Trabajo Fin de Grado
Ingeniería Aeroespacial

Sistema de detección y localización de nubes mediante estéreo visión

Autor: Raquel González Morata

Tutor: Eduardo Fernández Camacho

Dpto. Teoría de Ingeniería de Sistemas y Automática
Escuela Técnica Superior de Ingeniería

Sevilla, 2013

Proyecto Fin de Grado
Ingeniería Aeroespacial

Sistema de detección y localización de nubes mediante estéreo visión

Autor:
Raquel González Morata

Tutor:
Eduardo Fernández Camacho
Profesor titular

Dpto. de Ingeniería de Sistemas y Automatica
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla
Sevilla, 2022

Proyecto Fin de Grado: Sistema de detección y localización de nubes mediante estéreo visión

Autor: Raquel González Morata

Tutor: Eduardo Fernández Camacho

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

Presidente:

Vocales:

Secretario:

Acuerdan otorgarle la calificación de:

Sevilla, 2022

El Secretario del Tribunal

Agradecimientos

La elaboración de este proyecto conlleva el final de una de las etapas más importantes de mi vida. Comencé el grado de Ingeniería Aeroespacial siendo una persona, y lo acabaré siendo otra con muchos más conocimientos, aprendizajes y vivencias. En este camino, he tenido la oportunidad de conocer personas de las que he aprendido mucho, tanto profesores como compañeros, y me siento enormemente agradecida por ello.

Quiero agradecer a mis más cercanos compañeros el apoyo recibido todos estos años y las experiencias compartidas.

Un agradecimiento especial a mi familia sin la que, sin duda, nada de esto habría sido posible. Han compartido mis alegrías y pesares, empujándome cada día a superarme un poco más. Por todo el esfuerzo que han hecho y por lo que estuvieron y ya no están.

Por último, agradecer a todo el profesorado que me ha hecho llegar hasta aquí y, en especial, a Eduardo por permitirme realizar este proyecto sin las nociones previas requeridas ampliando así mis conocimientos una vez más.

Raquel González Morata

Sevilla, 2022

Resumen

La actual situación del mundo frente al cambio climático impulsa a la sociedad a mostrar un mayor interés por las energías renovables, como es la energía solar entre otras. Así mismo, este interés va acompañado de la búsqueda de una mayor eficiencia en el proceso de extracción de la energía, siendo éste el origen del trabajo aquí presente.

Partiendo de la idea de la propiedad de una planta de placas solares, este proyecto tiene como objetivo desarrollar un sistema capaz de detectar y localizar las nubes, permitiendo que futuros proyectos puedan aprovechar la información resultante para el control del funcionamiento de dichas placas que pueden ver alterado su rendimiento en función del sombreado producido por las nubes.

Para lograr el propósito mencionado se hace uso del procesamiento de imágenes y la técnica de estéreo visión.

La detección de objetos se puede realizar mediante varios métodos de procesamiento de imagen que se expondrán a lo largo de este trabajo. Sin embargo, las características del objeto de estudio en este caso, como son la opacidad y los contornos de las nubes en constante cambio, limitan el campo de trabajo. De esta manera, se ha optado por un sistema de detección basado en el algoritmo YOLO ("*You Only Look Once*") y en el conjunto de métodos del valor umbral, también conocido como Thresholding.

Por otro lado, la localización de la nube, una vez detectada, se lleva a cabo mediante la utilización de dos cámaras fijas orientadas al cielo capturando imágenes en el mismo momento. La técnica de estéreo visión será la encargada de extraer la información requerida del objeto en 3D a partir de las imágenes obtenidas.

Todo ello se desarrolla en Python y, principalmente, con el uso de la biblioteca libre de visión artificial OpenCV.

Abstract

The current world situation in the presence of climate change is driving society to show greater interest in renewable energies, such as solar energy, among others. Likewise, this interest is accompanied by the search for greater efficiency in the energy extraction process, which is the origin of the work presented here.

Starting from the idea of the ownership of a solar panel plant, this project aims to develop a system capable of detecting and locating clouds, allowing future projects to leverage the resulting information to control the operation of these panels that can see their performance altered depending on the shading produced by the clouds.

To achieve the aforementioned purpose, image processing and stereo vision technique are used.

Object detection can be performed using different image processing methods that will be presented throughout this paper. However, the characteristics of the object of study in this case, such as opacity and constantly changing cloud contours, limit the field of work. Therefore, a detection system based on the YOLO ("You Only Look Once") algorithm and the set of thresholding methods, has been chosen.

On the other hand, the localization of the cloud, once detected, is carried out using two fixed cameras oriented to the sky and capturing images at the same time. The stereo vision technique will be responsible for extracting the required 3D information of the object from the images obtained.

All this is developed in Python and, mainly, with the use of the free artificial vision library OpenCV.

Índice

ÍNDICE DE FIGURAS

1 INTRODUCCIÓN

La realización de este trabajo tiene como objetivo desarrollar un sistema que, junto a otros proyectos vinculantes, sirva de apoyo a la extracción eficiente de energía de placas solares. En concreto, el sistema creado realiza su aportación detectando y localizando las nubes presentes en el cielo.

Para llevar a cabo el propósito de este trabajo se emplean técnicas de procesamiento de imagen y estéreo visión, siendo una de las principales herramientas a utilizar la biblioteca libre de visión artificial OpenCV. Ésta, a pesar de ser programada en C++, presenta conectores para otros lenguajes como Python, para el que, además, ofrece documentación explicativa. Por lo tanto, el mencionado lenguaje Python será el empleado para la programación necesaria.

Puesto que se parte desde el desconocimiento tanto del procesamiento de imagen como del lenguaje Python, es necesaria la elaboración consciente de un estudio previo y posterior adquisición de conocimientos sobre las distintas posibilidades que existen para detectar objetos mediante procesamiento de imagen, así como el cálculo de la localización mediante estéreo visión, todo ello implementado en Python. Con ello se pretende llegar a un resultado satisfactorio.

En cuanto a la detección de objetos, los métodos fundamentales son Template Matching, Haar Cascade, y YOLO. Estos se exponen con detalle en el apartado 2, aunque finalmente se opta por este último en base a los argumentos presentes en dicho apartado. Además, a la hora de identificar los contornos de los objetos detectados se utilizarán métodos Thresholding como base principal, pudiendo aparecer pequeñas variaciones.

En el caso de la localización del objeto mediante estéreo visión, se emplean dos cámaras fijas con igual orientación hacia el cielo. Las imágenes capturadas por ellas en un mismo instante de tiempo proporcionarán, con los cálculos adecuados, información 3D del objeto. Un mayor desarrollo de este concepto se muestra en el apartado 3.

Para finalizar, en el apartado 3 y 4 se mostrarán las conclusiones y resultados del trabajo, y posibles mejoras y/o ampliaciones para futuros proyectos respectivamente.

2 DETECCIÓN DEL OBJETO

Como se ha mencionado anteriormente, lo primero a realizar en este trabajo es la detección del objeto de estudio a partir de una imagen digital. Antes de la identificación de contornos del objeto en cuestión, es necesario que el sistema creado sea capaz de distinguir una nube del resto de cuerpos que pueden presentarse en una instantánea del cielo como, por ejemplo, aviones o pájaros. Puesto que existen distintas maneras de dar solución a este problema, a continuación, se va a mostrar el estudio detallado realizado que resulta en la elección final. Así mismo, el estudio previo incluye los factores involucrados en la posterior identificación de contornos.

2.1. Template Matching

Esta técnica es una de las más sencillas para identificar un objeto en una imagen. Su funcionamiento consiste resumidamente en buscar y localizar una imagen con función de plantilla en otra imagen de mayor tamaño. Para ello, se requiere aportar la imagen plantilla cuyos píxeles serán comparados con los pertenecientes a la imagen mayor en busca de una correspondencia.

Las funciones principales a utilizar son **cv.matchTemplate(*image, template, method*)** y **cv.minMaxLoc()**. [1]

La primera de ellas es la responsable de la comparación de forma que, si la imagen plantilla (*template*) tiene un tamaño (wxh), esta imagen se irá comparando con todas y cada una de las distintas cuadrículas del mismo tamaño (wxh) que puedan ser generadas en la imagen (*image*) de mayor tamaño (W x H). El resultado obtenido es una imagen en escala de grises de tamaño (W–w+1, H–h+1) donde el valor de cada píxel indica su similitud con la imagen plantilla. La mayor similitud puede estar asociada al mínimo o máximo valor del píxel en función del método empleado [2]:

- **cv.TM_CCORR**: Template Matching Cross correlation

  Este método se basa en la multiplicación de los valores de cada par de píxeles implicados en la comparación correspondiente entre ambas imágenes para luego sumar todos los productos obtenidos. De esta forma, la mayor correspondencia coincide con el máximo valor resultante de todas las sumas generadas tras realizar todas las comparaciones posibles.

- **cv.TM_CCOEFF**: Template Matching Correlation Coefficient

  Al igual que el método anterior, este también está basado en la multiplicación. Sin embargo, implica que los píxeles más ocuros tomen valores negativos y los más claros, valores positivos. Al superponer ambas imágenes, si la multiplicación de los píxeles resulta positiva conlleva correspondencia. Por el contrario, si resulta negativa implica disparidad.

- **cv.TM_SQDIFF**: Template Matching Sum of Square Difference

  En esta ocasión los valores de cada par de píxeles son restados para luego realizar la suma del cuadrado de todas las restas. Es simple observar que, al emplear este método, la mayor correspondencia se asocia al mínimo valor de las sumas resultantes.

Por consiguiente, habrá que fijarse en el mínimo valor al usar **cv.TM_SQDIFF**, y en el máximo valor al usar **cv.TM_CCOEFF** y **cv.TM_CCORR**.

Por otro lado, la función **cv.minMaxLoc()** recibe como entrada la imagen en escala de grises proveniente de **cv.matchTemplate** y devuelve como resultado los valores mínimo y máximo, así como sus posiciones.

2.2. Haar Cascade

Este método de detección de objetos está basado en el machine learning en el que un clasificador es entrenado mediante imágenes positivas y negativas, entendiendo por imágenes positivas aquellas en las que aparece el objeto a identificar y por imágenes negativas, aquellas donde no aparece el objeto en cuestión.

La técnica de la cascada a la que hace referencia su nombre consiste en concatenar clasificadores débiles que analizando de forma independiente pequeñas partes de la imagen no tienen éxito, pero combinándolos ofrecen un buen resultado.

Para ello, se emplean *features* (Figura 1) que hacen referencia al bloque de píxeles encontrados en una ventana de detección cerrada. Cada *feature* toma el valor generado por la resta de la suma de los pixeles que se encuentran en el rectángulo blanco menos la suma de los píxeles situados en el rectángulo negro. Utilizando estos *features* se puede comparar el resultado obtenido en las imágenes positivas con el obtenido en las negativas para tomar una decisión. Como se ha mencionado previamente, esto se produce de forma concatenada, es decir, aplicando un grupo de *features* por cada etapa. [3]

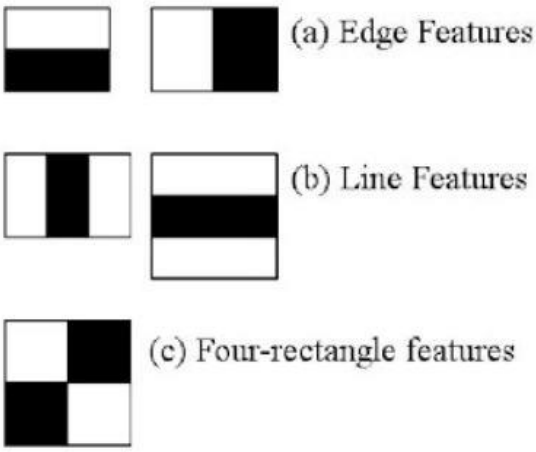

Figura 1. Haar Cascade *features* [4]

Además del OpenCV, para este método se necesita una herramienta como Cascade Trainer GUI [5]. Esta herramienta es un programa utilizado para entrenar, testear y mejorar modelos de clasificación en cascada que, a partir de un data set de imágenes positivas y negativas, crea un archivo .xml conteniendo el clasificador buscado y empleado posteriormente en Python.

2.3. YOLO ("*You Only Look Once*")

Este algoritmo, como su nombre indica, es capaz de detectar el objeto deseado con tan sólo una lectura de la imagen, siendo probablemente el más rápido con respecto a los demás. Consta de una única red neuronal convolucional encargada de dividir la imagen en regiones, para las cuales predice cuadros de identificación o *bounding boxes* y probabilidades. Las cajas son ponderadas a partir de las probabilidades predichas y se procede a eliminar aquellas que no lleguen a un valor límite. Finalmente, se analiza la posible detección doble del objeto y se descarta aquella que tiene menor precisión. [6] [7]

Las implementaciones existentes de YOLO tienen la ventaja de que tras su descarga e instalación es fácil detectar objetos con el uso de los pesos ponderados proporcionados. Sin embargo, estos pesos corresponden al

entrenamiento de sets de datos específicos que pueden o no contener el objeto que se desea detectar. Por ello, habría que realizar un entrenamiento customizado partiendo de un set de datos como ocurría con el método Haar Cascade.

