



**MÁSTER EN INGENIERÍA DE SISTEMAS Y CONTROL**

UNIVERSIDAD NACIONAL DE EDUCACIÓN A DISTANCIA
UNIVERSIDAD COMPLUTENSE DE MADRID

## TRABAJO FIN DE MÁSTER

CURSO 2021/2022
Convocatoria de Junio

# Detección de fallos de impresión 3D por medio de visión artificial

**Autor:** Yao Junwei
**Directores:** Dictino Chaos García
Jesús Chacón Sombría
Jacobo Sáenz Valiente

Departamentos de Informática y Automática (UNED)
Departamentos de Arquitectura de Computadores y Automática (UCM)

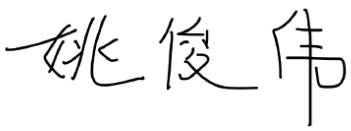
- **Máster En Ingeniería De Sistemas Y Control**
- **Título**: Detección de fallos de impresión 3D por medio de visión artificial
- **Proyecto De Tipo A:** Proyecto específico propuesto por un profesor.
- **Autor:** Yao Junwei
- **Directores:** Dictino Chaos García
  Jesús Chacón Sombría
  Jacobo Sáenz Valiente

**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: Yao Junwei

Firma del alumno



# Resumen

La impresión 3D por medio de deposición de material fundido es una tecnología de rápida expansión, pero no está carente de problemas. Aunque, en teoría, la impresión es totalmente automatizada, existen numerosos problemas que pueden producirse durante la impresión y que requieren una supervisión humana (desalineación de las capas, temperatura inadecuada del plástico, despegue de la pieza de la cama de impresión, etc.).

En este trabajo se pretende utilizar realimentación visual del proceso de impresión 3D para detectar en tiempo real posibles defectos y problemas en la impresión.

Mediante el preprocesamiento de la imagen se obtienen los contornos de la pieza, que se pueden comparar con los esperados. Después se utilizan algoritmos como ORB (***O****riented fast and* ***R****otated* ***B****rief*) que permiten describir la imagen y obtener los puntos característicos, mediante los cuáles finalmente se obtiene la similitud entre la imagen impresa y el dibujo de diseño. Esta similitud se puede usar como una medida del éxito de la impresión 3D y para juzgar si hay errores en la impresión.

**Palabras clave:** Visión Artificial, Procesamiento De Imágenes, ORB (*Oriented Fast and Rotated Brief*), Impresión 3D.

# Abstract

3D printing via molten material deposition is a rapidly expanding technology, but it is not without its problems. Although printing is theoretically fully automated, there are numerous issues that can occur during printing that require human supervision (layer misalignment, improper plastic temperature, part lifting off the print bed, etc.).

In this work it is intended to use visual feedback of the 3D printing process to detect possible defects and problems in printing in real time.

Through the preprocessing of the image, the contours of the piece are obtained, which can be compared with the expected ones. Algorithms such as ORB (***O****riented fast and* ***R****otated* ***B****rief*) are then used to describe the image and obtain the characteristic points, through which the similarity between the printed image and the design drawing is finally obtained. This similarity can be used as a measure of the success of the 3D print and to judge if there are errors in the print.

**Keywords:** Machine Vision, Image Processing, ORB (*Oriented Fast and Rotated Brief*), 3D Printing

# Índice general

# Figuras

# Formulas



# Tablas

# 1. Introducción Y Objetivos

## 1.1 Introducción

La impresión 3D es un avance muy importante de las tecnologías de fabricación por adición, donde un objeto tridimensional es creado mediante la superposición de capas sucesivas del material.

La tecnología de impresión 3D tiene muchas ventajas, algunas de ellas son:

- Alto grado de libertad.
- Fácil utilización
- Alta velocidad de impresión
- Costo relativamente bajo.

Los técnicos o ingenieros pueden aprender a operar impresoras de forma sencilla. Esto brinda una gran comodidad a varios estudios de ingeniería, como robótica, ingeniería mecánica e ingeniería de comunicaciones. Sin embargo, cuando se imprime en 3D, las impresoras suelen causar defectos de impresión por varios motivos, entre los que se incluyen: la temperatura del banco de trabajo (es demasiado alta o baja), la boquilla está bloqueada, los consumibles de impresión no son adecuados, etc.[11]

La impresión 3D lleva un tiempo relativamente largo, en cuanto a la producción de un prototipo (por lo general, puede llevar de 2 a 3 horas).

Para encontrar y resolver problemas de impresión por primera vez, los técnicos deben dedicar mucho tiempo de su trabajo a la supervisión de la impresión, lo cual puede suponer bajo rendimiento para una empresa el tener un técnico realizando esta supervisión. Por lo tanto, este trabajo intentará utilizar la tecnología de visión artificial para juzgar si existe un defecto en la impresión, comparando la imagen impresa del prototipo con la imagen del diseño.


***Figura 1.1.1 Impresora 3D de Prusa i3 con un espacio de trabajo de 200x200x200mm[11]***

El algoritmo utilizado y descrito a lo largo de este trabajo usa visión artificial. Las imágenes impresas mediante impresoras 3D pueden tener cambios en la rotación, el estiramiento y la ampliación comparándolo con el prototipo a imprimir.

Teniendo en cuenta lo mencionado anteriormente será necesario utilizar un algoritmo que tenga en cuenta todos estos posibles cambios, por lo que tiene que asumir las siguientes capacidades:

- El algoritmo de visión artificial usado debe ser resistente a la ampliación, el estiramiento y la rotación de la imagen.
- La velocidad de cálculo del algoritmo debe ser rápida con una precisión de reconocimiento alta.
- El algoritmo debe ser capaz de funcionar con un número pequeño de muestras.

El algoritmo de visión artificial utilizado principalmente en este trabajo es el algoritmo ORB (*Oriented Fast and Rotated Brief*). ORB este algoritmo fue propuesto por el autor *Rublee* en ICCV en 2011, y su objetivo principal es mejorar el rendimiento en tiempo real de los algoritmos convencionales actuales, como SIFT (*Scale-Invariant Feature Transform*) o SURF (*Speeded Up Robust Features*).

Cuando la relación de transformación entre los pares coincidentes de dos vistas de la imagen es pequeña, el rendimiento se puede aproximar al algoritmo SIFT utilizando este algoritmo el tiempo de cálculo se reduce considerablemente.[1]

El rendimiento en tiempo real del algoritmo ORB proporciona una buena aplicación en dispositivos móviles, al mismo tiempo, en comparación con el algoritmo SURF, el algoritmo ORB funciona mejor en el caso de la rotación de imágenes.[22]

El algoritmo ORB, SURF o SIFT puede procesar imágenes en tiempo real, a diferencia de los algoritmos de inteligencia artificial como el aprendizaje profundo.[2]

Los algoritmos anteriores no necesitan un gran número de datos de entrenamiento para completar el trabajo.

En resumen, finalmente la elección del algoritmo ORB como el algoritmo de visión artificial usado para este trabajo ha sido elegido teniendo en cuenta lo mencionado anteriormente.

El algoritmo ORB se utiliza con frecuencia, un ejemplo de de uso de este algoritmo es el siguiente:

- Uso en la conducción autónoma donde se usar el algoritmo SLAM *(Simultaneous localization and mapping)*. Una rama muy importante del algoritmo SLAM es combinar el algoritmo ORB y el algoritmo SLAM. La razón principal para usar el algoritmo ORB-SLAM es que el algoritmo ORB tiene un alto rendimiento en tiempo real y alta precisión. El algoritmo muestra una alta practicabilidad y escalabilidad en la práctica.[7]

Además de ORB (*Oriented Fast and Rotated Brief*), el principal algoritmo de visión artificial es el preprocesamiento de imágenes, del cual se hablará más adelante.

Para tener un valor práctico para el procesamiento general, todas las imágenes utilizadas en este trabajo se tomaron de objetos reales donde algunas imágenes tienen problemas como: sobreexposición, enfoque impreciso y alto brillo.

A continuación, se puede usar algoritmos de reconocimiento de imágenes para

procesar imágenes impresas en 3D e imágenes prototipo y obtener la similitud de las imágenes comparando las imágenes.

Según demanda, se podría ampliar aún más a la línea de producción de la fábrica para realizar escaneos 3D de productos recién producidos y analizar si hay defectos de producción.

## 1.2 Objetivos

El objetivo principal del trabajo es automatizar el diagnóstico de errores, defectos y problemas de impresión 3D.

Este objetivo a su vez se descompone en los siguientes objetivos secundarios:

1. Determinación de errores y defectos comunes cuya detección pueda ser automatizada.
2. Preprocesamiento de una imagen para minimizar la influencia de la iluminación y ruido de fondo.
3. Descripción y extracción de características de la imagen.
4. Comparación de las características de las imágenes reales con el modelo de impresión para realizar el diagnóstico.

## 1.3 Organización de la memoria

Este trabajo se dividirá en 5 partes para presentar el trabajo:

1. En la primera parte, se presentará la línea general del trabajo, así como los objetivos.

2. En la segunda parte, se presentará el *hardware*, el *software* y las bibliotecas que han utilizado para realizar este proyecto de fin de máster.

3. La tercera parte se centrará en la realidad, cómo se procesa la imagen para eliminar el ruido de fondo, cómo funciona el algoritmo ORB (*Oriented Fast and Rotated Brief*) y cómo se utiliza el comparador BFmatcher *(Brute Force Matcher)* etc.

