MÁSTER EN INGENIERÍA DE SISTEMAS Y CONTROL

UNIVERSIDAD COMPLUTENSE DE MADRID
UNIVERSIDAD NACIONAL DE EDUCACIÓN A DISTANCIA

TRABAJO FIN DE MÁSTER

[CURSO 2021/2022]
Convocatoria de Septiembre

DETECCIÓN DE FALLOS DE IMPRESIÓN 3D POR MEDIO DE VISIÓN ARTIFICIAL

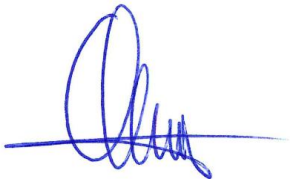
Autora: Sara Ortiz Martínez
Directores: Jacobo Sáenz Valiente
Jesús Chacón Sombría
Dictino Chaos García

- **Máster en Ingeniería de Sistemas y Control.**
- **Título:** Detección de fallos de impresión 3D por medio de visión artificial.
- **Tipo de proyecto:** Proyecto de tipo A. Proyecto específico propuesto por un profesor.
- **Autora:** Sara Ortiz Martínez
- **Directores:** Jacobo Sáenz Valiente, Jesús Chacón Sombría y Dictino Chaos García.

Autorización

Autorizamos a la Universidad Complutense y a la UNED a difundir y utilizar con fines académicos, no comerciales y mencionando expresamente a sus autores, tanto la memoria de este Trabajo Fin de Máster, como el código, la documentación y/o el prototipo desarrollado.

Firmado:



Firma del alumno

RESUMEN

La visión artificial es una disciplina que actualmente está presente en muchas tecnologías debido a que permite producir información numérica o simbólica de las imágenes del mundo real para que puedan ser tratados por un ordenador. Por eso, con la ayuda de la visión artificial es posible automatizar diversos procesos tanto en la industria como en las aplicaciones a nivel usuario.

Utilizando la visión artificial en el proceso de la impresión 3D, se consigue perfeccionar el proceso de impresión para que el resultado final tenga una mayor calidad, además también permite poder realizar la detección de defectos de impresión tanto en modo local como en remoto.

Este aumento de la calidad y optimización de la impresión es el objeto de interés del presente trabajo, por lo tanto, a través de la detección del contorno de los objetos de las imágenes, se realiza una comparación para determinar si existe o no fallo de impresión en cada capa de impresión y en su caso actuar adecuadamente.

LISTADO PALABRAS CLAVE: Impresión 3D, visión artificial, procesado de imágenes, automatización.

ABSTRACT

Computer vision is a discipline that is currently present in many technologies, due to the fact that it makes it possible to get numerical or symbolic information of the real images in order to be processed by a computer. With this, it is possible to analyse and understand images of the real world in order to achieve a better perception of the environment. With the help of computer vision it is possible to automate several processes both in industry and in user applications.

By using machine vision in the 3D printing process, it is possible to perfect the printing process so that the final result has a higher quality, and it also makes it possible to detect printing defects both locally and remotely.

This increase in the quality and optimization of the printing is the object of interest of this project, therefore, through the detection of the contour of the objects in the images, a comparison is made to determine whether or not there is a printing error in each printing layer.

KEYWORDS: 3D Printing, computer vision, Image processing, automation.

ÍNDICE

1. Introducción 11
1.1 Impresión 3D 11
2. Historia de la Impresión 3D 13
2.1.1 La impresora 3D 16
2.1.2 Problemas de impresión 17
2.2 Visión Artificial 20
3. Objetivo del Proyecto 22
4. Organización de la Memoria 22
5. Arquitectura del Sistema 24
6. Software 27
6.1 Slic3r 27
6.2 Octoprint 27
6.2.1 Octolapse 29
6.3 Python 29
6.3.1 Librerías 31
7. Fundamento del Procesado de las Imágenes 33
7.1 Modelo de Color HSV 33
7.2 Algoritmo Canny 38
7.3 Filtrado de la Imagen 40
8. Desarrollo del Algoritmo 46
8.1 Procesamiento de la Imagen 3D (Imagen de referencia) 46
8.1.1 Conversión de imagen 3D a 2D 46
8.1.2 Procesado del color de la imagen 47
8.1.3 Contorno del objeto 48
8.2 Procesamiento Imagen Real 51
8.2.1 Eliminación brillo de la imagen 51
8.2.2 Filtrado de la imagen 52
8.3 Comparación de las Imágenes 55
9. Validación del Algoritmo 57
10. Conclusiones y Trabajos Futuros 74
10.1 Conclusiones 74

10.2 Trabajos Futuros 75
11. Lista de Referencias y Bibliografía 76

ANNEXO 1 PROGRAMA PYTHON

FIGURAS

Figura 1 Esquemático modelado por deposición fundida [3] 14
Figura 2 Especificación Prusa i3. [5] 16
Figura 3 Defecto warping en impresión 3D. [6] 17
Figura 4 Defecto deslaminación en impresión 3D. [6] 17
Figura 5 Defecto desplazamiento de capas en impresión 3D. [6] 18
Figura 6 Defecto hilos en impresión 3D. [6] 18
Figura 7 Defecto sobrecalentamiento en impresión 3D. [6] 19
Figura 8 Diagrama de flujo 26
Figura 9 Conexión SSH 28
Figura 10 Octopi 29
Figura 11 Ranking mundial de lenguajes de programación más usados.[12] 30
Figura 12 Modelo de color HSV. [18] 33
Figura 13 Valor: +5. a) Imagen original. b) Imagen modelo de color HSV. c) Imagen en escala de grises desde RGB. d) Imagen en escala de grises desde HSV 35
Figura 14 Valor: +15. a) Imagen original. b) Imagen modelo de color HSV. c) Imagen en escala de grises desde RGB. d) Imagen en escala de grises desde HSV 36
Figura 15 Valor: +25. a) Imagen original. b) Imagen modelo de color HSV. c) Imagen en escala de grises desde RGB. d) Imagen en escala de grises desde HSV 37
Figura 16 Umbral histéreis. Algoritmo Canny. [22] 39
Figura 17 Kernel 3x3 filtro promedio 40
Figura 18 Ejemplo imagen filtro promedio. [19] 40
Figura 19 Filtrado promedio, kernel: a) 5x5. b) 7x7. c) 9x9. d) 11x11. e) 13x13 41
Figura 20 Contorno Canny a imagen con filtro promedio, kernel: a) 5x5. b) 7x7. c) 9x9. d) 11x11. e) 13x13 41
Figura 21 Ejemplo imagen filtro mediana. [19] 42

Figura 22 Filtrado mediana, kernel: a) 5x5. b) 7x7. c) 9x9. d) 11x11. e) 13x13 42
Figura 23 Contorno Canny a imagen con filtro mediano kernel: a) 5x5. b) 7x7. c) 9x9. d) 11x11. e) 13x13 42
Figura 24 Campana de Gauss 43
Figura 25 Ejemplo filtro gaussiano. [19] 43
Figura 26 Filtrado promedio kernel 11x11 + gaussiano, kernel: a) 5x5. b) 7x7. c) 9x9. d) 11x11 44
Figura 27 Contorno Canny a imagen con filtro promedio kernel 11x11 + Gaussian, Kernel: a) 5x5. b) 7x7. c) 9x9. d) 11x11 44
Figura 28 Filtrado mediana kernel 11x11 + gaussiano, kernel: a) 5x5. b) 7x7. c) 9x9. d) 11x11 44
Figura 29 Contorno Canny a imagen con filtro mediana kernel 11x11 + gaussiano, kernel: a) 5x5. b) 7x7. c) 9x9. d) 11x11 45
Figura 30 Detección zona de interés en imagen con filtro mediana kernel 11x11 + gaussiano, kernel: a) 5x5. b) 7x7. c) 9x9. d) 11x11 45
Figura 31 Imagen 3D (Slic3r) 46
Figura 32 Imagen referencia (2D) 47
Figura 33 Tratamiento color imagen referencia a) original, b) escala de grises, c) binaria 48
Figura 34 Detección de contorno en imagen referencia. 49
Figura 35 Rectángulo en área de interés 50
Figura 36 Imagen de referencia final 50
Figura 37 a) imagen real inicial. b) Imagen en espacio de color HSV 51
Figura 38 a) imagen real inicial. b) Imagen en espacio de color HSV. c) Imagen en escala de grises 52
Figura 39 Imagen con filtro mediana 52
Figura 40 Imagen con filtro mediana y gauss 53
Figura 41 Imagen con algoritmo Canny 53
Figura 42 a) Imagen real contorno. b) Rectángulo área de interés en imagen real 54
Figura 43 Imagen real final 54
Figura 44 Imagen comparada 56

Figura 45 a) Imagen original. b) Imagen HSV. c) Imagen en escala de grises 57
Figura 46 a) Filtro mediana. B) filtro mediana + gauss 57
Figura 47 Canny 58
Figura 48 a) Rectángulo área interés. b)Imagen escalada 58
Figura 49 Imagen comparada 59
Figura 50 a) Imagen original. b) Imagen HSV. c) Imagen en escala de grises 60
Figura 51 a) Filtro mediana. B) filtro mediana + gauss 60
Figura 52 Canny 61
Figura 53 a) Rectángulo área interés. b) Imagen escalada 61
Figura 54 Imagen comparada 62
Figura 55 a) Imagen original. b) Imagen HSV. c) Imagen en escala de grises 63
Figura 56 a) Filtro mediana. b) filtro mediana + gauss 63
Figura 57 Canny 64
Figura 58 a) Rectángulo área interés. b) Imagen escalada 64
Figura 59 Imagen comparada 65
Figura 60 a) Imagen original. b) Imagen HSV. c) Imagen en escala de grises 66
Figura 61 a) Filtro mediana. B) filtro mediana + gauss 66
Figura 62 Canny 67
Figura 63 a) Rectángulo área interés. b) Imagen escalada 67
Figura 64 Imagen comparada 68
Figura 65 Objeto contornos complejos_procesado color. a) Referencia, b) Imagen en escala de grises y c) imagen en binario 69
Figura 66 a) Rectángulo área interés. b) Imagen escalada final 69
Figura 67 a) Imagen original. b) Imagen HSV. c) Imagen en escala de grises 70
Figura 68 a) Filtro mediana. b) filtro mediana + gauss 70
Figura 69 Canny 71
Figura 70 a) Rectángulo área interés. b) Imagen escalada 71
Figura 71 Imagen comparada 72

TABLAS

Tabla 1 Resumen resultados defectos analizados 73

1. Introducción

1.1 Impresión 3D

La impresión 3D es una tecnología relativamente joven y aunque ya está implementada en muchos sectores, en especial en la industria, todavía queda mucho por desarrollar y explotar en cuanto a esta tecnología se refiere**.** La impresión 3D consiste en la fabricación de objetos tridimensionales mediante la superposición aditiva de capas. Dicha impresión se realiza a partir de datos de diseño asistido por ordenador (CAD). Por lo tanto, esta tecnología permite mucha versatilidad ya que una misma impresora puede fabricar infinidad de productos distintos de diferentes materiales. Permite también gran flexibilidad y prototipado rápido lo que supone una gran ventaja para la industria, siendo la creación rápida de prototipos un grupo de técnicas diversas que se utilizan para fabricar un modelo físico a escala en el menor tiempo posible utilizando datos de un modelo digital [1]. El uso de prototipos puede emplearse para la validación de algunos requisitos del producto en etapas tempranas, esto da lugar a que los posibles errores se puedan detectar al principio del proceso y se puedan solventar antes del desarrollo del producto final. Por lo tanto, al finalizar el proceso, se obtiene un importante ahorro en coste y tiempo.

A día de hoy, son múltiples los sectores como la aeronáutica, la automoción, la gastronomía, la ciencia y la sanidad o el mundo del entretenimiento los que han implementado el uso de la impresión 3D como un proceso que forma parte de su día a día. Por otro lado, debido a que el coste de la impresora 3D es cada vez más competitivo ha hecho que sea accesible también a nivel usuario y que muchas personas tengan la oportunidad de disponer de una impresora 3D en su hogar. Además, este amplio desarrollo también ha sido posible debido a que existen diversas herramientas *software* de código abierto que hacen sencillo el control y la monitorización de la impresión 3D. Sin embargo, la impresión 3D también presenta algunas desventajas como las que se muestran en el capítulo 2.1.2, por lo tanto, el

trabajo de la visión artificial para la detección automática de defectos es crucial para aumentar la calidad del producto fabricado en impresión 3D a la mayor brevedad posible.

2. Historia de la Impresión 3D

A continuación se resumen algunos de los hitos más importantes de la historia de la impresión 3D:
La primera patente relacionada con la tecnología fue presentada en mayo de 1980 por el Dr. Hideo Kodama, del Instituto Municipal de Investigación Industrial de Nagoya, que describe un sistema de creación rápida de prototipos de fotopolímero. La idea consistía en utilizar material de fotopolímero, expuesto a la luz UV, para endurecer una pieza. Lamentablemente, nunca consiguió los fondos para hacerla realidad, pero fue un primer paso para la historia de la impresión 3D [1].

En 1984, Chuck Hill presentó una solicitud de patente sobre un método de creación de prototipos basado en la estereolitografía (patente concedida en 1986). Más tarde creó la empresa 3D Systems, que se convirtió en la más importante del sector y vendió las primeras impresoras 3D de estereolitografía en el mercado en el año 1988. También creó el nuevo formato de archivo (STL: estereolitografía) que proporciona la descripción de la geometría de la superficie de los objetos [1].

En 1989 S. Scott Crump, junto con su esposa e inventora Lisa Crump, inventa y patenta un nuevo método de fabricación aditivo llamado modelado de deposición fundida. [2]. El modelado por deposición fundida es una tecnología de fabricación aditiva que construye piezas calentando y extruyendo filamentos a través de una pequeña boquilla. Este método de fabricación suele trabajar con termoplásticos o materiales compuestos. La boquilla sigue trayectorias controladas por ordenador, como en las máquinas de control numérico por ordenador (CNC), mientras extruye el material para dibujar capas una encima de otra y crear la pieza [3]. En la siguiente Figura 1 se observa un ejemplo de las principales partes que componen este tipo de impresoras:

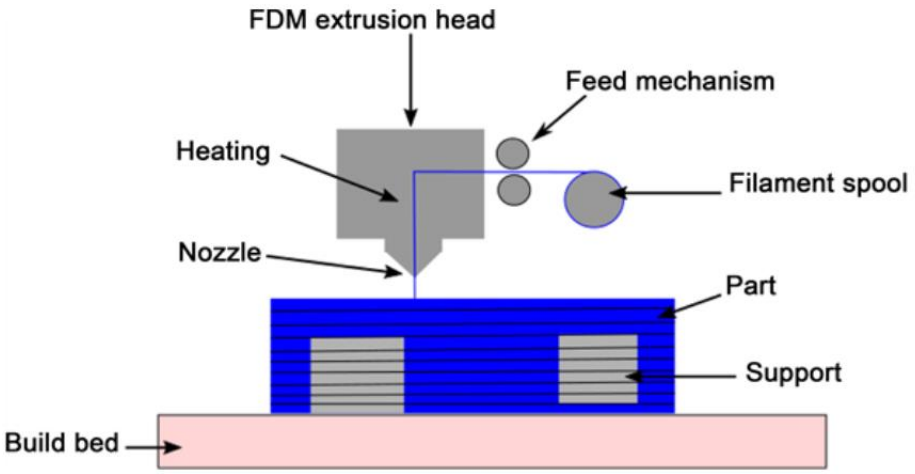


Figura 1 Esquemático modelado por deposición fundida [3]

Sin embargo, no fue hasta 1992 cuando se comercializaron las primeras impresoras de impresión 3D del tipo SLA (estereolitográfico). Estas presentaban algún defecto e imperfecciones, pero eran capaces de llevar a cabo la fabricación de objetos finales capa por capa, haciendo el rayo ultravioleta solido el fotopolímero y construyendo de esta forma el objeto en cuestión [2].