2.4. Identificación de contornos

Los métodos anteriores nos ofrecen la posibilidad de categorizar los objetos capturados por una cámara. Sin embargo, no es suficiente para su localización, es primordial el reconocimiento de sus contornos.

Los contornos en una imagen, en el supuesto ideal, se caracterizan por el cambio brusco de intensidad del píxel. Pero, en la realidad, esto no ocurre así ya que depende del muestreo de la digitalización y del ruido digital aleatorio que pueda presentarse. La aparición del ruido digital se debe a factores como el tipo de lente de la cámara, la intensidad de la luz o la temperatura, y perjudica a la imagen produciendo variaciones de manera aleatoria en la intensidad de sus píxeles.

Primeramente, como se ha justificado, se debe aplicar una previa eliminación del ruido en la imagen para simplificar la identificación de los contornos. OpenCV ofrece una gran variedad de procedimientos para llevarlo a cabo mediante filtros de paso-bajo [8]. El inconveniente de los filtros paso-bajo es que eliminan los bordes además del ruido. Sin embargo, como veremos a continuacion, existen métodos para la eliminación del ruido sin repercusión en los bordes. En el procesamiento de imagen, en el que aparecen involucradas las convoluciones, se utiliza la matriz kernel. Dicha matriz está asignada a un píxel núcleo y define el tamaño de la convolución y los pesos aplicados a ella.

- **Averaging.** Este procedimiento simplemente toma la media de los píxeles vecinos al núcleo que se encuentran bajo el área de la matriz kernel y sustituye el valor de dicho núcleo por el resultado obtenido. Se consigue mediante la función **cv.blur()**, cuyas entradas son la imagen y el tamaño de la matriz kernel.

- **Gaussian Blurring.** Este método es similar, siendo utilizada en esta ocasión la matriz kernel gaussiana. Con él se consigue un suavizado más uniforme. Para ello, se necesita la función **cv.GaussianBlur()**, cuyas entradas son la imagen, el tamaño kernel y las desviaciones estándar en ambas direcciones.

- **Median Blurring.** Como su nombre indica, este procedimiento realiza la misma función que el Averaging pero con la obtención de la mediana. Es bastante útil frente al ruido sal y pimienta, es decir, aquel que cubre la imagen de forma dispersa con píxeles blancos y negros. La funcion a emplear es **cv.medianBlur()**, siendo las entradas las mismas que las de **cv.blur()**.

- **Bilateral Filtering.** Esta técnica tiene como ventaja la eliminacion eficaz del ruido manteniendo la nitidez de los bordes. Mientras que los otros métodos tan solo tienen en cuenta los píxeles vecinos sin considerar su intensidad y si pertenecen o no a un borde, este hace uso del filtro gaussiano en cuanto al espacio, pero también aplica un filtro gaussiano función de la diferencia de intensidad de los píxeles. De esta forma, solo se suavizan aquellos píxeles con intensidades próximas a la del núcleo. Todo esto se consigue con el uso de la función **cv.bilateralFilter()**, aportando la imagen, el díametro del vecindario, sigmaColor y sigmaSpace. Un valor alto de sigmaColor implica que los colores más lejanos de la vecindad se mezclarán. Por otro lado, un valor alto de sigmaSpace se traduce en una influencia de los píxeles más alejados entre ellos siempre y cuando sus colores estén lo suficientemente cercanos.

Habiendo resuelto el problema del ruido, OpenCv proporciona dos funciones con las que encontrar contornos y dibujarlos: **cv.findContours()** y **cv.drawContours()**. La primera de ellas necesita la imagen a estudiar, el

modo de recuperación de contorno y el método de aproximación. El modo de recuperación elegido simplemente indica los contornos a devolver, pudiendo devolver solo los contornos externos, todos los contornos sin relación de jerarquia, todos los contornos organizados por jerarquía de dos niveles, o todos los contornos con una jerarquía completa de contornos anidados. En cuanto al método de aproximación, en función del escogido, principalmente se pueden almacenar todos los puntos encontrados pertenecientes a una forma o solo aquellos puntos que no sean redundantes.

No obstante, se recomienda emplear imágenes binarias para una mejor precisión.Así pues, se debe aplicar el método Thresholding o el método Canny Edge Detection.

- **Thresholding.** Es una técnica de binarización donde los píxeles toman un valor u otro en función de un valor umbral establecido. Encontramos el thresholding simple con la función **cv.threshold()**, donde el valor umbral es introducido como entrada manualmente por el usuario. Por otra parte, está el thresholding adaptativo con la función **cv.adaptativeThreshold()**. Este último determina el valor umbral para un píxel en base a una pequeña región alrededor de él, por lo que está cambiando constantemente en busca del más adecuado. En función del método indicado como entrada puede calcular el valor umbral como la media de los píxeles vecinos menos una constante C (también entrada) o como la suma ponderada gaussiana de los valores vecinos menos la constante C. Además, para ambas funciones, se puede indicar la forma de binarización, eligiendo asi los valores que toman los píxeles tras ser evaluados. [9]

- **Canny Edge Detection.** Este algoritmo incluye tres procesos. El primero de ellos es encontrar el gradiente de intensidad de la imagen en las direcciones horizontal y vertical. Después, se eliminan los píxeles que pueden no formar parte del contorno mediante la técnica de supresión non-maximum, donde cada píxel es estudiado para ver si es un máximo local en su vecindario en la dirección del gradiente, resultando en una imagen binaria con bordes adelgazados. Por último, se aplica un umbral por histéresis, para lo que se aporta a la función **cv.Canny()** un valor mínimo y un valor máximo de Threshold. Así, los píxeles con valores de gradiente de intensidad mayores al máximo son considerados del borde, los menores al mínimo son descartados, y los intermedios son juzgados en base a su conectividad. [10]

Durante todo este procesamiento se trabaja con imágenes en escala de grises, ya que simplifican los mecanismos hacia la binarización y, además, aportan la información suficiente.

Con toda la informacion recabada, podríamos ya realizar la identificación de los contornos. Sin embargo, algunos objetos de estudio, como las nubes, tienen complejidades, pudiendo fallar el método seleccionado en función de las características de la imagen. Existen una serie de métodos llamados **fixed thresholding**, para los que el valor umbral impuesto ha sido estudiado detalladamente.

Un valor interesante de estudiar es el ratio normalizado de B/R (canal azul/canal rojo) de la imagen, que es capaz de mantener el contraste incluso con ruido de por medio. Distinguimos aquí entre imagen unimodal e imagen bimodal: las imágenes unimodales están constituidas por el mismo elemento (nube o cielo), mientras que las bimodales tienen ambos elementos. Una investigación de la Universidad Jiaotong de Beijing [11] demuestra que si transformamos las imágenes en color a imágenes ratio B/R y estudiamos los histogramas orrespondientes, obtenemos un histograma de poca variación para imágenes unimodales e histogramas de mayor variación para imágenes bimodales. Este aspecto es fundamental para el reconocimiento de contornos. Las imágenes unimodales, más complejas para aplicar una simple técnica thresholding, se pueden tratar siguiendo este factor. En la ecuación 2.2 se muestra el ratio normalizado y en la Figura 2 la causa por la que es empleado el normalizado en lugar del normal. Para algunas imágenes donde el ruido está bastante presente, la

transformación a imagen de ratio B/R es indiferente. Sim embargo, la transformación a imagen de ratio normalizado muestra el contraste buscado.

$$ratio = \frac{B}{R} \tag{2.1}$$

$$ratio\ normalizado = \frac{ratio - 1}{ratio + 1} = \frac{B - R}{B + R} \tag{2.2}$$

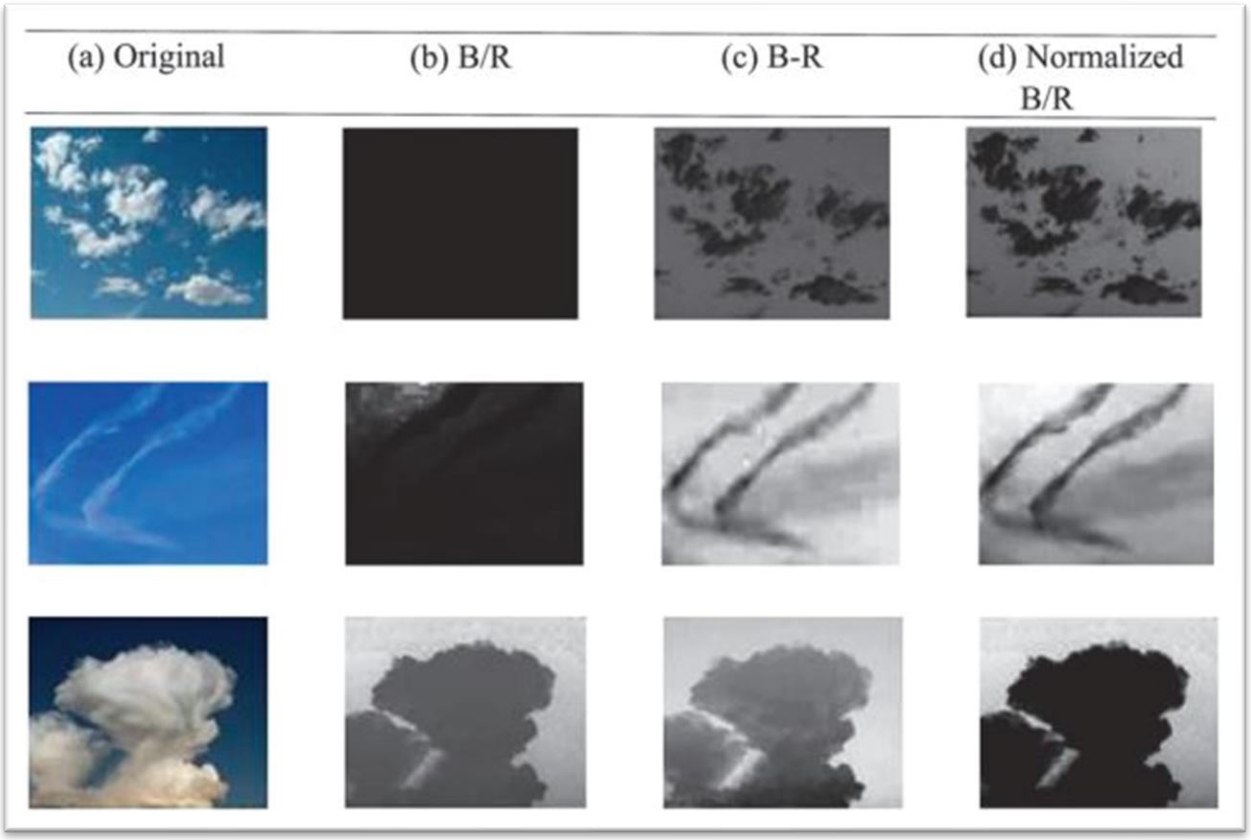

Figura 2. Representación de las imágenes originales y las asociadas a sus valores de B/R, B-R y B/R normalizado [11]

3 LOCALIZACIÓN DEL OBJETO

Actualmente, son muchas las situaciones que requieren localizar los objetos de la manera más precisa posible, siendo un claro ejemplo los radares de carretera. La calidad de la imagen tomada por la cámara influye no sólo en el procesamiento digital que conlleva la detección de objetos, sino también en la técnica de estéreo visión como veremos más adelante en los resultados. Ahora, antes de realizar la programación del proceso, es necesario tener saber los conocimientos básicos sobre los que se apoya esta técnica, destacando la correspondencia entre imágenes como lo más complejo.

3.1 Geometría epipolar

Tomar una imagen no es más que proyectar el espacio 3D en el espacio 2D. Dado que se pierde la información de profundidad del objeto capturado, es inevitable preguntarse si es posible su extracción a partir de la propia imagen tomada.

Siguiendo la Figura 3, donde **X** es un punto en el espacio 3D al que se puede acceder desde el punto **C1** (3D) siguiendo la dirección del vector **V1** o desde el punto **C2** (3D) siguiendo la dirección del vector **V2**, es fácil comprobar que la obtención de **X** con el único conocimiento de **C1** y **V1** es imposible. Sin embargo, con la triangulación conseguida con el conocimiento de **C2** y **V2** también, si es posible hallar su posición.

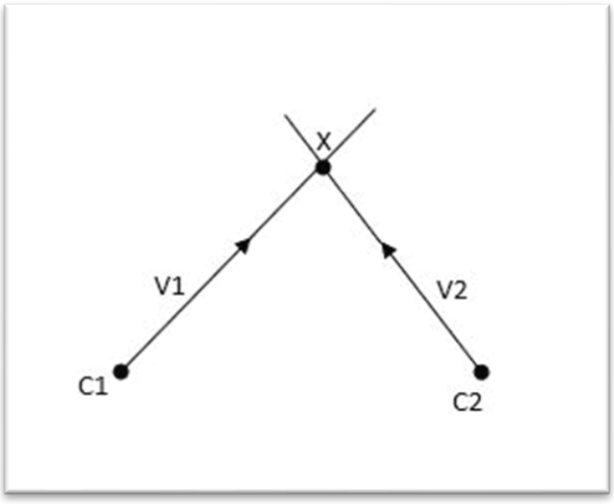

Figura 3. Triangulación

Algo similar ocurre con la localización de un punto 3D a través de la información proporcionada por una imagen. Por lo tanto, se necesita tomar dos imágenes.