4. En la cuarta parte les mostraré los resultados del trabajo, En esta parte se analizarán los problemas y soluciones que se puedan encontrar en la ejecución de este trabajo.

5. En la quinta sección, se presentarán las conclusiones del trabajo y su extensibilidad futura.

# 2. Estado Del Arte

En este capítulo se presenta una descripción de los métodos y técnicas que se van a utilizar en el trabajo, haciendo referencia a distintos trabajos que muestran sus aplicaciones.

## 2.1 Introducción de OpenCV

OpenCV[12] es una biblioteca de *software* de aprendizaje automático y visión por computadora multiplataforma lanzada bajo la licencia Apache2.0 (código abierto), que puede ejecutarse en los sistemas operativos: Linux, Windows, Android y Mac iOS. Es liviano y eficiente: consta de una serie de funciones de C y una pequeña cantidad de clases de C++, proporciona interfaces en lenguajes como Python, Ruby y MATLAB, e implementa muchos algoritmos generales en procesamiento de imágenes y visión por computadora.[12]

OpenCV está escrito en lenguaje C++, tiene interfaces C++, Python, Java y MATLAB y es compatible con Windows, Linux, Android y Mac OS. OpenCV está orientado principalmente a aplicaciones de visión en tiempo real y utiliza instrucciones MMX y SSE cuando están disponibles, y hoy también Proporciona soporte para C#, Ch, Ruby y GO.


***Figura 2.1.1 Logo de OpenCV [12]***

El mercado de la visión por computadora es muy grande y continúa creciendo actualmente por lo que no existe una API estándar para esto. Existen aproximadamente tres tipos de *software* de visión por computadora en la actualidad:[17]

1. Código de investigación (lento, inestable, independiente e incompatible con otras bibliotecas)

2. Herramientas comerciales de alto costo (como Halcon, MATLAB+Simulink)

3. Algunas soluciones especiales que se basan en *hardware* (como videovigilancia, sistemas de control de fabricación, equipos médicos) son el *statu quo* actual, y las API estándar simplificarán el desarrollo de programas y soluciones de visión artificial. OpenCV se esfuerza por convertirse en una API estándar de este tipo.

OpenCV está comprometido con las aplicaciones en tiempo real en el mundo real. Ha mejorado considerablemente su velocidad de ejecución a través de la escritura de código C optimizado, y puede obtener una velocidad de procesamiento más rápida al comprar la biblioteca de funciones multimedia de alto rendimiento IPP de Intel (*Integrated Performance Primitives*).

La siguiente figura 2.1.2 muestra la comparación del rendimiento entre OpenCV y otras bibliotecas de funciones de visión actuales. En la mayoría de los casos, el rendimiento de cálculo de OpenCV es mejor que otras bibliotecas de funciones de visión actuales.[17]

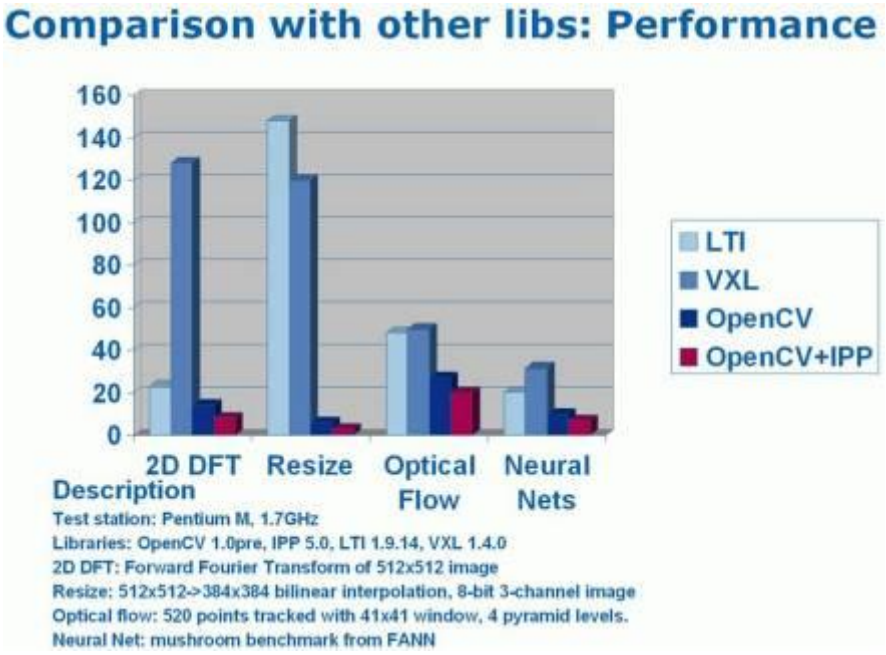



***Figura 2.1.2 Comparación de OpenCV y otras librerías de visión artificial [17]***

Debido a las ventajas anteriores, en la realización de este trabajo, se utilizan una gran cantidad de funciones de la biblioteca OpenCV y se completan los diversos diseños de manera eficiente y precisa con el lenguaje Python. Las funciones de la biblioteca OpenCV están altamente integradas, son fáciles de usar y rápidas.

El que OpenCV sea de código abierto facilita que programadores e ingenieros mantengan el código actualizado e implementen y depuren las funciones utilizadas a lo largo de este proyecto.

## 2.2 Introducción de Algoritmo ORB (Oriented Fast and Rotated Brief)

El algoritmo ORB es la abreviatura de *Oriented Fast and Rotated Brief,* que se puede usar para crear rápidamente vectores de características para puntos clave en una imagen, y así identificar objetos en las imágenes de forma sencilla.[1] *FAST y BRIEF* son los algoritmos de detección de características y de creación de vectores.

ORB en primer lugar busca regiones especiales en la imagen, llamadas puntos clave. Los puntos clave son pequeñas regiones prominentes en una imagen, como las esquinas, que tienen un cambio brusco en el valor de píxel de claro a oscuro.

ORB luego calcula el vector de características correspondiente para cada punto clave. Los vectores propios creados por el algoritmo ORB contienen solo 1 y 0 y se denominan vectores propios binarios. El orden de los 1 y los 0 cambia según un punto clave en particular y el área de píxeles que lo rodea. Este vector representa patrones de intensidad alrededor de puntos clave, por lo que se pueden usar varios vectores de características para identificar áreas más grandes o incluso objetos específicos en una imagen.

Además del algoritmo ORB (*Oriented Fast and Rotated Brief)*, los algoritmos de puntos característicos principales incluyen el algoritmo SIFT (*Scale-Invariant Feature Transform*) y el algoritmo SURF (*Speeded Up Robust Features*). Con base en datos de otros investigadores[22], es posible comprender cómo se desempeña cada algoritmo en diferentes aspectos.

Table 4. Results of comparing the image with its scaled image.

| | Time (sec) | Kpnts1 | Kpnts2 | Matches | Match rate (%) |
|---|---|---|---|---|---|
| SIFT | 0.25 | 248 | 1210 | 232 | 31.8 |
| SURF | 0.08 | 162 | 581 | 136 | 36.6 |
| ORB | 0.02 | 261 | 471 | 181 | 49.5 |

***Figura 2.2.1 La comparación de imágenes de diferentes escalas[22]***

Table 7. Results of the image matching by adding 30 % of salt and pepper noise.

| | Time (sec) | Kpnts1 | Kpnts2 | Matches | Match rate (%) |
|---|---|---|---|---|---|
| SIFT | 0.115 | 248 | 242 | 132 | 53.8 |
| SURF | 0.059 | 162 | 385 | 108 | 39.48 |
| ORB | 0.027 | 261 | 308 | 155 | 54.48 |

***Figura 2.2.2 La comparación de imágenes con ruido[22]***

Según los datos de este artículo, el algoritmo ORB tiene el tiempo de ejecución más corto y la tasa de coincidencia más alta en la escena de *zoom* de imagen y la imagen con ruido. Para la escena de impresión 3D, la imagen de impresión 3D generalmente tiene algo de escalado y rotación en comparación con el prototipo de diseño, y puede tener algo de ruido, y el algoritmo ORB funciona mejor en ambas escenas.

A través del algoritmo ORB, el algoritmo SIFT, el algoritmo SURF puede extraer los puntos característicos de la imagen. Este trabajo utilizará el algoritmo ORB con el mejor efecto de procesamiento para calcular los puntos característicos de la imagen. La imagen se puede describir matemáticamente a través de los puntos característicos. Estos puntos característicos tendrán un significado positivo para el trabajo de comparación posterior.

### 2.2.1 Encontrar los puntos característicos mediante FAST

El primer paso en la detección de características ORB es encontrar puntos clave en la imagen, que utiliza el algoritmo FAST.[18]

FAST es la abreviatura de *Features from Accelerated Segments Test*, que puede seleccionar rápidamente puntos clave. Los pasos del algoritmo son los siguientes[2]:

1. Determinar el valor del parámetro de umbral h para el punto característico seleccionado.

2. Para cualquier píxel p en la imagen, FAST compara los 16 píxeles en el círculo con el punto p como el centro. Si el valor del gris en el círculo es menor que lp - h (lp es el valor de gris del punto p) o hay más de 8 píxeles cuyo valor de gris es mayor que lp + h en total, entonces se selecciona el píxel p como punto clave.