Fue en 1999 cuando la impresión 3D se comenzó a utilizar en el campo de la medicina, se consiguió usar en un recubrimiento sintético a partir de las células del paciente en cuestión y con esto dar solución a determinados órganos. Éste fue un momento clave en la historia de la impresión 3D. [4]
En 2005 el Dr. Adrian Bowyer funda RepRap, en la Universidad de Bath, una iniciativa de código abierto para construir una impresora 3D que tiene las técnicas para reproducirse a sí misma. La visión de este proyecto es el de democratizar la fabricación de unidades de distribución de bajo coste RepRap a las personas de todo el mundo [1].

En 2006 se construye la primera máquina del tipo SLS (Sintetización de Láser Selectivo) viable. Básicamente, este tipo de máquina utiliza un láser para fundir materiales en el proceso de impresión 3D. Este descubrimiento abre las puertas a la personalización masiva y a la demanda de fabricación de piezas industriales, y más tarde, prótesis.

Ese mismo año, Object, un proveedor de materiales e impresoras 3D, crea una máquina con la capacidad de imprimir en múltiples materiales, incluyendo polímeros y elastómeros. La máquina permite que una parte sea fabricada con una gran variedad de densidades y propiedades de material [2].

Sobre la base del éxito del sueño RepRap, en 2009 Makerbot crea kits para montar tu propia impresora [4].

Desde 2012 el crecimiento de la industria 3D ha aumentado desproporcionadamente año con año. Cada vez más marcas presentan en el mercado nuevas soluciones de impresión 3D que permiten fabricar piezas no sólo en plásticos si no en resinas, metales y materiales orgánicos.

Sin duda, la impresión 3D y manufactura aditiva gana más terreno cada día y seguirá escribiendo una misma historia: Transformar las cadenas de valor y cómo fabricamos cada día.

2.1.1 La impresora 3D

La impresora 3D utilizada para este proyecto es el modelo Prusa i3 MK3S+ con un espacio de trabajo 200x200x200 mm. Este modelo de impresora entre otras características, posee una base térmica con láminas de acero desmontables, nivelación automática de la cama mediante malla, sensor de filamento, recuperación de pérdida de energía (modo pánico) y características de seguridad [5]. En la Figura 2 se resumen las características técnicas más importantes de la impresora:

Volumen de Impresión	25×21×21 cm (9.84"×8.3"×8.3")
Altura de capa	0.05 - 0.35 mm
Nozzle	0.4mm por defecto, amplia gama de otros diámetros/boquillas compatibles
Diámetro del filamento	1.75 mm
Materiales compatibles	Amplia gama de termoplásticos, incluyendo PLA, PETG, ASA, ABS, PC (Policarbonato), CPE, PVA/BVOH, PVB, HIPS, PP (Polipropileno), Flex, nGen, Nylon, Carbon filled, Woodfill y otros materiales rellenos.
Velocidad máxima de desplazamiento	200+ mm/s
Temperatura máxima de la boquilla	300 °C / 572 °F
Temperatura máxima de la base	120 °C / 248 °F
Extrusor	Accionamiento directo, engranajes Bondtech, hotend E3D V6
Superficie de impresión	Láminas de acero magnético desmontables(*) con diferentes acabados de superficie, base calefactable con compensación de esquinas frías
Dimensiones de la impresora (sin bobina)	7 kg, 50×55×40 cm; 19.6×21.6×15.7 in (X×Y×Z)
Consumo de energía	Ajustes PLA: 80W / Ajustes ABS: 120W

Figura 2 Especificación Prusa i3. [5]

2.1.2 Problemas de impresión

Existen multitud de errores que pueden ocurrir durante la impresión 3D y que pueden dar lugar a un resultado de impresión no deseado. Los defectos más frecuentes son los que se detallan a continuación, éstos han sido extraídos de la información incluida en [6]:

Pandeo o *Warping*: Se puede producir por dos factores. Por falta de adherencia a la cama. Esto puede ocurrir por una altura incorrecta de la boquilla respecto de la base o un mal nivelado, pero también por falta de adherencia con la superficie de impresión. El segundo factor puede ser la contracción térmica e las capas superiores. Al encogerse estas capas, crean una tensión en las capas inferiores y hace que se pierda la adherencia a la cama.

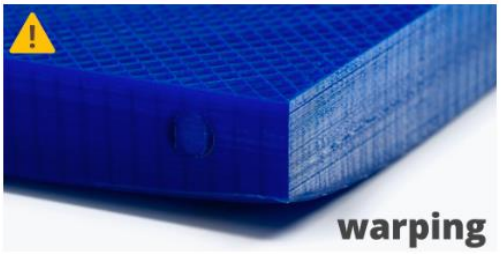


Figura 3 Defecto warping en impresión 3D. [6]

Deslaminación o *cracking*: Este defecto se presenta cuando las capas de una pieza no están bien unidas. Esto puede ocurrir por una mala adherencia entre capas o que haya una buena adherencia entre capas pero que debido a una diferencia de temperatura entre diferentes partes de la pieza, hace que se deforme hasta tal punto, que afecte a la separación entre capas.

Figura 4 Defecto deslaminación en impresión 3D. [6]

Desplazamiento de capas: Desplazamientos horizontales de la pieza, este defecto ocurre cuando existe desplazamiento de la pieza en algún eje. Puede ocurrir en el eje X, en el Y o incluso en ambos de manera simultánea. Las causas pueden ser un exceso de temperatura de los motores, falta de potencia de los motores o un fallo mecánico del mecanismo de la impresora.

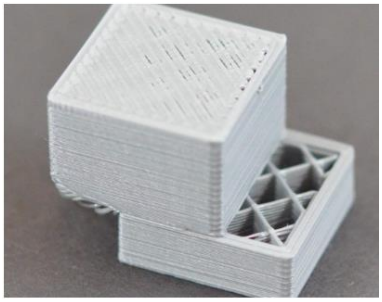


Figura 5 Defecto desplazamiento de capas en impresión 3D. [6]

Hilos o *stringing*: Este defecto de impresión ocurre cuando al moverse el fusor entre dos piezas de la impresión, gotea una pequeña cantidad de plástico que al solidificarse se adhiere a ambas partes dejando un pequeño hilo entre ellas. El problema puede ser debido a una alta temperatura de fundido, a una incorrecta distancia de retracción o a una baja velocidad de movimiento del fusor.

Figura 6 Defecto hilos en impresión 3D. [6]

Sobrecalientamiento: Este defecto ocurre cuando se funde el plástico a una temperatura excesiva, lo que provocará deformación en las distintas capas. Esto se debe a que al fundirse a una temperatura demasiado alta, el material no tiene tiempo a enfriarse y es por esto que las capas pueden quedar caídas o hundidas.

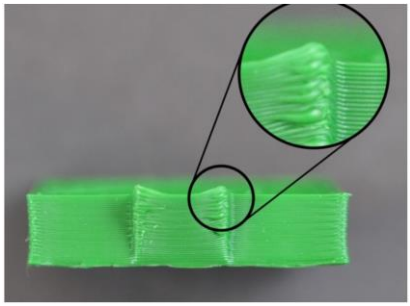


Figura 7 Defecto sobrecalentamiento en impresión 3D. [6]

2.2 Visión Artificial

La visión artificial permite que los ordenadores y los sistemas obtengan información de imágenes digitales, videos y otras entradas visuales. La visión artificial funciona de manera muy similar a la visión humana, excepto que los humanos tienen una ventaja, la experiencia y los contextos aprendidos para diferenciar entre los objetos, la distancia entre ellos, si son dinámicos o estáticos, o si hay algo mal en una imagen. El objetivo de la visión artificial es entrenar a las máquinas para que sean capaces de realizar estas funciones. La visión artificial se utiliza en diversas industrias como son la industria energética, automotriz, fabricación [7].

La Industria 4.0 está marcada por procesos de automatización y digitalización muy desarrollados y por el uso de la electrónica y las tecnologías de la información (TI) en la fabricación y los servicios. La integración y el análisis de datos masivos malintencionados en tiempo real optimizarán los recursos en el proceso de fabricación y lograrán un mejor rendimiento. La computación móvil, la computación en la nube, *big data* y el *IoT* (*Internet of things*) son las tecnologías clave de la Industria 4.0 [8].

La visión artificial es una de las tecnologías clave a la hora de llevar a cabo una estrategia de digitalización industrial ya que en los últimos años ha permitido incrementar mecanismos dentro del control de calidad, como ejemplos: detección de defectos, verificación de montajes, trazabilidad, metrología, lectura de códigos, etc.

El origen como tal de la visión artificial industrial se remonta a los años 60 y al origen de un prototipo automatizado basado en cámaras de visión y sistemas de procesamiento de las imágenes captadas. Con ello se buscaba la forma de tener acceso a cómo eran ciertas estructuras y al análisis de su contenido, a través del procesamiento de imágenes con ayuda de ordenadores y *software*.

La historia de la visión artificial marca su hito en la década de los 80 con el desarrollo de la ingeniería informática y la creación de procesadores más sofisticados y rápidos, dando lugar a microprocesadores capaces de captar, procesar y reproducir imágenes tomadas por una cámara a la que podían estar conectada de forma remota.

Es aquí donde se comienzan a explorar cómo los sistemas de visión artificial pueden capturar imágenes de forma automatizada y reproducir las características visuales de objetos o espacios a través de la interpretación de los datos que puede realizar un *software* destinado a esta labor [9].

Como se ha mencionado anteriormente, la visión artificial supone un gran avance en el sector industrial ya que permite mejorar los procesos de producción y garantizar una mayor calidad. En el presente proyecto, se analiza la detección de fallos de piezas impresas en 3D a través de la visión artificial.

3. Objetivo del Proyecto

El objetivo principal del presente proyecto consiste en automatizar la detección de defectos de impresión 3D para conseguir optimizar los procesos de impresión y detectar a tiempo posibles errores mecánicos o de funcionalidad de la impresora para evitar seguir imprimiendo cuando existan fallos, con el derroche de material y energía que eso supone, así como posibles daños en la impresora debidos a atascos o colisiones con la pieza.

Por tanto, este objetivo principal se descompone en los siguientes objetivos secundarios:

- Binarización de la imagen de referencia (modelo digital).
- Procesado y filtrado de la imagen del objeto obtenida al finalizar cada capa de impresión para minimizar la influencia de brillo y ruido ocasionados por agentes externos.
- Evaluación del resultado de la impresión de cada capa del objeto para determinar si existen errores de impresión.

4. Organización de la Memoria

Para presentar el trabajo realizado, se ha dividido la memoria en 5 partes principales:

1. En la primera parte, se introducen los conceptos de impresión 3D y visión artificial, así como la ventaja que se genera cuando se combinan ambas tecnologías y algunas de las desventajas. Ambos conceptos son clave para el desarrollo del proyecto. También se describen los objetivos del presente trabajo. (Apartados: 1 y 3).
2. En la segunda parte, se presenta la arquitectura del sistema a alto nivel, con el fin de obtener una visión general del desarrollo del algoritmo. (Apartado: 5)
3. En la tercera parte del proyecto se presenta el *software*, *hardware* y las bibliotecas que se han utilizado y se desarrollan los fundamentos

teóricos de los distintos algoritmos y modelos utilizados para el procesamiento de las imágenes. (Apartados: 6 y 7).

4. En la cuarta parte del proyecto se describe en profundidad el desarrollo del algoritmo y la validación de dicho algoritmo con diversos ejemplos. (Apartados: 8 y 9).
5. Por último, se presentan las conclusiones y trabajos futuros. (Apartado: 10).

5. Arquitectura del Sistema

En este apartado se detalla la arquitectura del sistema a alto nivel y se enumeran los pasos realizados en la lógica del programa:

1. Activar *Octolapse* desde *Octoprint* (ver apartado 6.2).
2. Inicio de la impresión.
    - Se exporta el código '*Gcode*' del objeto desde Slic3r a *Octoprint* y se añade el siguiente código '*Gcode'* al finalizar cada capa para que la impresión se detenga al finalizar cada capa.

```
M401      ; Almacena posición actual
G0 X0 Y0  ; Mueve cabezal a coordenadas X=0 Y=0
M25       ; Detiene la impression.
```

3. Preguntar al usuario nº capas totales del objeto a imprimir.
4. Bucle *for* para ir comparando capas sucesivamente, hasta que nº capa actual sea igual a nº capas totales o hasta que se detecte un fallo.
    - Procesamiento imagen 3D (referencia). (Ver apartado 8.1).
        - Conversión imagen 3D a 2D.
        - Binarización imagen de referencia.
        - Detección de contornos.
        - Detección de objeto de interés.
        - Almacenar imagen referencia final (*Imagen final 1*, *ver* Figura 8).
    - Conexión SSH para acceder a las imágenes realizadas por *Octolopase* al finalizar cada capa. Estas imágenes se almacenan en la SD de la Raspberry pi.
    - Procesamiento imagen real. (Ver apartado 8.2).
        - Eliminación de brillo de la imagen mediante función HSV y transformación imagen en escala de grises.
        - Filtrado imagen.
        - Detección de contornos.
        - Detección de objeto de interés

- Almacenar imagen real final (*Imagen final 2, ver* Figura 8).

- Comparar ambas imágenes (ver apartado 8.3):
  - Se calcula el número total de píxeles del contorno de la imagen de referencia.
  - Se calcula el número de píxeles del contorno de la imagen comparada.

    - Si no hay fallo → Se reanuda la impresión.

    - Si hay fallo → *plugin stop printing*. Se detiene programa y se detiene impresión. Se muestra error por pantalla.

A continuación se muestra más detalle en el diagrama de flujo del programa:

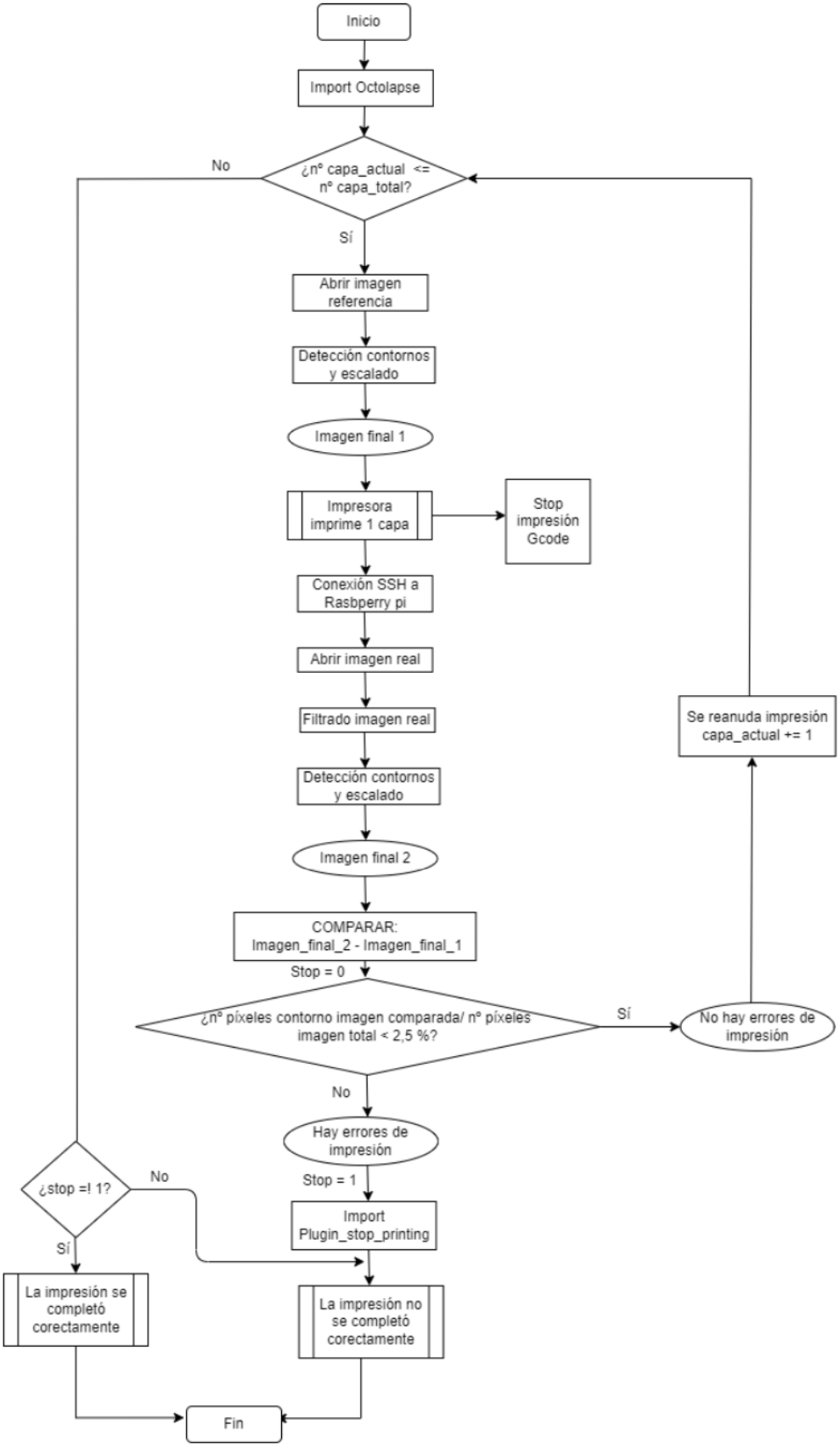


Figura 8 Diagrama de flujo

6. Software

Para la realización de este trabajo, se han utilizado distintas herramientas *software*, las cuales se describen a continuación:

6.1 Slic3r

Slic3r es una herramienta *software* que se desarrolló en 2011 como resultado de un proyecto comunitario de código abierto y sin ánimo de lucro. Esta herramienta es utilizada para convertir un modelo 3D en instrucciones para la impresora 3D. El procedimiento consiste en dividir el objeto a imprimir en capas horizontales calculando los datos que la impresora 3D necesita como trayectorias y cantidad de material necesario. En relación al proyecto, se utiliza *Slic3r* para la visualización de los objetos 3D de referencia [10].