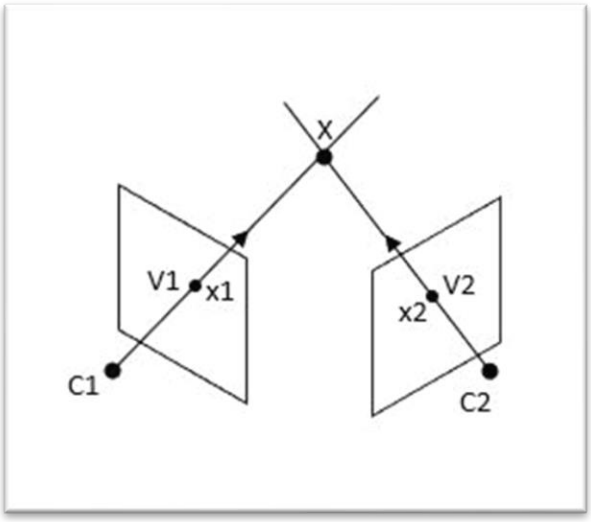

Figura 4. Triangulación 3D

Una mejor explicación se tiene observando la Figura 4. Para la obtención de **X**, ya se ha dejado constancia de que se necesitan como datos los valores de **C1**, **C2**, **V1**, y **V2**. No obstante, tomando dos imágenes, a priori solo se conocen las posiciones de las cámaras, **C1** y **C2**, y la posición de **X** proyectada en ambas imágenes, **x1** y **x2**. Con esta información, se pueden calcular **V1** y **V2** y, por consiguiente, la posición de **X**.

El inconveniente de esta triangulación generada como solución al problema es hallar la correspondencia de un píxel de una imagen en la otra.

Si se procede a buscar píxeles que tengan la misma intensidad, aparece una infinidad de correspondencias. Una mejor alternativa es estudiar la intensidad de los píxeles vecinos en una ventana de tamaño (nxn) píxeles siendo n un número impar. Aun así, la cantidad de correspondencias encontradas puede ser alta y, además, la capacidad de procesamiento del ordenador puede hacer el proceso lento. Aquí entra en juego la geometría epipolar.

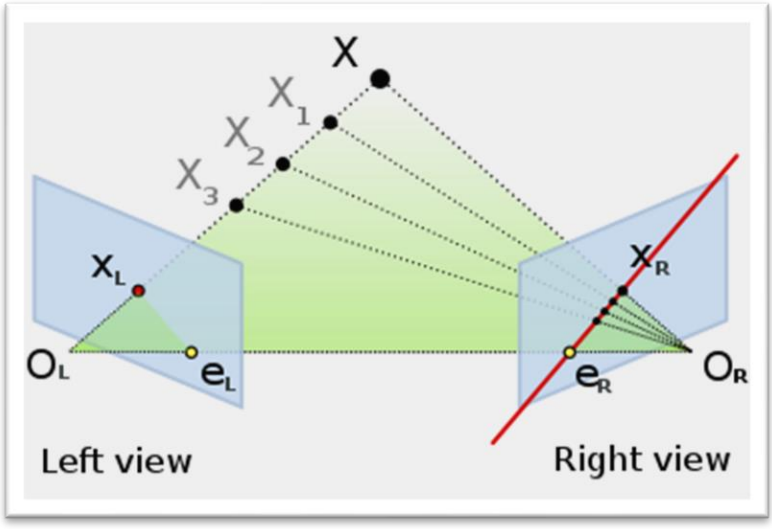

Figura 5. Geometría epipolar [12]

La geometría epipolar se refleja en la Figura 5. Las posiciones de las cámaras son $\mathbf{O_L}$ y $\mathbf{O_R}$, y **X** el objeto cuya profundidad se desea conocer. Si se captura una imagen de la recta que une los puntos $\mathbf{O_L}$, $\mathbf{x_L}$, y **X**, se obtiene la línea roja sobre la que debe estar la proyección de **X** en la imagen de la cámara derecha, es decir, $\mathbf{x_R}$. De esta manera, la búsqueda de la correspondencia se limita a la línea roja, también conocida como **línea epipolar**.

La unión de las dos cámaras resulta en la **línea base** y su intersección con los planos de las imágenes origina los epipolos correspondientes (**e_L**, **e_R**). Finalmente, podemos definir el **plano epipolar** con la **línea base** y el vector **O_L-X**. La **línea epipolar** mencionada se obtiene de la intersección de dicho plano con el plano de la imagen derecha.

En definitiva, la geometría epipolar reduce y simplifica la búsqueda de correspondencia de píxeles a la línea epipolar, para cuya obtención solo son necesarios los datos de las posiciones de las cámaras y el vector que une una de las cámaras con la proyección del objeto en cuestión sobre su imagen.

A lo realizado hasta ahora se le puede aplicar una simplificación más: planos de imágenes paralelos.

En esta simplificación, las líneas epipolares se mantienen paralelas a la línea base de forma que un píxel ocupando la posición (x1, y1) tiene su correspondencia en la otra imagen en la posición (x2, y2=y1). [13]

3.2 Estéreo vision

La visión estereoscópica tiene la capacidad de generar una imagen tridimensional combinando la información aportada por dos imágenes dimensionales. Su funcionamiento está basado en el concepto de disparidad binocular.

El efecto de la disparidad se puede entender primeramente con un sencillo ejemplo. Supongamos la existencia de una cámara fija capturando dos objetos separados entre sí una cierta distancia a lo largo del eje de profundidad. Si ambos objetos se trasladan paralelamente la misma distancia, la cámara percibirá mayor desplazamiento del objeto más cercano. La Figura 6 representa este ejemplo mediante geometría donde, sin duda, manteniendo la longitud del cateto opuesto, el ángulo de visión desde la cámara **C1** depende de la longitud del cateto contiguo.

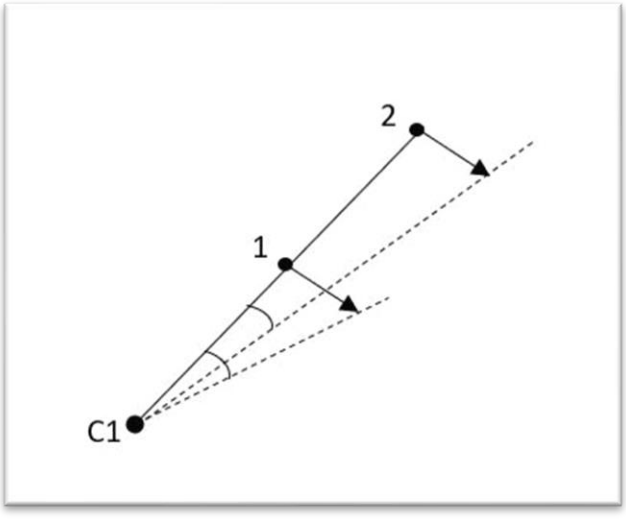

Figura 6. Efecto de disparidad

La disparidad sigue la siguiente ecuación matemática:

$$disparidad = x - x' = \frac{B \cdot f}{Z} \tag{3.1}$$

donde **B** es la distancia entre las cámaras, **f** la distancia focal característica de la cámara, **Z** la profundidad del punto detectado, **x** la distancia en píxeles del centro de la imagen al píxel correspondiente a la proyección del punto **X** sobre la imagen izquierda y **x'** la distancia en píxeles del centro de la imagen al píxel correspondiente a la proyección del punto **X** sobre la imagen derecha

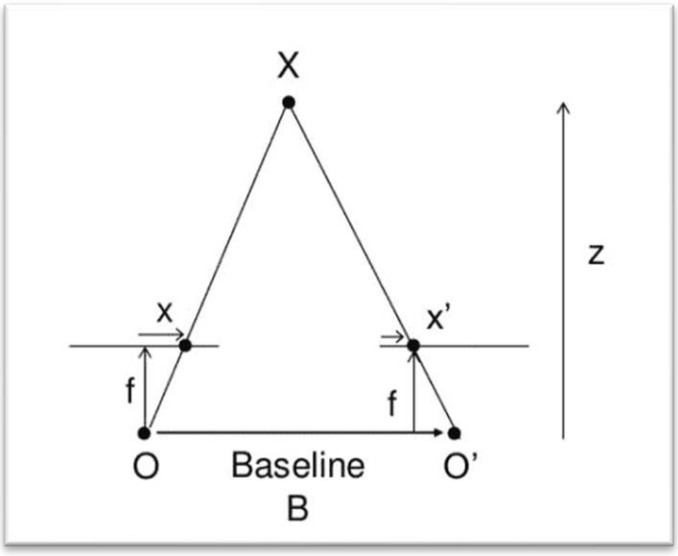

Figura 7. Disparidad [14]

Volviendo al objetivo principal, es decir, la localización de la nube, el dato importante a calcular es la profundidad **Z**, la cual nos proporcionará la altura de la nube con respecto a las cámaras.

Además, siguiendo la geometría expuesta en la Figura 8 y sabiendo que el punto **X**, su proyectado sobre la imagen, y el origen de la cámara se encuentran sobre la misma línea, se puede calcular la distancia tridimensional (x real) del punto **X** a la cámara **C1** únicamente con los datos de los catetos contiguos, que en este caso son la distancia focal y la profundidad calculada, y los catetos opuestos, que son la distancia **x** del píxel estudiado al centro de la imagen y **x real** que es la íncógnita del problema.

$$x\,real = \frac{x \cdot B}{x - x'} \tag{3.2}$$

$$y\,real = \frac{y \cdot B}{x - x'} \tag{3.3}$$

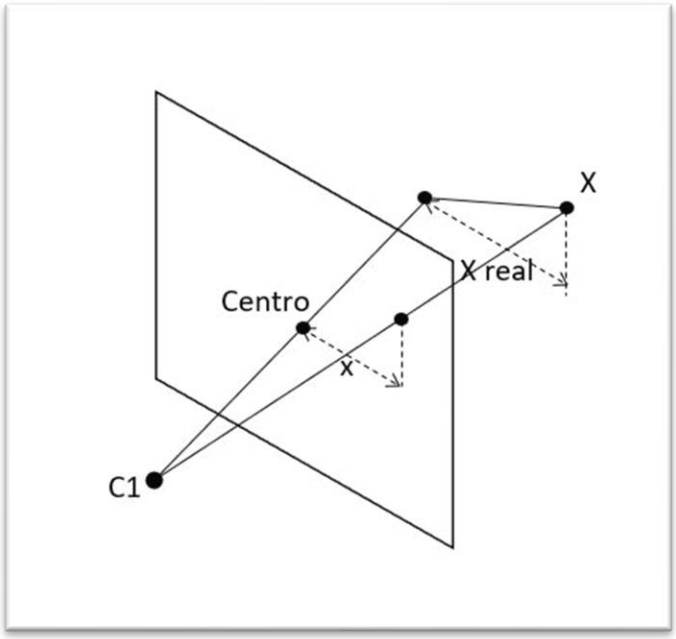

Figura 8. Trigonometría para cálculo de **x** e **y** del objeto 3D

4 RESULTADOS

Una vez desarrolladas las bases fundamentales para la realización del sistema, es necesaria su evaluación para tomar las decisiones pertinentes en la búsqueda de un programa satisfactorio. A continuación, se muestran las decisiones tomadas con respecto a las distintas alternativas y las suposiciones o simplificaciones realizadas para concluir en el Codigo Final, al que se le aplicará una serie de imágenes para comprobar su funcionamiento.

4.1 Análisis

Antes de proceder con el estudio de las bases teóricas, es importante dejar constancia de que aquellas imágenes en las que no aparezca refrencia alguna, son imágenes personales tomadas por la cámara de un iPhone o de un Xiaomi.

La primera detección y clasificación del objeto se puede efectuar, como se estudió en el apartado 2, con una variedad de métodos. Evaluaremos cúal de los mencionados es el más adecuado para las nubes.

El principal inconveniente del **Template Matching** para el caso de las nubes reside en la comparación con la imagen plantilla que hay que proporcionar, ya que esta técnica no es eficiente si la imagen buscada sufre cambios, algo que las nubes padecen de forma constante.

A continuación, empleando el Código 1, obtenido del proporcionado por OpenCV [1] con pequeñas modificaciones, se mostrará la baja precisión de la técnica.