Con los pasos anteriores, se puede saber que los puntos característicos clave determinados por FAST están ubicados en áreas con cambios rápidos en el nivel de gris, y tales áreas generalmente determinan algún tipo de borde.

En la siguiente figura 2.2.1 vemos un gato donde los puntos verdes son los puntos característicos de la imagen determinados por FAST.[18] El contorno de los ojos del gato y el área coloreada de la nariz en la figura son las áreas marginales de la imagen. Los bordes definen los límites del gato, así como los límites del área de la cara, por lo que estos puntos clave nos permiten identificar este gato en lugar de cualquier otro objeto o fondo en la imagen.

***Figura 2.2.1 Encontrar puntos característicos de fotos de gatos mediante el algoritmo FAST [18]***

### 2.2.2 Como el BRIEF genera descriptores de imagen

BRIEF es la abreviatura de *Binary Robust Independent Elementary Features[18]*. Su función es crear un vector de características binario basado en un conjunto de puntos clave, también conocido como descriptor de características binarias, que es un vector de características que contiene solo 1 y 0.

Como muestra la figura 2.2.2, cada punto clave en BRIEF se describe mediante un vector de características binarias, que normalmente es una cadena de 128 a 512 bits que contiene solo 1 y 0.

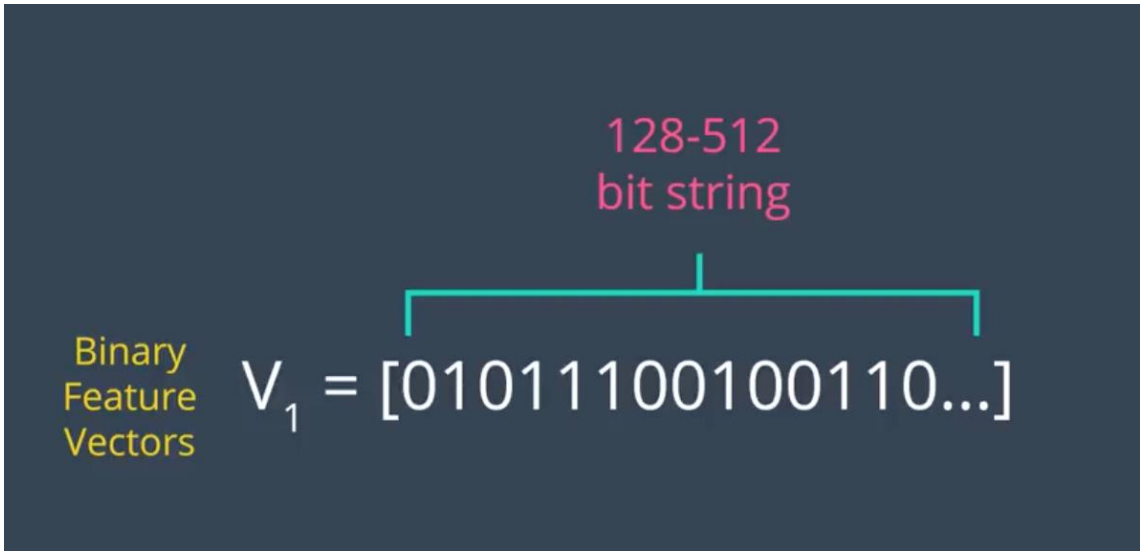



***Figura 2.2.2 Los descriptores de BRIEF [18]***

La mayor ventaja del algoritmo BRIEF para generar descriptores de funciones binarias es que se puede almacenar en la memoria de manera muy eficiente y se puede calcular rápidamente, y permite que BRIEF se ejecute en dispositivos con recursos informáticos muy limitados (como los móviles).

A través del algoritmo BRIEF, podemos describir la imagen utilizando los descriptores BRIEF, que es equivalente a describir y mantener la información de puntos característicos de la imagen en forma matemática. Al comparar la información de diferentes imágenes, podemos analizar las imágenes. En este trabajo lo más importante es analizar la similitud de las imágenes a través de la comparación de imágenes. Esta es también la base de realización de este trabajo.

### 2.2.3 Mejoras BRIEF para la rotación de escala

El algoritmo BRIEF puede generar descriptores de características binarias, pero no satisface la invariancia de escala y rotación de las imágenes, por lo que no se puede usar directamente en la coincidencia de imágenes con requisitos más altos. Especialmente teniendo en cuenta que no se puede garantizar que el patrón capturado en el entorno real esté en la posición correcta cada vez, y el ángulo de la cámara puede causar problemas como el estiramiento, la rotación y el plegado de la imagen.

Para superar esta deficiencia, el BRIEF se mejora de la siguiente manera:

**1. Invariancia de Escala**

Para que las características satisfagan la invariancia de escala, el algoritmo BRIEF crea una "pirámide de imagen". Una "pirámide de imagen" es una representación a múltiples escalas de una sola imagen, que consta de una serie de versiones de diferente resolución de la imagen original. [2]

Como se puede ver en la figura 2.2.3, la reducción de resolución de la imagen contiene menos píxeles y reduce el tamaño en un factor de 1/2.

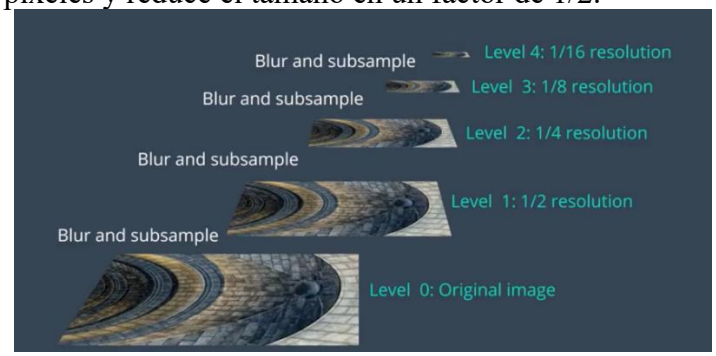



***Figura 2.2.3 "Pirámide de imagen" BRIEF [2]***

Una vez que ORB ha creado una “pirámide de imagen” como se puede ver en la figura 2.2.4, utiliza el algoritmo FAST para encontrar rápidamente puntos clave de imágenes de diferentes tamaños en cada nivel.

Dado que cada nivel de la pirámide consta de una versión más pequeña de la imagen original, los objetos de la imagen original también se reducen en tamaño en cada nivel de la pirámide. Al determinar los puntos clave en cada nivel, el ORB puede descubrir eficientemente puntos clave para objetos de diferentes tamaños, de modo que el ORB logre una invariancia de escala parcial.

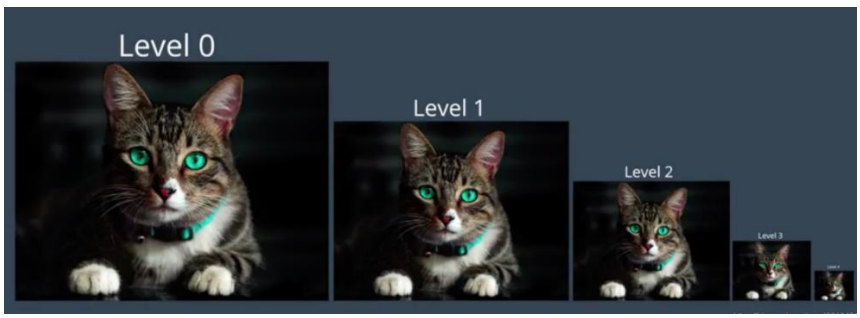



***Figura 2.2.4 Ejemplo de "Pirámide de imagen" [18]***

## 2. Invariancia de rotación

Para satisfacer la invariancia de rotación, ORB asigna a cada punto clave una orientación que depende de cómo varía la escala de grises alrededor de ese punto clave.

ORB selecciona una imagen en el nivel 0 de la "pirámide de imagen" y calcula la orientación de los puntos clave en esa imagen. Primero se calcula el centroide de la intensidad en un cuadro del tamaño especificado centrado en el punto clave. El centroide de intensidad se puede ver como la ubicación del nivel de gris de píxel promedio en un área determinada. Después de calcular el centroide de intensidad, se obtiene la orientación del punto clave dibujando un vector desde el punto clave hasta el centroide de intensidad.[19]

En la imagen 2.2.5 se muestra un ejemplo donde el punto clave es la dirección inferior izquierda, porque el brillo del área seleccionada aumenta en dicha dirección.

