arriba para guardar el punto de inicio real mientras se va deformando a medida que añadimos más imágenes para formar la panorámica.

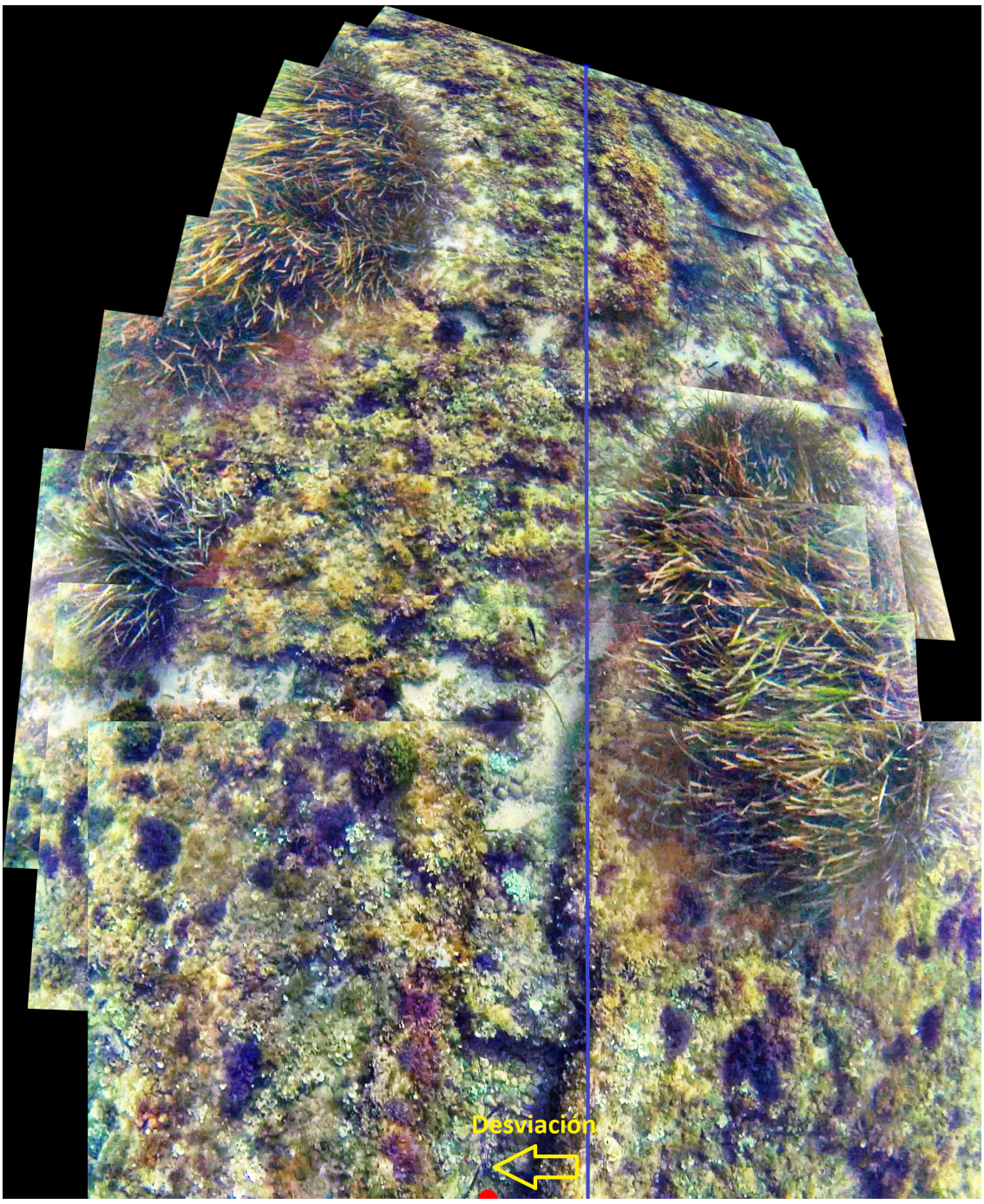


Figura 10: Imagen que muestra la desviación de la nave.

Si la cámara se ha movido lateralmente, debería haber una distancia entre ambos centros. Para darle un valor tangible lo he pasado a metros, esto lo he podido hacer

porque tenemos una distancia 'real' teórica entre centros de una imagen y otra, unos 0.8 metros. Este número viene de que el vehículo se mueve a 0.8 metros por segundo cuando esta en modo grabación y he extraído dos imágenes por segundo del vídeo.

He puesto 'real' teórica porque para tener una distancia real valida deberíamos tener alguna especie de referencia en las imágenes, por ejemplo un doble puntero láser con el que tener siempre una distancia de referencia al tomar imágenes.

4.3.2 Primer acercamiento a la detección de vegetación

Con el poco tiempo restante que me queda, he decidido encarrilar el trabajo a un apartado que interesa mucho a investigadores del fondo marino **la detección de vegetación**. Es una funcionalidad que me hubiera gustado haber expandido y hecho a consciencia ya que es clave para hacer investigaciones sobre la calidad del agua o la degradación del fondo marino y un largo etcetera.

He dado una pequeña pincelada al tema aprovechando que hago detección del punto de color azul para el cálculo de la desviación y intentar buscar las tonalidades verdes, tan características de la plantas, e intentar hacer una máscara con ella. Es básicamente pasar la imagen al modo de color HSV y buscar todos los píxeles que estén en un rango definido por nosotros. Después se cuenta cuántos píxeles no son cero (negro) y junto con el total de píxeles se puede obtener una pequeña referencia del porcentaje de vegetación que hay en la imagen panorámica.

Remarco que es un inicio y no es ni de cerca la mejor forma de detectar vegetación. Lo ideo sería obtener una buena base de imágenes de train y junto con unos clasificadores en cascada ir detectando por patrones que es es arena, roca y planta.

5 Pruebas y resultados

5.1 Tratamiento de imágenes

5.1.1 Prueba 1 - Ecualización ordinaria

La primera prueba se basa en la ecualización del histograma ordinaria en los canales RGB, usando la función de OpenCV equalizaHist en cada canal de color.

Imagen original y su histograma:

Figura 11: Frame original usado para prueba 1

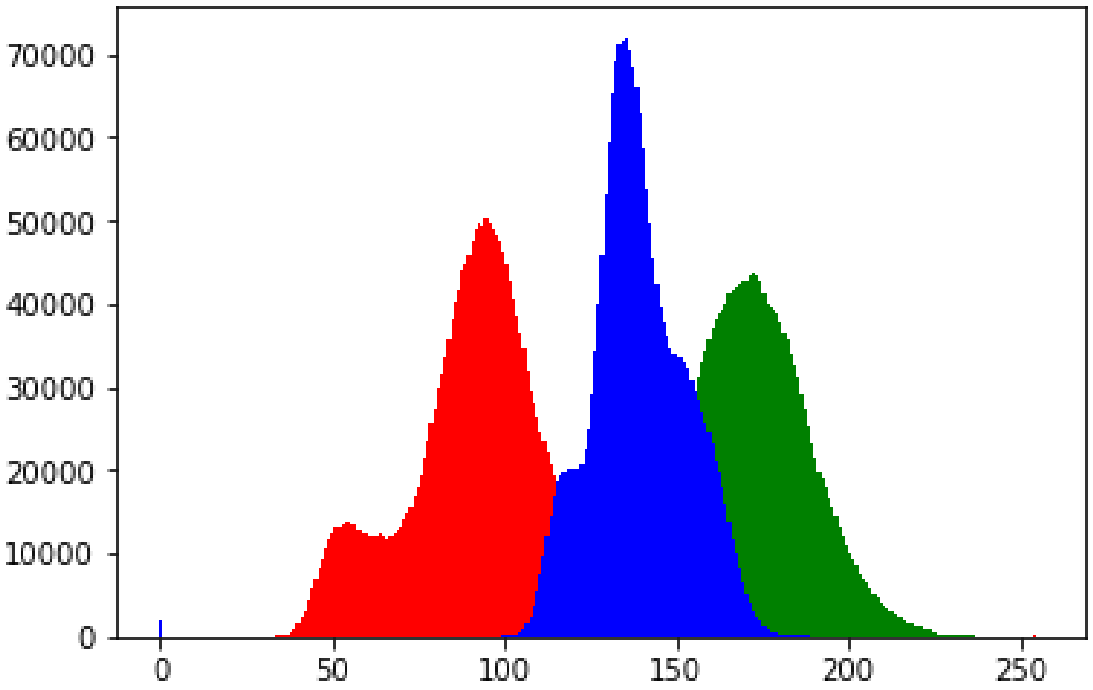


Figura 12: Histograma del frame original de la prueba 1

Imagen ecualizada y su histograma:

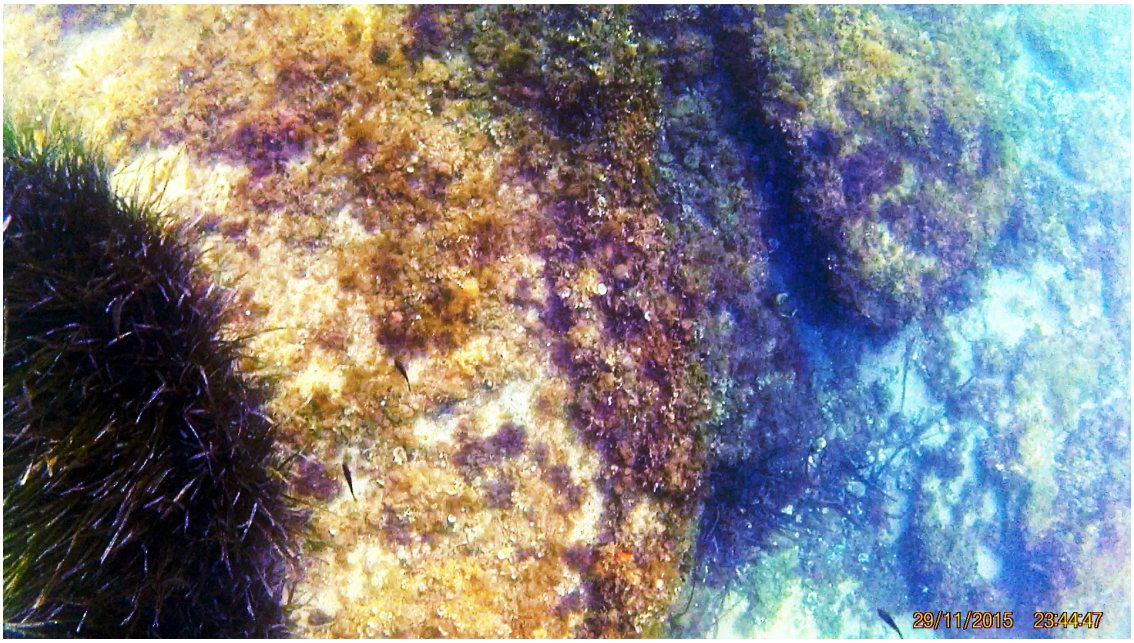


Figura 13: Imagen ecualizada del Frame original de la prueba 1

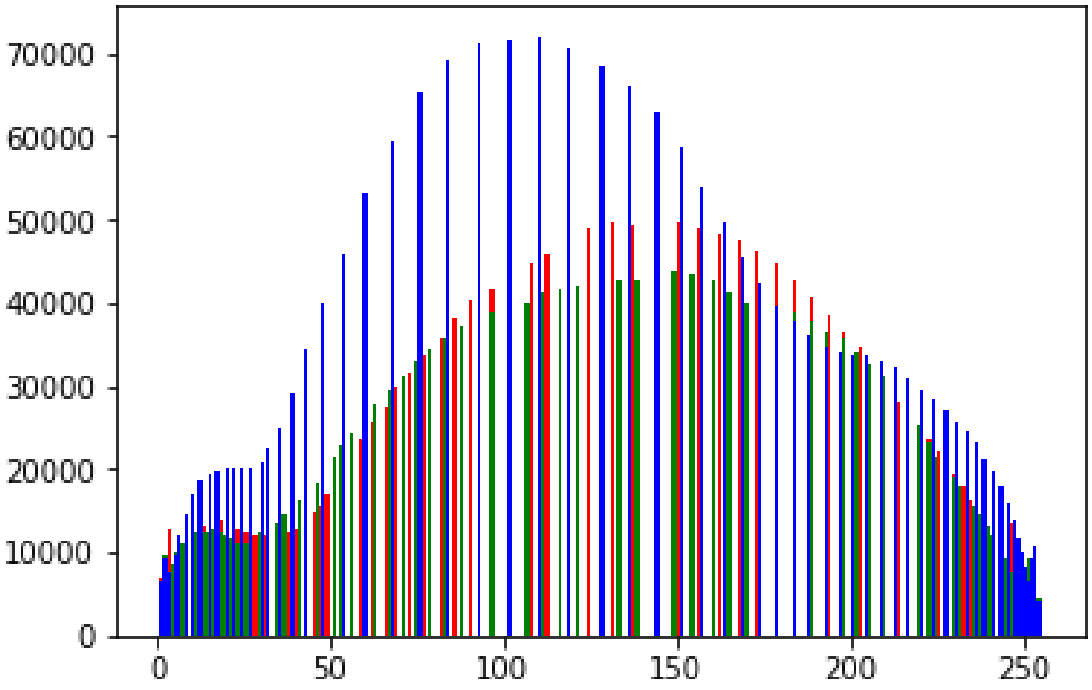


Figura 14: Histograma de la imagen ecualizada de la prueba 1

Una vez visto el resultado podemos apreciar una mejora en el contraste, tenemos de nuevo el color de rojo presente en la imagen. Por el contrario como punto negativo los niveles de colores no son realistas, los valores de rojo son demasiado altos y los azules no han bajado lo necesario. Al tener una parte más profunda en la imagen, la ecualización no ha hecho un buen trabajo en esa región.

Podemos apreciar que la imagen es demasiado brillante, esto ocurre porque hemos repartido los máximos de los canales RGB, no hemos tocado el nada en el modo de color HSV en el cuál podemos variar los niveles de brillo.

El tiempo de computación es el más bajo, era de esperar ya que la complejidad de la función es muy baja. Los tiempos están en el rango de los 0.013 y 0,015 segundos de ejecución.

5.1.2 Prueba 2 - Ecualización ordinaria modificada

Adicionalmente a lo hecho en Prueba 1, ahora se añade una ecualización de histograma ordinaria a los canales S y V del modo de color HSV, después de la ecualización en RGB.

Imagen original y su histograma:

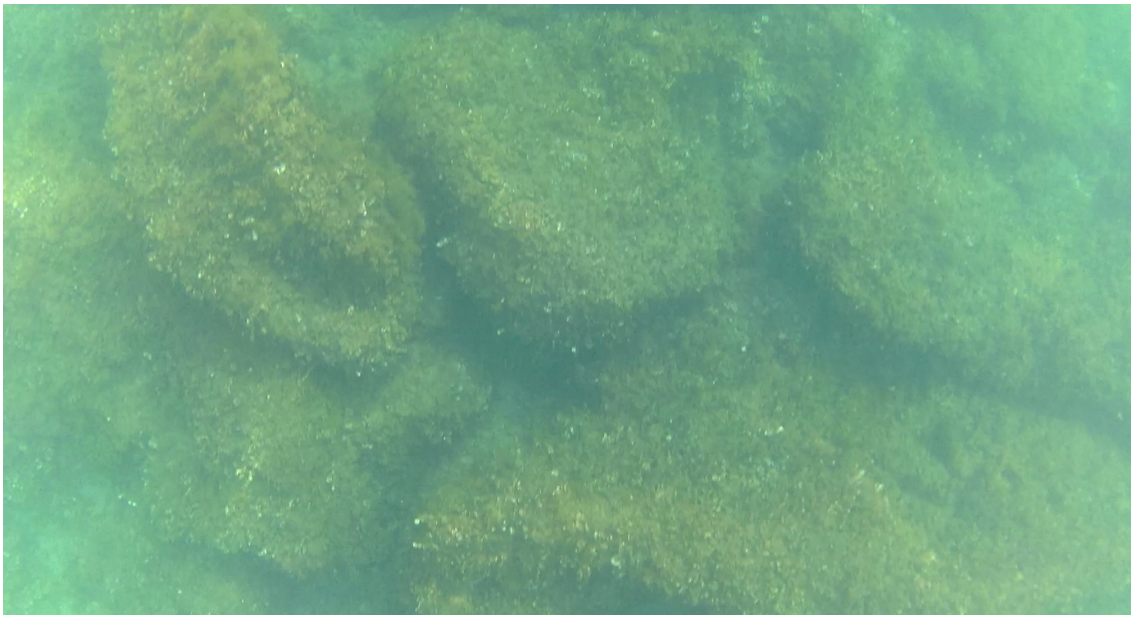


Figura 15: Frame original usado para la prueba 2

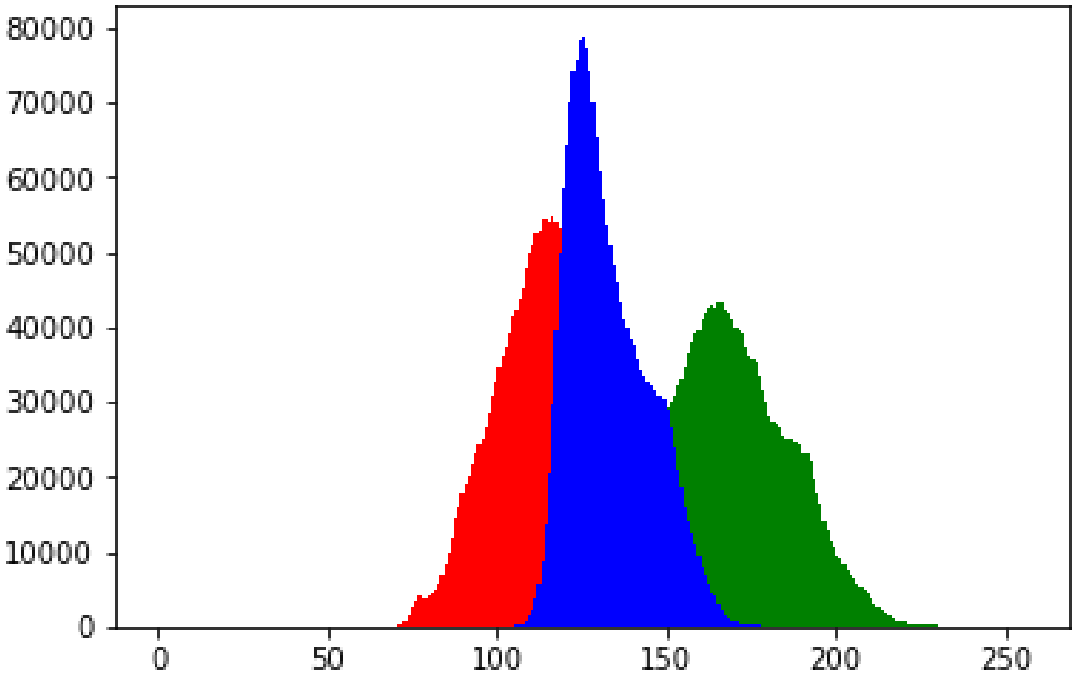


Figura 16: Histograma del frame original usado en la prueba 2

Imagen ecualizada (RGB + HSV) y su histograma:

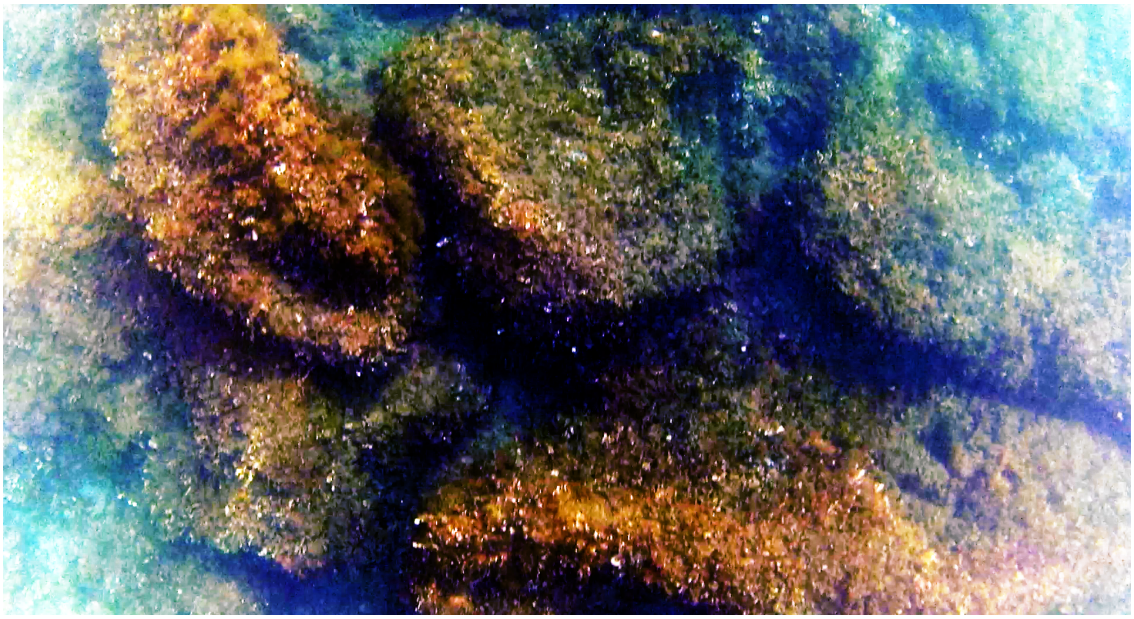


Figura 17: Imagen ecualizada del frame original de la prueba 2

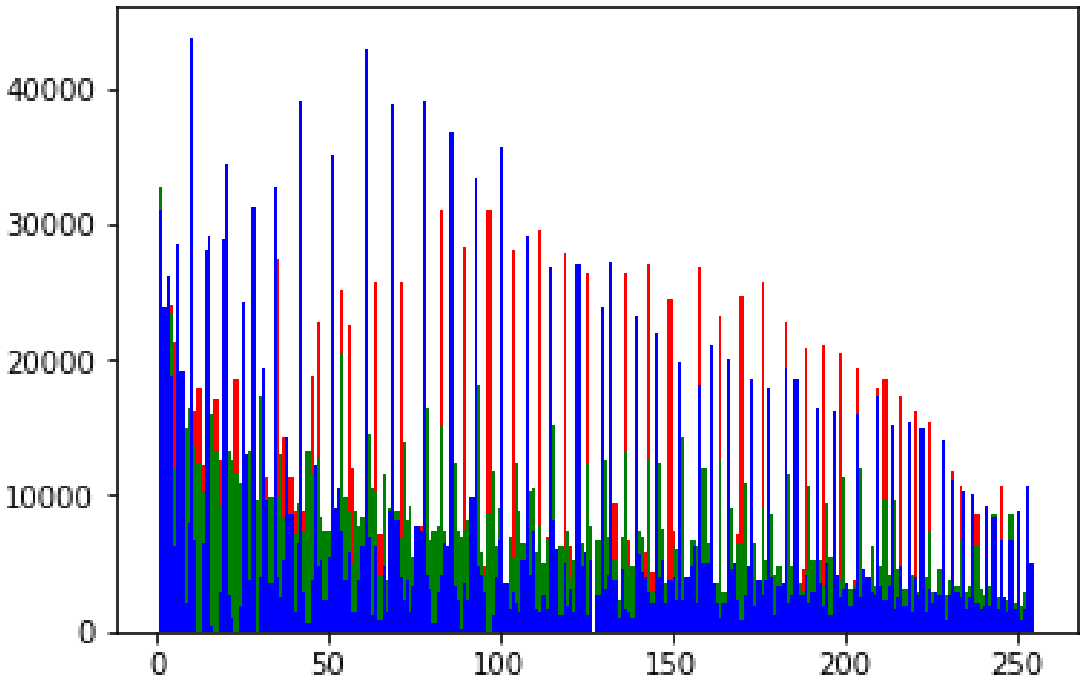


Figura 18: Histograma de la imagen ecualizada en la prueba 2

Vuelto a hacer el mismo proceso que en el paso 1 para tener una comparación con la misma imagen.