Código 1: Detección de nube mediante Template Matching

```
import cv2

#Leer y cambiar de tamaño la imagen donde se quiere buscar y la imagen
# platilla, respectivamente
img = (cv2.resize(cv2.imread('fotos/f03.JPG', 0), (0, 0),
                  fx=0.2, fy=0.2))
template = (cv2.resize(cv2.imread('fotos/nube.jpg', 0), (0, 0),
                       fx=0.2, fy=0.2))
h, w = template.shape

# Probaremos los distintos métodos
methods = [cv2.TM_CCOEFF, cv2.TM_CCORR, cv2.TM_SQDIFF]

for method in methods:
    img2 = img.copy()
    result = cv2.matchTemplate(img2, template, method)
    min_val, max_val, min_loc, max_loc = cv2.minMaxLoc(result)
    if method in [cv2.TM_SQDIFF]:
        location = min_loc
    else:
        location = max_loc

    bottom_right = (location[0] + w, location[1] + h)
    cv2.rectangle(img2, location, bottom_right, 255, 5)
    cv2.imshow('Match', img2)
```

```
cv2.waitKey(0)
cv2.destroyAllWindows()
```

Figura 9. Imagen donde buscar correspondencia

Figura 10. Imagen plantilla a buscar.

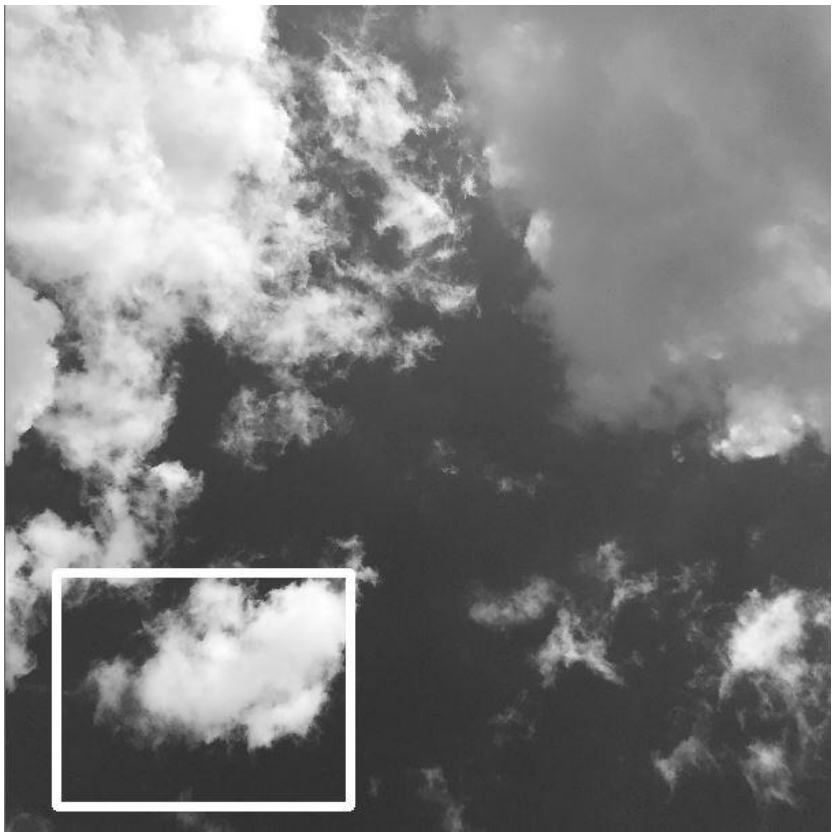

Figura 11. Template Matching Correlation Coefficient

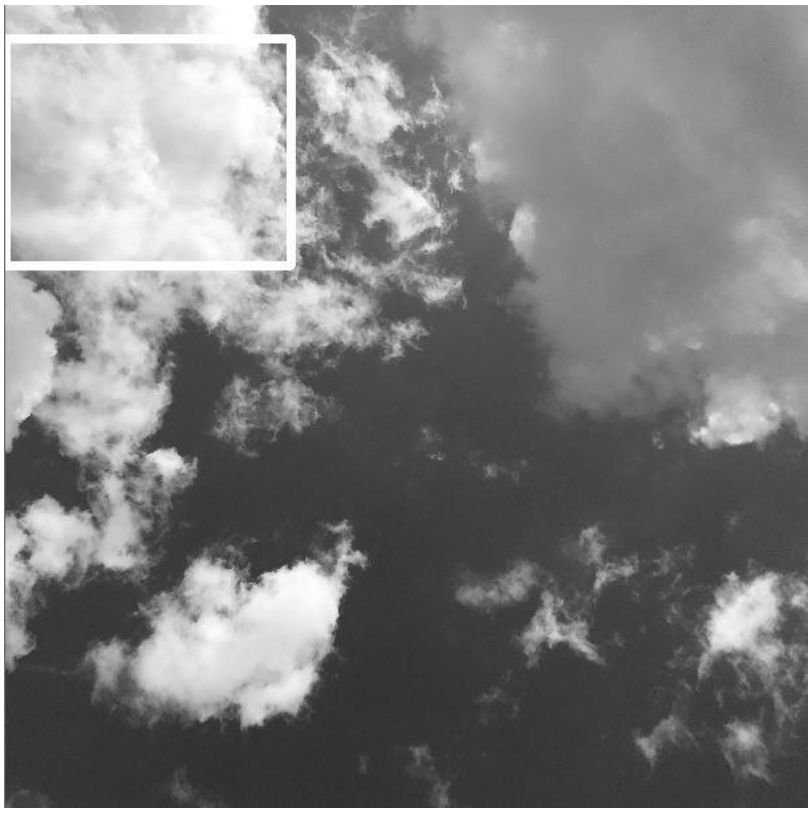

Figura 12. Template Matching Cross Correlation

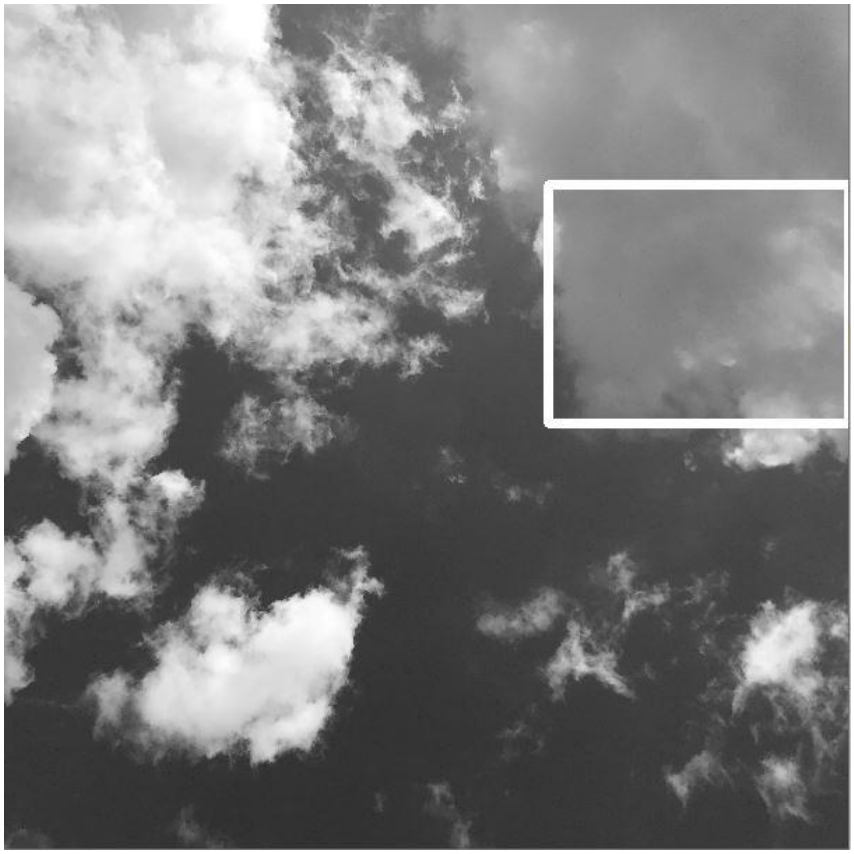

Figura 13. Template Matching Sum of Square Difference

En las Figuras 11, 12 y 13 se observa la mala calidad de la técnica explicada anteriormente, ya que con una imagen plantilla no se puede contemplar las distintas formas que puede tomar una nube. Incluso aplicando un Multi-Template Matching no sería el método más adecuado.

Quedando descartado notoriamente el Template Matching, estudiamos ahora un poco más en profundidad la implementación de YOLO.

Su implementación oficial, Darknet [15], consta de archivos entrenados para la detección de las siguientes clases u objetos:

[person, bicycle, car, motorcycle, airplane, bus, train, truck, boat, traffic light, fire hydrant, stop sign, parking meter, bench, bird, cat, dog, horse, sheep, cow, elephant, bear, zebra, giraffe, backpack, umbrela, handbag, tie, suitcase, frisbee, skis, snowboard, sports ball, kite, baseball bat, baseball glove, skateboard, surfboard, tennis racquet, bottle, wine glass, cup, fork, knife, spoon, bowl, banana, apple, sándwich, orange, broccoli, carrot, hot dog, pizza, donut, cake, chair, couch, potted plant, bed, dining table, toilet, tv, laptop, mouse, remote, keyboard, cell pone, microwave, oven, toaster, sink, refrigerator, book, clock, base, scissors, teddy bear, hair drier, toothbrush]

Debido a la ausencia de la clase “cloud” (nube) en el listado, habría que entrenar al clasificador con un nuevo data set. Esto, por lo tanto, habría que realizarlo tanto para el método YOLO como para el Haar Cascade.

Ante la obligatoriedad del entrenamiento, se opta por lo tanto por el método YOLO, ya que éste tiene la característica de ser el más rápido por su funcionamiento de una única lectura de imagen.

Para la realización del entrenamiento customizado de YOLO se sigue uno de los tutoriales aportados por PySource **[9]**.

Primeramente, se requiere un data set con un gran número de imágenes de nubes para que no surjan grandes problemas de precisión. Con este fin, se recurre a la librería icrawler de Python con la que se pueden obtener imágenes de Bing fácilmente tras su instalación empleando el Código 2 [17].

Código 2: Descarga de imágenes de nubes

```
from icrawler.builtin import BingImageCrawler

classes=['clouds']
number=100
for c in classes:
    bing_crawler=BingImageCrawler(storage={'root_dir':'imagenes'})
    bing_crawler.crawl(keyword=c,filters=None,max_num=number,offset=0)
```

A continuación, para la identificación del objeto en las distintas imágenes se emplea el software Labellmg que precisa etiquetar manualmente todas y cada una de las detecciones realizadas por el usuario (Figura 14). A mayor dedicación y precisión creando los cuadros identificadores, mayor será la precisión del clasificador resultante.

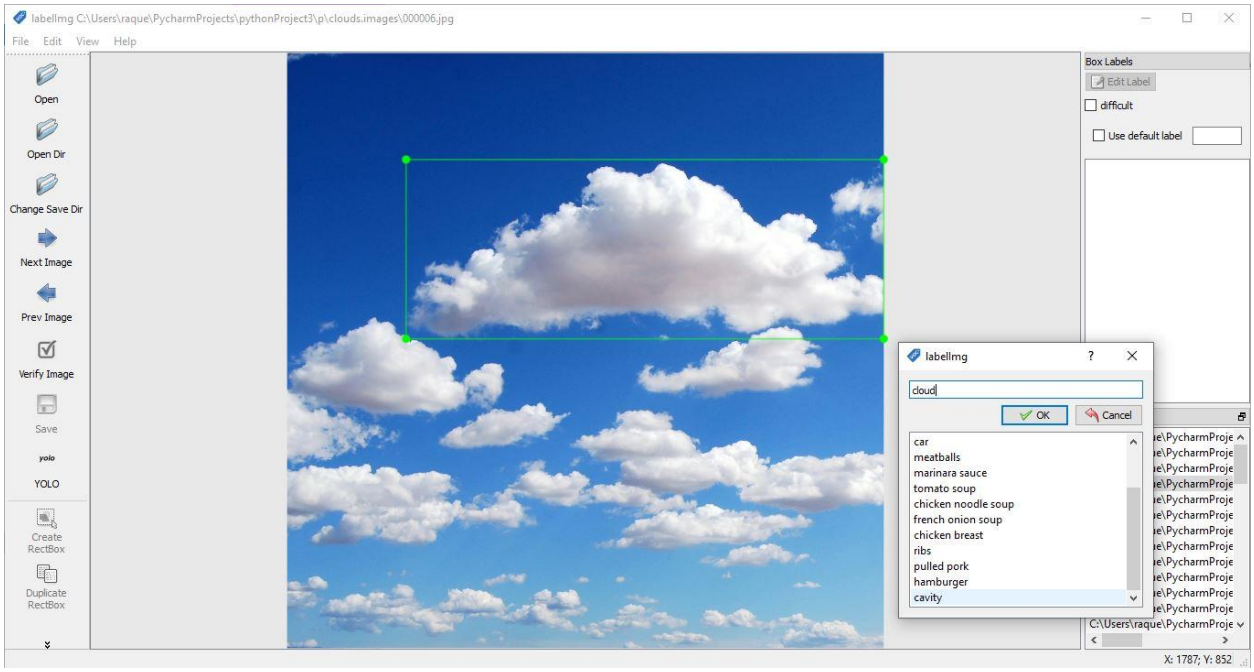

Figura 14. Software Labellmg

Por cada imagen debidamente identificada y guardada, el software genera un archivo de texto con la información de las clases detectadas en ellas y las correspondientes coordenadas de sus cuadros identificadores.