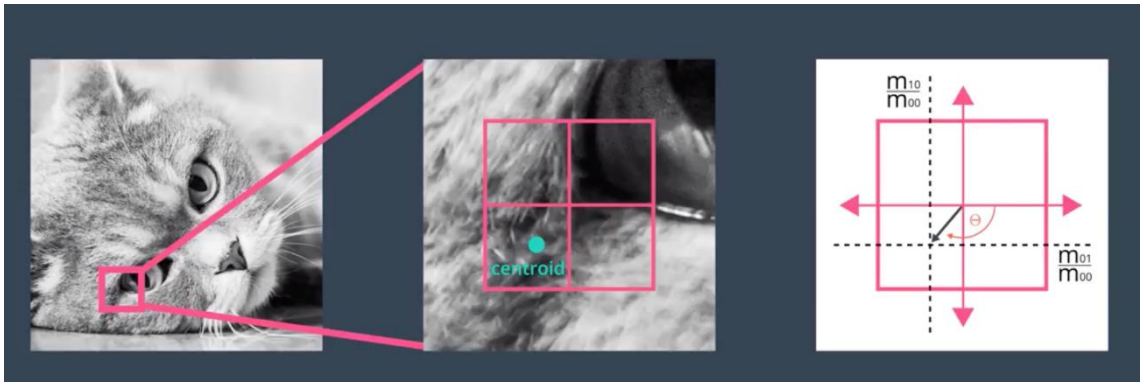



***Figura 2.2.5 La orientación de un punto clave[19]***

Después de asignar una orientación a cada punto clave en la imagen del nivel 0 de pirámide, ORB ahora se repite el mismo proceso para todas las demás imágenes de nivel de pirámide. Es importante tener en cuenta que, en cada nivel de pirámide de imagen, el tamaño del área alrededor de los puntos clave utilizados para confirmar la orientación no se reduce, por lo que la misma área cubrirá un área más grande de la imagen en cada nivel de pirámide, lo que resultará en diferentes tamaños de punto clave.

## 3. rBRIEF (*Rotation-Aware Brief*)

ORB ahora crea vectores de características usando una versión modificada de BRIEF, llamada rBRIEF (*Rotation-Aware BRIEF*). [1]

rBRIEF crea los mismos vectores para los puntos clave independientemente de la orientación del objeto, lo que hace que la rotación del algoritmo ORB sea invariable, lo que significa que puede detectar los mismos puntos clave en una imagen girada hacia cualquier ángulo. De esta manera, el rBRIEF es invariable sin importar cómo se amplíe y gire la imagen. Esto también sienta las bases para el trabajo posterior.[2]

## 2.3 Introducción de BFmatcher (Brute Force Matcher)

Después de describir la imagen usando el algoritmo ORB (*Oriented Fast and Rotated Brief*) como se describe en la sección anterior, en el segundo paso debemos hacer coincidir los puntos clave de la imagen.

En este trabajo se utiliza el algoritmo BFmatcher (*Brute Force Matche*r)[6] para el emparejamiento. BFmatcher es un método común para la coincidencia de puntos de características bidimensionales. BFmatcher siempre intenta todas las coincidencias posibles, de modo que siempre pueda encontrar la mejor coincidencia.

El método de procesamiento BFmatcher es muy simple. Primero se selecciona un punto clave en la primera imagen, luego se realiza la prueba de distancia del descriptor con cada punto clave de la segunda imagen, y finalmente se devuelve el punto clave más cercano. Básicamente, el proceso es el siguiente en curso:

1. Se extraen los vectores de características N y M de las dos imágenes respectivamente.

2. Se hacen coincidir los vectores de características de N y M para encontrar el mejor emparejamiento.

3. Se envía el resultado coincidente y se localiza el resultado en un píxel específico de la imagen.

Luego, combinado con el algoritmo de coincidencia ORB anterior, podemos optimizar todo el proceso en los siguientes pasos:

1. Como se muestra en la figura 2.3.1 se lee la imagen que se emparejará 1 y la imagen que se emparejará 2 en forma de imágenes en escala de grises.


***Figura 2.3.1 Las dos imágenes entradas de gris***

2. Como se muestra en la figura 2.3.2 los puntos característicos son detectados por el algoritmo ORB, y los puntos claves obtenidos de las dos imágenes se almacenan en sus correspondientes vectores de descriptores.

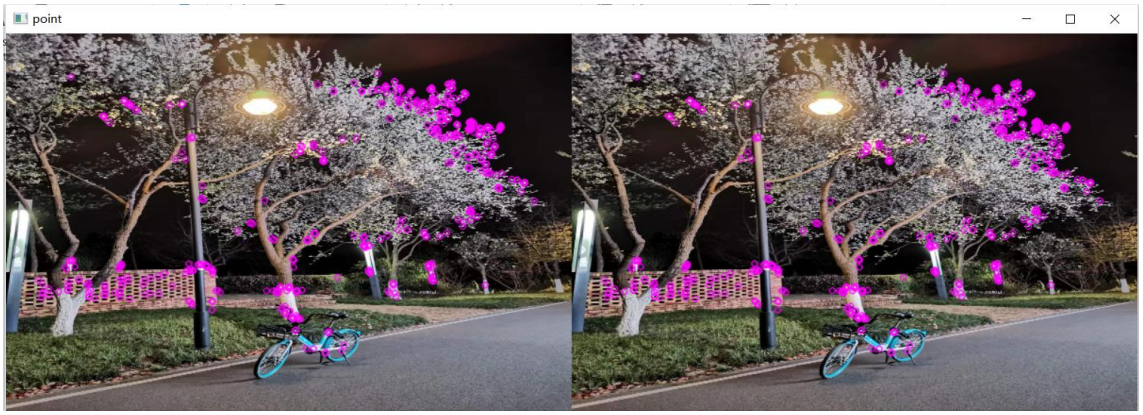



*Figura 2.3.2 Los puntos de clave de ORB*

3. Según los dos vectores del descriptor que se han extraído en el paso anterior, el mejor emparejamiento se realiza a través de BFmatcher y se almacena en DMatch.

4. Como se muestra en la figura 2.3.3 se define la imagen coincidente de salida y, a continuación, se utiliza el método drawMatches para dibujar los puntos característicos y los resultados coincidentes en las dos imágenes en Coincidencias y visualizarlos.

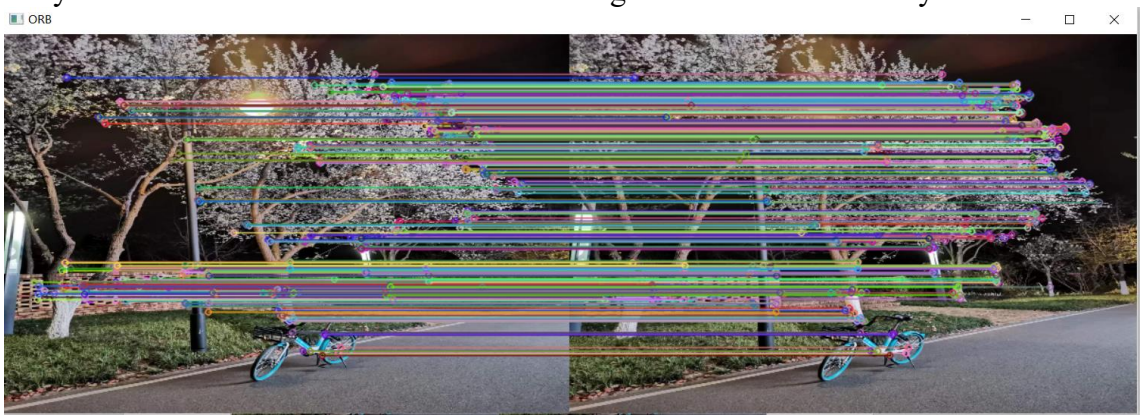



*Figura 2.3.3 La imagen salida de BFmatcher*

## 2.4 Introducción al preprocesamiento de la imagen

Dado que se necesita procesar imágenes reales con frecuencia, se debe considerar el efecto del ruido en este trabajo. Para evitar la influencia del ruido en el algoritmo ORB, se deben realizar varios preprocesamientos en la imagen para eliminar el ruido.

### 2.4.1 Filtro mediano de luminancia (medianBlur)

En una escena de impresión 3D real, debido a las características de algunos materiales de impresión, (colores y configuraciones de la cámara), a menudo puede haber reflejos serios o leves en las fotos tomadas.

De acuerdo con la descripción del algoritmo ORB, este tipo de cambio de brillo rápido, abrupto y extremo se identificará como el punto clave de la imagen, luego, en la primera parte, se debe reducir la magnitud del cambio de brillo. Para reducir el efecto del cambio de brillo en este proyecto se usa un filtro mediano.[21]

El filtro mediano es un método de filtrado no lineal. El filtro mediano consiste en reemplazar el valor central con el valor mediano de los píxeles en el cuadro de convolución para lograr el propósito de eliminar el efecto del ruido. La imagen procesada por el filtro mediano será más suave y el efecto del ruido en la imagen se reducirá en gran medida. Se reemplaza el valor del píxel actual con el valor mediana de todos los valores de píxel en el vecindario.[21]

El filtro mediano tomará los valores del píxel actual y los píxeles adyacentes que lo rodean (hay un número impar de píxeles en total), ordenará estos valores de píxel y luego usará el valor de píxel en el medio como el valor de píxel actual.

Por ejemplo, para la siguiente matriz de la figura 2.4.1:

| | | | | | | |
|---|---|---|---|---|---|---|
| 55 | 58 | 22 | 55 | 22 | 60 | 168 |
| 123 | 17 | 66 | 33 | 77 | 68 | 14 |
| 47 | 22 | 97 | 95 | 94 | 25 | 14 |
| 68 | 66 | 93 | **78** | 90 | 171 | 82 |
| 69 | 99 | 66 | 91 | 101 | 200 | 192 |
| 98 | 88 | 88 | 45 | 36 | 119 | 47 |
| 88 | 158 | 3 | 88 | 69 | 211 | 234 |

***Figura 2.4.1 La matriz original[21]***

El procedimiento sería el siguiente[21],

1. Se establece el vecindario en un tamaño de 3x3 (el área de convolución), se ordena los valores de píxeles en su vecindario de 3x3 (se acepta el orden ascendente o descendente) y, después de ordenar en orden ascendente, el valor de la secuencia es [66,78,90 ,91, 93, 94, 95, 97, 101].