Imagen ecualizada (unicamente modo color RGB, exactamente como prueba 1) y su histograma:

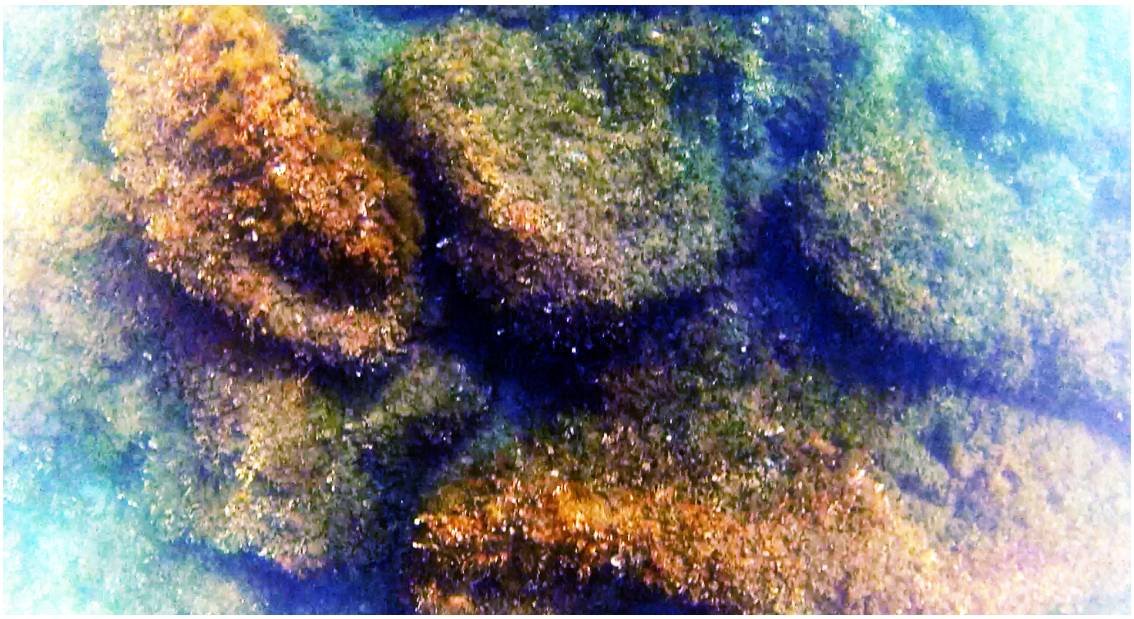


Figura 19: Imagen ecualizada usando frame original de la prueba 1

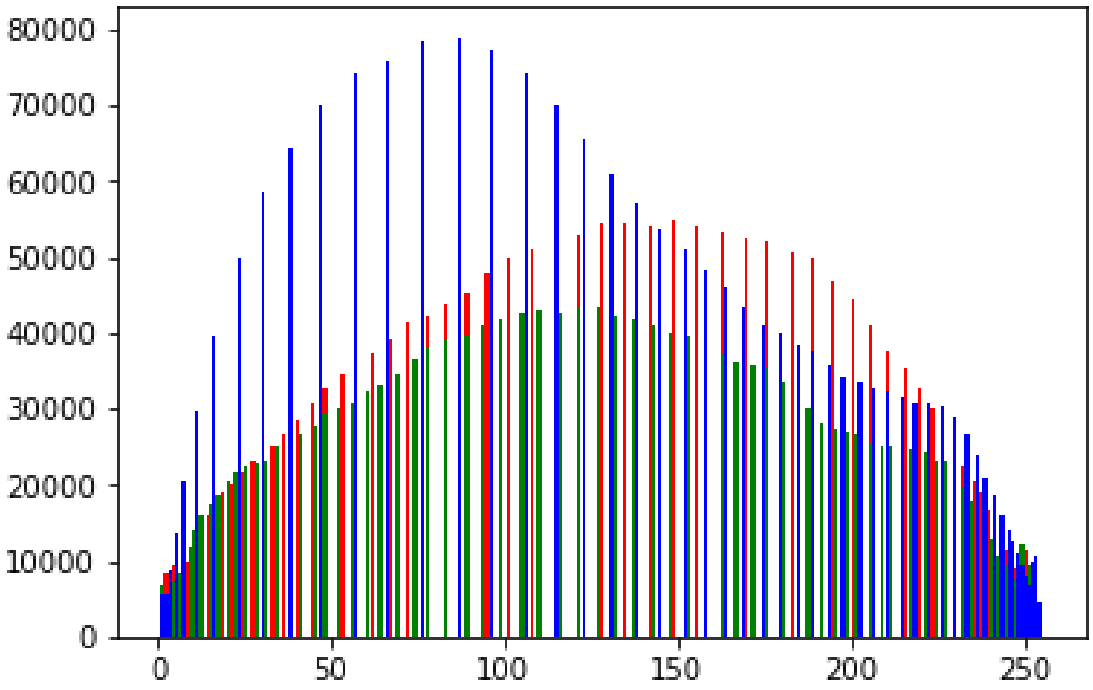


Figura 20: Histograma de la imagen ecualizada anterior

Podemos apreciar que la ecualización de los canales S y V del modo de color HSV han resultado en tener una mejor saturación y un menor nivel de iluminación en la imagen. Aunque aún sufrimos niveles de color poco realistas y algo de sobre brillo por los laterales de la imagen que son las partes más alejadas (esto ocurre por la deformación que ejerce la óptica de la lenta).

El histograma nos indica una menor saturación en los niveles bajos y altos en los canales de color RGB, hace un mejor reparto de picos por el histograma. En general podemos decir que hemos ganado algo más de calidad en comparación.

El tiempo de computación de esta función esta en el rango de los 0.067 a 0.084 segundos, son casi 5 veces más que la ordinaria sin el retoque en el modo de color HSV, pero igualmente es un tiempo más que razonable por la ganancia obtenida adicional.

5.1.3 Prueba 3 - Ecualización adaptativa

En esta prueba tenemos una ecualización adaptativa del histograma, usando la función exposure.equalize_adapthist de la libreria Skimage.

Imagen original y su histograma:

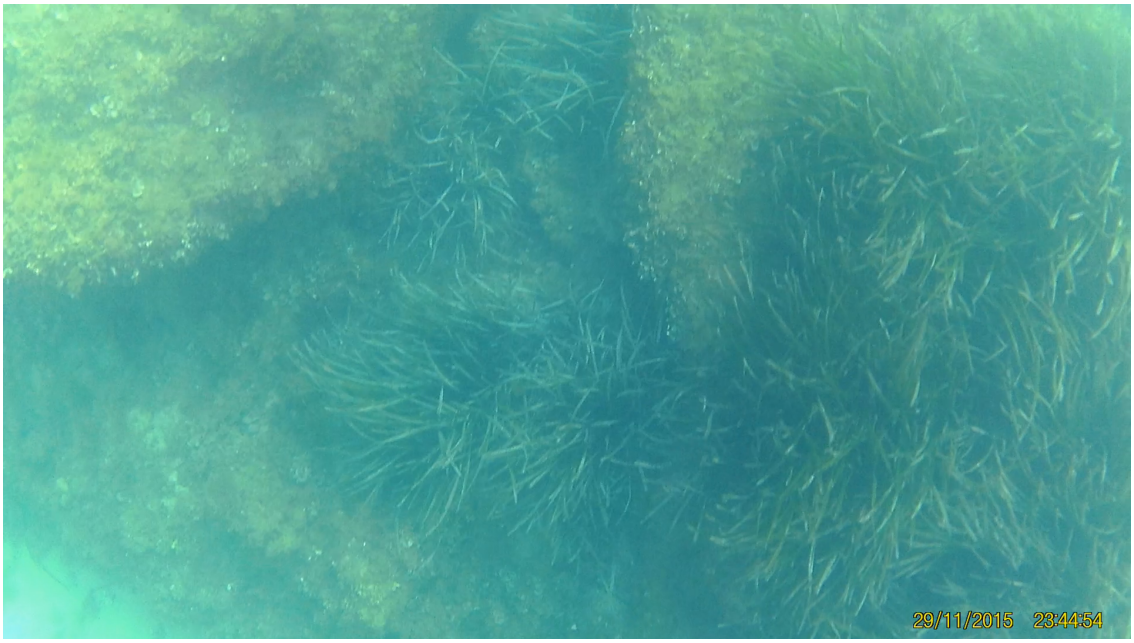


Figura 21: Frame original para la prueba 3

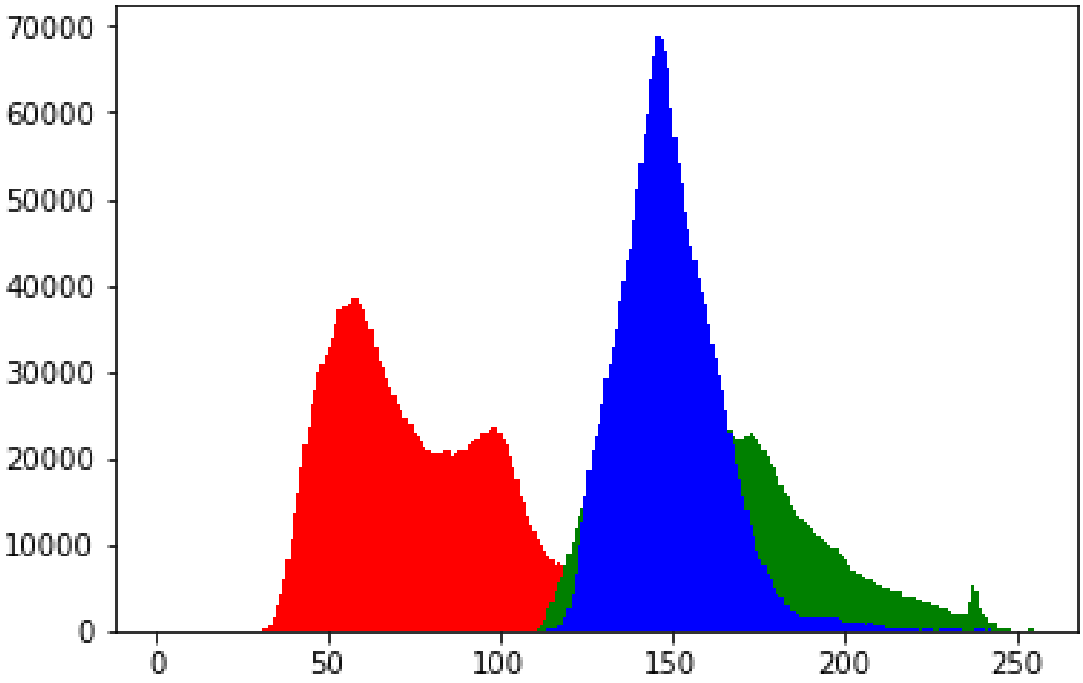


Figura 22: Histograma del frame original de la prueba 3

Imagen ecualizada (ecualizado adaptativo) y su histograma:

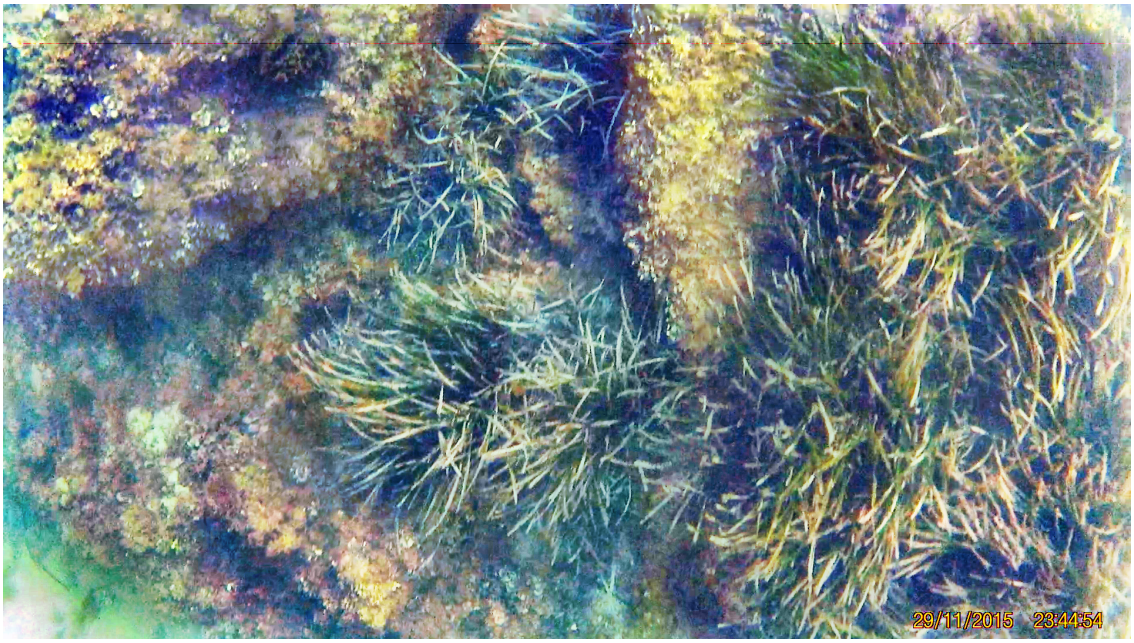


Figura 23: Imagen ecualización adaptativa de la prueba 3

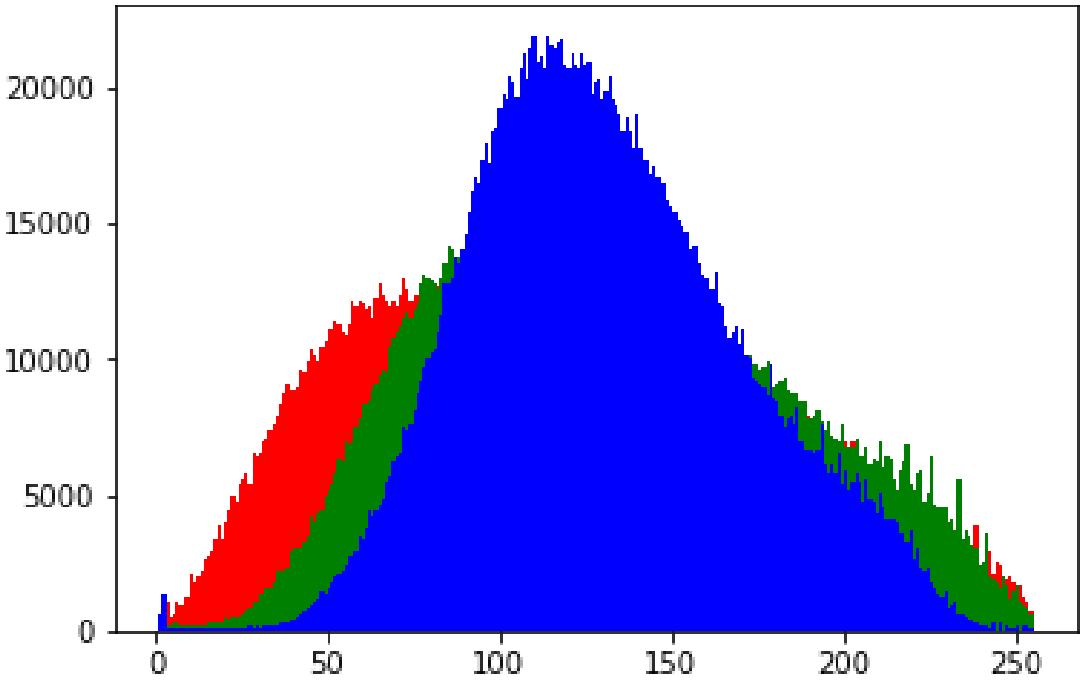


Figura 24: Histograma de la imagen ecualizada de la prueba 3

Después de ver el resultado de la ecualización adaptativa podemos concluir que hemos obtenido un resultado deseado. Al utilizar un algoritmo que realiza ecualizaciones locales, comprobamos que las partes más profundas o con más distancia

son ecualizadas correctamente, al contrario que en los métodos ordinarios. El histograma sigue una forma fiel al original pero con una mayor amplitud de valores, motivo por el cual el problema de los colores poco realistas es solucionado.

El tiempo de computación es del rango de 0.47 a 0.6 segundos. El tiempo de ejecución con respecto el ordinario (con retoque HSV) ha aumentado en algo más de 7 veces. No cabe duda que si queremos el mejor resultado deberemos hacer la ecualización adaptativa. Ahora, si buscamos una aplicación de mejora a tiempo real, quizás no es la mejor solución por el retraso que podría aportar este método.

5.1.4 Prueba 4 - Ecualización adaptativa con límite de recorte

Imagen original y su histograma:

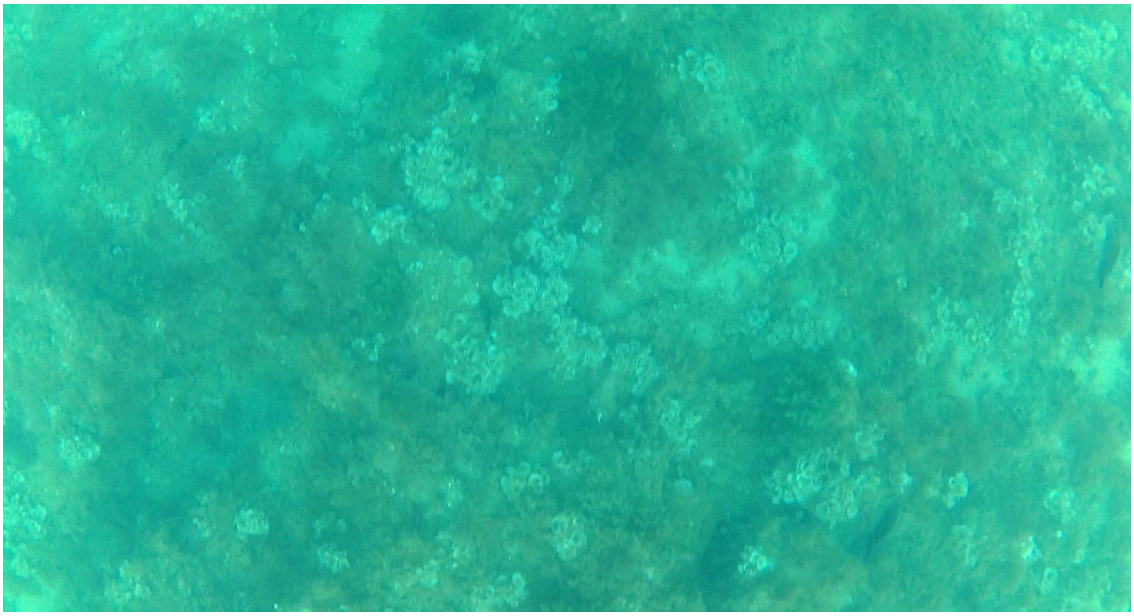


Figura 25: Frame original para la prueba 4

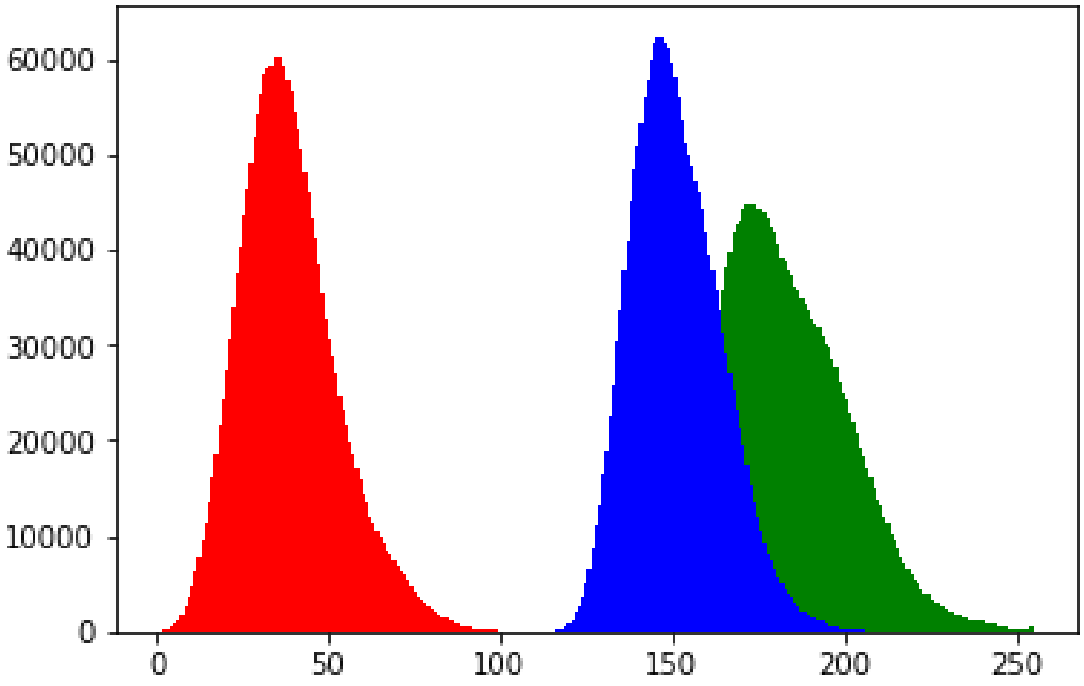


Figura 26: Histograma del frame original para la prueba 4

Imagen ecualizada (ecualizado adaptativo) con un límite de recorte **bajo** y su histograma:

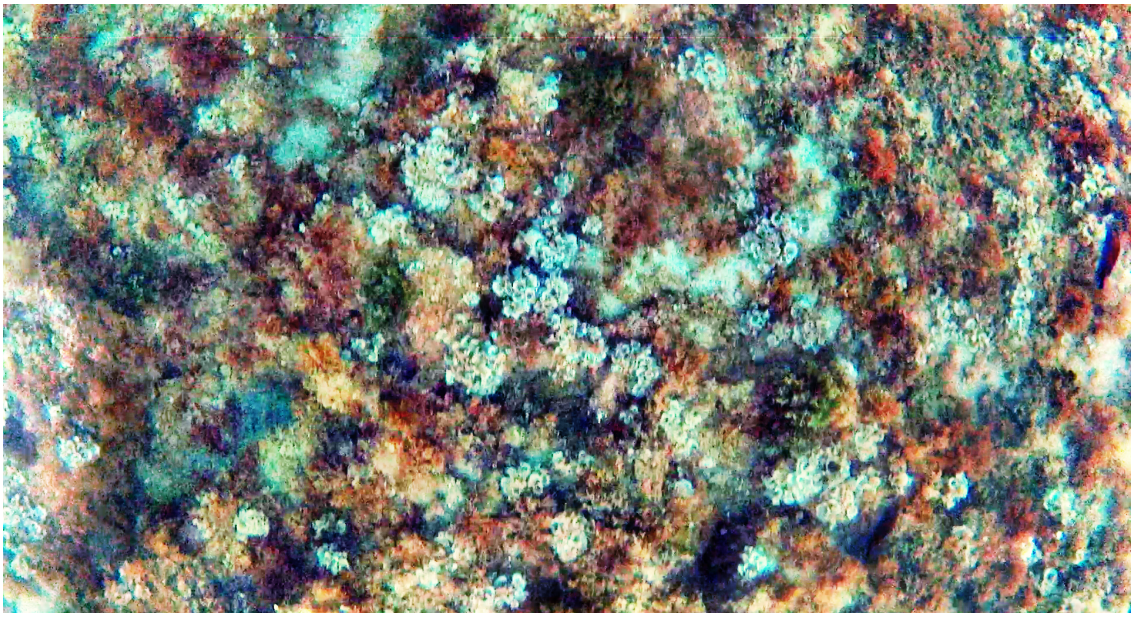


Figura 27: Imagen ecualización adaptativa con límite de recorte bajo

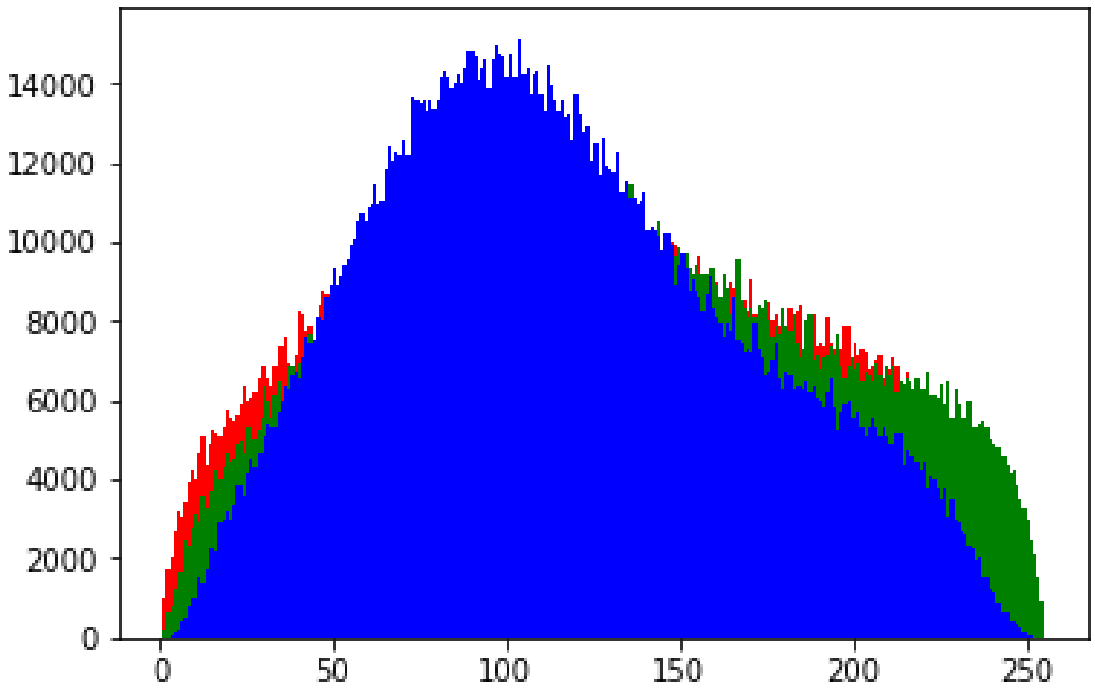


Figura 28: Histograma de la imagen ecualizada con límite de recorte bajo

Imagen ecualizada (ecualizado adaptativo) con un límite de recorte **alto** y su histograma:

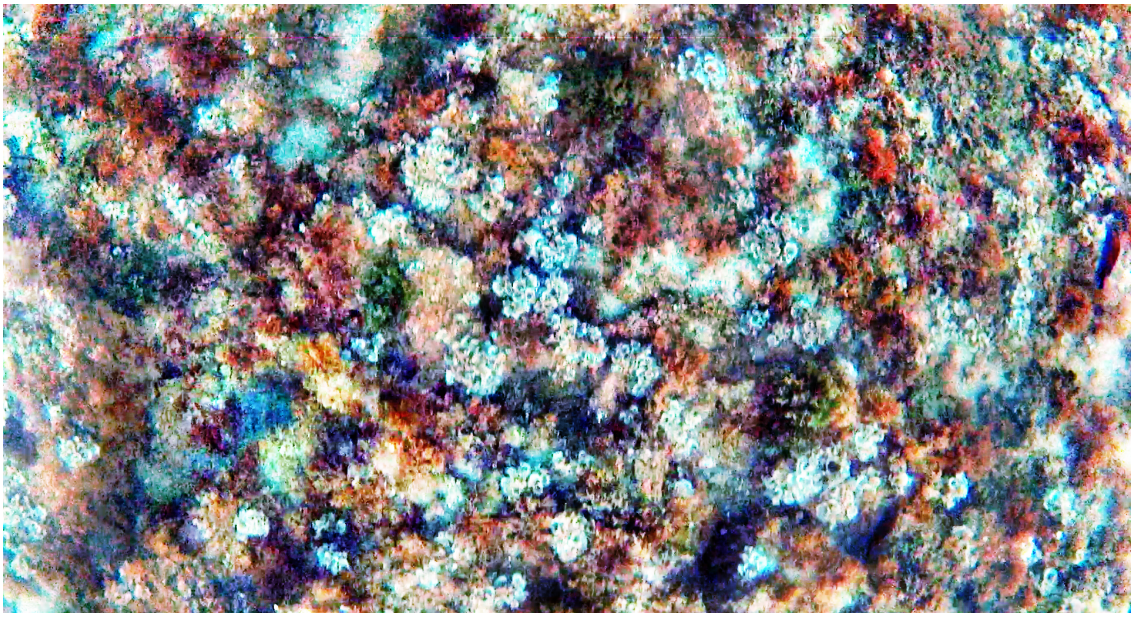


Figura 29: Imagen ecualización adaptativa con límite de recorte alto

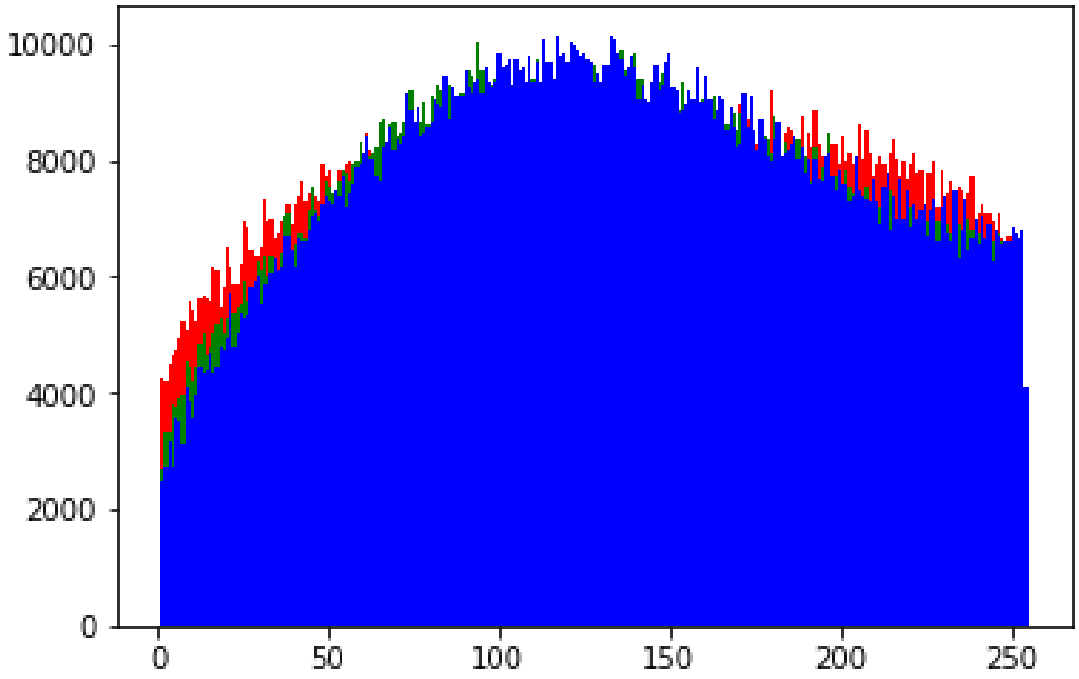


Figura 30: Histograma de la imagen ecualizada con límite de recorte alto

Al tener un límite de recorte más bajo controlamos que los valores de salida no se disparen, podemos apreciar que con un límite más alto el histograma sufre una alteración bastante más pronunciada en los valores de píxeles provocando que los valores se saturen y que exista un nivel de blanco irreal haciendo que parezca que la imagen adquiere colores más blanquinosos.

El tiempo de computación sigue en el mismo rango de segundos tanto con un límite de recorte algo como uno bajo.

5.1.5 Prueba 5 - Ecualización ordinaria modificada con filtro enfoque

Prueba usando un filtro de convulsión para obtener una ganancia en los bordes, creando un efecto de enfoque. He podido comprobar que no existe mejora a la hora de obtener una mejor interpretación del color, esto ocurre porque únicamente estamos haciendo cambios pequeños en los valores de la imagen. Por lo que esta prueba queda como anécdota y no proporciona nada a nuestro proyecto.

Esta idea surge por la carencia de calidad en las imágenes tomadas, pretende una pre mejora antes de la ecualización del histograma de forma ordinaria (con retoque en HSV).

Imagen original y su histograma:

Figura 31: Frame original para la prueba 5

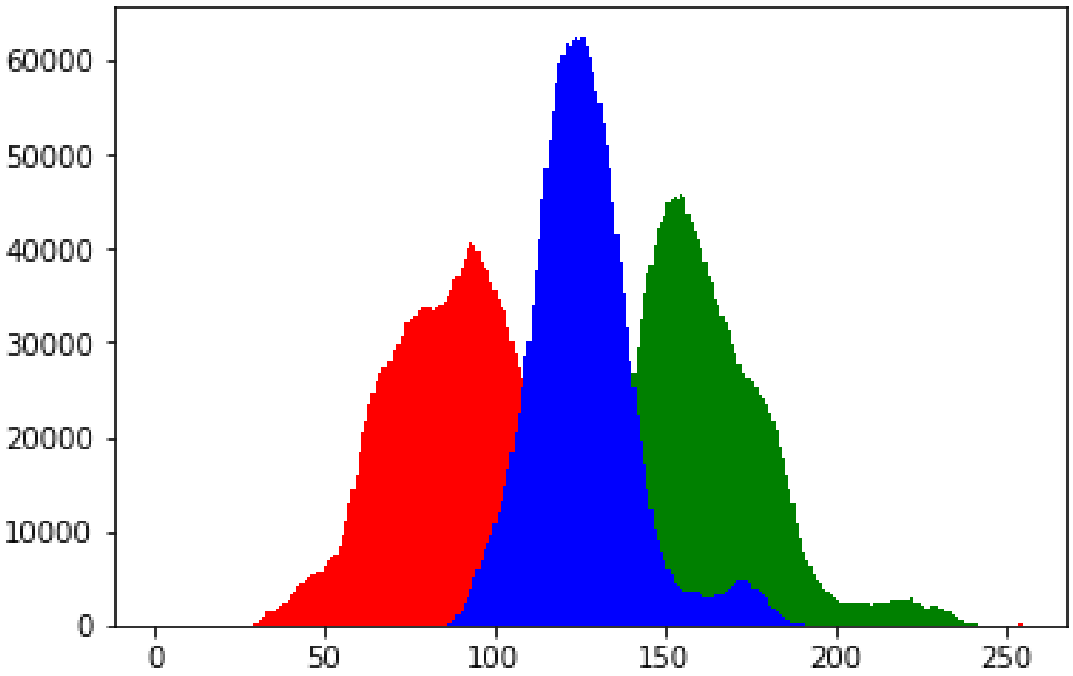


Figura 32: Histograma del frame original para la prueba 5

Imagen ecualizada (ecualizado ordinario) **sin** filtro y su histograma:

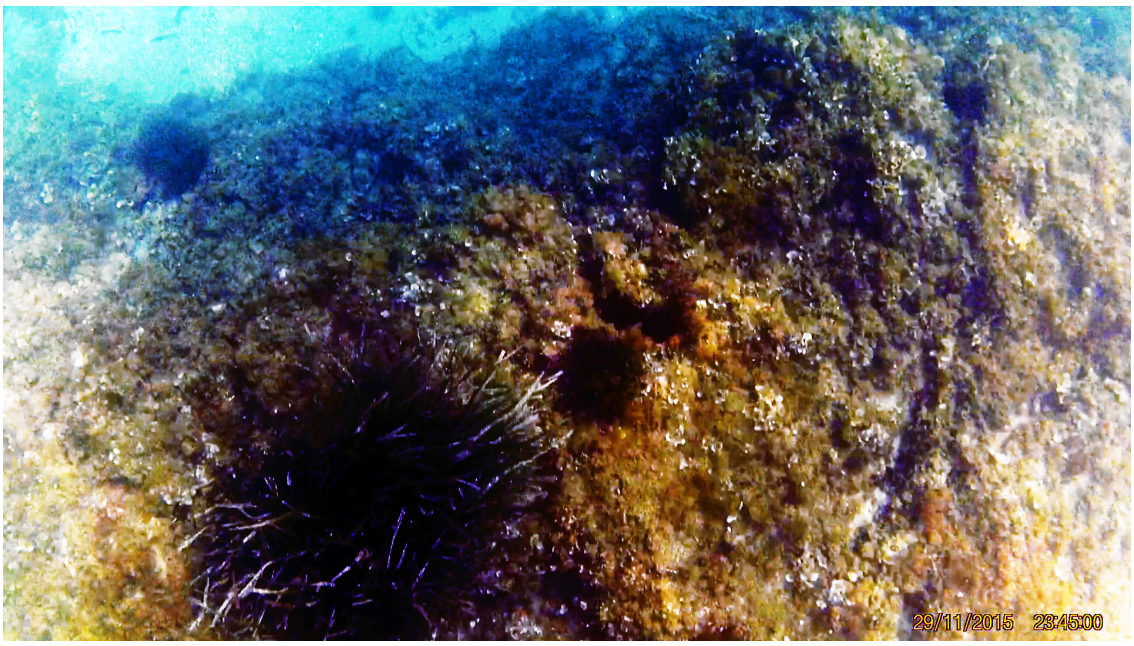


Figura 33: Imagen ecualización ordinaria sin filtro

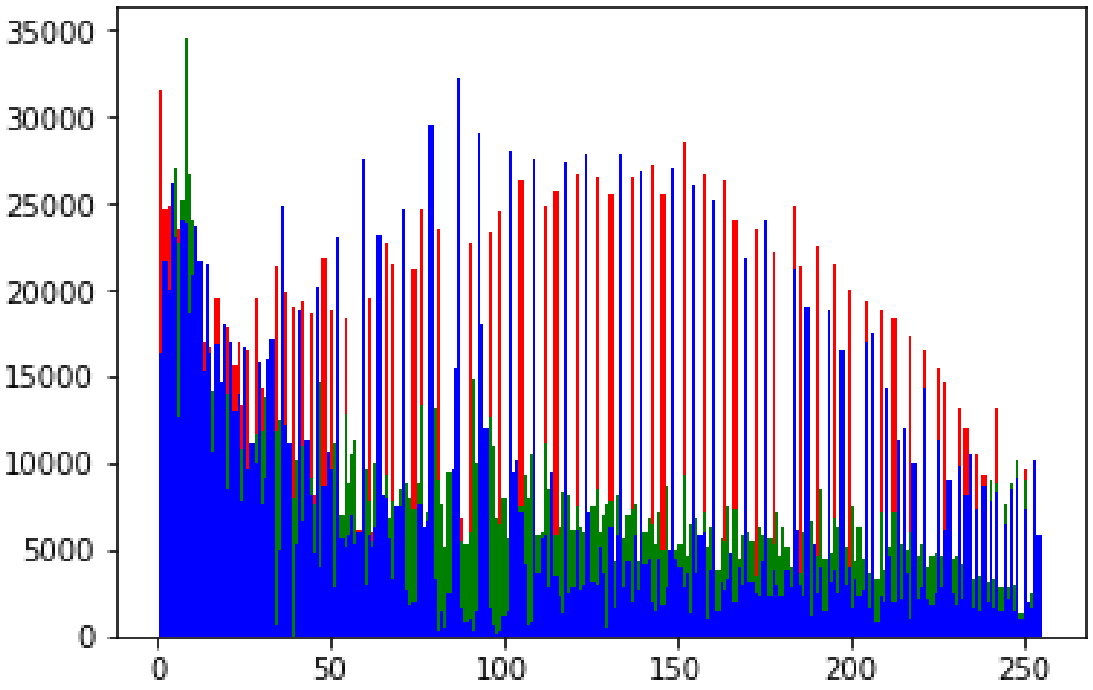


Figura 34: Histograma de la imagen con ecualización ordinaria sin filtro

Imagen ecualizada (ecualizado ordinario) **con** filtro y su histograma:

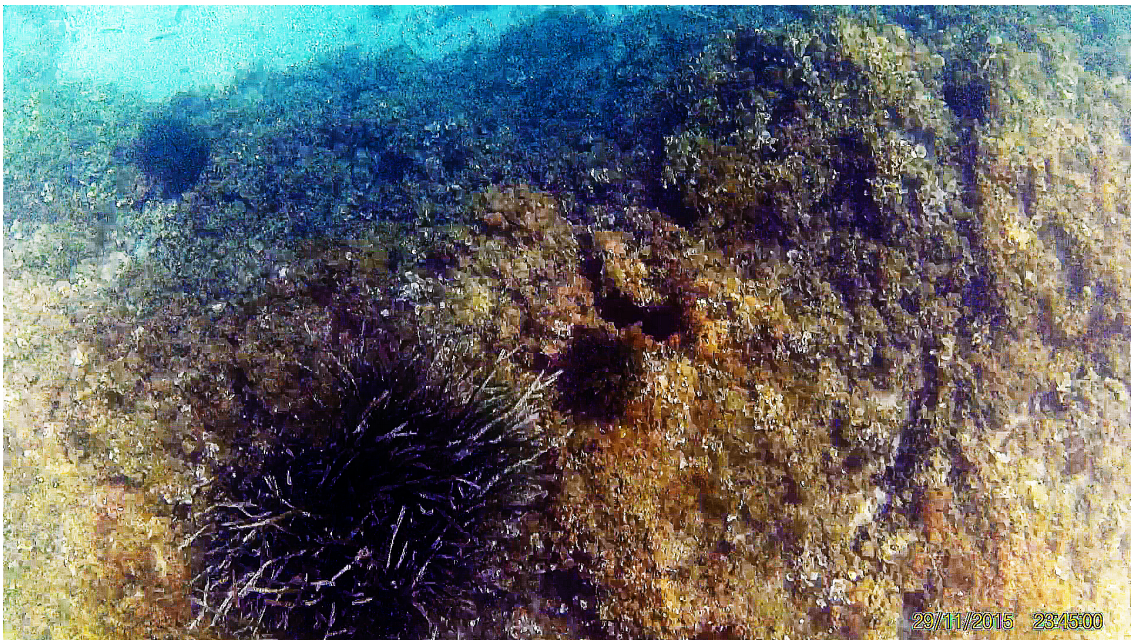


Figura 35: Imagen ecualización ordinaria con el filtro

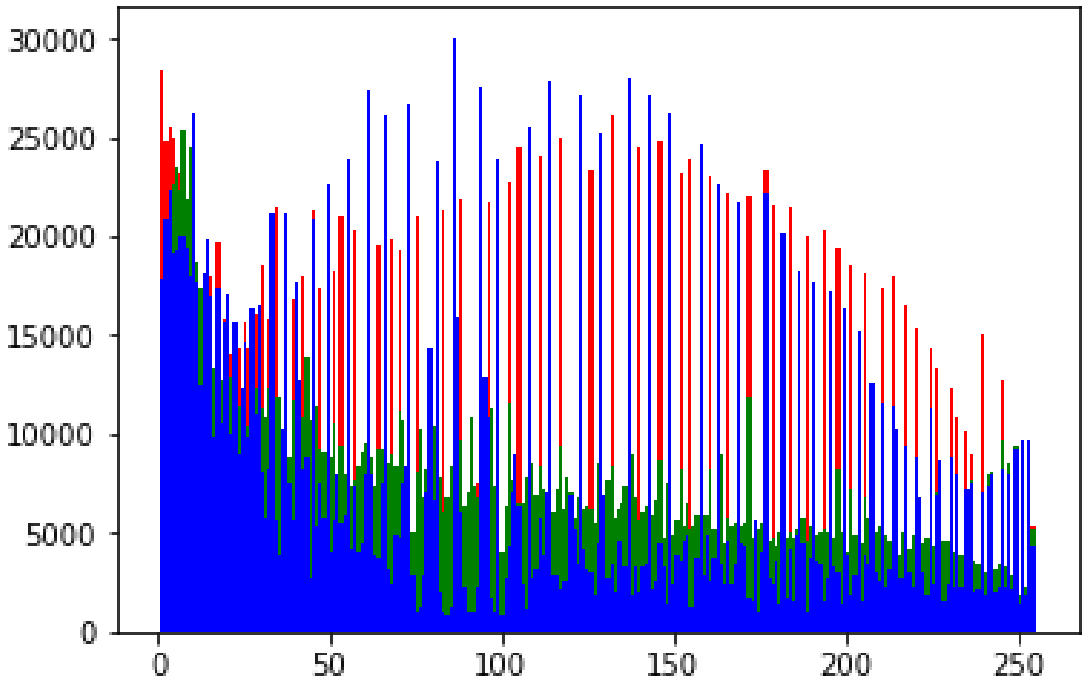


Figura 36: Histograma de la imagen con ecualización ordinaria con filtro

Se puede observar que no hay cambio apreciable y significativo en el histograma. Se confirma que no hemos ayudado a nada al objetivo de mejorar el contraste. En algunas pruebas el cambio ha sido algo más apreciable pero básicamente no deja de ser un movimiento de píxeles de un valor a otro. El tiempo de computación adicional al del ecualizado es despreciable.

5.2 Creación de panorámica

5.2.1 Prueba 6 - Realización panorámica con matcher de fuerza bruta

La prueba consiste es crear una panorámica con un cierto número de imágenes. Se busca que la panorámica revele si hay desviación en el rumbo de la nave, cuando creamos una panorámica es fácil de ver ya que veremos una imagen irregular. En concreto esta prueba se ha realizado con nueve imágenes secuenciales, supones que siempre van una después de la otra ya que la nave sigue un camino recto. La ejecución del código ha ido formando la panorámica del resultado de la anterior con la siguiente.