Una vez completada esta operación, se realiza el entrenamiento y, por lo tanto, la obtención de los archivos necesarios para utilizar el clasificador, como los pesos ponderados, con la ayuda de los archivos proporcionados.

Finalmente, ejecutando el **Código 3**, con modificaciones, se obtienen los resultados de las Figuras 15 y 16. Observamos que, disminuyendo el límite de confianza de los cuadros identificadores creados por el

clasificador, se detecta un mayor número de nubes, aunque hay un cuadro presente del que se podría prescindir. A pesar de que la precisión se puede aumentar, como hemos visto, variando este factor, se decide realizar otro entrenamiento con mayor dedicación de manera que en el **Código Final** se combinen, como se mostrará más adelante, la solución de ambos.

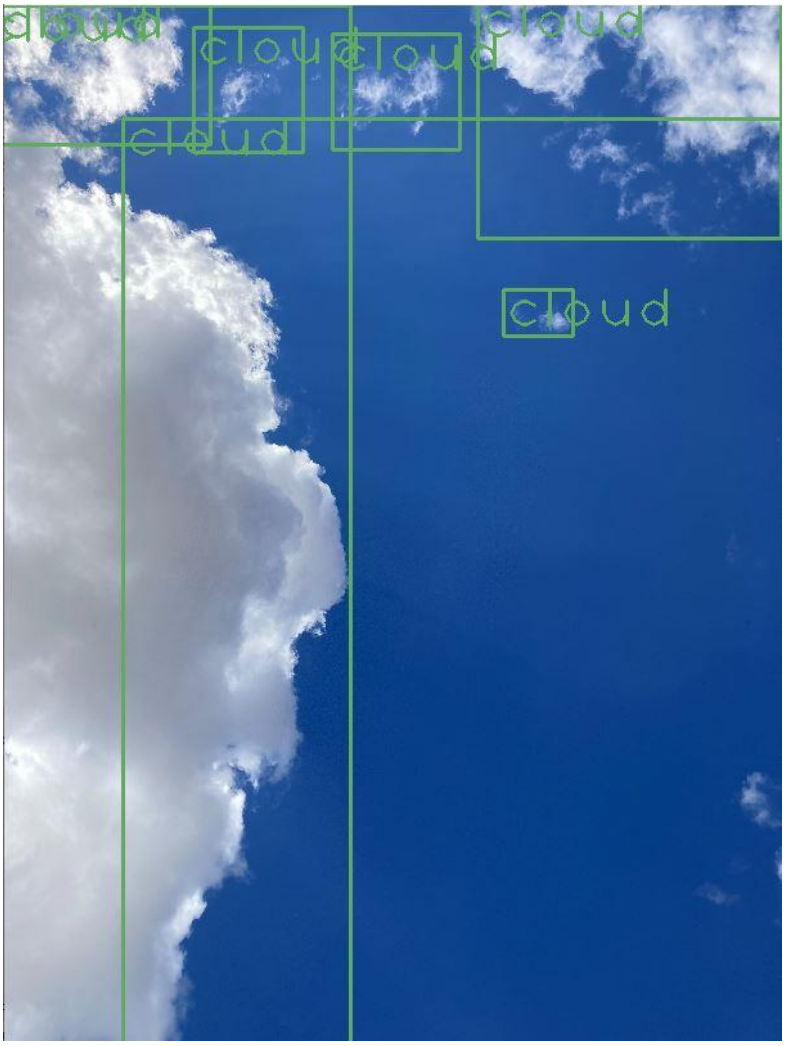

Figura 15. YOLO, límite de confianza 0.2

Figura 16. YOLO, límite de confianza 0.5

Código 3: YOLO

```
import cv2
import numpy as np

# Cargar archivos YOLO
net = (cv2.dnn.readNet("yolov3_training_last1.weights",
                       "yolov3_testing.cfg"))

# Clases a identificar
classes = ["cloud"]

# Ruta de la imagen
img_path = 'fotos/f03.jpg'

output_layers = net.getUnconnectedOutLayersNames()
colors = np.random.uniform(0, 255, size=(len(classes), 3))

# Leer imagen y cambiar tamaño
img = cv2.imread(img_path)
img = cv2.resize(img, None, fx=0.2, fy=0.2)
height, width, channels = img.shape
```

```
# Detectar objetos
blob = (cv2.dnn.blobFromImage(img, 0.00392, (416, 416), (0, 0, 0),
                              True, crop=False))

net.setInput(blob)
outs = net.forward(output_layers)

# Mostrar información en imagen sobre objetos detectados
class_ids = []
confidences = []
boxes = []
for out in outs:
    for detection in out:
        scores = detection[5:]
        class_id = np.argmax(scores)
        confidence = scores[class_id]
        if confidence > 0.3:
            center_x = int(detection[0] * width)
            center_y = int(detection[1] * height)
            w = int(detection[2] * width)
            h = int(detection[3] * height)

            # Coordenadas del rectángulo identificador
            x = int(center_x - w / 2)
            y = int(center_y - h / 2)

            boxes.append([x, y, w, h])
            confidences.append(float(confidence))
            class_ids.append(class_id)

# Proporcionar los índices de aquellas bounding boxes que cumplen
# con un límite de confianza. Probamos com 0.2 y 0.5
indexes = cv2.dnn.NMSBoxes(boxes, confidences, 0.5, 0.4)
font = cv2.FONT_HERSHEY_PLAIN
for i in range(len(boxes)):
    if i in indexes:
        x, y, w, h = boxes[i]
        label = str(classes[class_ids[i]])
        color = colors[class_ids[i]]
        cv2.rectangle(img, (x, y), (x + w, y + h), color, 2)
        cv2.putText(img, label, (x, y + 30), font, 3, color, 2)


cv2.imshow("Image", img)
key = cv2.waitKey(0)

if len(indexes) == 0:
    print('No hay nubes')
```

Continuando con los mismos pasos del estudio previo, es momento de identificar el contorno de las nubes. Por ello, teniendo presente lo expresado en el apartado 2.4, se decide tratar la imagen de distinta manera en función del valor obtenido en la desviación típica de sus valores del ratio B/R normalizado. Así, el límite determinado es 0.2. Por lo cual, aquellas imágenes que presenten una desviación típica de mayor valor que el límite, serán bimodales y se les aplicará un sencillo proceso de thresholding. Sin embargo, para las imágenes unimodales cuya desviación tipica sea inferior al límite, el tratamiento consistirá en la aplicacion del thresholding a la imagen del ratio B/R normalizado con un umbral determinado por la media y la desviación estándar de dichos valores. El valor límite 0.2 se ha impuesto tras calcular la desviación estándar del ratio B/R de varias imágenes

y estudiar sus resultados.

Por último, con respecto al cálculo de la localización de las nubes, nos acogeremos a la simplificación mostrada en el apartado 3.2 de los planos de las imágenes de ambas cámaras paralelos. Las pruebas a realizar al **Código Final** en el siguiente apartado, implicarán tanto imágenes tomadas por móvil personal como imágenes obtenidas de internet debidamente referenciadas. Asi mismo, se expondrá el método empleado para la correspondencia entre imágenes basada en la geometría epipolar.

4.2 Sistema Final

Finalmente, el trabajado desarrollado queda resumido en las siguientes líneas. El **Código Final**, encargado de ejecutarlo todo, incluye el **Código 3** perteneciente al apartado 4.1 y llama a las funciones definidas en los **Código calculostd** y **Código calculocnts**.

Su funcionamiento comienza cargando los dos archivos pertenecientes a los entrenamientos realizados a YOLO y las imágenes procedentes de la cámara izquierda y la cámara derecha. Seguidamente, mediante el método YOLO se procede a representar gráficamente la detección de las nubes conseguida en la imagen de la cámara izquierda con uno de los archivos de entrenamiento cargados anteriormente. Sin embargo, puesto que el tamaño de la imagen puede sobrepasar el tamaño de la pantalla del ordenador, en este caso 17 pulgadas, ésta se redimensiona antes de llevar a cabo todo el proceso empleando un factor escala que permite mantener a la imagen con un gran tamaño para su observación sin superar los límites. Una vez finalizado este paso, el sistema vuelve a estudiar la imagen de la cámara izquierda, pero esta vez con el archivo generado del otro entrenamiento. De esta manera nos aseguramos de que, si uno de los entrenamientos no detecta alguna nube, ésta pueda ser detectada por el otro. Para no colapsar la imagen de cuadros identificadores, se comparan los detectados por un archivo con los detectados por el otro y se establece un límite de 75 píxeles por el cual aquellos cuadros cuyos centros estén a una distancia menor a la establecida no serán representados gráficamente. En la Figura 17 se tiene un ejemplo de detección de nubes donde aparecen en amarillo los cuadros identificadores de la primera detección y en verde los de la segunda. Por otro lado, en la Figura 18, se muestra otro ejemplo con un avión y su estela presentes en la imagen que, sin embargo, no son detectados por el sistema.

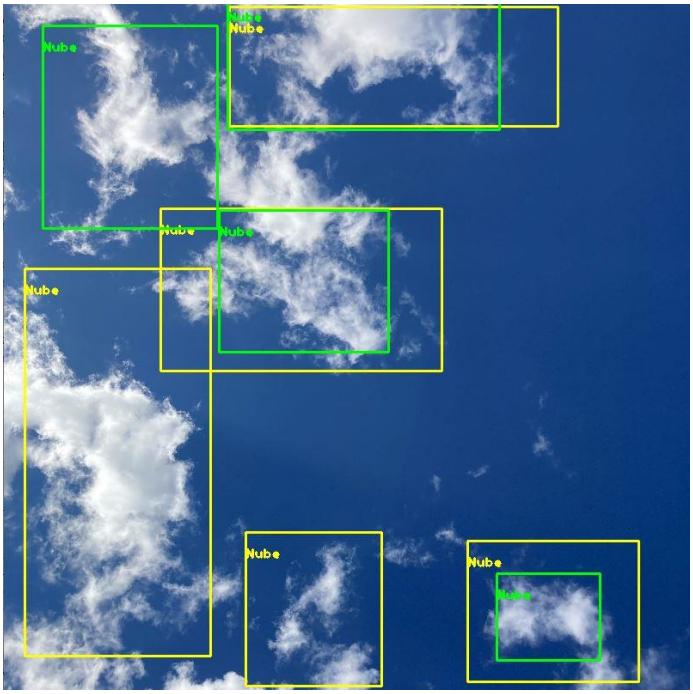

Figura 17. Detección por YOLO

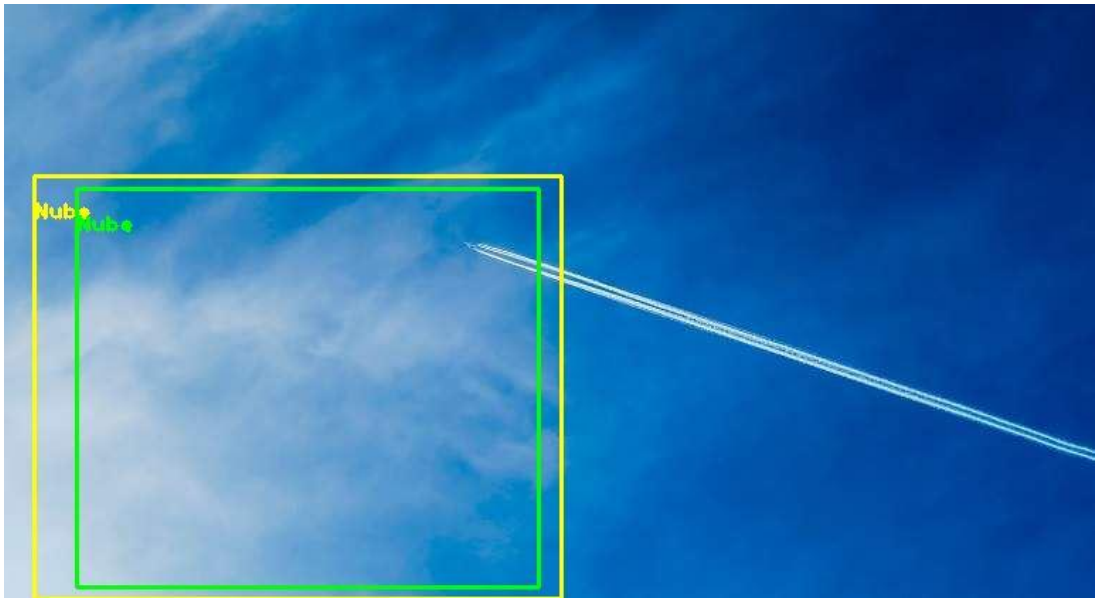

Figura 18. Detección por YOLO con estela de avión [18]

A continuación, si se generan cuadros identificadores de nubes en la imagen de la cámara izquierda, se procede a la identificación de sus contornos para luego buscar correspondencia en la imagen de la cámara derecha. A partir de este momento, se emplean las versiones en escala de grises de ambas imágenes. En este punto, se llama a las funciones **calculostd** y **calculocnts**, presentes en los **Códigos calculostd y calculocnts** respectivamente.