6.2 Octoprint

Es una aplicación de código abierto y gratuita que permite el control de impresoras 3D. Permite controlar las impresiones 3D de manera remota. Comúnmente se suele instalar en una Rasberry Pi como es en el caso de este proyecto. *Octoprint* ofrece también la posibilidad de desarrollar *Plugins* libres y accesibles por el resto de usuario para mejorar el control de la impresora 3D y ampliar características de la impresora como visión, seguridad, etc. [11].

El objetivo del proyecto es detectar fallos de impresión, por lo tanto, al finalizar cada capa, aparecerá por pantalla en la impresora si hay o no errores de impresión. Para ello, se ha creado un plugin en Python que mandará el mensaje a la impresora. Esto se detallará más adelante en el apartado 8.3.

Para la instalación de *Octoprint* en la Rasberry pi es necesario descargarse la imagen preconfigurada de *Octoprint* (llamada *Octopi)* en la tarjeta SD de la

Raspberry pi. Una vez descargado, se inserta la tarjeta SD en la Raspberry pi y se accede a ella a través de conexión SSH, añadiendo como *Host Name*: 'Octopi'.

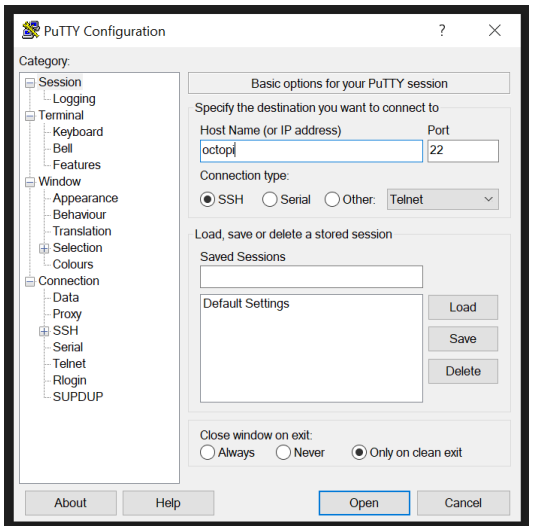


Figura 9 Conexión SSH

A continuación se pide los siguientes datos:

- Login as: pi
- pi@octopi's password: raspberry

En cuanto se establece la conexión, es posible acceder a *Octopi. Octopi* es una imagen preconfigurada para la Raspberry Pi que contiene la aplicación Octoprint. a través de la web oficial: http://octopi desde donde tenemos acceso al control en remoto de nuestra impresora 3D, creación de *plugins*, visión del código de impresión, etc.:

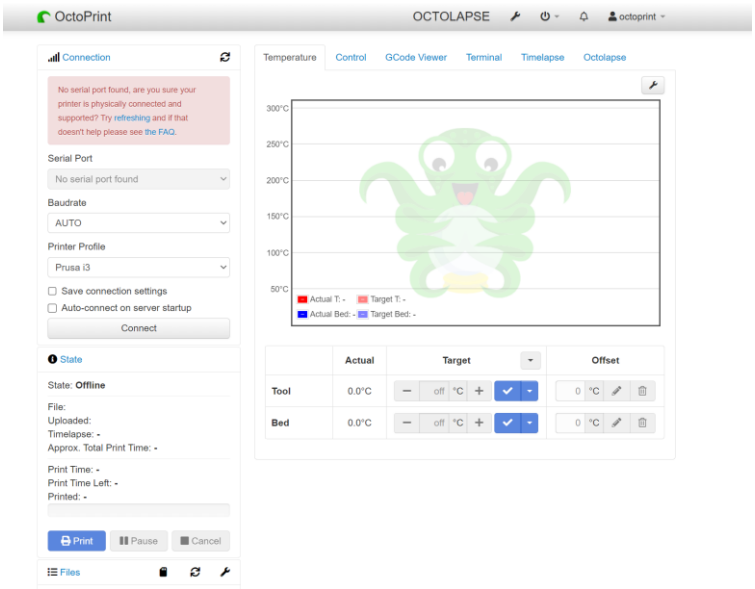


Figura 10 Octopi

6.2.1 Octolapse

Octolapse es uno de los *plugin* más usados en *Octoprint*. Consiste en un plugin mediante el cual se realiza una fotografía al objeto de impresión al finalizar cada capa. Esto lo utilizaremos durante la realización del proyecto para ir comparando estas imágenes obtenidas al finalizar cada capa de impresión con la imagen de referencia.

6.3 Python

Es un lenguaje de alto nivel de programación orientado a objetos. Ofrece una semántica dinámica integrada. Fue creado a finales de 1980 por Guido Van Rossum en el centro para las matemáticas y la informática en los Paises Bajos [13]. Las principales ventajas de Python son que es un código abierto y que debido a su clara sintaxis y legitiblidad hacen que se cree un código fácil de leer y comprender. Esto ha facilitado que el uso de Python ha ido ganando relevancia con el paso de los años. Actualmente y según un estudio estadístico publicado en 2022 por *Stack Overflow,* [12], Python se sitúa en el 4º puesto de lenguajes de programación más usados a nivel mundial. En este estudio han participado 71.547 personas, de las cuales un 53.421 son desarrolladores profesionales y 6.239 son usuarios en periodo de aprendizaje. Este es el resultado de la encuesta ordenado

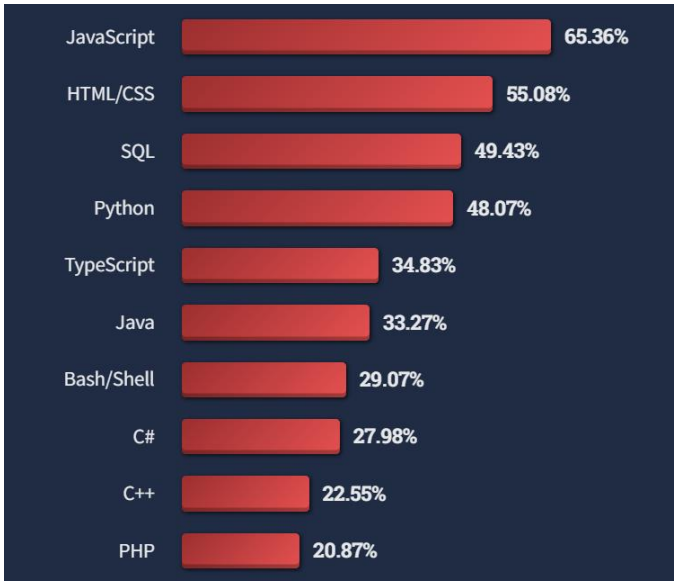


Figura 11 Ranking mundial de lenguajes de programación más usados. [12]

Python ha sido el lenguaje escogido para el desarrollo del presente proyecto debido a las ventajas anteriormente mencionadas.

6.3.1 Librerías

En Python existen infinidad de librerías de código abierto que pueden ser usadas. Las librerías utilizadas para la realización del proyecto han sido las siguientes:

Matplotlib: Librería especializada en la creación de gráficos a partir de datos contenidos en listas o *arrays* en dos dimensiones. Proporciona una API orientada a objetos para la creación de gráficos en aplicaciones que utilizan interfaz gráfica de usuario de propósito general. Fue creada por John D. Hunter en el año 2003 [14]. Esta libería se ha utilizado durante el desarrollo del código del presente proyecto para poder trabajar con la imagen 3D de referencia y transformarla en imagen 2D para el posterior tratamiento de imagen.

OpenCV: OpenCV (Open Computer Vision, Visión artificial Abierta). Es la libería por excelencia para el desarrollo de algoritmos de visión artificial y machin learning. Es una libería de código abierto que incluye varios cientos de algoritmos de visión por ordenador. En el desarrollo del proyecto se han utlizado diversas funciones de esta libería. Fue creada por Intel Corporation en en el año 2000 [15]. Esta librería se ha utilizado en el presente proyecto para el tratado de las imágenes (filtrado, cambio modelo de color) así como para detectar los contornos de las imágenes.

Numpy: Numpy (Numerical Python) Es una librería especializada en el cálculo numérico y el análisis de datos, especialmente para un gran volumen de datos. Es una librería de cogido abierto que realiza procesamiento con datos contenidos en vectores (arrays) multidimensionales y contiene una serie de rutinas para procesar esos arrays. Fue creada por Travis Oliphant en el año 2005 [16]. Mediante esta librería se ha calculado el número de puntos finales de la imagen comparada para determinar si hay o no defectos en la impresión 3D.

SSH Paramiko: Librería utilizada para el acceso al directorio de Raspberry pi mediante conexión SSH. Esta conexión es utilizada para poder abrir las imágenes realizadas por la impresora que son almacenadas en la tarjeta SD de la Raspberry pi.

7. Fundamento del Procesado de las Imágenes

Para el procesamiento de las imágenes reales tomadas por la impresora 3D ha sido necesario emplear diferentes procesos ya que las imágenes presentan ruido y brillo. En este apartado se muestra la investigación que ha sido realizada para llegar a la resolución final descrita en el apartado 8.

7.1 Modelo de Color HSV

El modelo de color HSV se define por los 3 parámetros que conforman el nombre, tono o matiz (Hue), Saturación (Saturation) y luminosidad o valor (Value) [17]. A continuación se definen cada una de ellas:

El tono o matiz (H) se define como un ángulo en el círculo cromático, por lo tanto, puede alcanzar valores entre 0° y 360°. 0° corresponde al color rojo, 120° corresponde al color verde y 240° corresponde al color azul.

La saturación (S) se declara como porcentajes y, por lo tanto, puede alcanzar valores entre 0% y 100% (o 0 a 1). Una saturación del 100% significa un color completamente saturado y puro, cuanto menor es la saturación, más se vuelve el color a un gris neutro.

El valor de luminosidad (V) también se da como un porcentaje, donde 0% significa que no hay brillo (por lo tanto, negro) y 100% de brillo total, por lo tanto, un espectro entre el color puro (saturación del 100%) y el blanco (saturación de 0 %)

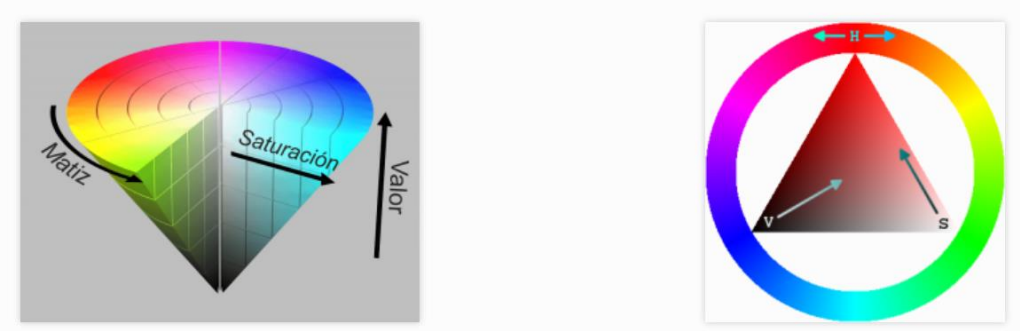


Figura 12 Modelo de color HSV. [18]

Con el ajuste de estos tres parámetros es posible seleccionar el color en función de la tonalidad, la saturación, así como controlar el brillo de cada píxel de la imagen. Esta última función, es la que se ha utilizado para la realización del proyecto. El motivo de escoger este modelo de color para el tratamiento de las imágenes reales era poder tener la capacidad de eliminar el brillo de las imágenes y así poder hacer una correcta detección de contornos. Por lo tanto, en este caso particular, se ha trabajado con la variación del valor de “V”.

Para determinar cuál sería el valor óptimo que habría que sumar a los píxeles de la imagen original para conseguir la eliminación del brillo, se han hecho diversas pruebas con distintos valores de “V” hasta determinar cuál es el valor correcto que nos permite eliminar el brillo de la imagen, sin generar ruido adicional o pérdida de información de la imagen.

Para ello, se han probado diferentes imágenes con distintos valores de incremento del valor “V”, a continuación se muestra un ejemplo con 3 valores distintos de “V”. En las siguientes imágenes se muestra la diferencia de convertir la imagen original (a) en escala de grises pasando directamente desde modelo RGB (c) o convertir la imagen original pasando al modelo de color HSV (b) para llegar a la escala de grises habiendo modificado el valor de “V” (d).

Se muestran 3 ejemplos con diferentes valores de “V”:

- Incrementando el valor de V : +5:

Como se puede observar en la imagen Figura 13, si se convierte directamente la imagen original, Figura 13 a) en escala de grises, Figura 13 c), ésta presenta mucho brillo que luego va a imposibiltar la detección de contornos. Se obseva que el restultado en escala de grises es mejor si se incrementa el valor de “V”, ver Figura 13 d). Sin embargo, incrementando el valor en 5 no es suficiente para eliminar de manera total el brillo de la imagen. Sería necesario incrementarlo un poco más, para conseguir una elimación de brillo más completa.