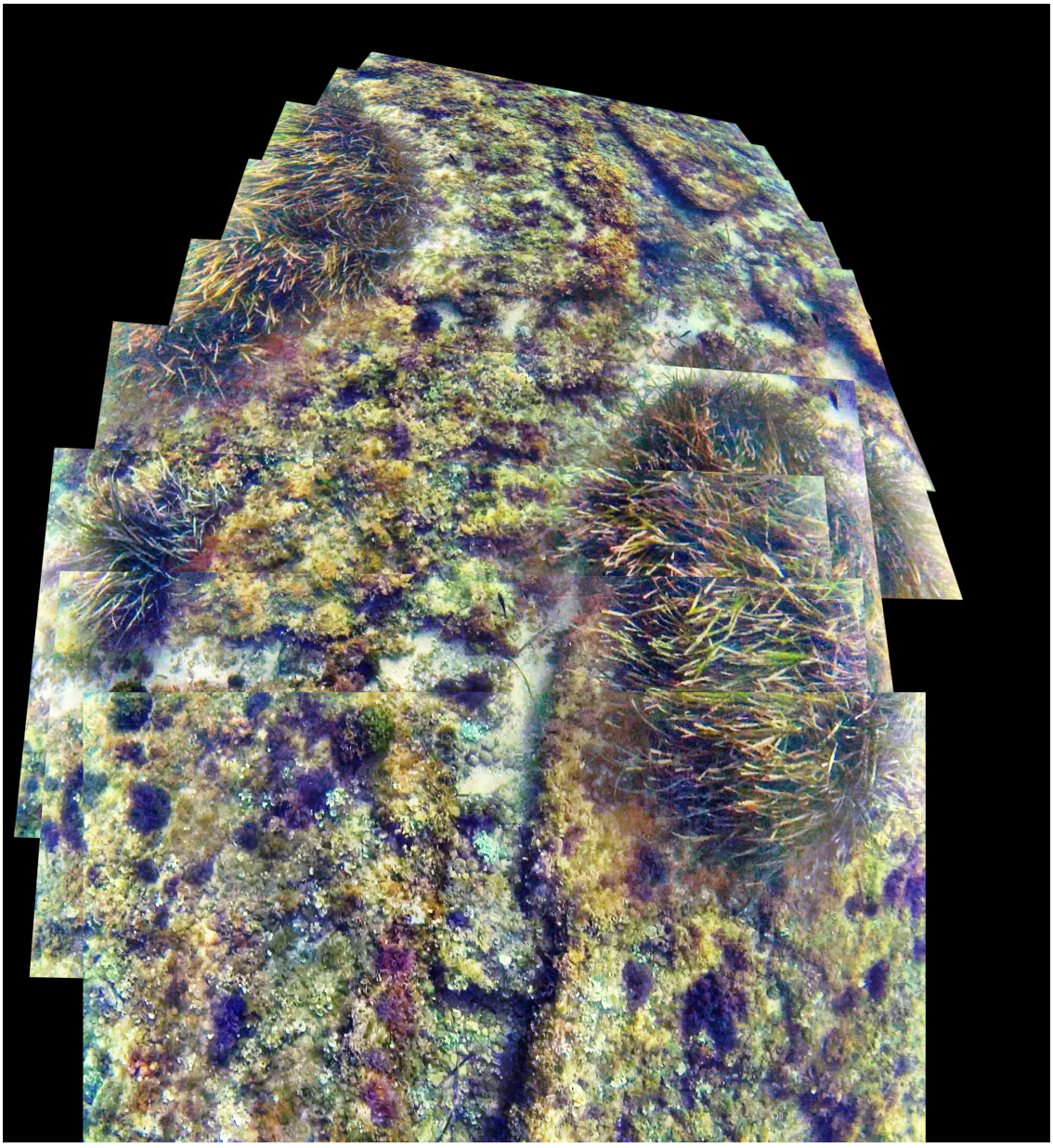


Figura 37: Imagen panorámica resultante

Podemos observar como la trayectoria no es recta, conforme vamos avanzando para hacer coincidir las características o puntos clave hay que proyectar y hacer traslaciones.

El tiempo de ejecución de esta panorámica es de algo más de 20 minutos, muy superior a ejemplos de imágenes más 'sencillas' de unir. Esto ocurre por la cantidad de puntos de interés que existen a la hora de hacer el match y la proyección que debe realizarse. Sería necesario una optimización del matcher para obtener una buena mejora en tiempo.

5.2.2 Prueba 7 - Realización panorámica con matcher FLANN

Esta prueba tiene el propósito de conseguir el mismo resultado que con un matcher de fuerza bruta (prueba 6) pero en un tiempo de computación mucho menor. El problema es que al revisar varias veces mi código y no funcionar, me decidí a buscar si era un causa de la propia librería OpenCV. En definitiva, que es un bug conocido, en su repositorio de github (dejo enlace en las referencias) se ha reportado y aún no hay un merge de la rama con la solución ni tampoco una release desde la corrección. Por lo que no he podido realizar la prueba, pero seguramente el resultado hubiera sido el esperado ya que cualquier algoritmo mínimamente optimizado es mejor que la fuerza bruta.

5.2.3 Prueba 8 - Realización panorámica por parejas

Para la realización hemos tenido que usar un match de fuerza bruta, ya que el matcher basado en FLANN no lo tenemos disponible. Se ha querido hacer esta prueba para como pre prueba a realizar una panorámica por diferentes hilos de ejecución y así descender el tiempo de computación requerido.

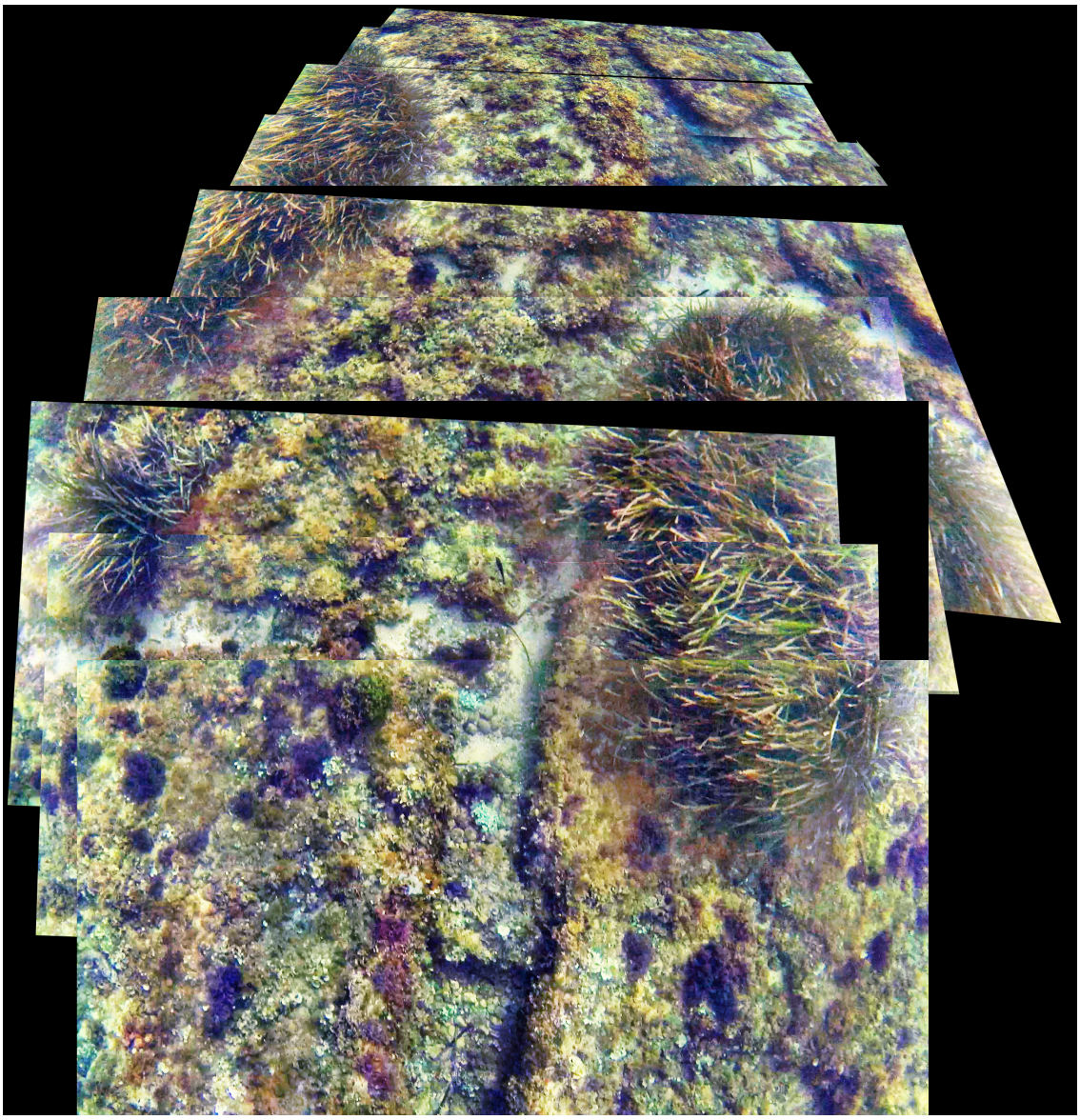


Figura 38: Imagen panorámica por parejas de imágenes

El resultado no ha sido el esperado. El principal problema que tenemos es la lente con la que se han tomado las fotos, gran angular, que es el típico en las cámaras deportivas. Esto hace que se obtenga más información del terreno, pero difícil de encuadrar a la hora de unir varias imágenes porque la óptica deforma la imagen. Con la prueba 6 el resultado se obtenía un buen resultado porque se hacia proyección sobre la unión.

5.2.4 Prueba 9 - Cálculo desviación

Cuando ya hemos demostrado de manera visual que existe una desviación, pensé en poder dar unos valores, aunque fueran una aproximación. Ya que para dar una distancia verídica deberíamos haber dispuesto de las herramientas ya en la expedición a la hora de tomar las imágenes.

De la panorámica resultante en la prueba 6 tenemos que la desviación total es de 0.32 metros.