La función **calculostd** recibe como entrada la imagen BGR y extrae los tres canales de color. Con los canales B y R calcula los valores de ratio normalizado de la ecuación 2.2, eliminando los valores NaN que puedan aparecer si el denominador toma valor 0, y obtiene la desviación estándar de ellos. Además, debido a los posibles valores negativos que puede tomar el ratio normalizado y su rango de

comportamiento, se escala el conjunto de valores de la imagen al rango [0,256) para posibilitar su representación gráfica. Así, la función devuelve la desviación estándar, el conjunto de valores de ratio normalizado, el conjunto de valores de ratio normalizado escalado y un vector de color. El conjunto de valores de ratio normalizado y el vector color se emplearán para su representación y comparación con el valor de la desviación estándar.

Por otro lado, la desviación estándar es utilizada como entrada junto a la imagen para la función **calculocnts**. Como ya se ha explicado anteriormente, esta función categoriza la imagen en unimodal o bimodal en función de si el valor obtenido de la desviación estándar supera o no el límite 0.2. Si se supera, la imagen es bimodal y, por lo tanto, se aplica la técnica **Bilateral Filtering** a la imagen para suavizar ruido manteniendo bordes para, después, utilizar el detector de contornos **Canny**. Por último, antes de hacer uso de la función que devuelve los píxeles de los contornos en vectores, se procesa la imagen mediante una matriz kernel que realiza una dilatación, consiguiendo unificar y suavizar los contornos. Si, por el contrario, la desviación estándar no supera el valor límite, la imagen es considerada unimodal. El proceso, entonces, consiste en aplicar el método simple **Thresholding** al conjunto de valores de ratio normalizado escalado, donde el valor umbral es función de la media y desviación típica de dichos valores. Finalmente, esta imagen también incluye el proceso de dilatación.

Las Figuras 19 y 20 constituyen un primer ejemplo de imagen unimodal, donde el elemento principal son las nubes. La desviación típica asociada a esta imagen es 0.13, valor que coincide con la Figura 20, donde se observa poca variación y un único pico mayor.

Figura 19. Imagen unimodal [19]

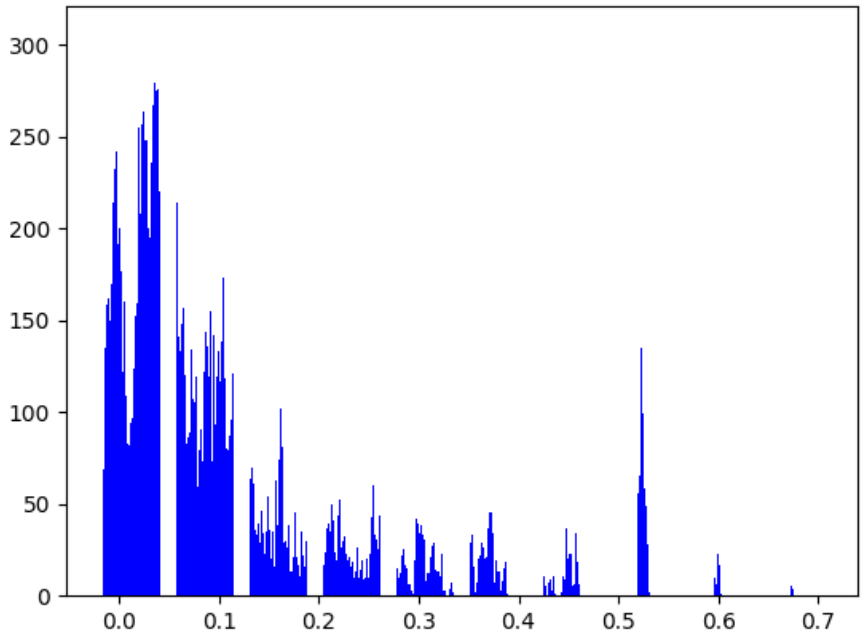

Figura 20. Valores ratio normalizado de Fig.19

Las siguientes Figuras 21 y 22 muestran el caso de una imagen bimodal procesada como tal, cuya desviación típica es 0.32.

Figura 21. Imagen bimodal

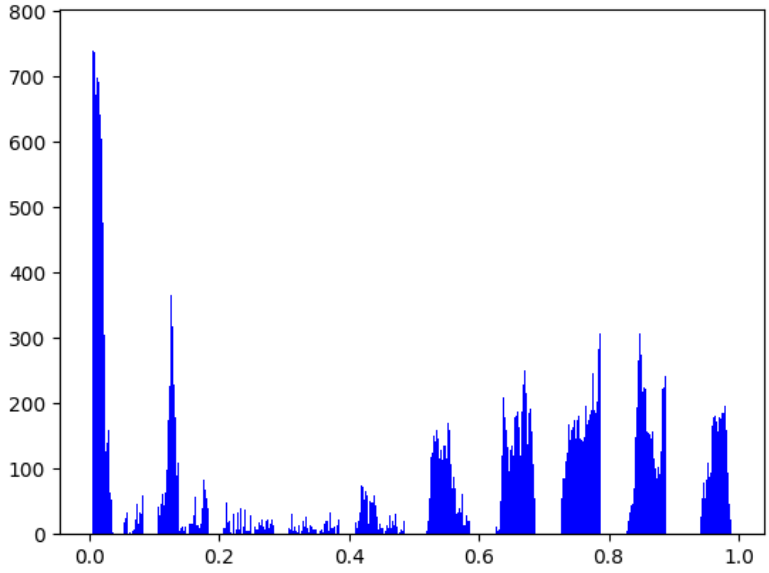

Figura 22. Valores ratio normalizado de Fig.21

Se presenta ahora un ejemplo de mala precisión de una imagen en un día con niebla, cuya desviación estándar es 0.026 y el comportamiento del sistema se muestra en las Figuras 23 y 24. Como se observa, la gráfica correspondiente a los valores de ratio normalizado presenta mayor variación que la del primer ejemplo.

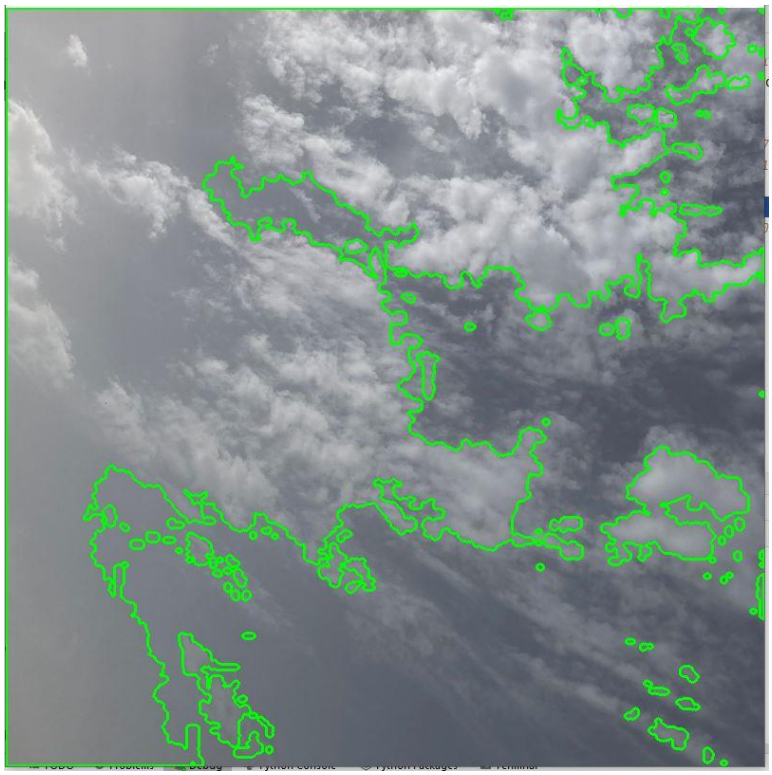

Figura 23. Imagen unimodal día nublado

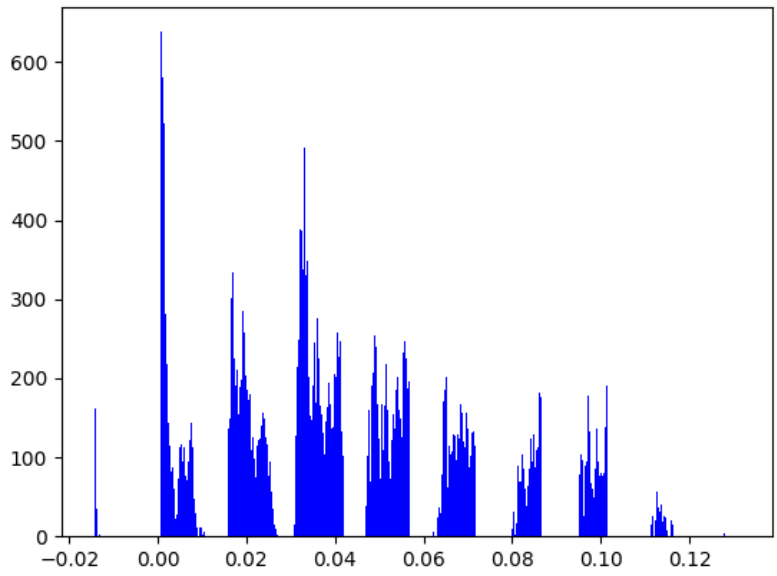

Figura 24. Valores ratio normalizado de Fig.23

Una vez almacenados los contornos en vectores, es momentos de buscar correspondencias. Antes de comenzar, se establecen algunos datos que serán utilizados más adelante:

- **Separación entre cámaras.** Las imágenes tomadas por móvil personal fueron tomadas a 25 metros de distancia y las imágenes de internet pueden o no venir con esa información. Este parámetro se modifica en función de las imágenes a tratar, y para aquellas capturas donde se desconoce el dato, se suponen distancias lógicas y similares a las mencionadas anteriormente.
- **Distancia focal de la cámara.** Otro dato a modificar en función de las cámaras. En este caso, el móvil empleado tiene una distancia focal de 35 mm.
- **Tamaño del píxel.** Dependiendo de la resolución de la cámara, tendrá un valor u otro. De nuevo, en este caso, puesto que la resolución es de 326 píxeles por pulgada, el tamaño del píxel es 25.4mm/326píxel.
- **Tamaño de celda SDA.** Este valor esta prefijado a 9 pero puede modificarse en función de los píxeles vecinos que se quieran incluir en la comparación. El uso de la Suma de Diferencias Absolutas será explicado a continuación.
- **Límites.** Debido al error de precisión ocasionado por la muestra manual de fotos, además de buscar correspondencia ligada a la línea horizontal según lo establecido por la geometría epipolar para la simplificación de planos de imágenes paralelos, también será necesario buscar en líneas horizontales superiores e inferiores abarcadas por un límite. El establecido aquí es 10.

El sistema toma los contornos de la imagen izquierda y realiza una selección de píxeles de dichos contornos para no estudiarlos todos y reducir el tiempo computacional. En lugar de buscar el valor de cada uno de los píxeles seleccionados de la imagen izquierda en la imagen derecha, se utiliza el método de Suma de Diferencias Absolutas similar al mencionado en el apartado 2.1. Mediante esta técnica se comparan bloques de píxeles. Cada par de píxeles asociados es restado para luego sumar

todas las restas obtenidas en valor absoluto. Aquella comparación que resulte en el mínimo valor SDA, cercano a cero, será la correspondencia buscada. Ya que la búsqueda se realiza a lo largo de 21 líneas horizontales, se decide simplificar el proceso de manera que la comparación solo se realizará con los píxeles de los contornos de la imagen de la cámara derecha que se encuentren en dichas líneas.

Una nube puede tener una correspondencia falsa, por lo que para descartarla se sigue el siguiente proceso. El sistema va almacenando las variaciones verticales correspondientes a los píxeles de un contorno. Si la variación más frecuente se repite más de 10 veces, entonces suponemos que el error de precisión ha llevado a la nube a desplazarse dicha cantidad de una imagen a otra. Sin embargo, si la variación más frecuente no alcanza las 10 repeticiones, se supone falsa correspondencia.

Posteriormente, aunque cada píxel de correspondencia posea una variación vertical y una disparidad propias, se toman la variación vertical y disparidad más frecuentes de todos los hallados para representar la correspondencia del contorno completo en la imagen de la derecha.

Paralelamente, empleando las ecuaciones 3.1, 3.2 y 3.3 y realizando las conversiones pertinentes entre pulgada y milímetro, se calculan las profundidades de cada nube detectada y posición 3D.

Lo siguiente son los resultados respecto al cálculo de la localización del objeto de las pruebas realizadas al sistema conjunto, donde en lugar de mostrar todas las coordenadas 3D, se indica la coordenada media en el eje X del contorno y la coordenada media en el eje Y. Para entender los valores positivos y negativos de los valores medios de las coordenadas basta con observar la Figura 25 donde el centro de coordenadas es la cámara de la izquierda, ya que están calculadas con respecto a esta imagen.