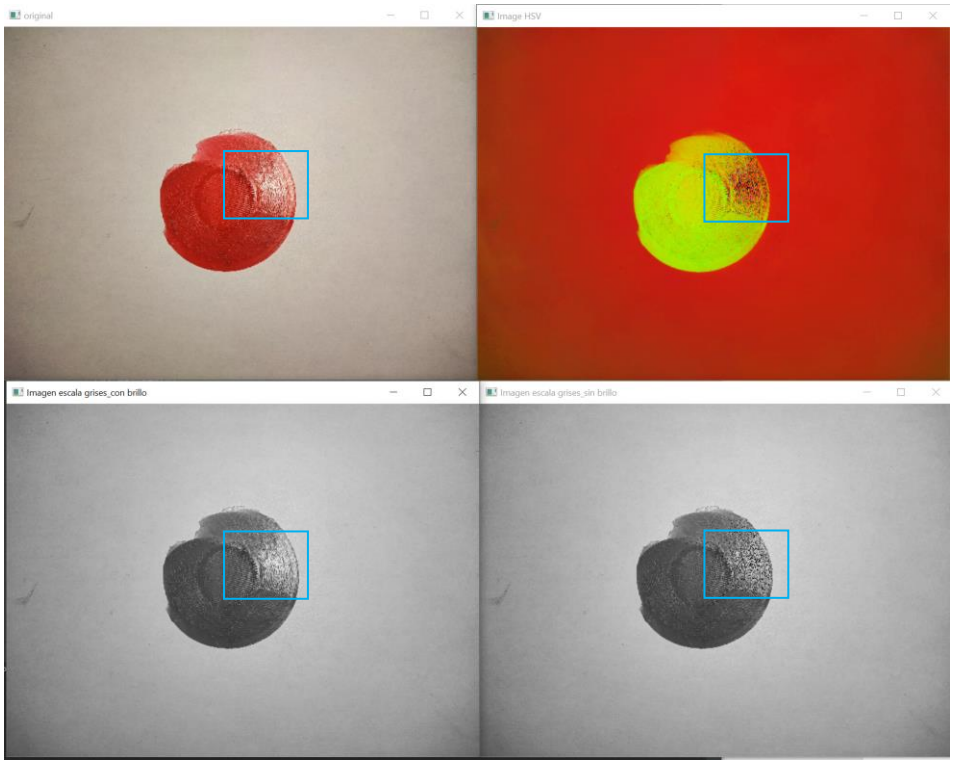


Figura 13 Valor: +5. a) Imagen original. b) Imagen modelo de color HSV. c) Imagen en escala de grises desde RGB. d) Imagen en escala de grises desde HSV

- Incrementando el valor de V : +15:

En este caso se observa que se consigue una eliminación del brillo de la imagen practicamente total ver Figura 14 d), sin añadir ruido adicional que pueda ocasionar problemas para la detección de contorno:

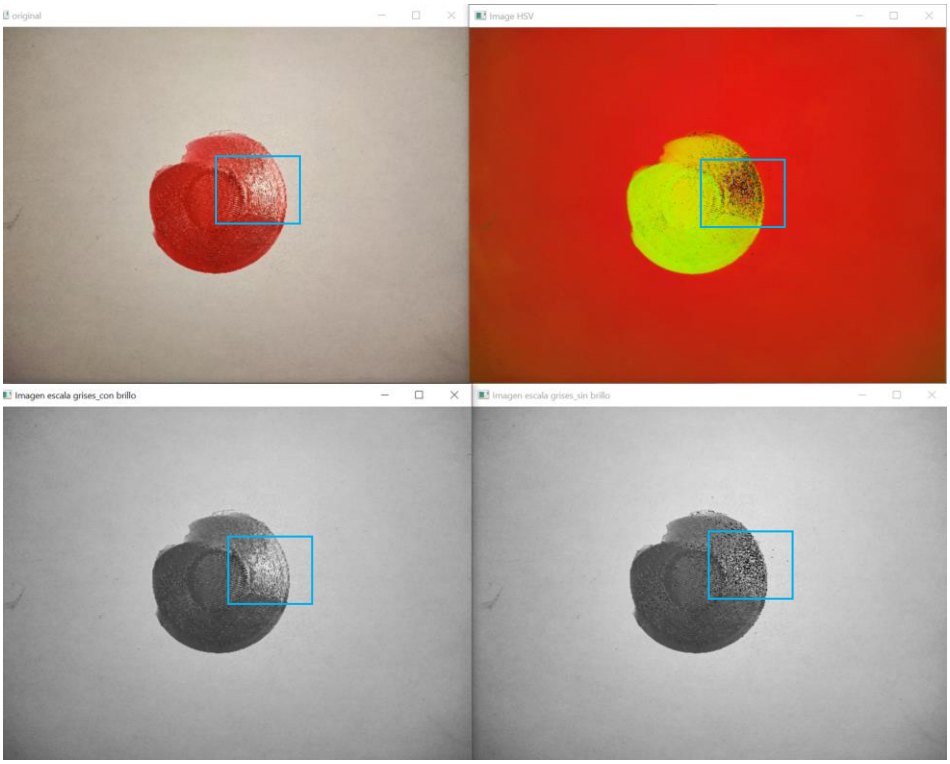


Figura 14 Valor: +15. a) Imagen original. b) Imagen modelo de color HSV. c) Imagen en escala de grises desde RGB. d) Imagen en escala de grises desde HSV

- Incrementando el valor de V : +25:

En este último ejemplo se puede apreciar que además de eliminar el brillo de la imagen, se genera ruido externo en el contorno no deseado como se muestra en la Figura 15 d:

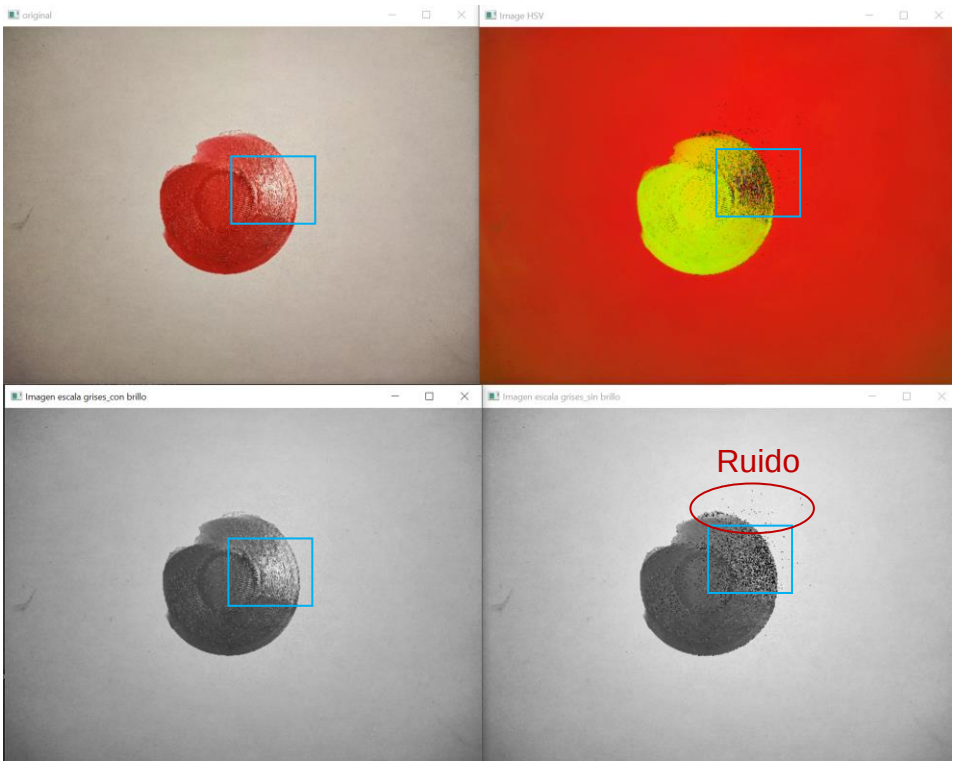


Figura 15 Valor: +25. a) Imagen original. b) Imagen modelo de color HSV. c) Imagen en escala de grises desde RGB. d) Imagen en escala de grises desde HSV

El estudio realizado para obtener el valor adecuado de “V” ha sido llevado a cabo probando más valores. Sin embargo en este apartado se muestran 3 ejemplos representativos para explicar cómo afecta la variación de este valor. Finalmente se concluye que el valor óptimo para eliminar el brillo no deseado de la imagen sin añadir ruido adicional es aumentar 15 al valor de “V”.

7.2 Algoritmo Canny

El algoritmo Canny es un algoritmo de detección de bordes a escala múltiple propuesto por John F. Canny en 1986, cuyo objetivo es encontrar un algoritmo óptimo de detección de bordes. Se utiliza ampliamente en el campo del procesamiento de imágenes y está en constante mejora e innovación. El operador Canny elige los puntos de borde mediante un enfoque de umbral, por lo que el valor del umbral tiene una importante influencia en los resultados de la detección de bordes de la imagen [21].

Para conseguir un funcionamiento óptimo, el algoritmo internamente realiza las siguientes etapas [22]:

- Reducción del ruido: Filtra el ruido de la imagen mediante un filtro gaussiano. Para conseguir suavizar el ruido de la imagen.
- Encontrar el gradiente de intensidad de la imagen: El gradiente define dos valores: la dirección en la cual el cambio de intensidad alcanza el máximo valor y la magnitud de esa dirección. Para ello el algoritmo utiliza cuatro filtros para detectar los bordes horizontales, verticales y diagonales.
- Supresión de falsos máximos: Una vez obtenida la magnitud y dirección de la variación de intensidad de la imagen, se realiza un escaneo completo para eliminar los píxeles que no forman parte del borde y así afinar los bordes encontrados. Para esto se compara el valor de cada píxel con los píxeles cercanos en la dirección del gradiente (perpendicular al borde). Si el valor del píxel en cuestión es mayor que sus píxeles contiguos, entonces este es considerado un máximo local y el algoritmo lo acepta. De lo contrario, si el píxel resulta no ser un máximo local, entonces es suprimido. El resultado final será una imagen con bordes muy finos.

- Umbral de histéresis: Esta etapa trata de aumentar la precisión de los contornos encontrados, para ello, se fijan dos valores de umbral *minVal* y *maxVal.* Los bordes con intensidad mayores que *maxVal* (A) serán aceptados como pertenecientes a los bordes mientras que los píxeles con intensidad menor a minVal serán descartados (D). Por otro lado, los píxeles con valores de intensidades entre *minVal* y *maxVal* serán aceptados o no dependiendo de a qué píxeles estén conectados. Si estos píxeles con intensidades intermedias están conectados a píxeles que han sido descartados (C), éstos serán también descartados, por el contrario si están conectados a píxeles que han sido considerados parte del borde, se considerarán también parte del borde (B). Los puntos A, B, C y D vienen representados en la Figura 16:

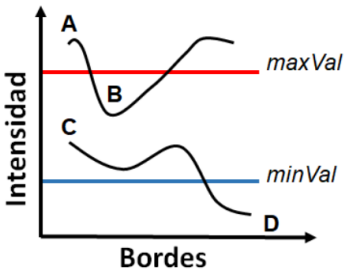


Figura 16 Umbral histéreis. Algoritmo Canny. [22]

En el desarrollo del proyecto, como se explica en el siguiente apartado 7.3 el algoritmo Canny se aplica a la imagen en escala de grises con filtro mediana y gauss. Para este proyecto se han elegido unos valores de umbral: minVal: 40 y maxVal: 150.

7.3 Filtrado de la Imagen

Una vez obtenida la imagen en escala de grises sin brillo tras aplicar el modelo de color HSV (ver apartado 7.1), comienza el proceso de filtrado. El objetivo de este proceso es difuminar la imagen para eliminar el ruido de la misma y para obtener solo las formas principales de la imagen. Para ello es necesario aplicar filtros paso bajo con los que se elimina el ruido de alta frecuencia. Con esto se consigue suavizar el ruido y los bordes de la imagen. La librería OpenCV ofrece, entre otros, tres filtros de paso bajo que se definen a continuación: Filtro promedio, filtro mediana y filtro gaussiano.

FILTRO PROMEDIO

El filtro promedio se realiza convolucionando la imagen con un filtro de caja normalizado. De este modo, el filtro toma el promedio de todos los píxeles bajo el área de la máscara o kernel y remplaza al elemento central por este valor promedio. En el caso de que se utilice un filtro de caja normalizado 3x3 el kernel con el que se convoluciona la imagen se vería asi:

$$K = \frac{1}{9}\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$

Figura 17 Kernel 3x3 filtro promedio

Este sería un ejemplo de procesamiento de una imagen tras aplicar filtro promedio con un *kernel* 5x5.

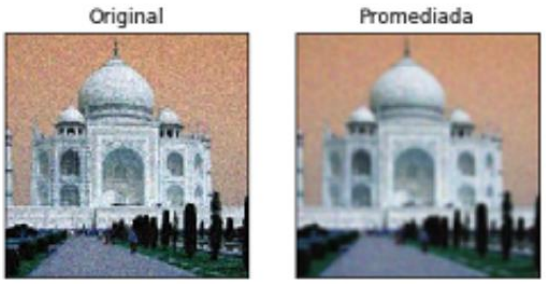


Figura 18 Ejemplo imagen filtro promedio. [19]

Para verificar que el filtro promedio es un buen filtro para el tipo de imagen real del presente proyecto, se ha hecho un investigativo con 5 diferentes tamaños de *kernel*, en orden de aparición 5x5, 7x7 ,9x9 ,11x11 y 13x13:

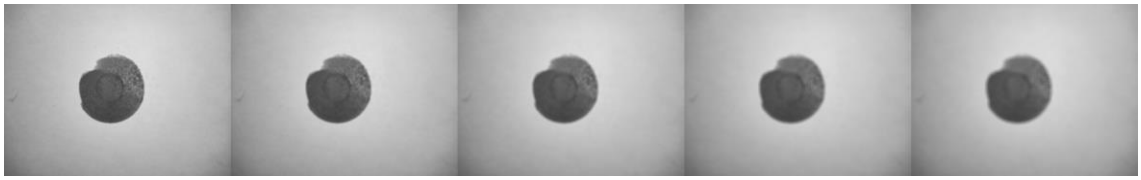


Figura 19 Filtrado promedio, kernel: a) 5x5. b) 7x7. c) 9x9. d) 11x11. e) 13x13

Tras este filtrado, se prueba a calcular el contorno mediante el algoritmo de Canny a las 5 imágenes anteriores. Como se aprecia en la Figura 20, no se obtiene un buen resultado, ya que aplicando el algoritmo a un filtrado de imagen con *kernel* de 5x5 Figura 20 a) la zona de brillo de la imagen no se consigue filtrar bien y es detectado por Canny como contornos. Sin embargo, aumentando el tamaño del *kernel* a valores mayores de 9x9 (Figura 20 d y Figura 20 e) vemos que esa zona de brillo se difumina y no se detecta con el algoritmo de Canny pero tampoco se detecta el borde completamente externo del objeto.

Figura 20 Contorno Canny a imagen con filtro promedio, kernel: a) 5x5. b) 7x7. c) 9x9. d) 11x11. e) 13x13

FILTRO MEDIANA

El proceso del filtro mediana es idéntico al del filtro promedio, salvo que se calcula la mediana de todos los elementos del *kernel* y se sustituye ese valor por el del píxel central. Este filtro se utiliza para eliminar el ruido de sal y pimienta. A continuación, se muestra un ejemplo de filtrado (*kernel* de 5x5):

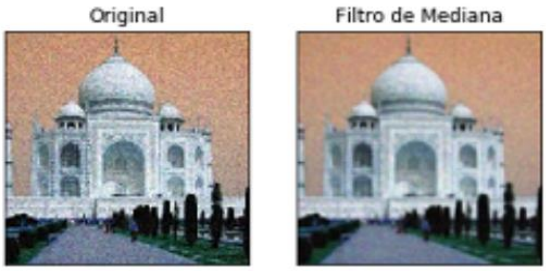


Figura 21 Ejemplo imagen filtro mediana. [19]

Utilizando este filtrado mediana en la imagen real del proyecto, se ha hecho un investigativo aplicando a la imagen en escala de grises un filtro mediana con distintos valores de *kernel*. El resultado se muestra a continuación por orden de aparición: 5x5, 7x7, 9x9, 11x11 y 13x13:

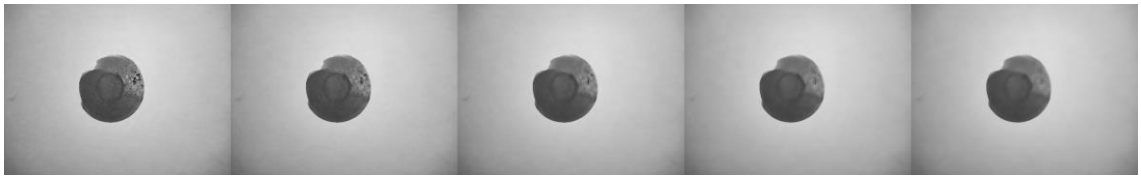


Figura 22 Filtrado mediana, kernel: a) 5x5. b) 7x7. c) 9x9. d) 11x11. e) 13x13

Tras este filtrado, se prueba a calcular el contorno mediante el algoritmo de Canny a las 5 imágenes anteriores de la Figura 22. Como se aprecia en la Figura 23, se obtiene un mejor resultado que tras la aplicación del filtro promedio, ya que en este caso al aumentar el tamaño del *kernel* eliminamos el ruido producido por las imperfecciones de la imagen a causa de la luminosidad pero no perdemos tanta información del contorno externo. Vemos que la imagen donde obtenemos la información completa del contorno sin ruido de la zona de brillo es la Figura 23 d) con filtro mediana *kernel* 11x11:

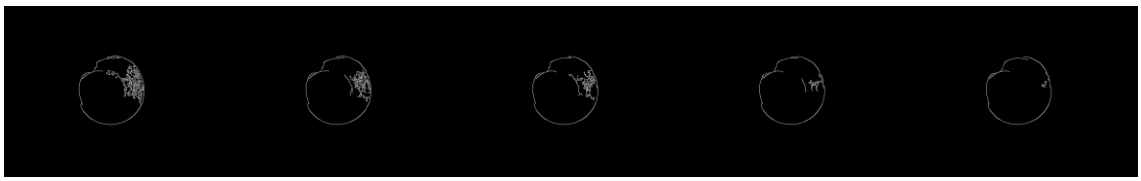


Figura 23 Contorno Canny a imagen con filtro mediano kernel: a) 5x5. b) 7x7. c) 9x9. d) 11x11. e) 13x13

FILTRO GAUSSIANO

El ruido gaussiano produce pequeñas variaciones en la imagen. Tiene su origen en diferencias de ganancias del sensor, ruido en la digitalización, etc. [20]. En este caso, igual que en los filtros explicados anteriormente, también se aplica un *kernel* pero con el filtro gaussiano se tiene en cuenta la geometría de la campana de Gauss y se otorga mayor peso al píxel central:

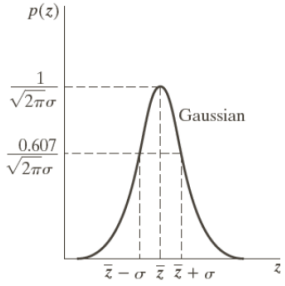


Figura 24 Campana de Gauss

A continuación se muestra un ejemplo de imagen a la que se le ha aplicado un filtro gaussiano con *kernel* 5x5:

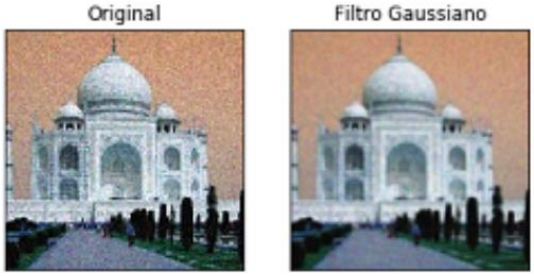


Figura 25 Ejemplo filtro gaussiano. [19]

Por lo tanto, retomando la investigación del filtrado de la imagen del presente proyecto, se prueba a aplicar el filtrado de Gauss tras el filtro promedio visto anteriormente. Se selecciona el mejor resultado de filtro promedio con *kernel* 11 x 11 Figura 19 d) y sobre esa imagen, se aplica un filtrado gaussiano con *kernel* ordenado por orden de aparición: 5x5, 7x7, 9x9 y 11x11 tal y como se muestra en la Figura 26.

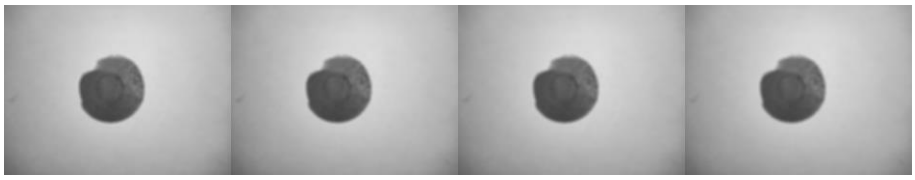


Figura 26 Filtrado promedio kernel 11x11 + gaussiano, kernel: a) 5x5. b) 7x7. c) 9x9. d) 11x11

Tras este filtrado, se prueba a calcular el contorno mediante el algoritmo de Canny a las 4 de la Figura 26. Sin embargo, como se aprecia en las imágenes de la Figura 27 no se obtiene un buen resultado ya que el algoritmo canny no es capaz de detectar los contornos de las imágenes anteriores:

Figura 27 Contorno Canny a imagen con filtro promedio kernel 11x11 + Gaussian, Kernel: a) 5x5. b) 7x7. c) 9x9. d) 11x11

Por lo tanto, descartando el filtro promedio, se prueba a aplicar el filtro gaussiano tras aplicar el filtro mediana. Para ello seleccionamos el mejor resultado tras aplicar el filtro mediana que fue la imagen resultante al aplicar *kernel* 11 x 11, Figura 22 d) y sobre ella aplicamos filtro gaussiano con los siguientes tamaños de *kernel* ordenados por orden de aparición: 5x5, 7x7, 9x9 y 11x11.

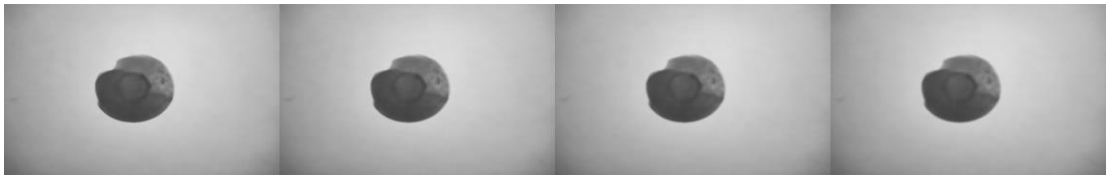


Figura 28 Filtrado mediana kernel 11x11 + gaussiano, kernel: a) 5x5. b) 7x7. c) 9x9. d) 11x11

Tras este filtrado, se prueba a calcular el contorno mediante el algoritmo de Canny a las 4 imágenes mostradas en Figura 28. Se puede apreciar en la Figura 29 que en esta ocasión sí que se obtiene un buen resultado de la

información del contorno externo y se ha eliminado el ruido producido por la zona con brillo.

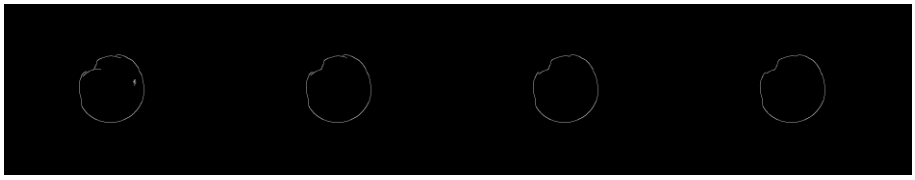


Figura 29 Contorno Canny a imagen con filtro mediana kernel 11x11 + gaussiano, kernel: a) 5x5. b) 7x7. c) 9x9. d) 11x11

Finalmente sobre el resultado de Canny, probamos a detectar el contorno con un cuadrado para posteriormente eliminar el fondo sobrante y vemos que se obtiene un buen resultado tanto con el filtrado gaussiano con *kernel* de 11x11, Figura 30 c) como con *kernel* de 13x13 Figura 30 d):

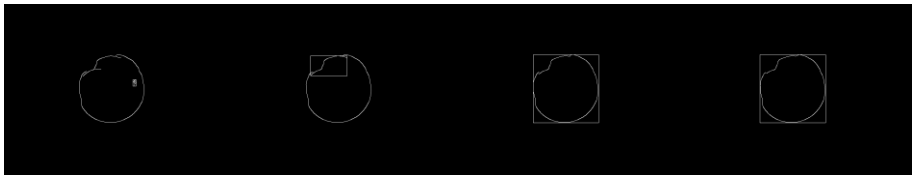


Figura 30 Detección zona de interés en imagen con filtro mediana kernel 11x11 + gaussiano, kernel: a) 5x5. b) 7x7. c) 9x9. d) 11x11

Tras los resultados anteriores, se determina que el filtrado con mejores resultados es Filtro mediana *kernel* 11x11, Figura 22 d) y posteriormente filtro gaussiano *kernel* 11x11, Figura 28 d).

8. Desarrollo del Algoritmo

En los siguientes apartados se realiza una explicación en detalle del proceso seguido para poder resolver los objetivos fijados para este proyecto. Para explicar el desarrollo del algoritmo utilizado se escoge una imagen a modo de ejemplo, sin embargo, el algoritmo se ha probado con diferentes imágenes como se muestra más adelante en el apartado 9.

8.1 Procesamiento de la Imagen 3D (Imagen de referencia)

El primer paso de la impresión 3D consiste en diseñar el objeto que se desea imprimir. Existen diferentes herramientas *software* para la creación y visualización de objetos en 3D, sin embargo, en este caso la imagen 3D de referencia dada, se ha extraído del *software* Slic3r. Por lo tanto, este será el punto de partida del proyecto:

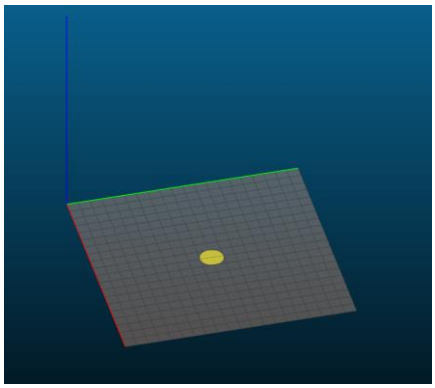


Figura 31 Imagen 3D (Slic3r)

8.1.1 Conversión de imagen 3D a 2D

El primer paso del programa realizado en Python consistirá en abrir la imagen 3D en formato.stl. Para ello, se utiliza la librería *Matplotlib*. Seguidamente, se procede a girar la imagen para visualizar el objecto en vista de planta mediante la función *axes.view.init* de la librería *Matplotlib*, a dicha función se le pasan como parámetros los ángulos de acimut y elevación, en este caso para tener la perspectiva deseada los valores serían: acimut = 0º y elevación = 90º , además se eliminan los ejes de coordenadas, las cuadrículas del fondo de la imagen y se guarda dicha imagen. A

continuación, mediante la función *escalar* se escala la imagen a tamaño (640,480). A continuación, en Figura 32, se muestra el resultado:

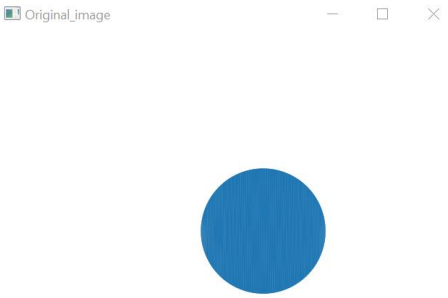


Figura 32 Imagen referencia (2D)

8.1.2 Procesado del color de la imagen

En este momento ya se dispone de una imagen de tamaño conocido en 2D, el siguiente paso por tanto, consistirá en empezar a procesar la imagen con el objetivo de conseguir el contorno de la misma, detectar el área de interés y eliminar el fondo restante. Como estas imágenes extraídas de Slic3r no presentan brillos ni otro tipo de ruido, no será necesario aplicar filtros. De este modo, dentro de la función *color_bin_image*, haciendo uso de la librería *OpenCV,* mediante la función *cvtColor* transformamos la imagen original a escala de grises, para posteriormente con la misma función *cvtColor*, obtener la imagen en binario. Para obtener la imagen en espacio de color binario, el valor de umbral se calcula mediante la binarización de Otsu, la cual se basa en el histograma de las imágenes en escala de grises. La idea principal es maximizar la varianza entre clases en el histograma. Por otro lado, se necesita minimizar la varianza dentro de la clase. [24]. Como resultado, la binarización de Otsu calcula automáticamente el umbral por el cual los valores de los píxeles a partir de un determinado valor pasarán a valer 255 (color blanco) o 0 (color negro) La Figura 33 muestra el progreso desde la imagen original Figura 33 a) en 2D, pasando por escala de grises Figura 33 b) y finalmente la imagen en binario Figura 33 c):

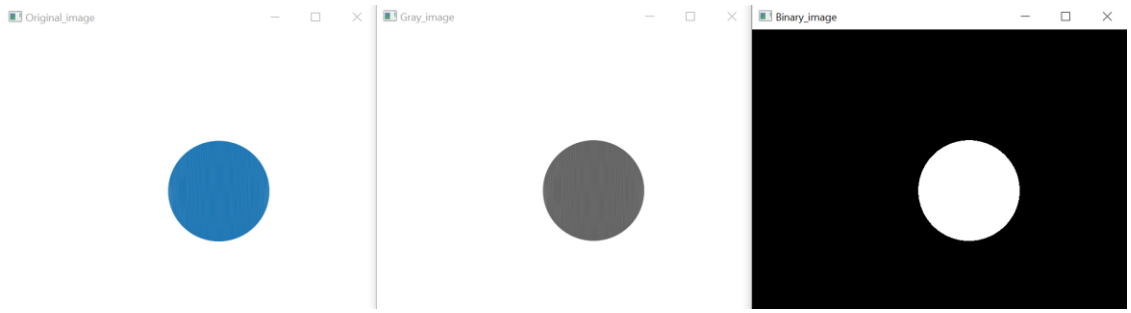


Figura 33 Tratamiento color imagen referencia a) original, b) escala de grises, c) binaria

8.1.3 Contorno del objeto

En la función *contours* se procede a detectar el contorno del objeto de interés de nuestra imagen. Esta función se utiliza tanto para detectar el contorno de la imagen de referencia (en espacio binario), como para detectar contornos de la imagen real después de aplicar el algoritmo *Canny* que veremos más adelante.

En ambos casos, la función utilizada es la misma, se trata de la función *findContours* de la librería *OpenCV*, a la cual se le pasan los siguientes parámetros: [25] [26]

1. Imagen origen.
2. Jerarquía de contornos. En este caso se usa la jerarquía *CV_RETR_TREE* la cual crea una jerarquía completa y ordenada de todos los contornos de la imagen.
3. Método: Configura el método para la detección de contornos, se escoge el método *CV_CHAIN_APPROX_SIMPLE*, el cual en segmentos horizontales, verticales y diagonales solo almacena los puntos extremos, de este modo es más óptimo.

El siguiente paso consiste en dibujar el contorno calculado anteriormente. Para evitar contornos falsos producidos por zonas ruidosas (cuando se trata de imágenes reales con brillo y ruido), solo se considera un contorno válido aquel que sea mayor de 100. Por lo tanto, todo aquel contorno mayor de 100 se dibujará en una imagen creada mediante una máscara con todos los valores a 0 (color negro) y sobre ella, se dibuja el contorno seleccionado, ver

Figura 34. El dibujo del contorno se realiza con la función de *OpenCV drawContours* y tiene como entradas:

1. Imagen sobre la que se dibuja el contorno, en este caso la máscara creada anteriormente: mask1.
2. El contorno calculado: contours.
3. Selección de contornos, en este caso como queremos todos los contornos calculados, seleccionamos: -1.
4. Color del contorno: 255 (color blanco).
5. Espesor del contorno: 1

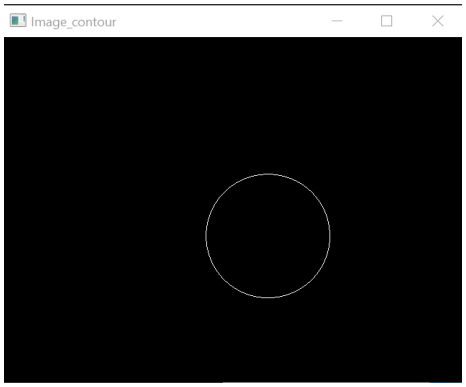


Figura 34 Detección de contorno en imagen referencia.

A continuación, se calcula el rectángulo o cuadrado que englobe el contorno externo del objeto de interés. Se realiza con la función de la librería *OpenCV boundingRect,* la cual solo tiene como parámetro de entrada el primer contorno guardado según la jerarquía de la función *findContours*.

Este orden asegura que el primer contorno será el contorno más externo. Además, también se dibuja el rectángulo anterior sobre la imagen de contorno, con la función de la librería *OpenCV rectangle.* El resultado se muestra en la Figura 35:

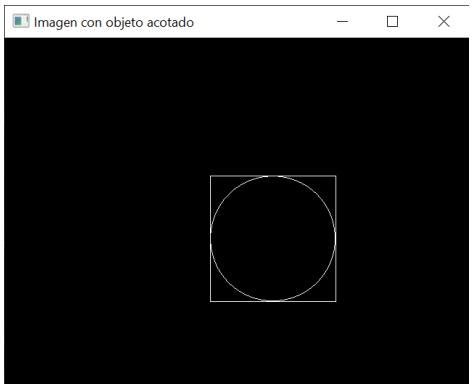


Figura 35 Rectángulo en área de interés

El último paso de la función *contours*, consiste en recortar el tamaño de la imagen al tamaño del cuadrado o rectángulo calculado para eliminar el fondo sobrante de nuestra imagen. El resultado final se puede apreciar en la Figura 36:

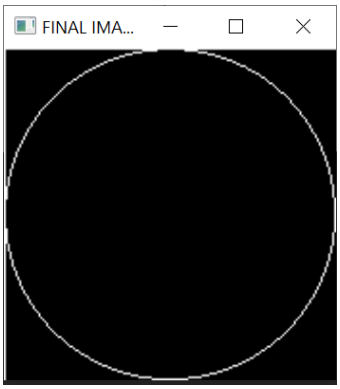


Figura 36 Imagen de referencia final

8.2 Procesamiento Imagen Real

En este apartado se detallarán los pasos a seguir para el procesado de la imagen real tomada por la impresora 3D al finalizar cada capa de impresión.