2. En esta secuencia, el valor en la posición central (también llamado valor mediana) es "93", así que se reemplaza el valor de píxel original 78 con este valor como el nuevo valor de píxel para el punto actual.

El efecto de filtro mediano es como se muestra en la figura 2.4.2:

| | | | | | | |
|---|---|---|---|---|---|---|
| 55 | 58 | 22 | 55 | 22 | 60 | 168 |
| 123 | 17 | 66 | 33 | 77 | 68 | 14 |
| 47 | 22 | 97 | 95 | 94 | 25 | 14 |
| 68 | 66 | 93 | **93** | 90 | 171 | 82 |
| 69 | 99 | 66 | 91 | 101 | 200 | 192 |
| 98 | 88 | 88 | 45 | 36 | 119 | 47 |
| 88 | 158 | 3 | 88 | 69 | 211 | 234 |

***Figura 2.4.2 La matriz de medianBlur[21]***

En la imagen 2.4.3 utiliza el filtro mediano de diferentes tamaños del área de convolución para procesar la misma imagen. El tamaño utilizado a la izquierda es 3, y el al derecho es 25. Se puede ver que, si el tamaño del área de convolución seleccionado es más grande, el efecto de filtrado será mejor, pero al mismo tiempo, los puntos característicos de la imagen se volverán discretos, lo que también afectará negativamente al algoritmo ORB. Entonces el tamaño del área de convolución es muy importante para el trabajo.

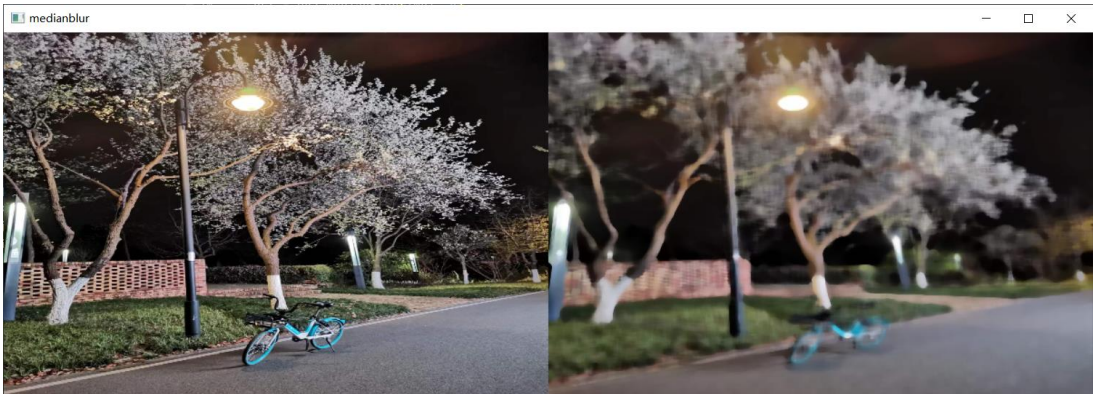



***Figura 2.4.3 El efecto de procesamiento del filtro mediano.***

### 2.4.2 Modelo de color HSV

HSV (*Hue, Saturation, Value*) es un espacio de color creado por A. R. Smith en 1978 basado en las características intuitivas del color, también conocido como Hexcone Model.[16]

Como se muestra en la figura de 2.4.4, los parámetros del color en este modelo son: Matiz (H), Saturación (S), Luminosidad (V).

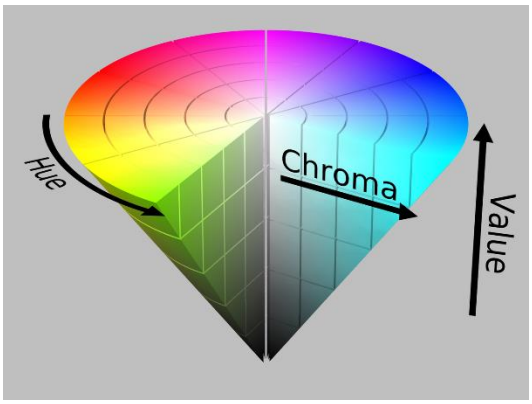



***Figura 2.4.4 Modelo de HSV[16]***

**Matiz (H)**

Medido por ángulo, el rango de valores es [0°,360°], comenzando desde el rojo y contando en sentido antihorario, el rojo es 0°, el verde es 120° y el azul es 240°. Sus colores complementarios son: amarillo a 60°, cian a 180° y morado a 300°.

**Saturación (S)**

La saturación S indica qué tan cerca está el color del color espectral. Un color que se puede ver como el resultado de un color espectral mezclado con blanco. Cuanto mayor sea la proporción de colores espectrales, mayor será el grado de color cercano a los colores espectrales, mayor será la saturación de color. La saturación es alta y el color es profundo y vivo. El componente de luz blanca del color espectral es 0 y la saturación es la más alta. Por lo general, el valor oscila entre 0% y 100%, cuanto mayor sea el valor, más saturado será el color.

**Luminosidad (V)**

La luminosidad indica el brillo del color. Para el color de la fuente de luz, el valor de luminosidad está relacionado con el brillo del iluminante; para el color del objeto, este valor está relacionado con la transmitancia o reflectancia del objeto. Por lo general, el valor oscila entre 0 % (negro) y 100 % (blanco).

### 2.4.3 Canny

Como se mencionó anteriormente, podemos evitar la influencia del ruido extrayendo los contornos de la imagen. Entonces, además de procesar los valores HSV, ¿hay una mejor manera de procesar imágenes? Después de buscar en diferentes fuentes, se llega a la conclusión de que se puede usar la función *Canny* para el procesamiento de imágenes.[14]

El detector de bordes *Canny* es un algoritmo de varios pasos para detectar bordes en cualquier imagen de entrada. Podemos obtener además el contorno de la imagen procesada, como se muestra a continuación:

1. **Eliminar el ruido de la imagen**

   Debido a que el borde de la imagen es muy vulnerable al ruido, para evitar detectar la información del borde incorrecto, la imagen generalmente se filtra para eliminar el ruido. El propósito del filtrado es suavizar algunas áreas que no son de borde con texturas débiles para obtener bordes más precisos. En el proceso de procesamiento real, generalmente se usa el filtrado gaussiano para eliminar el ruido en la imagen.

2. **Calcula el gradiente de los píxeles de la imagen**

   Como la dirección del gradiente de cada píxel siempre es vertical se puede juzgar aproximadamente la posición del borde de la imagen calculando la dirección del gradiente.

3. **Determinar si el píxel es borde**

   Se calcula si el pixel es el valor máximo de la misma dirección del gradiente en el punto del pixel circundante, en el caso de que se cumpla, es probable que los pixeles actuales sean los bordes de la imagen.

4. **Según la demanda se filtra el borde**

   De acuerdo con los pasos anteriores, se han obtenido los bordes de la imagen. Entre estos bordes, algunos son bordes reales de la imagen y otros son bordes introducidos por el ruido.
   En el siguiente paso, se pueden reservar algunos bordes según sea necesario. Se pueden establecer dos umbrales, uno es el umbral alto *maxVal* y el otro es el umbral bajo *minVal*.
   Si el gradiente del borde actual es mayor que el *maxVal* del borde alto, se retiene y se define como un "borde fuerte", y si es menor que el *minVal* del umbral bajo, se elimina. Si el gradiente está entre los dos, se determina si conservar el borde actual de acuerdo con si es adyacente a un borde fuerte.

En las figuras de 2.4.7 y 2.4.8, se muestra el efecto de la función *Canny*. El umbral utilizado es:

$$MaxVal = 150, minVal = 50$$

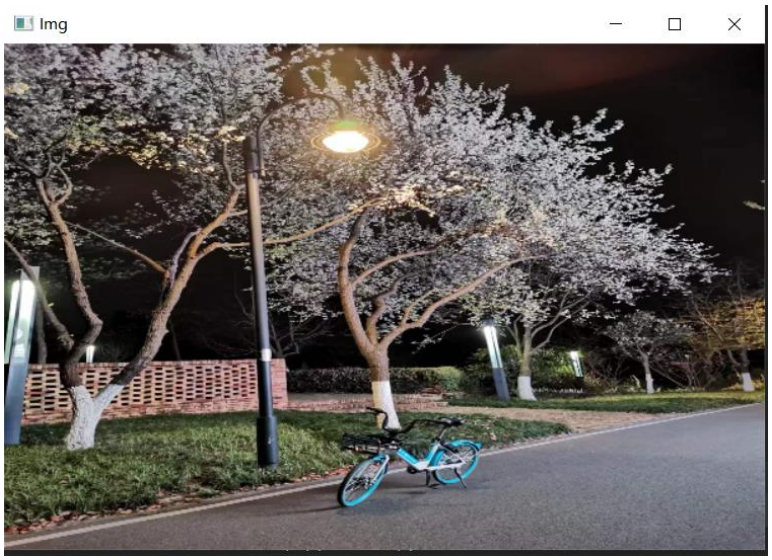



***Figura 2.4.5 Imagen Original***


***Figura 2.4.6 Canny***

La imagen que se obtiene es similar a la imagen filtrada por la función HSV, pero *Canny* permite extraer los contornos de toda la imagen de forma más completa. Combinado con el trabajo anterior, se obtiene mejores resultados de procesamiento. Se puede usar métodos de procesamiento de imágenes en combinación o usar un solo método de procesamiento según los requisitos para obtener el mejor efecto de procesamiento de imágenes.