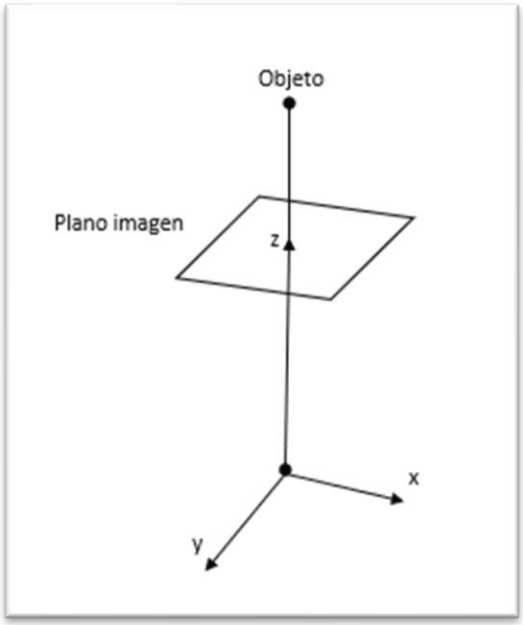

Figura 25. Sistema de coordenadas de la cámara

1. Imagen tomada de internet a la que se le ha hecho dos recortes simulando una cámara izquierda y otra derecha. [20] Suponemos una separación entre ellas de 400 metros.

Figura 26. Contornos imagen cámara izquierda

Figura 27. Contornos imagen cámara derecha

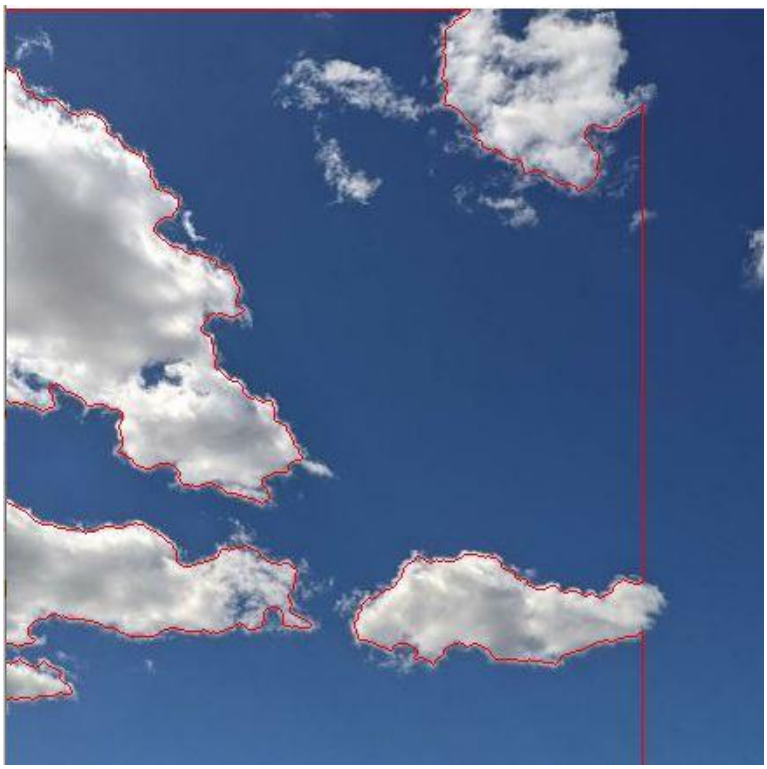

Figura 28. Contornos con correspondencia

Contorno	Profundidad (m)	Posición X media (m)	Posición Y media (m)
1	2604.13	-56.96	148.15

Figura 29. Posición 3D

Al no ser imágenes capturadas por cámaras reales, el sistema no calcula bien la profundidad, asignando la misma para todas las nubes, ya que todas sufren el mismo desplazamiento.

2. Par de imágenes tomadas de internet de un coche en movimiento [21]. Suponemos una distancia entre cámaras o capturación de imágenes de 400 metros. El resto de los parámetros permanece intacto.

Figura 30. Imagen de cámara izquierda

Figura 31. Imagen de cámara derecha

Figura 32. Contornos con correspondencia

Contorno	Profundidad (m)	Posición X media (m)	Posición Y media (m)
1	2364.28	676.22	505.26
2	2428.17	-142.24	-263.22

Figura 33. Posición 3D

3. En este caso, las imágenes están tomadas por móvil personal a una distancia de 25 metros.

Figura 34. Imagen de cámara izquierda

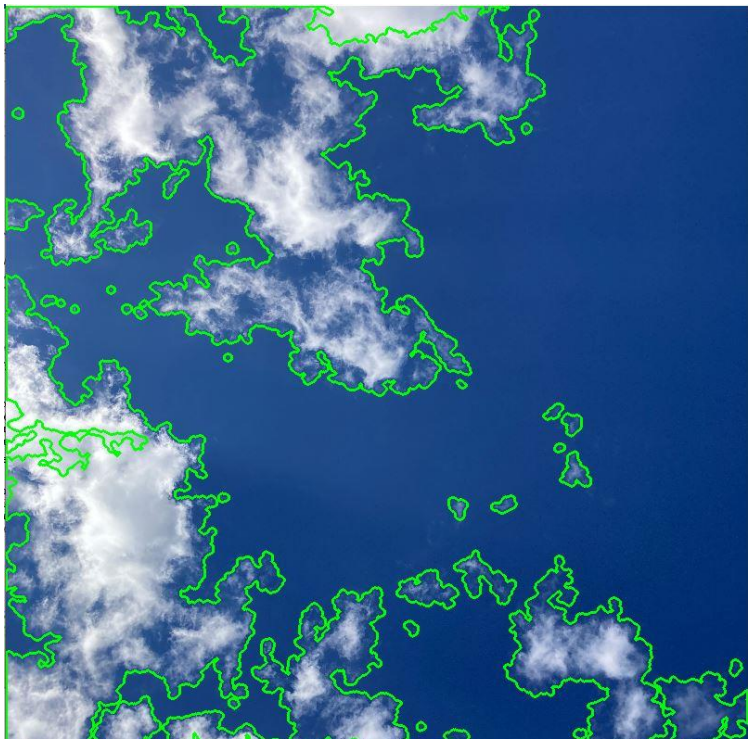

Figura 35. Imagen de cámara derecha

Figura 36. Contornos con correspondencia

Contorno	Profundidad (m)	Posición X media (m)	Posición Y media (m)
1	415.94	89.11	216.56
2	591.06	-383.46	222.72
3	574.77	-111.23	-262.51

Figura 37. Posición 3D

Aunque a simple vista no se aprecia, la toma de imágenes por la misma cámara con una separación en tiempo de poco menos de un minuto altera los resultados ya que la nube cambia de aspecto. Es por ello, que para algunas nubes no encuentra correspondencia.

5 CONCLUSIONES Y DESARROLLOS FUTUROS

A lo largo de este documento se han ido explicando y demostrando las bases del proyecto con el que se ha intentado desarrollar una ayuda para la extracción óptima de energía solar de placas solares mediante el procesamiento de imagen digital y estéreo visión. Con los resultados expuestos en el apartado 4, se puede visualizar una serie de conclusiones con respecto al proyecto realizado, así como propuestas de trabajos futuros.

5.1 Conclusiones

Se extraen las siguientes conclusiones y aportaciones:

- El método YOLO como detección de objetos es altamente eficaz. En el caso de las nubes, se puede comprobar que parte de ellas, en ocasiones, queda fuera del cuadro identificador. Sin embargo, detecta las nubes sin importar su densidad u opacidad, y el defecto anterior no tendría mucha posibilidad de mejora debido al aspecto cambiante de las nubes.
- La identificación de contornos desarrollada no es del todo efectiva ya que las funciones empleadas se basan en multiud de parámetros modificables como entrada. Esto hace que los parámetros establecidos funcionen para unas imágenes pero no para otras. Las imágenes que más complejidad conllevan, en este aspecto, son aquellas que capturan el cielo nublado. A pesar de la existencia de métodos de umbral adaptativos, el resultado conseguido no es preciso. Hemos visto como se puede recurrir a los canales de color de la imagen para optimizar la identificación de los contornos mediante el ratio normalizado de B/R y, aún obteniendo resultados comparables, algunos contornos se quedan fuera del alcance.
- La geometría epipolar juega un importante papel en la búsqueda de la correspondencia entre imágenes, simplificando dicha búsqueda a una única línea en la imagen. Se puede realizar una simplificación mayor, si se consideran, como hemos estudiado, paralelos los planos de las imágenes.
- El sistema planteado consta de dos cámaras fijas supuestas en el mismo plano y con igual orientación. Sin embargo, debido a la manipulación manual de ellas, pueden percibirse errores a la hora de hallar correspondencias. La ausencia de cámaras establecidas de forma profesional con los parámetros de orientación y posición adecuados con respecto a la simplificación de los planos de imagen de ambas cámaras paralelos, implica la búsqueda de correspondencia entre imágenes a lo largo de más de una línea horizontal, es decir, la propia línea horizontal establecida por el píxel a buscar más k líneas horizontales por encima y menos k líneas por debajo. Esto conlleva un tiempo de procesamiento no óptimo.
- La técnica de visión estéreo, a pesar del problema de la correspondencia mencionado anteriormente, en general ofrece muy buenos resultados,

5.2 Trabajos futuros

A la vista de los resultados y conclusiones obtenidos, se plantean propuestas que, desarrolladas, pueden lograr mejoras en la funcionalidad del sistema aquí presente y, además, ampliar el campo de operación:

- La propuesta actual consta de un sistema basado en dos cámaras fijas extendible a varias cámaras dispuestas paralelamente con la misma orientacion y en el mismo plano, pero una alternativa mucho más interesante es el **uso de cámaras móviles**. Aunque este proyecto podría extenderse también al

uso de cámaras móviles, solo permitiría el movimiento de las cámaras en una misma línea. Para cámaras móviles en cualquier dirección, el sistema debería incluir un proceso de calibración y reproyección de ambas imágenes en un mismo plano.

- Debido a la falta de precisión de la identificación de contornos, una posible mejora sería realizar un **estudio** más detallado de la información proporcionada por un **data set de imágenes** lo suficientemente amplio para poder **establecer ciertos parámetro**s en función de las distintas condiciones de las imágenes.
- Utilizar un proceso global de estéreo visión de mejor calidad que no necesite la búsqueda de correspondencia a lo largo de varias líneas, **mejorando así el tiempo computacional**.
- Extender los resultados mejorados de este proyecto hacia otro sistema capaz de calcular la **cobertura de las nubes**.
- **Categorizar** las nubes detectadas en sus diferentes variedades (cirros, cúmulos, nimboestratos, …) para usos meteorológicos.

REFERENCIAS

[1] «OpenCV: Template Matching,» [En línea]. Available: https://docs.opencv.org/4.x/d4/dc6/tutorial_py_template_matching.html.

[2] «stack overflow,» [En línea]. Available: https://stackoverflow.com/questions/55469431/what-does-the-tm-ccorr-and-tm-ccoeff-in-opencv-mean.

[3] E. Á. J. Ambrogio, «Reconocimiento de objetos a través de la metodología Haar Cascades,» *ARTÍCULOS PRESENTADOS A RADI | TECNOLOGÍA DE LA INFORMACIÓN Y COMUNICACIÓN.*

[4] «OpenCV: Cascade Classifier,» [En línea]. Available: https://docs.opencv.org/3.4/db/d28/tutorial_cascade_classifier.html.

[5] «AMIN,» [En línea]. Available: https://amin-ahmadi.com/cascade-trainer-gui/.

[6] «Wikipedia,» [En línea]. Available: https://es.wikipedia.org/wiki/Algoritmo_You_Only_Look_Once_(YOLO).

[7] Enrique, «Medium,» 12 Mayo 2018. [En línea]. Available: https://medium.com/@enriqueav/detecci%C3%B3n-de-objetos-con-yolo-implementaciones-y-como-usarlas-c73ca2489246.

[8] «OpenCV: Filtering,» [En línea]. Available: https://docs.opencv.org/4.x/d4/d13/tutorial_py_filtering.html.

[9] «OpenCV: Thresholding,» [En línea]. Available: https://docs.opencv.org/4.x/d7/d4d/tutorial_py_thresholding.html.

[10] «OpenCV: Canny,» [En línea]. Available: https://docs.opencv.org/4.x/da/d22/tutorial_py_canny.html.

[11] Q. Li, «A Hybrid Thresholding Algorithm for Cloud Detection on Ground-Based,» *JOURNAL OF ATMOSPHERIC AND OCEANIC TECHNOLOGY*, vol. 28, pp. 1286-1296, 2011.

[12] «Wikipedia: Epipolar geometry,» [En línea]. Available: https://en.wikipedia.org/wiki/Epipolar_geometry.

[13] «Learn OpenCV,» [En línea]. Available: https://learnopencv.com/introduction-to-epipolar-geometry-and-stereo-vision/.

[14] «Learn OpenCV,» [En línea]. Available: https://learnopencv.com/depth-perception-using-stereo-camera-python-c/.

[15] «pjreddie,» [En línea]. Available: https://pjreddie.com/darknet/yolo/.