8.2.1 Eliminación brillo de la imagen

En este caso, las fotografías tomadas siempre se realizarán desde la parte superior de la impresora y pueden presentar brillo y ruido pues va a depender de cómo esté iluminado el espacio en cada momento. Por ello, el primer paso de la función *color_image* consiste en convertir la imagen original de BGR a HSV con la función *cvtColor* de la libería *OpenCV*. A través de esta conversión, cada píxel de la imagen constará de 3 componentes H (matiz), S (saturación) y V (Brillo). Para eliminar el brillo de la imagen real, se modifica el valor de V. Por lo tanto, en primer lugar, a través de la función de la librería *OpenCV*, *split,* se extraen los valores de H, S y V de cada píxel y se le suma 15 al valor de "V" Tras diversas pruebas, 15 es el valor óptimo para eliminar brillo de la imagen sin perder información necesaria, (ver párrafo 7.1 para más información).

A continuación, en la Figura 33 se muestra la comparativa de la imagen inicial Figura 33 a) y la imagen convertida a HSV Figura 33 b):

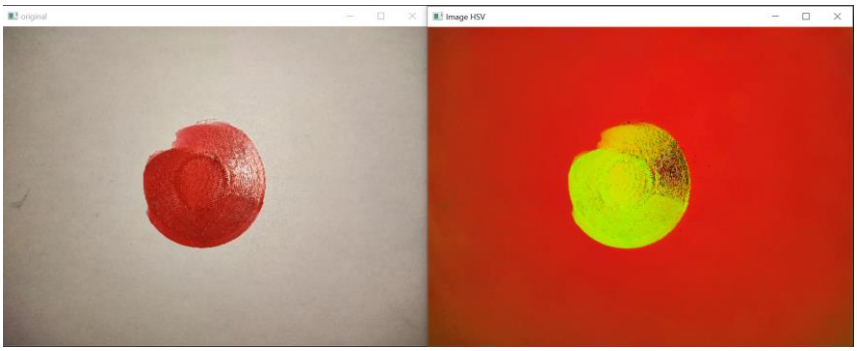


Figura 37 a) imagen real inicial. b) Imagen en espacio de color HSV

El siguiente paso consiste en convetir la imagen de HSV a escala de grises, utilizando de nuevo la función *cvtColor.* Como se puede apreciar en la Figura 38 c), la imagen resultado en escala de grises no presenta brillos:

Figura 38 a) imagen real inicial. b) Imagen en espacio de color HSV. c) Imagen en escala de grises

8.2.2 Filtrado de la imagen

Seguidamente, con la función *filter_image* y tras el investigativo realizado definido en el apartado 0, se realiza un filtrado de la imagen con filtro mediana para eliminar el ruido de sal y pimienta, el filtrado se lleva a cabo a través de la función *medianBlur* de la librería *OpenCV* con un tamaño de *kernel* de 11x11. El resultado obtenido se muestra en la Figura 39:

Figura 39 Imagen con filtro mediana

Para eliminar el ruido gaussiano de la imagen, tras realizar el filtro mediana, se aplica un filtro gaussiano. La función es *GaussianBlur* de librería *OpenCV,* a la función se le pasa como parámetro el valor de sigma o anchura de la campana de gauss que en nuestro caso dejamos como 0 para que se

calcule automáticamente. El resultado del filtrado, se observa en la Figura 40:

Figura 40 Imagen con filtro mediana y gauss

Por último, aplicamos el algoritmo *Canny* para hallar los contornos de la imagen. El algoritmo *Canny* se ha definido anteriormente en el párrafo 7.2. A continuación en la Figura 41, se muestra el resultado de aplicar dicho algoritmo para la detección de bordes:

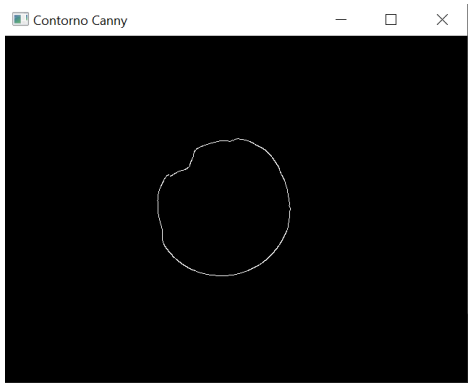


Figura 41 Imagen con algoritmo Canny

La última función del procesamiento de la imagen real es la función *contours,* esta función es la misma que la aplicada a la imagen de referencia y se ha detallado en el apartado 8.1.3. Como resultado obtenemos en primer lugar el contorno de nuestro objeto Figura 42 a) sobre la máscara creada, en segundo lugar en la Figura 42 b) se puede ver el rectángulo creado en torno al objeto de interés y finalmente en la Figura 43, se observa la imagen final recortada.

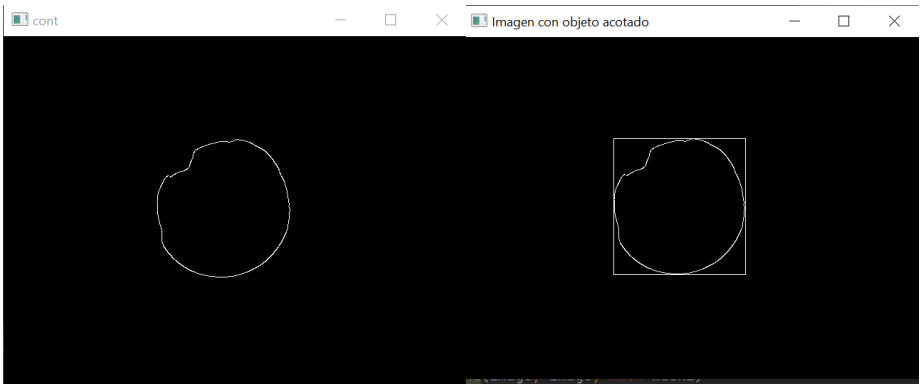


Figura 42 a) Imagen real contorno. b) Rectángulo área de interés en imagen real

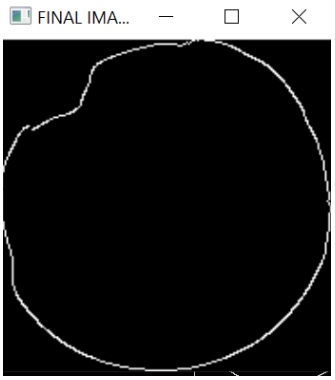


Figura 43 Imagen real final

8.3 Comparación de las Imágenes

Finalmente se procede a comparar las dos imágenes finales obtenidas. Para ello, en la función *comparar* se restan los valores de ambas imágenes para ver cuántos puntos distintos se obtienen. Esta resta de imágenes se realiza con la función *subtract* de la librería *OpenCV* [27]. Para determinar si la impresión se ha realizado correctamente o no, se calcula el número de píxeles que forman el contorno de la imagen comparada, tras diversas pruebas se determina que si ese número es menor a un 2.5% del número total de píxeles de la imagen, aparecerá el siguiente mensaje:

"No hay errores en la impresión."

Si por el contrario, el número de píxeles de la imagen comparada es mayor a un 2,5% del número total de píxeles de la imagen, se considera que sí que hay fallos en la impresión y aparecerá el siguiente mensaje:

"Hay errores en la impresión. DETENER IMPRESIÓN."

A continuación, se envía un plugin *Plugin_stop_printing* para comandar una señal de parada a la impresora y así detener la impresión. Además, el valor *stop* que devuelve la función *comparar* pasará a valer 1, y por lo tanto, el programa finalizará mostrando también el siguiente mensaje por pantalla:

"La impresión no se completó correctamente. Última capa impresa: *nº capa actual* / *nº capa total*. "

En este proyecto, las imágenes finales tras su respectivo procesado se han escalado a 350x350 píxeles, es decir, las imágenes tienen 122.500 píxeles en total. Por lo tanto, un 2.5% sería 3062 píxeles. Es decir, diremos que la impresión no presenta fallos si el número total de píxeles que tiene la imagen resultante comparada es menor a 3062 píxeles. En caso contrario, se concluirá que la impresión sí que presenta defectos.

En este caso concreto, el resultado obtenido tras la comparación de ambas imágenes es 4004 píxeles, el resultado se puede ver en la Figura 44:

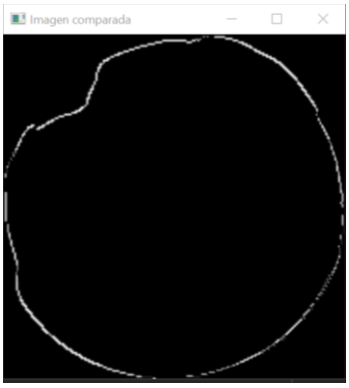


Figura 44 Imagen comparada

Por lo tanto, como el número de píxeles de la imagen comparada es mayor a 3062, se muestra el siguiente error:

"*Hay errores en la impresión. DETENER IMPRESIÓN.* "

A continuación, a través de la función *import*, se envía un plugin *Plugin_stop_printing* para comandar una señal de parada a la impresora y así detener la impresión.

Stop =1

Y aparece el siguiente mensaje final:

"*La impresión no se completó correctamente. Última capa impresa: 1 /1.* "

9. Validación del Algoritmo

Para verificar el procedimiento anteriormente descrito, en este apartado se muestran diferentes ejemplos con y sin defectos del objeto disco utilizado a lo largo de la memoria. Además, al final también se incluye el análisis de un objeto con contornos complejos para probar la validación también con este tipo de objetos más complejos:

- **Disco. Defecto 2:**

1. Eliminación brillo de la imagen:

Como se aprecia en la siguiente Figura 45 se consigue eliminar el brillo de la imagen real, transformando primero la imagen a modelo HSV Figura 45 b) para posteriormente obtener la imagen en escala de grises sin brillo, Figura 45 c):

Figura 45 a) Imagen original. b) Imagen HSV. c) Imagen en escala de grises

2. Filtrado de la imagen:

A continuación, se filtra la imagen hasta obtener una imagen difuminada donde predomina el defecto de impresión que en este caso es falta de material en el interior del objeto:

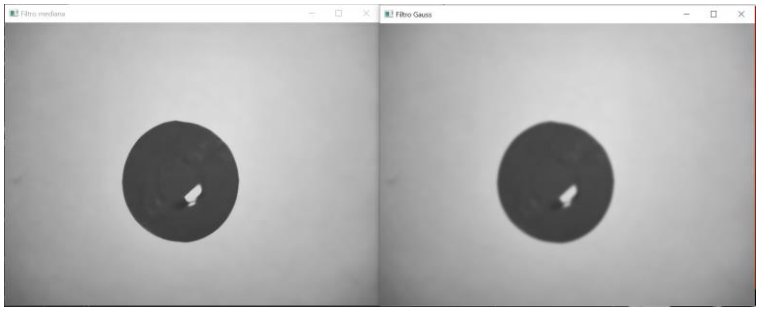


Figura 46 a) Filtro mediana. B) filtro mediana + gauss

3. Algoritmo Canny

En la Figura 47 se observa la obtención de los contornos principales de la imagen, en este caso el contorno exterior y el contorno del defecto:

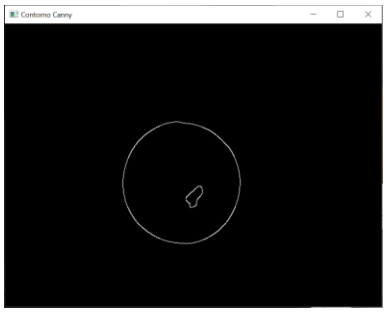


Figura 47 Canny

4. Detección del área de interés

Se procede a eliminar el fondo de la imagen, para poder procesar únicamente el objeto de interés, ver Figura 48 b):

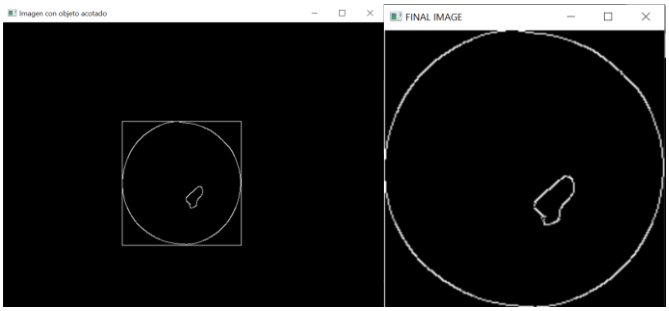


Figura 48 a) Rectángulo área interés. b)Imagen escalada

5. Comparar imágenes:

Por último se obtiene como resultado la comparación entre la imagen real y la imagen obtenida en la Figura 48 b), el resultado se puede apreciar en la Figura 49 donde se aprecia con total claridad el defecto situado en el interior del objeto:

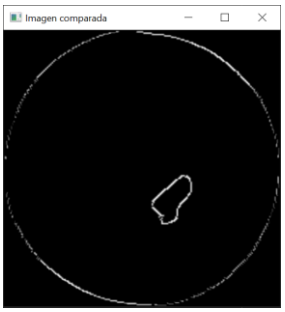


Figura 49 Imagen comparada

El número total de píxeles de la imagen comparada es 3764 > 3062 (2,5% de los píxeles totales), por lo tanto, se detecta error en la impresión.

- **Disco. Defecto 3:**

1. Eliminación brillo de la imagen:

Como primer paso, se consigue eliminar el brillo de la imagen real, transformando primero la imagen a modelo HSV Figura 50 b) para posteriormente obtener la imagen en escala de grises sin brillo, Figura 50 c):

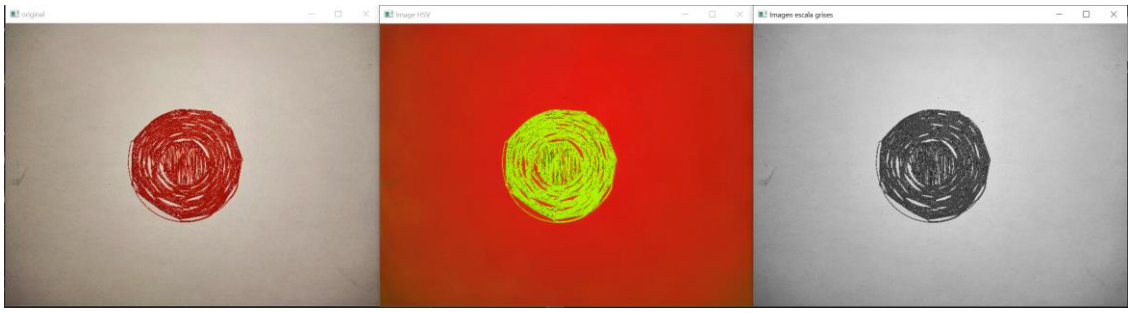


Figura 50 a) Imagen original. b) Imagen HSV. c) Imagen en escala de grises

2. Filtrado de la imagen:

A continuación, se realiza el filtrado de la imagen para difuminar la imagen y que sea más fácil obtener los contornos principales. Se puede apreciar en la Figura 51 b) que en este caso se pierde parte de la información de la imagen real ya que hay pequeños defectos que se difuminan al aplicar el filtrado, sin embargo, los defectos principales sí que siguen apareciendo:

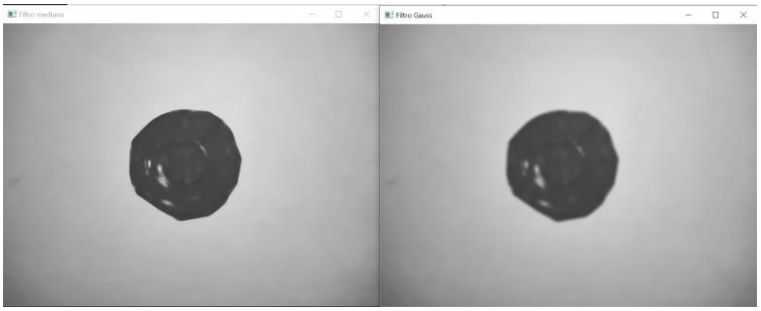


Figura 51 a) Filtro mediana. B) filtro mediana + gauss

3. Algoritmo Canny:

Mediante el algoritmo *Canny* se detecta el contorno externo del objeto y parte de los contornos de los defectos internos, ver Figura 62:

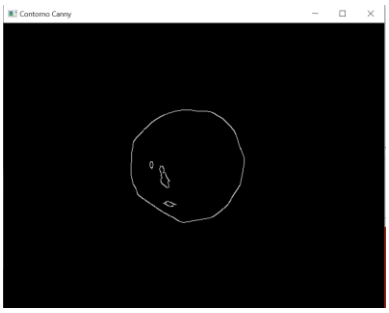


Figura 52 Canny

4. Detección del área de interés:

Se procede a eliminar el fondo de la imagen, para poder procesar únicamente el objeto de interés, ver Figura 53 b):

Figura 53 a) Rectángulo área interés. b) Imagen escalada

5. Comparar imágenes:

Por último se obtiene como resultado la comparación entre la imagen real y la imagen obtenida en la Figura 53 b), el resultado se puede apreciar en se aprecia con total claridad el defecto situado en el interior del objeto:

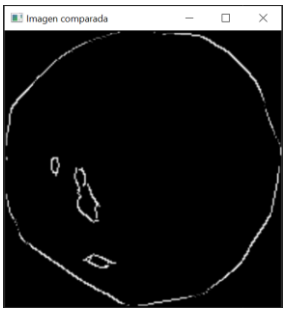


Figura 54 Imagen comparada

El número total de píxeles de la imagen comparada es 5003 > 3062 (2,5% de los píxeles totales), por lo tanto, se detecta error en la impresión.