# 3. Problemas y Pasos de resolución

Esta sección se centrará en el problema real de este trabajo, detectar los fallos de impresión 3D por medio de visión artificial, donde pasa solventarlos es necesario utilizar fórmulas matemáticas y soporte de software explicado en secciones anteriores.

Este capítulo discutirá específicamente los problemas y soluciones encontrados al realizar la comparación de imágenes impresas en 3D e imágenes prototipo en 4 partes:

- Clasificar los problemas a detectar
- Preprocesamiento de la imagen
- Detección y coincidencia de puntos característicos
- Pasos de resolución y la configuración

## 3.1 Clasificar los defectos a detectar

En primer lugar, se comienza a procesar la imagen real. Si se quiere procesar bien la imagen, el primer paso es observar la imagen real y analizar los defectos con las imágenes para intentar corregirlos o evitarlos a través de varios algoritmos.

Las siguientes fotos (Figura 3.1.1, Figura 3.1.2, Figura 3.1.3, Figura 3.1.4, Figura 3.1.5) fueron tomadas en escenas reales.

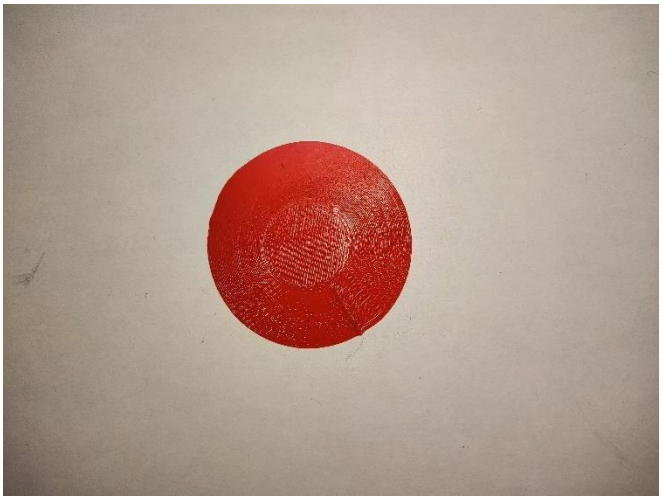


***Figura 3.1.1 Foto 1***

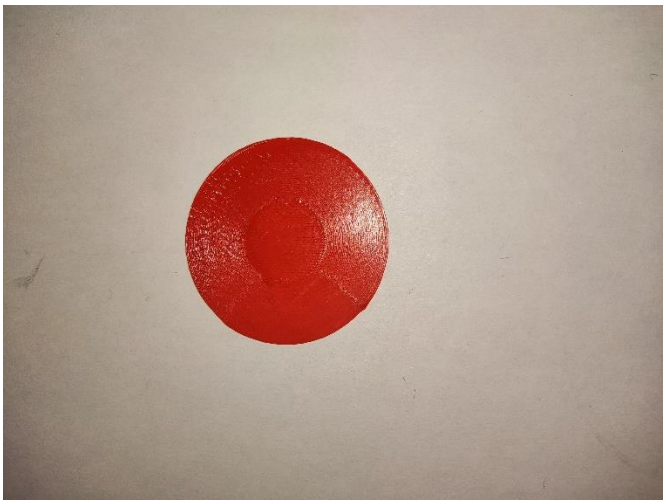


*Figura 3.1.2 Foto 2*

*Figura 3.1.3 Foto 3*

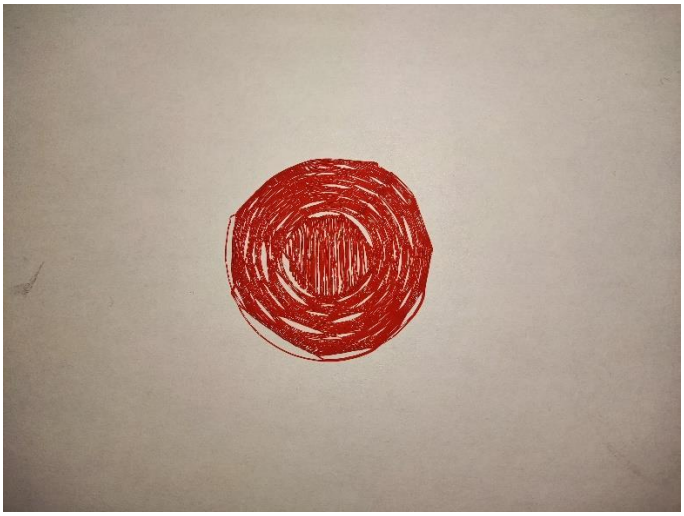


*Figura 3.1.4 Foto 4*

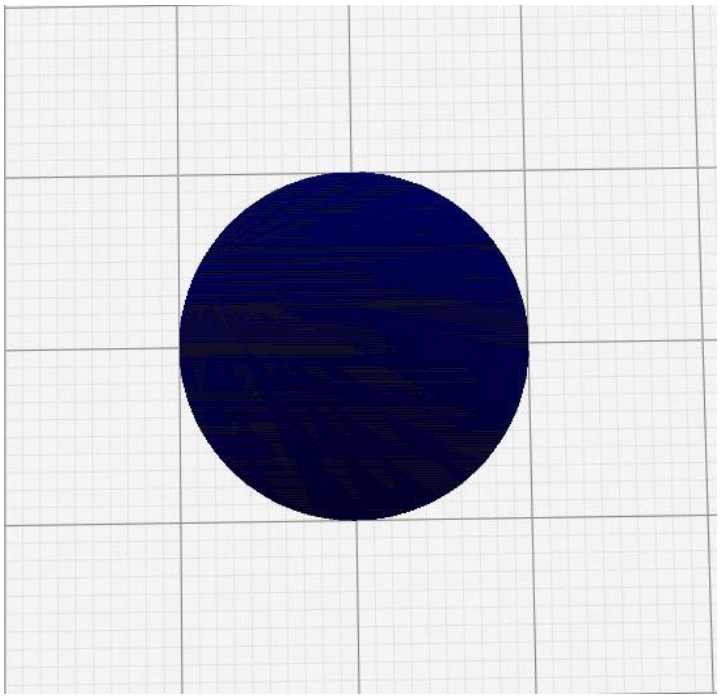


***Figura 3.1.5 Foto de Diseño***

En las 4 imágenes anteriores se detecta que si se realiza una comparación entre las diferentes fotos con la última imagen (esta es la original y el resultado deseado) 1 y 2 muestran buenos resultados, pero la impresión muestra un área reflectante de alto brillo, lo que conduce a un brillo desigual en todo el patrón. Los aspectos más destacados son el ruido de iluminación.

Las fotos 3 y 4 son claramente resultados de impresiones deficientes con varios fallos de impresión. Los fallos de impresión de estas dos imágenes son típicos. Hay muchas razones por las que la impresión puede fallar, por ejemplo:

- La boquilla de impresión 3D está bloqueada y el material de impresión no se puede expulsar normalmente
- La baja temperatura del banco de trabajo de la impresora 3D hace que el material de impresión no pueda plastificarse normalmente
- La impresora 3D carece de material de impresión, lo que resulta en material de eyección insuficiente.

La manifestación más típica de este tipo de error de impresión es la gran diferencia entre la imagen impresa y el diseño del prototipo. Esta diferencia generalmente se refleja en el cambio de forma de la imagen impresa, y deberíamos usar esto como punto de partida para el procesamiento de imágenes.

Al mismo tiempo, al comparar la imagen impresa y el diseño del prototipo, podemos observar que, ya sea que la impresión sea exitosa o no, existe una gran diferencia entre los dos. Esto está relacionado con las características de la impresión 3D. La impresión 3D consiste en depositar el material de impresión capa por capa a través de la boquilla de impresión. Cuando diseñamos prototipos de impresión 3D, diseñamos el objeto impreso como un objeto completo. En otras palabras, las impresoras 3D tienen una precisión limitada, lo que significa que debemos ignorar las diferencias causadas por la precisión de impresión en el procesamiento de imágenes posterior. fenómeno de la impresión 3D y no un fallo de impresión.

De acuerdo con la descripción anterior, se pueden extraer 3 conclusiones:

1. Hay una gran diferencia entre la imagen de impresión 3D y el diseño del prototipo, y las dos imágenes no se pueden comparar directamente. En la comparación de imágenes subsiguiente, es necesario ignorar esta parte de la diferencia en la imagen, que es causada por la precisión de impresión en lugar del error de impresión.
2. La mayoría de los fallos de impresión provocarán el cambio del contorno de la imagen. Extraer el contorno de la imagen puede ser más efectivo para la comparación de imágenes.
3. No importa cuán buena o mala sea la impresión, debemos procesar la imagen para minimizar el efecto del ruido de fondo.

## 3.2 Preprocesamiento de la imagen

En esta sección, se discute específicamente cómo se preprocesan las imágenes.

En primer lugar, se han determinado dos puntos de preprocesado de imagen según el apartado anterior:

1. Reducir el impacto del ruido en las imágenes impresas en 3D.
2. Extraer los contornos de las imágenes impresas en 3D y dibujos de prototipos.