[16] S. Canu, «PySource,» [En línea]. Available: https://pysource.com/2020/04/02/train-yolo-to-detect-a-custom-object-online-with-free-gpu/.

[17] «PyPI,» [En línea]. Available: https://pypi.org/project/icrawler/.

[18] «Ecodiario El Economista,» [En línea]. Available: https://ecodiario.eleconomista.es/viralplus/noticias/8778271/11/17/Que-es-el-rastro-blanco-que-dejan-los-aviones.html.

[19] «FreePik,» [En línea]. Available: https://www.freepik.es/fotos-premium/cielo-nublado-azul-dia-claro-soleado_17044538.htm.

[20] «Xiaomi Community,» [En línea]. Available: https://c.mi.com/thread-3300501-1-0.html.

[21] «Stereocopy Blog,» [En línea]. Available: https://stereoscopy.blog/2022/03/04/stereoscopic-3d-photography-with-a-single-lens/.

ANEXO

Código Final

```
import cv2 as cv
import numpy as np
import matplotlib.pyplot as plt
from calculostd import calculostd
from calculocnts import calculocnts
import math as mt
from statistics import mode

# Archivos de distintos entrenamientos de YOLO
archivos = (['yolov3_training_last.weights',
             'yolov3_training_last1.weights'])

# Ruta de imágenes de ambas cámaras
camaraizq = 'fotos/carizq.jpg'
camaradcha = 'fotos/cardcha.jpg'

coordenadas0 = []
color1 = (0, 255, 255)
color2 = (0, 255, 0)
font = cv.FONT_HERSHEY_PLAIN


fx, fy = [1, 1]
for m, archivo in enumerate(archivos):
    # Cargar YOLO
    net = cv.dnn.readNet(archivo, "yolov3_testing.cfg")

    output_layers = net.getUnconnectedOutLayersNames()

    # Cargar imagen de cámara izquierda
    if m == 0:
        img = cv.imread(camaraizq)
        if img.shape[0] >= 900:
            fy = 800 / img.shape[0]
        if img.shape[1] >= 1600:
            fx = 1500 / img.shape[1]
        factorescala = min([fx, fy])
        img = (cv.resize(img, None, fx=factorescala, fy=factorescala,
                         interpolation=cv.INTER_CUBIC))
        imagen = img.copy()
        height, width, channels = img.shape

    # Detectar nube
    blob = (cv.dnn.blobFromImage(img, 0.00392, (416, 416), (0, 0, 0),
                                 True, crop=False))

    net.setInput(blob)
    outs = net.forward(output_layers)

    # Mostrar información de nubes detectadas en imagen
    confidences = []
    boxes = []
```

```
    for out in outs:
        for detection in out:
            scores = detection[5:]
            class_id = np.argmax(scores)
            confidence = scores[class_id]
            if confidence > 0.3:
                center_x = int(detection[0] * width)
                center_y = int(detection[1] * height)
                if m == 0:
                    coordenadas0.append([center_x, center_y])
                w = int(detection[2] * width)
                h = int(detection[3] * height)

                # Rectangle coordinates
                x = int(center_x - w / 2)
                y = int(center_y - h / 2)

                boxes.append([x, y, w, h])
                confidences.append(float(confidence))

    if m == 0:
        indexes0 = cv.dnn.NMSBoxes(boxes, confidences, 0.5, 0.4)
        indexes = indexes0
    else:
        indexes1 = cv.dnn.NMSBoxes(boxes, confidences, 0.5, 0.4)
        indexes = indexes1

    for i in range(len(boxes)):
        if i in indexes:
            if m == 0:
                x, y, w, h = boxes[i]
                label = 'Nube'
                cv.rectangle(img, (x, y), (x + w, y + h), color1, 2)
                cv.putText(img, label, (x, y + 30), font, 1, color1, 2)
            else:
                flag = 0
                b = boxes[i][0:2]
                for coordenada in coordenadas0:
                    a = coordenada
                    distancia = np.linalg.norm(np.array(a) - np.array(b))
                    if distancia < 75:
                        flag = 1
                        break
                if flag == 0:
                    x, y, w, h = boxes[i]
                    label = 'Nube'
                    cv.rectangle(img, (x, y), (x + w, y + h),color2, 2)
                    cv.putText(img, label, (x, y + 30), font, 1,color2, 2)


if len(indexes0)  == 0 and len(indexes1) == 0:
    print('No se detectan nubes')
    cv.imshow("Cielo sin nubes detectadas", img)
else:
    cv.imshow("Nubes detectadas", img)

    grayizq = cv.cvtColor(imagen,cv.COLOR_BGR2GRAY)
    stdizq, divizq, normalizeddivizq, colorizq = calculostd(imagen)

    imgdcha = cv.imread(camaradcha)
    imgdcha = (cv.resize(imgdcha,(imagen.shape[1],imagen.shape[0]),
                         interpolation=cv.INTER_CUBIC))
```

```
    graydcha = cv.cvtColor(imgdcha,cv.COLOR_BGR2GRAY)
    stddcha, divdcha, normalizeddivdcha, colordcha = calculostd(imgdcha)

    fondo = imgdcha.copy()
    cntsizq = calculocnts(imagen, stdizq)
    cntsdcha = calculocnts(imgdcha, stddcha)

    cntscomp = []
    for contour in cntsdcha:
        cv.drawContours(imgdcha, contour, -1, (0, 255, 0), 2)
        for pixel in contour:
            cntscomp.append(pixel)

    separacion = 400  # Distancia entre cámaras
    focal = 35  # distancia focal de camaras en mm
    tampixel = 25.4/326 # mm/píxel (326 pixeles por pulgada)
    profundidades = []
    xcontorno = []
    ycontorno = []
    xCnts = []
    yCnts = []
    bucle = 13000
    SDA = []
    minimos = []
    sum = 0
    despl = []
    disparityCnts = []
    centrox = imagen.shape[1] // 2 #suponemos el mismo centro para ambas
fotos
    # img.shape[0]= height, img.shape[1]= width
    centroy = imagen.shape[0] // 2
    tam = 9 # tamaño celda SDA
    linf = mt.floor(tam / 2)
    lsup = mt.ceil(tam / 2)
    limites = 10 # límites de
    verticales = []
    pixelguardado = []
    SDApixeles = []
    bandera = 0
    xmedia = []
    ymedia = []

    for i, contour in enumerate(cntsizq):
        cv.drawContours(imagen, contour, -1, (0, 255, 0), 2)
        disparityCnts.append('Contorno ' + str(i))
        seleccion = contour[0:len(contour):8]
        for pixel in seleccion:
            fila = pixel[0, 1]
            columna = pixel[0, 0]
            # Los ´píxeles del contorno vienen como (columna,fila)
            if (linf - 1) < fila < (imagen.shape[0] - lsup) and\
                    (linf - 1) < columna < (imagen.shape[1] - lsup):
                for k in range(-limites, limites + 1):
                    if (linf - 1) < (fila + k) < (fondo.shape[0] - lsup):
                        for pixelcomp in cntscomp:
                            if (pixelcomp[0, 1] == (fila + k)
                                    and (linf - 1)<pixelcomp[0, 0]
                                    <(fondo.shape[1] - lsup)):
                                bandera = 1
                                V1 = fila -linf
                                V2 = fila +lsup
                                V3= columna - linf
```

```
                            V4= columna + lsup
                            V5= fila + k - linf
                            V6= fila + k + lsup
                            V7= (pixelcomp[0, 0] - linf)
                            V8= (pixelcomp[0, 0] + lsup)
                            resta = (np.array(grayizq[V1:V2, V3:V4],
                                    dtype='int64')
                                     - np.array(graydcha[V5:V6,V7:V8],
                                                dtype='int64'))
                            restabs = np.absolute(resta)
                            for element in restabs:
                                for num in element:
                                    sum = sum + num
                            if sum < bucle:
                                bucle = sum
                                pixelguardado = pixelcomp
                        sum = 0
                    if bandera == 0:
                        SDA.append([13000])
                        pixelguardado = []
                    else:
                        SDA.append([bucle])
                        bandera = 0
                        bucle = 13000
                else:
                    SDA.append([13000])
                    pixelguardado = []
                SDApixeles.append(pixelguardado)
            minSDA = min(SDA)
            vertical = SDA.index(minSDA) - limites
            pixelresultante = SDApixeles[SDA.index(minSDA)]
            if len(pixelresultante) != 0:
                indexcolumna = pixelresultante[0, 0]
                desplazamiento = columna - indexcolumna
                despl.append(desplazamiento)
                verticales.append(vertical)
            SDA = []
            SDApixeles = []
            minimos = []
    p = 0
    if len(verticales) != 0:
        kfinal = mode(verticales)
        for element in verticales:
            if element == kfinal:
                p = p + 1
    if p > 10 and len(despl) != 0:
        disparity = mode(despl)
        disparityCnts.append(disparity)
        for pixel in contour:
            fila = pixel[0, 1]
            columna = pixel[0, 0]
            if disparity > 0:
                indice = columna - disparity
                if 0 <= indice < fondo.shape[1]:
                    fondo[fila + kfinal, indice] = 0, 0, 255
                xreal = ((columna - centrox) * separacion)/disparity
                yreal = ((fila - centroy) * separacion)/disparity
                xcontorno.append(xreal)
                ycontorno.append(yreal)
        xCnts.append(xcontorno)
        yCnts.append(ycontorno)
        xcontorno=[]
```

```
            ycontorno = []

            depth = (separacion * focal) / (disparity*tampixel)
            profundidades.append('Contorno ' + str(i))
            profundidades.append(depth)
        else:
            disparityCnts.append('No correspondencia')

        despl = []
        verticales = []

    for i,contour in enumerate(xCnts):
        xmedia.append(np.mean(xCnts[i]))
        ymedia.append(np.mean(yCnts[i]))

    cv.imshow('B/R ratio normalizado camara izquierda', normalizeddivizq)
    plt.figure(1)
    plt.hist(divizq, color=colorizq)
    plt.show()
    cv.imshow('B/R ratio normalizado camara derecha', normalizeddivdcha)
    plt.figure(2)
    plt.hist(divdcha, color=colordcha)
    plt.show()
    cv.imshow('Contornos camara izquierda', imagen)
    cv.imshow('Contornos camara derecha', imgdcha)
    cv.imshow('Contornos localizados', fondo)

cv.waitKey(0)
cv.destroyAllWindows()
```

Código calculostd

```
import cv2 as cv
import numpy as np

def calculostd(img):
    b, g, r = np.double(cv.split(img))

    BRdif = b - r
    BRsum = b + r

    with np.errstate(divide='ignore', invalid='ignore'):
        div = BRdif / BRsum
        div = np.nan_to_num(div)

    out = np.zeros(div.shape, np.double)
    normalizeddiv = (np.uint8(cv.normalize(div, out,
                                           255.0, 0.0, cv.NORM_MINMAX)))

    color = []
    for i in range(div.shape[1]):
        color.append('blue')

    std = np.std(div, dtype='float64')

    return (std,div,normalizeddiv,color)

cv.waitKey(0)
```

Código calculocnts

```
import cv2 as cv
import numpy as np

def calculocnts(img,std):
    if std > 0.2 : # imagen bimodal

        img_gray = cv.cvtColor(img, cv.COLOR_BGR2GRAY)

        bilateral = (cv.bilateralFilter(img_gray, 3, 5,
                                        50, borderType=cv.BORDER_CONSTANT))

        canny = cv.Canny(bilateral, 15, 70)

        kernel = cv.getStructuringElement(cv.MORPH_ELLIPSE, (7, 7))
        dilated = cv.dilate(canny, kernel)

        cnts, _ = (cv.findContours(dilated.copy(), cv.RETR_EXTERNAL,
                                   cv.CHAIN_APPROX_NONE))
        print(f'{len(cnts)} contour(s) found!')

    else: # imagen unimodal

        b, g, r = np.double(cv.split(img))

        BRdif = b - r
        BRsum = b + r

        with np.errstate(divide='ignore', invalid='ignore'):
            div = BRdif / BRsum
            div = np.nan_to_num(div)

        out = np.zeros(div.shape, np.double)
        normalizeddiv = (np.uint8(cv.normalize(div,
                                               out, 255.0, 0.0,
                                               cv.NORM_MINMAX)))

        umbral=(normalizeddiv.mean() - 0.2*np.std(normalizeddiv,
                                                  dtype='float64'))

        if umbral>90:
            threshold, thresh = (cv.threshold(normalizeddiv,
                                              umbral, 255,
                                              cv.THRESH_BINARY))
        else:
            threshold, thresh = (cv.threshold(normalizeddiv,
                                              umbral, 255,
                                              cv.THRESH_BINARY_INV))

        kernel = cv.getStructuringElement(cv.MORPH_ELLIPSE,(7,7))
        dilated = cv.dilate(thresh, kernel)

        cnts, _ = (cv.findContours(dilated, cv.RETR_EXTERNAL,
                                   cv.CHAIN_APPROX_NONE))
        print(f'{len(cnts)} contour(s) found!')

    return(cnts)
```