- **Disco. Defecto 4:**

1. Eliminación brillo de la imagen:

Como primer paso, para la mejor detección del contorno, se elimina el brillo de la imagen real, transformando primero la imagen a modelo HSV Figura 55 b) para posteriormente obtener la imagen en escala de grises sin brillo, Figura 55 c):

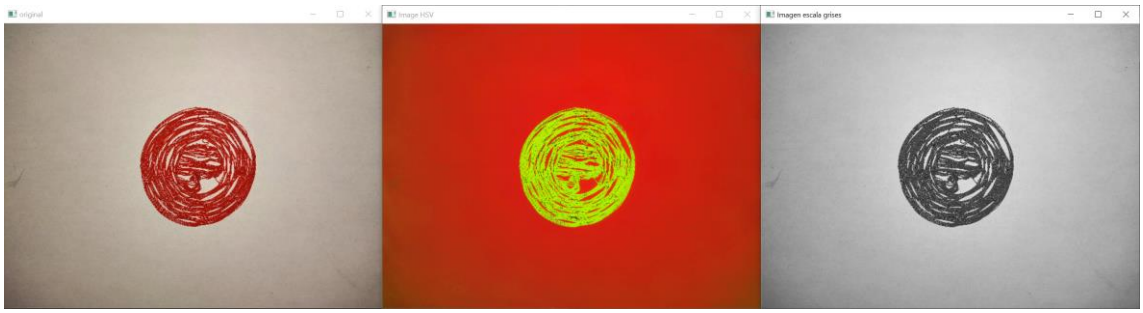


Figura 55 a) Imagen original. b) Imagen HSV. c) Imagen en escala de grises

2. Filtrado de la imagen:

A continuación, se realiza el filtrado de la imagen como se puede apreciar en la Figura 56 b), donde se observa que se obtiene una imagen con menor detalle pero donde se aprecian claramente los defectos de impresión:

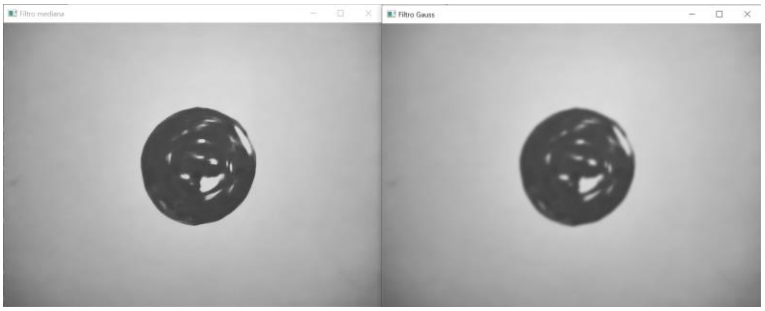


Figura 56 a) Filtro mediana. b) filtro mediana + gauss

3. Algoritmo Canny:

A través del algoritmo *Canny* se obtienen los contornos principales de la imagen, tanto el contorno externo como los internos debido a los defectos de impresión:

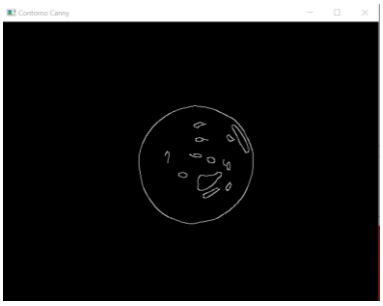


Figura 57 Canny

4. Detección del área de interés:

En la Figura 58 b) se observa el objeto de interés:

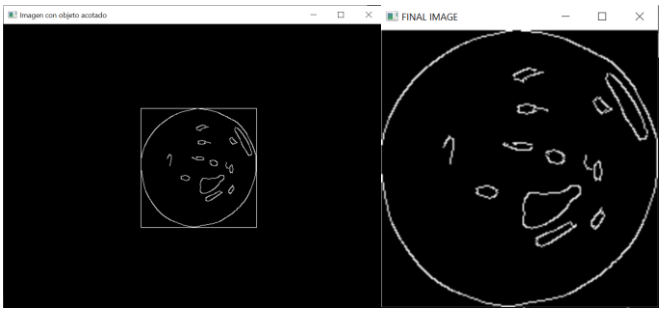


Figura 58 a) Rectángulo área interés. b) Imagen escalada

5. Comparar imágenes:

Finalmente, se obtiene el restultado de comparar la imagen anterior, Figura 58 b), con la imagen de referencia, obteniendo como resultado la siguiente imagen Figura 59 donde se aprecian diversos defectos de impresión:

Figura 59 Imagen comparada

El número total de píxeles de la imagen comparada es 8193 > 3062 (2,5% de los píxeles totales), por lo tanto, se detecta error en la impresión.

- **Disco. Ejemplo sin defecto:**

1. Eliminación brillo de la imagen:

En primer lugar, se elimina el brillo de la imagen real, transformando primero la imagen a modelo HSV Figura 60 b) para posteriormente obtener la imagen en escala de grises sin brillo, Figura 60 c):

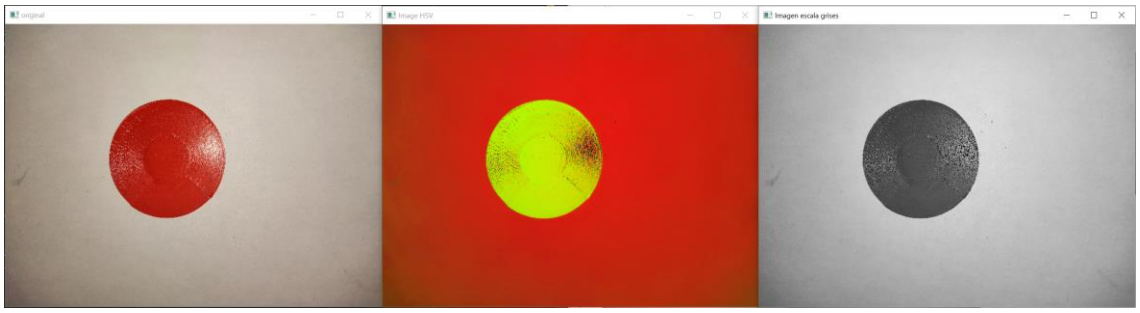


Figura 60 a) Imagen original. b) Imagen HSV. c) Imagen en escala de grises

2. Filtrado de la imagen:

En segundo lugar se filtra la imagen obtenida anteriormente, Figura 60 c), aunque en este caso no es necesario puesto que la imagen prácticamente no presenta ruido, por esa razón, el resultado obtenido tras filtrar la imagen Figura 61 b) es similar a Figura 60 c):

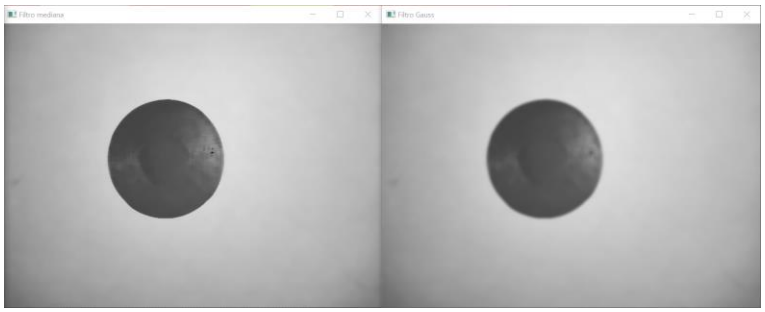


Figura 61 a) Filtro mediana. B) filtro mediana + gauss

3. Algoritmo Canny:

Se detecta el contorno externo de la imagen, en este caso es el único contorno a detectar, ver Figura 62:

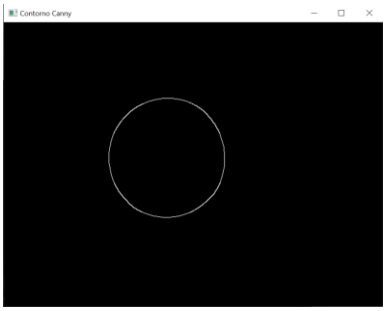


Figura 62 Canny

4. Detección del área de interés:

A continuación, se detecta el área de interés y se elimina el ruido de fondo, tal y como se muestra en Figura 63 b):

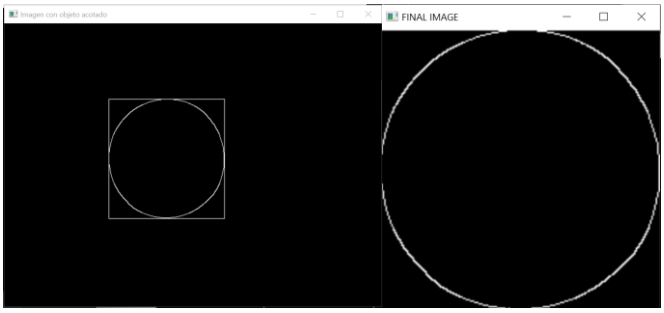


Figura 63 a) Rectángulo área interés. b) Imagen escalada

5. Comparar imágenes:

Finalmente se obtiene la imagen resultante, Figura 64, tras la comparación de la Figura 63 b) con la imagen de referencia. En este caso se obtienen algunos píxeles blancos en el contorno externo porque el grosor del contorno de la Figura 63 b) está ligeramente difuminado y tiene mayor grosor que el contoro de la imagen de referencia.

Figura 64 Imagen comparada

El número total de píxeles de la imagen comparada es 2867 < 3062 (2,5% de los píxeles totales), por lo tanto no hay errores en la impresión.

- **Objeto con contornos complejos**

1. Procesamos imagen referencia:

Siguiendo con el procedimiento definido en el apartado 8, se parte de la imagen de referencia, Figura 65 a), para posteriormente trasformarla en escala de grises Figura 65 b) y finalmente obtener la imagen en binario Figura 65 c):

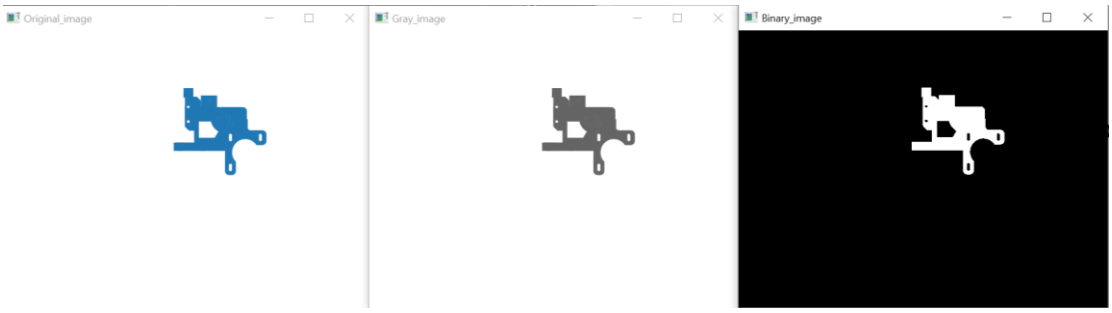


Figura 65 Objeto contornos complejos_procesado color. a) Referencia, b) Imagen en escala de grises y c) imagen en binario

A continuación, se detectan los contornos del objeto y se elimina el fondo de la imagen para obtener solo la información del objeto de interés, ver Figura 66 b):

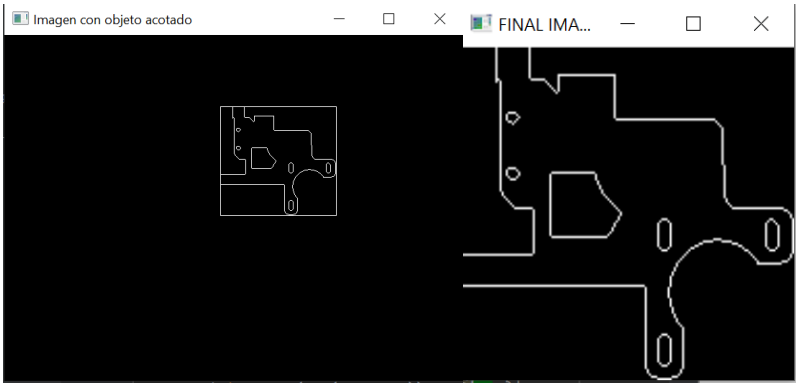


Figura 66 a) Rectángulo área interés. b) Imagen escalada final

A continuación se procesa la imagen real:

2. **Eliminación brillo de la imagen:**

Del mismo modo que ya se ha explicado anteriormente, se elimina el brillo de la imagen real, convirtiendo en primer lugar la imagen en modelo HSV (ver Figura 67 b)) para después obtener la imagen sin brillos tal y como se muestra en la Figura 67 c):

Figura 67 a) Imagen original. b) Imagen HSV. c) Imagen en escala de grises

3. **Filtrado de la imagen:**

A continuación se realiza el filtrado para eliminar el ruido sal y pimienta y así difuminar la imagen para conseguir obtener los contornos principales de la imagen, ver Figura 68 b):

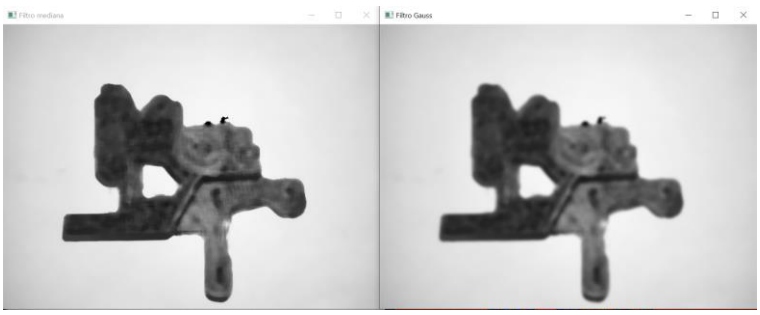


Figura 68 a) Filtro mediana. b) filtro mediana + gauss

4. Algoritmo Canny:

Mediante el algortimo *Canny* se detectan los contornos principales de la imagen, ver resultado en Figura 69:

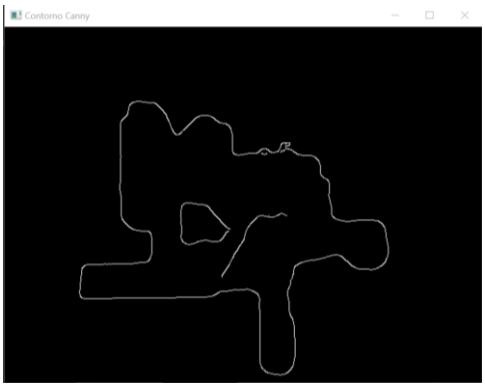


Figura 69 Canny

5. Detección del área de interés:

A continuación, se detecta el objeto de interés y se elimina el fondo de la imagen, ver Figura 70 b):

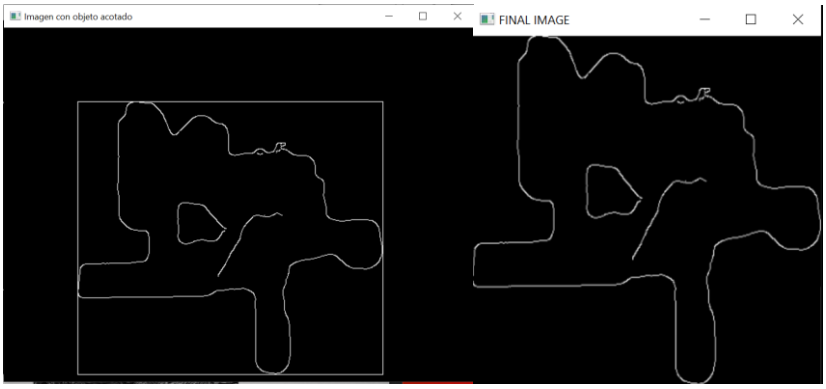


Figura 70 a) Rectángulo área interés. b) Imagen escalada

Finalmente, se obtiene el resultado tras la comparación entre el contorno obtenido de la imagen de referencia, Figura 65 c) y el contorno de la imagen real Figura 70 b). Como se puede apreciar en la siguiente imagen, se detectan diversos defectos de impresión:

Figura 71 Imagen comparada

El número total de píxeles de la imagen comparada es 3114 > 3062 (2,5% de los píxeles totales), por lo tanto hay errores de impresión.