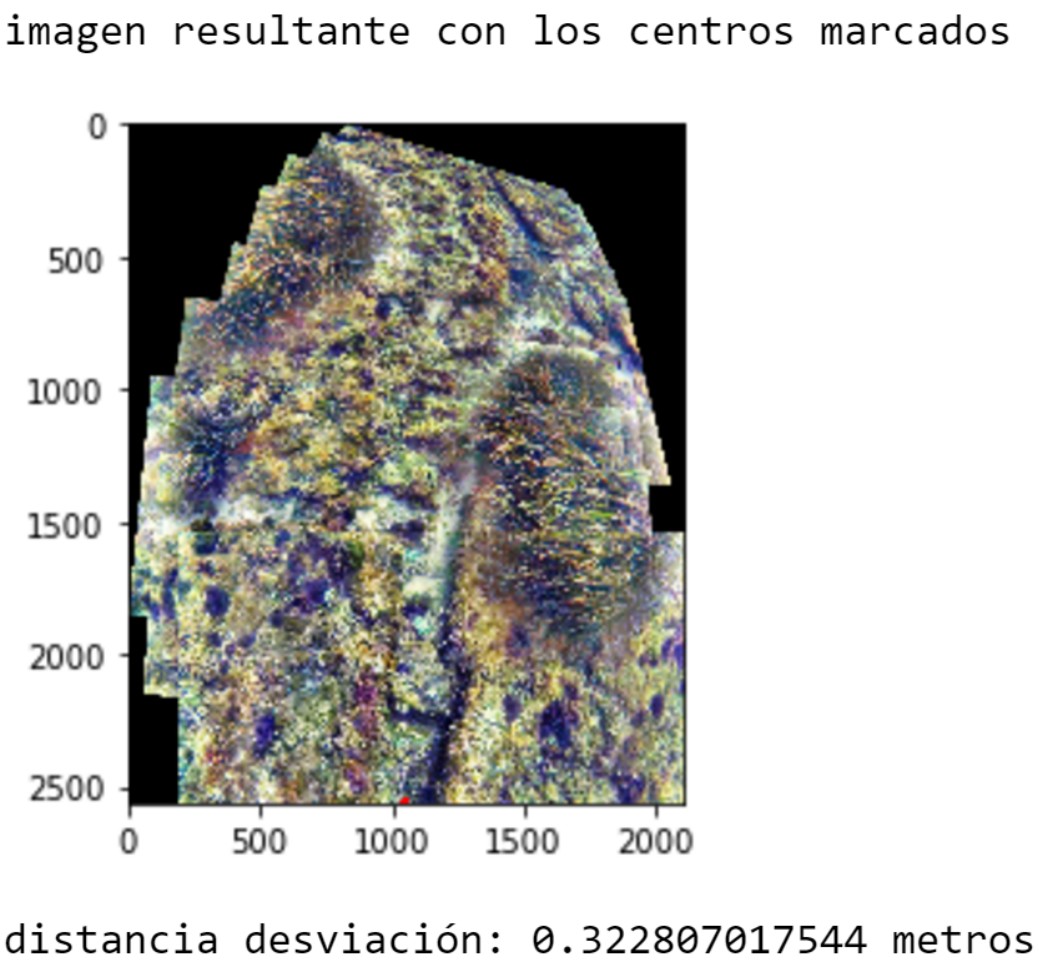


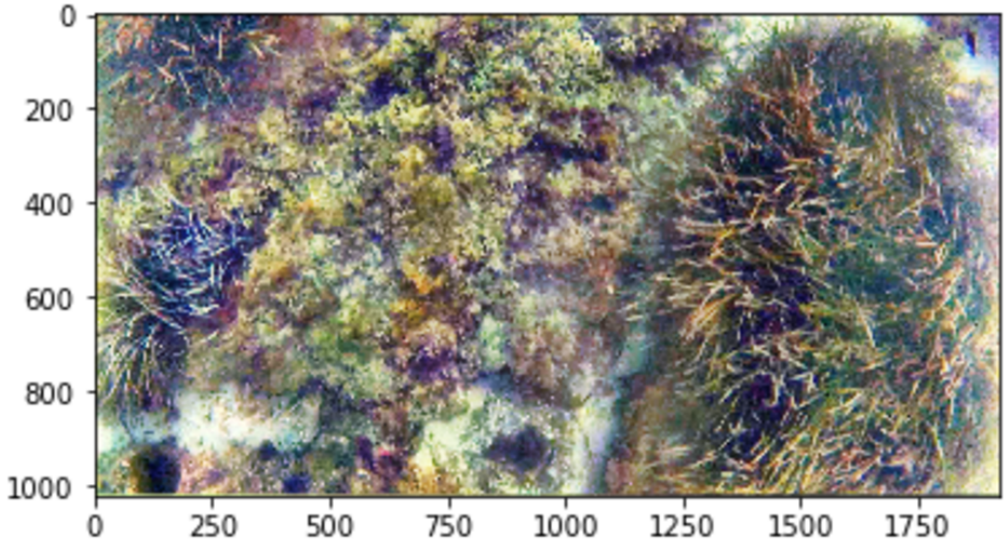
Figura 39: Imagen panorámica resultante más el cálculo de la desviación

Antes he dicho que el fabricante del sistema de navegación nos asegura una tasa de error alrededor del 3%, y tenemos unos 9 casi 10 metros recorridos con estas imágenes, por lo que se acerca bastante a nuestra tasa de error.

Debo dejar claro que es una forma aproximada y que sin una distancia de referencia en la foto es imposible calcular una distancia veridica.

5.2.5 Prueba 10 - Primera aproximación a la detección de vegetación

Para acabar he querido enganchar una última prueba con los trabajos futuros. Es el caso de la detección de vegetación. Me hubiera gustado poder hacer una funcionalidad de detección de vegetación útil y completa. Es algo que realmente da mucha vitalidad a los proyectos sobre expediciones, ya que suelen ser de estudios de vegetación o calidad del agua.




```
% de vegetación: 22.8911595797
```

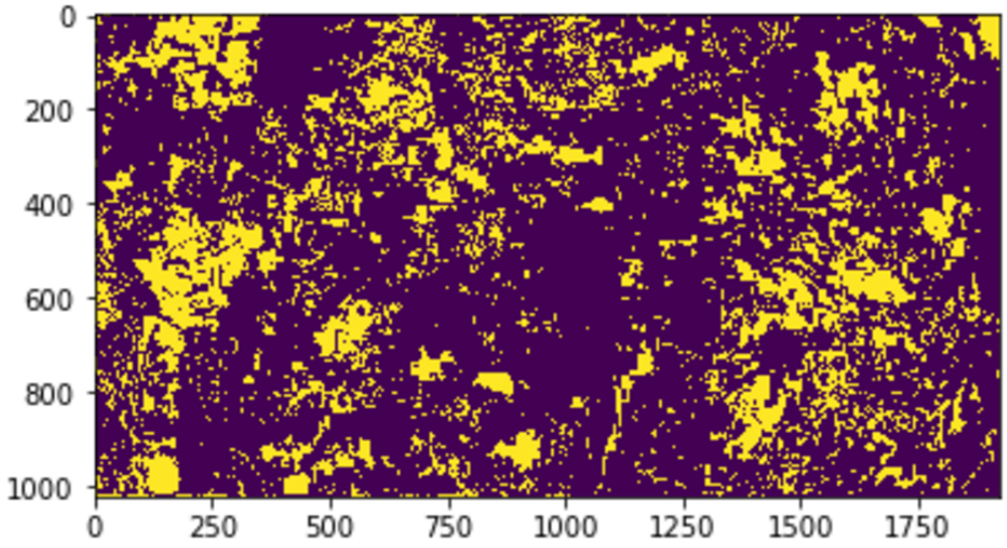


Figura 40: Resultado de la detección 'light' de vegetación.

Es otra forma aproximada y nada ideal de detectar vegetación, como ya he dicho en el desarrollo del método.

6 Conclusión y trabajos futuros

Valorando resultados creo que tenemos un claro ganador a la hora de tratar las imágenes, la ecualización adaptativa es la mejor opción aunque también la más costosa. Pese a esto, vale la pena ese tiempo adicional en la computación de la imagen. Los algoritmos CLAHE funcionan y nos proporcionan el contraste que tanto se pierde debajo del mar.

Pese a los buenos resultados obtenidos, tendría en cuenta previamente a tomar imágenes con cualquier cámara anclada a un vehículo o llevada por un submarinista, un estudio de la cámara y de las condiciones a las que nos vamos a enfrentar. Buscaría la lente adecuada para evitar deformaciones, como las que nos hemos encontrado con la cámara deportiva. Además haría una calibración de color, así sabríamos como altera la óptica el color y podríamos meteorizar la mejorar a equis niveles de equis canal. Por último, intentaría conseguir una manera de poder obtener en todo momento una distancia de referencia real, para poder calcular la desviación. Ya que es imposible que un vehículo submarino autónomo haga un trayecto perfecto, hay varias variables en contra, no disponer de sistema de navegación 100% fiable, corrientes, etcétera. Una mejora notable que nos quitaría ya del todo crear una mejora por software del color sería tener en el vehículo un buen sistema de iluminación montado.

Respecto a la deformación curva que genera la lente gran angular de la cámara deportiva usada, habría que hacer una corrección, para ello deberíamos hacer unas pruebas con objetos conocidos para tener unos patrones. Lo habitual es hacerlo con un tablero de ajedrez y calcular la distorsión angular, las lineas rectas parecen algo curvadas (se amplifica cuando nos alejamos del centro de la imagen), y la distorsión tangencial, algunas zonas de la imagen parecen más cerca o más lejos de lo esperado (ocurre cuando al hacer una imagen la lente no está alineada al plano de la imagen).

A la hora de buscar los puntos de interés en común de dos imágenes me hubiera gustado poder aplicar FLANN, ya que las imágenes con tantas características cuando se usa fuerza bruta los tiempos de computación se disparan. Cualquier proceso que optimice algún paso ya es más óptimo que la fuerza bruta.

Ha quedado claro que el mundo submarino da para mucho, siempre podríamos sacar alguna función que hacer con las imágenes tomadas en él.

Una posible expansión en un trabajo futuro, sin duda sería la detección de vegetación, aprovechando que usamos puntos de interés y características para hacer una panorámica podríamos por patrones y una buena batería de imágenes de training mirar de conseguir un cálculo de la vegetación que existe en un trayecto. Esto sería una funcionalidad muy interesante para númerosos cientificos que hacen una búsqueda de resultados sobre vegetación o calidad del agua.

Otro posible trabajo futuro es el de crear una aplicación con interfaz gráfica, con el fin de hacer la vida más fácil a personas que no están familiarizadas con entornos de programación. Se intentaría hacer una interfaz fácil, sencilla con la que marcar que funciones queremos de una imagen o de un vídeo origen.

Referències

[1] Peter Rowlands, British Society of Underwater Photographers.
Basic problems and solutions,
http://www.bsoup.org/Articles/Problems_Solutions.php, Reproduced from in focus 9 (Apr. 1985).

[2] Unidad de Tecnología marina (UTM)
Descripción de los AUV,
http://www.utm.csic.es/auv/index.php/inicio/recursos/equipos-autonomos/auv/vehiculos,

[3] Wikipedia
Histogram equalization,
https://en.wikipedia.org/wiki/Histogram_equalization.

[4] Wikipedia
Adaptive histogram equalization,
https://en.wikipedia.org/wiki/Adaptive_histogram_equalization#CLAHE.

[5] Nick
Automatic Color Correction: Underwater Photos,
https://loomsci.wordpress.com/2013/06/20/automatic-color-correction-underwater-photos/, 2013.

[6] Scikit-image
Exposure module,
http://scikit-image.org/docs/dev/api/skimage.exposure.html#skimage.exposure.equalize_adapthist.

[7] OpenCV tutorials
Changing Colorspaces,
http://opencv-python-tutroals.readthedocs.io/en/latest/py_tutorials/py_imgproc/py_colorspaces/py_colorspaces.html.

[8] Wikipedia
Kernel (image processing),
https://en.wikipedia.org/wiki/Kernel_(image_processing).

[9] Matthew Brown and David G. Lowe *Automatic Panoramic Image Stitching using Invariant Features*,
http://matthewalunbrown.com/papers/ijcv2007.pdf.

[10] OpenCV docs
Introduction to SIFT,
http://docs.opencv.org/trunk/da/df5/tutorial_py_sift_intro.html.

[11] Learn OpenCV
Homography Examples using OpenCV,
https://www.learnopencv.com/homography-examples-using-opencv-python-c/.

[12] OpenCV docs
Feature Detection and Description,
http://docs.opencv.org/3.0-beta/modules/features2d/doc/feature_detection_and_description.html.

[13] OpenCV docs
cv::BFMatcher Class Reference,
http://docs.opencv.org/3.1.0/d3/da1/classcv_1_1BFMatcher.html.

[14] OpenCV 3 python GitHub repository
FlannBasedMatcher::knnMatch fail with error,
https://github.com/opencv/opencv/issues/5667.

[15] Wikipedia
k-nearest neighbors,
https://es.wikipedia.org/wiki/K_vecinos_m%C3%A1s_pr%C3%B3ximos.

[16] Wikipedia
Scale-invariant feature transform (SIFT),
https://es.wikipedia.org/wiki/Scale-invariant_feature_transform.

[17] Adrian Rosebrock
Rotate images (correctly) with OpenCV and Python,
http://www.pyimagesearch.com/2017/01/02/rotate-images-correctly-with-opencv-and-python/ January 2, 2017.

[18] OpenCV Tutorials
Camera Calibration,
http://opencv-python-tutroals.readthedocs.io/en/latest/py_tutorials/py_calib3d/py_calibration/py_calibration.html.