A continuación se muestra un resumen de los defectos detectados con el programa analizado en el presente proyecto:

Nº defecto	Imagen real	% píxeles imagen comparada	Tipo de defecto	Resultado
1		3.27 % > 2.5 %	Error en el contorno	**Error detectado**
2		3.07 % > 2.5 %	Falta de relleno	**Error detectado**
3		4.08 % > 2.5 %	Falta de relleno + error en el contorno	**Error detectado**
4		6.69 % > 2.5 %	Falta de relleno	**Error detectado**
5		2.54 % > 2.5 %	Falta de relleno + error en el contorno	**Error detectado**

Tabla 1 Resumen resultados defectos analizados

10. Conclusiones y Trabajos Futuros

10.1 Conclusiones

El programa desarrollado en el presente proyecto ha sido capaz de detectar diferentes tipos de defectos de impresión, tanto defectos debidos a falta de relleno en el interior del objeto, como fallos en el contorno de la pieza impresa.

El análisis de estos defectos se ha realizado a través del siguiente algoritmo:

- Binarización de la imagen de referencia para obtener los contornos de referencia a través del algoritmo *Canny* y la función *cv2. Contours* de la librería *OpenCV.*
- Procesado de imagen real para eliminar brillos mediante función HSV, y filtrado de imagen mediante filtro mediana y Gauss. Con este filtrado la imagen resultante queda difuminada, por lo tanto no será posible detectar defectos muy leves, solamente serán detectables defectos más evidentes como errores en el contorno o falta de relleno en el interior del objeto.
- Obtención los contornos de la imagen real mediante el algoritmo *Canny* y la función *cv2. Contours* de la librería *OpenCV*. Se obtendrán los contornos de los defectos que sean visibles tras el difuminado realizado por el filtro mediana y gauss.
- Comparación de ambas imágenes. Se sustrae el valor de píxeles del contorno de la imagen real al contorno de la imagen de referencia. Se calcula el ratio del número de píxeles del contorno de la imagen comparada respecto al número total de píxeles de la imagen de referencia y se determina si ese valor es menor o mayor a 2.5%. Si es menor, no habrá errores en la impresión. Sin embargo, si ese ratio es mayor, sí que habrá errores en la impresión. Esta detección funcionará siempre y cuando el tamaño del objeto de la imagen real sea igual o similar al tamaño del objeto de referencia, esto se consigue en este trabajo, al almacenar solo el área bajo el

paralelogramo que engloba al objeto de interés y escalando dicha imagen. Por lo tanto, siempre se compararán imágenes con objetos de tamaños semejantes.

Para finalizar se puede concluir en que se han complido los objetivos establecidos.
En el trabajo se evalúan distintos procesamientos de imágenes, explicando las ventajas y desventajas de cada uno de ellos y explicando el criterio final establecido.
Además, se ha validado el algoritmo creado con distintos tipos de defecto y distintos objetos, como se muestra en el apartado 9.

10.2 Trabajos Futuros

El trabajo en general ha logrado resultados satisfactorios en cuanto a la detección de defectos de impresión. Sin embargo, existen algunos puntos que se pueden estudiar en un futuro para optimizar y mejorar el proceso:

- Extraer cada capa de referencia de un modelo 3D de manera automática. En el presente trabajo se ha considerado que cada capa se almacena en un modelo 3D distinto y almacenados en un archivo nombrado “nº capa.stl”.
- Implementar diferentes tipos de filtrado en función de la cantidad de brillo que presente la imagen. Con esto se podrá discernir entre imágenes con poco brillo que necesiten poco filtrado y por lo tanto se obtenga un resultado más nítido en el que se puedan detectar mayor cantidad de defectos.
- Añadir un algoritmo de detección de geometrías para optimizar el proceso de comparación de imágenes. De modo que se puedan obtener puntos característicos de cada imagen además de cuantificar el número de píxeles del contorno de la imagen comparada final.
- Crear plugin para reanudar la impresión si no se ha detectado fallo en la capa actual. De este modo, la impresora reanudará la impresión automáticamente.

11. Lista de Referencias y Bibliografía

[1] V. W. Sahana and G. T. Thampi, "3D printing technology in industry," 2018 2nd International Conference on Inventive Systems and Control (ICISC), 2018, pp. 528-533.

[2] Tridi. *Breve historia de la impresión 3D* https://www.tridimx.com/blog/historia-de-la-impresion-3d/

[3] Alafaghani, A., Qattawi, A. and Ablat, M. (2017) Design Consideration for Additive Manufacturing: Fused Deposition Modelling. Open Journal of Applied Sciences, 7, 291-318.

[4] Descubre Arduino. *Guía de inicio a la impresión 3D* .https://descubrearduino.com/impresion-3d/

[5] Prusa3D. *Original Prusa i3 MK3S+* https://www.prusa3d.com/es/producto/kit-original-prusa-i3-mk3s-3/?gclid=CjwKCAjww8mWBhABEiwAl6-2RaTO6lQsaIczd1Xq72GqrdhzbfXVCRFFQVkCt6YZBjuesGw9dmb3ZxoCRWQQAvD_BwE#Features

[6] Bitfab. *Guía visual para resolver problemas impresión 3D* https://bitfab.io/es/blog/problemas-impresion-3d/

[7] IBM. *¿Qué es la visión artificial?* https://www.ibm.com/es-es/topics/computer-vision

[8] Yang Lu, Industry 4.0: A survey on technologies, applications and open research issues, Journal of Industrial Information Integration,Volume 6, 2017.

[9] INFAIMON. *Historia de la visión artificial.* https://infaimon.com/blog/vision-2d-3d/historia-evolucion-vision-

artificial/#:~:text=La%20historia%20de%20la%20visi%C3%B3n%20artificial%20marca%20su%20hito%20en,que%20pod%C3%ADan%20estar%20conectada%20de

[10] Slic3r. *Open source 3D printing toolbox.* https://slic3r.org/about/

[11] 3D Natives. *Octorprint, la aplicación para controlar tu impresora 3D*. https://www.3dnatives.com/es/octoprint-aplicacion-control-impresora-210820202/

[12] Stack Overflow: *2022 Developer Survey* https://survey.stackoverflow.co/2022/#technology

[13] Wikipedia. *Historia de Python* https://es.wikipedia.org/wiki/Historia_de_Python

[14] Matplotlib*: Visualization with Python* https://matplotlib.org/

[15] OpenCV: https://opencv.org/about/

[16] Numpy: *What is NumPy?* https://numpy.org/doc/stable/user/whatisnumpy.html

[17] Stefan Trost Media. Software y Soluciones Web. *Modelos de Color* https://es.sttmedia.com/modelos-de-color

[18] Lledo Energía: *Colimetría III: Espacio de color RGB; HSV y HSL* https://lledoenergia.es/colorimetria-iii-espacios-de-color-hsl-hsv-y-rgb/

[19] UnyPython. *Suavizando imágenes con OpenCV* https://unipython.com/suavizando-imagenes-con-opencv/

[20] Wikipedia. *Ruido en la fotografía digital* https://es.wikipedia.org/wiki/Ruido_en_la_fotograf%C3%ADa_digital#:~:text=

Ruido%20gaussiano%3A%20produce%20peque%C3%B1as%20variaciones,ruido%20en%20la%20digitalizaci%C3%B3n%2C%20etc.

[21] Z. Xu, X. Baojie and W. Guoxin, "Canny edge detection based on Open CV," *2017 13th IEEE International Conference on Electronic Measurement & Instruments (ICEMI)*, 2017, pp. 53-56.

[22] Detección de bordes Canny – *OpenCV Python* https://noemioocc.github.io/posts/Detecci%C3%B3n-de-bordes-Canny-openCV-python/

[23] Towars Data Science. Andrew Udell. *Background Removal with Python* https://towardsdatascience.com/background-removal-with-python-b61671d1508a

[24] *B. D. Setiawan, A. N. Rusydi and K. Pradityo, "Lake edge detection using Canny algorithm and Otsu thresholding,"* 2017 International Symposium on Geoinformatics (ISyG)*, 2017, pp. 72-76.*

[25] Hetpro-store. *OpenCV findContours – Encontrar contornos y áreas.* https://hetpro-store.com/TUTORIALES/opencv-findcontours/#:~:text=En%20OpenCV%20findContours%20es%20una,binaria%20(blanco%20y%20negro).

[26] Acervo Lima. *¿Cómo detectar formas en imágenes en Python usando OpenCV?* https://es.acervolima.com/como-detectar-formas-en-imagenes-en-python-usando-opencv/

[27] Programación Python. *Comparación de imágenes en Python con <<OpenCV>> y <<numpy>>*. ttps://programacionpython80889555.wordpress.com/2019/10/01/comparacion-de-imagenes-en-python-con-opencv-y-numpy/

ANEXO 1

PROGRAMA PYTHON

1. PROGRAMA PRINCIPAL

TFM_Sara_Ortiz.py

```
#LIBRERIAS
import cv2
import numpy as np
from stl import mesh
from mpl_toolkits import mplot3d
from matplotlib import pyplot
import paramiko

#FUNCIÓN COLOR_BIN_IMAGE (Imagen referencia):
def color_bin_image (image):
    gray_img = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
   _, image_bin = cv2.threshold(gray_img, 0, 255, cv2.THRESH_BINARY_INV
+ cv2.THRESH_OTSU)
   return (image_bin)

#FUNCIÓN CONTOURS:
def contours (image):
   (contours, _) = cv2.findContours(image, cv2.RETR_TREE,
cv2.CHAIN_APPROX_SIMPLE)
   mask1 = np.zeros(image.shape[:2], np.uint8)
   for i in range(len(contours)):
      area = cv2.contourArea(contours[i])
      if area > 100:
         cv2.drawContours(mask1, contours, -1, 255, 1)

   img_and = cv2.bitwise_and(image, image, mask=mask1)
   x, y, w, h = cv2.boundingRect(contours[0])
   img_cuad = cv2.rectangle(img_and, (x, y), (x + w, y + h), 255, 1)
   img_rec = img_cuad[y:(y + h), x:(x + w)]
```

```
    img_final = cv2.resize(img_rec, (350, 350))
    cv2.imwrite('img_final.png', img_final)
    cv2.imshow("FINAL IMAGE", img_final)
    cv2.waitKey(0)
    return (img_final)

#FUNCIÓN COLOR_IMAGE (Imagen real):
def color_image (image):
    img_hsv = cv2.cvtColor(image, cv2.COLOR_BGR2HSV)
    h, s, v = cv2.split(img_hsv)
    v += 15
    final_hsv = cv2.merge ((h, s, v))
    image_bgr = cv2.cvtColor(final_hsv, cv2.COLOR_HSV2BGR)
    gray_img = cv2. cvtColor(image_bgr, cv2.COLOR_BGR2GRAY)
    return (gray_img)

#FUNCIÓN FILTER_IMAGE (Imagen real):
def filter_image(image):
    median = cv2.medianBlur(image, 11)
    gauss = cv2.GaussianBlur(median, (11, 11), 0)
    canny = cv2.Canny(gauss, 40, 150)
    return (canny)

#FUNCIÓN COMPARAR:
def comparar (img_final, img_final2):
    stop=0
    img_comp = cv2.subtract(img_final2, img_final)
    cv2.imshow('Imagen comparada', img_comp)
    cv2.waitKey(0)
    puntos_ref = np.sum(img_final != 0)
    puntos_diff = np.sum(img_comp != 0)
    print('contorno imagen ref', puntos_ref)
    print('contorno imagen comp', puntos_diff)
```

```
rate = (len(img_final)**2) * 0.025
if puntos_diff < rate:
    print("No hay errores en la impresión.")
else:
    print("Hay errores en la impresión. DETENER IMPRESIÓN.")
    stop=1
    import Plugin_stop_printing
return(stop)

#INICIO PROGRAMA
print('¿Cuántas capas tiene la imagen?')
total_layer = int(input())

#ACCESO IMAGEN REFERENCIA
figure = pyplot.figure()
axes = mplot3d.Axes3D(figure)
new_mesh =
mesh.Mesh.from_file('C:/Users/SaraOM/images/images_ref/test2.stl')
axes.add_collection3d(mplot3d.art3d.Poly3DCollection(new_mesh.vectors))
scale = new_mesh.points.flatten()
axes.auto_scale_xyz(scale, scale, scale)
axes.view_init(azim=0, elev=90)
pyplot.grid(False)
image2d= pyplot.axis('off')
pyplot.savefig('Imagen_ref.png')

 #BUCLE PROGRAMA:
for layer in range(total_layer):
    #PROCESAMENTO IMAGEN REFERENCIA:
    image1 =
cv2.imread('C:/Users/SaraOM/images/images_ref/'+str(layer+1)+'.png')
    image1 = escalar(image1)
    image1 = color_bin_image(image1)
    image1 = contours(image1)
```

```
    # PROCESAMENTO IMAGEN REAL:
     host = "octopi"
     port = 22
     username = "pi"
     password = "raspberry"
     ssh = paramiko.SSHClient()
     ssh.set_missing_host_key_policy(paramiko.AutoAddPolicy())
     ssh.connect(host, port, username, password)
     print('Verificando capa ' + str(layer + 1) + '...')
     image2 = cv2.imread('/home/pi/OctoPrint/Octolapse/data' + str(layer + 1) +
'.png')
    image2 = escalar(image2)
    image2 = color_image(image2)
    image2 = filter_image(image2)
    image2 = contours(image2)
    #COMPARACIÓN DE AMBAS IMÁGENES
    output = comparar(image1, image2)
    if output == 1:
        break

#FINAL DEL PROGRAMA
if (layer+1) == total_layer and output != 1:
        print('La impresión se completó correctamente.')
else:
    print('La impresión no se completó correctamente. Última capa impresa: Nº
'+str(layer+1)+'/'+str(total_layer)+'.')
```

2. PLUGIN

Plugin_stop_printing.py

```
import octoprint.plugin
#MOSTRAR MENSAJE POR PANTALLA:
class stop_printing(octoprint.plugin.StartupPlugin):
        def error_detected(self):
          self.logger.info('Hay un error en la impresión. DETENER IMPRESION.')

# IDENTIFICACIÓN PLUGIN
_plugin_name_ = 'stop_printing'
_plugin_version_= '1.0.0'
_plugin_description_ = 'Detener impresión si se detecta un error.'
_plugin_pythoncompat_='>=3.7,<4'
_plugin_implementation_ = stop_printing()
```