### 3.2.1 Cambiar el tamaño de la imagen

Para el primer paso del procesamiento de imágenes, se debe recortar la imagen e intentar mantener solo el área del patrón impreso. Esto se puede descartar como ruido de fondo en el área de impresión.

Hay muchas soluciones para recortar imágenes, la forma más fácil es configurar la cámara para que solo tome fotografías del área impresa, o puede programar un área fija para recortar. En este trabajo se establece un área fija de la imagen recortada para obtener mejores resultados.

### 3.2.2 Reducir el efecto del ruido

El segundo paso debería reducir el efecto del ruido en el área de impresión de la imagen. En la imagen de la sección anterior, se puede observar que el principal ruido en el área de impresión es el área de alto brillo, que se relaciona principalmente con la reflectividad de la luz del material impreso y las condiciones de iluminación. Además de cambiar el material de impresión y ajustar la posición e intensidad de la fuente de luz, también es una buena opción para procesar la imagen mediante filtro mediano.

Sin embargo, lo que se debe entender es que el uso del filtrado mediano para procesar imágenes inevitablemente dará como resultado que parte de la información de la imagen se filtre. Como se ilustra en la figura 3.2.1 que se utiliza el valor del área de convolución de 7, 11, 15, 21 desde izquierda a la derecha, la expresión específica en la imagen es que la imagen se vuelve más borrosa. Cuanto mayor sea el área de convolución que se elija para el filtrado mediano, mayor será el impacto en la imagen y mejor será el efecto del filtrado de ruido, por lo que debemos elegir el área de convolución hacer una elección.


***Figura 3.2.1 El filtrado mediano con diferente área de convolución***

Las figuras siguientes (Figura 3.2.2, 3.2.3, 3.2.4 y 3.2.5) son los resultados de rendimiento de cada foto impresa en diferentes áreas de convolución, de izquierda a derecha, las áreas de convolución son 7, 11, 15 y 21, respectivamente. Y se usó uniformemente la función *Canny* para extraer el contorno de la imagen. Cuanto mayor sea el área de convolución, mejor será el efecto de filtrado de ruido y mayor será el impacto en la imagen original.

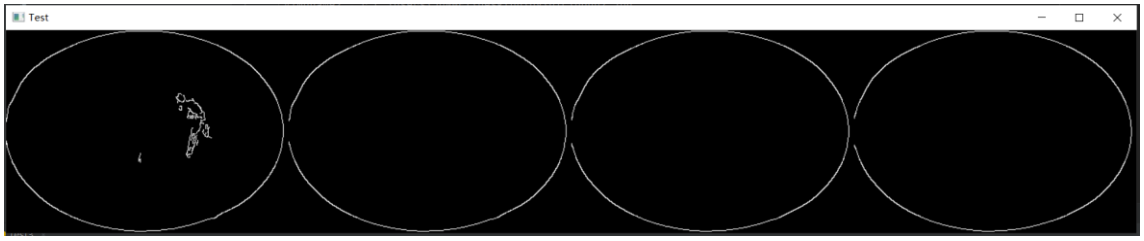



***Figura 3.2.2 Foto 1***

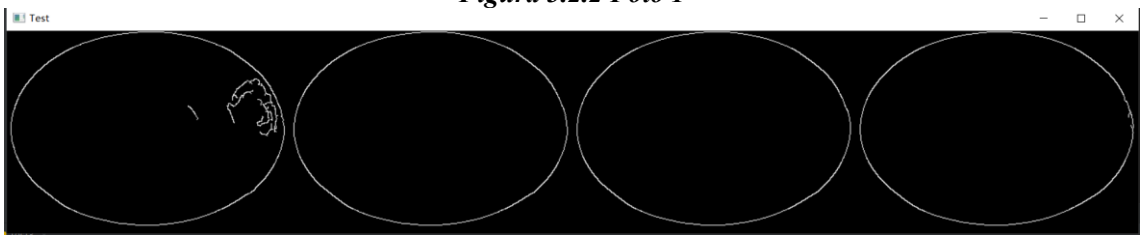



***Figura 3.2.3 Foto 2***

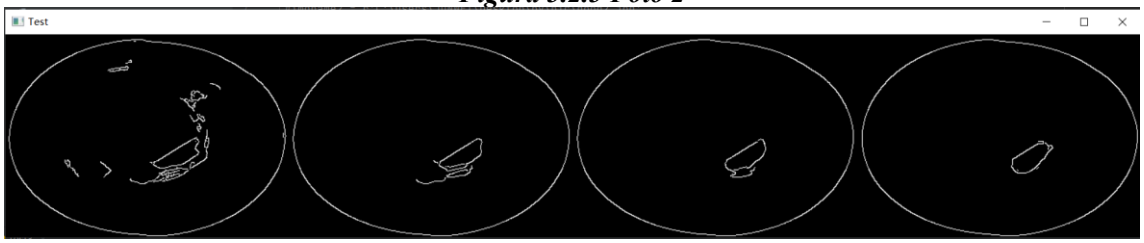



***Figura 3.2.4 Foto 3***

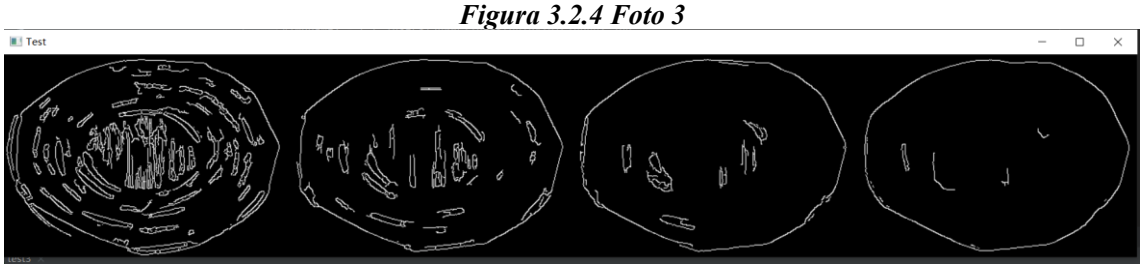



***Figura 3.2.5 Foto 4***

A partir de los resultados experimentales anteriores, se puede concluir que cuando el área de convolución es pequeña, la imagen puede retener más bordes de contorno para facilitar el trabajo de extracción de contorno posterior.

Cuando el área de convolución es grande, el rendimiento del filtrado de imágenes contra el ruido es mejor. Cuando el área de convolución alcanza de 11 a 15, el ruido se filtra casi por completo y se pueden conservar más bordes de contorno de los defectos de impresión. Se elige el número medio 13.

### 3.2.3 HSV

El tercer paso, debe ser extraer los contornos de la imagen. Después del procesamiento previo de la imagen en la parte anterior, se puede obtener una imagen relativamente limpia y sin ruido. Como se mencionó en la sección anterior, las imágenes se pueden procesar utilizando el modelo HSV o la función *Canny*. Dado que el fondo es blanco la mayor parte del tiempo cuando se imprime en 3D, Los fondos blancos o de colores claros suelen tener valores de tonalidad más bajos, por lo que se puede usar esta función para filtrar el fondo. Se suele usar HSV para separar la parte de la imagen impresa en la foto para obtener el contorno básico de la imagen.

El siguiente es el flujo de la operación:

1. Como se muestra en la figura 3.2.2 convertir imagen a imagen HSV.

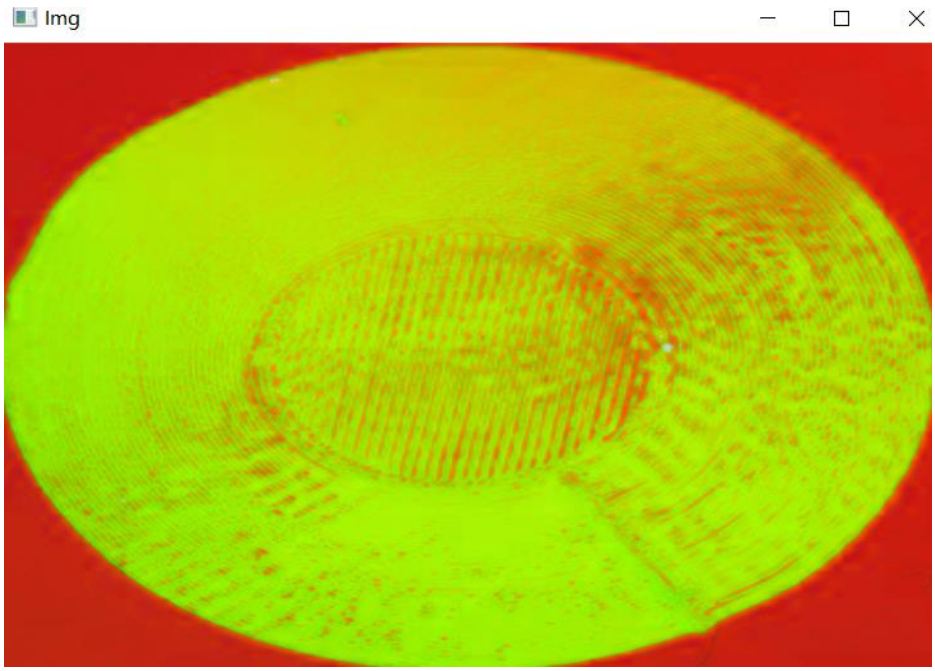



***Figura 3.2.6 Imagen de HSV***

2. Establecer el valor del filtro HSV

3. Como se muestra en la figura 3.2.3 filtrar valores de imagen

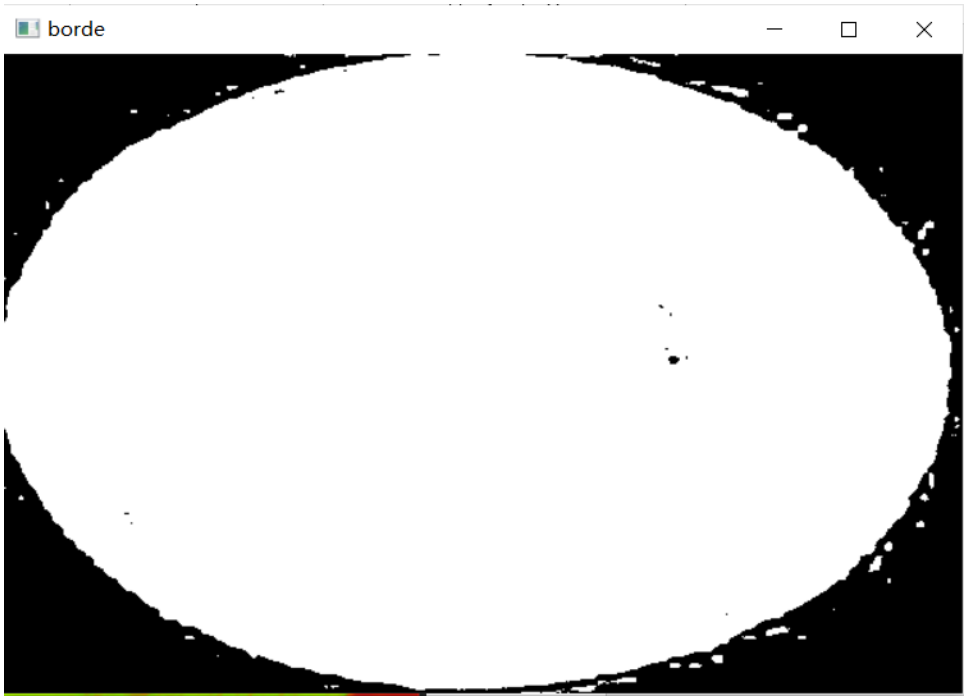



***Figura 3.2.7 Imagen Filtrado***

A través de los métodos anteriores, se puede obtener fácilmente el contorno de la imagen. Dado que el algoritmo de detección de puntos característicos trata con imágenes en escala de grises, no es sensible al color. Por lo tanto, se puede realizar un trabajo de seguimiento utilizando este resultado.

Se puede observar claramente el error de impresión en la figura 3.2.7. Pero hay dos problemas principales con este perfil:

1. La imagen de no logra resolver el efecto de ruido de la iluminación, y dentro hay un ruido que no debería existir.

2. Aparece ruido en los bordes exteriores de patrones y la transición no es suave.

Para resolver estos dos problemas, se puede adoptar el método de filtrado mediano para suavizar el borde de la imagen y suavizar la influencia del ruido de iluminación en la imagen y minimizar el ruido causado por cambios de brillo excesivos como las figuras de 3.2.8 se muestra.

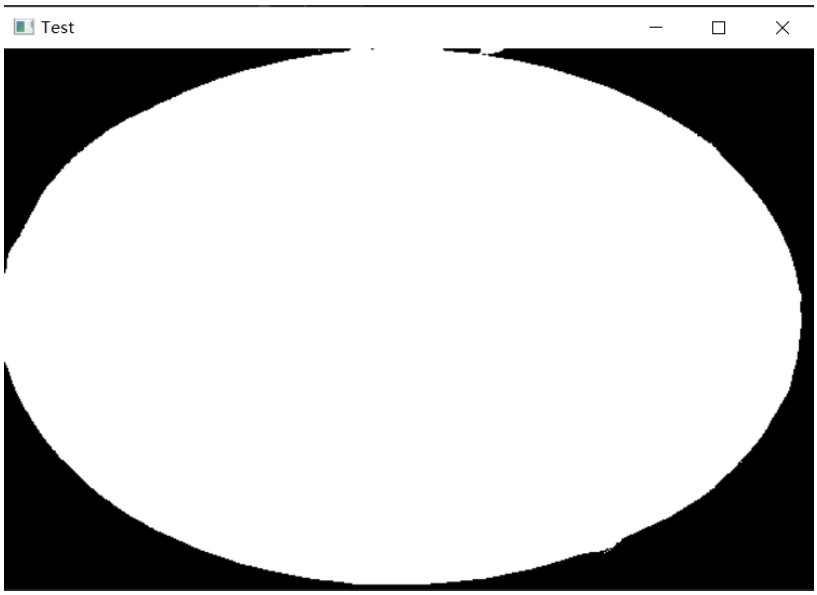



*Figura 3.2.8 El contorno de imagen con filtrado mediano*

Después de procesar los contornos extraídos por el método HSV, la imagen solo tendrá dos colores, 0 (negro) y 1 (blanco), que no se verán afectados por el efecto de brillo. Cabe señalar que HSV filtra y extrae el contorno del patrón a través de los diferentes tonos de la imagen impresa y el fondo. Si el color del material de impresión y el fondo son muy parecidos, el contorno de la imagen no se puede extraer a través de diferentes tonos como se muestra la figura 3.2.4 y 3.2.5.

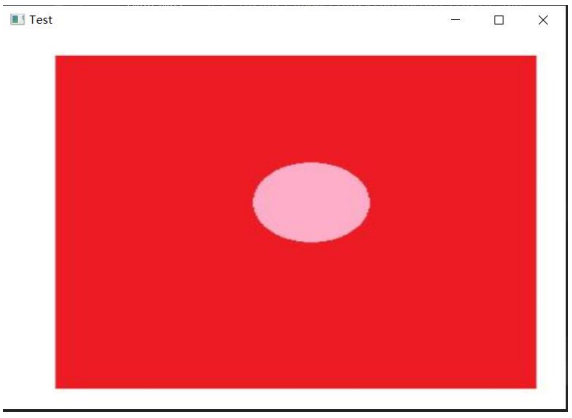



*Figura 3.2.9 La función HSV no puede extraer contornos en condiciones complejas 1*

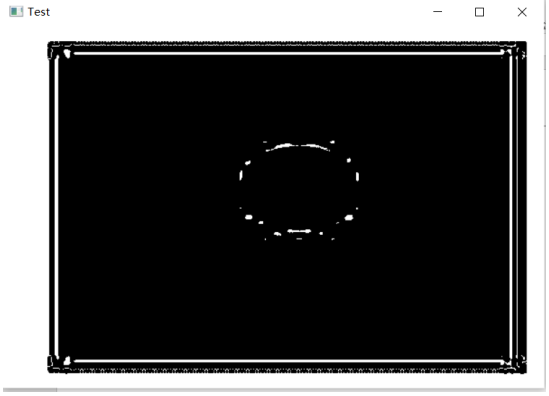



*Figura 3.2.10 La función HSV no puede extraer contornos en condiciones complejas 2*

### 3.2.4 Canny

A diferencia de la función HSV, la función *Canny* extrae los contornos de la imagen con una precisión muy alta. Y debido a que el principio de funcionamiento de la función *Canny* es extraer el contorno de la imagen calculando el gradiente del punto de píxel, no recibirá la interferencia del color de fondo[14]. Como se muestra en la figura 3.2.11, la función *Canny* puede extraer contornos de imágenes en escenarios donde la función HSV no funciona correctamente.

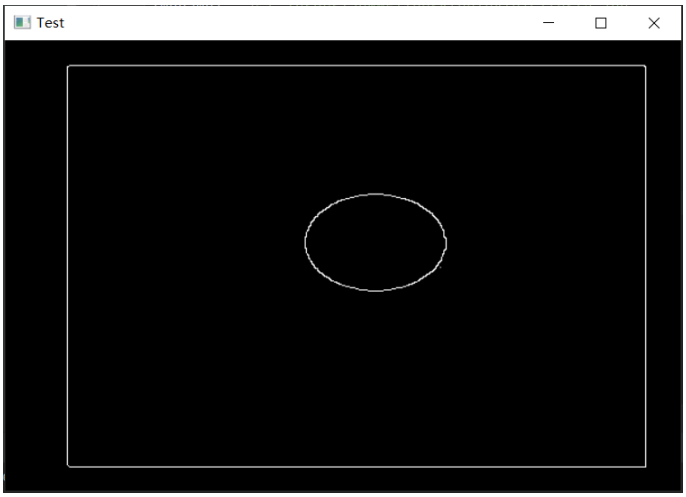



***Figura 3.2.11 La función Canny se puede extraer contornos en condiciones complejas***

Como se muestra en la figura 3.2.12 y 3.2.13, en la escena real, también existe una brecha significativa entre el desempeño de la función HSV y la función *Canny*. En la figura 3.2.13 la parte marcada es la parte de extracción de contorno sin éxito de HSV. Por lo general, los detalles del contorno extraídos por la función *Canny* son mejores. Debido a que los bordes del contorno son claramente visibles, la función *Canny* también puede extraer bordes para errores pequeños, pero la función HSV puede extraer bordes incompletos o ignorar defectos más pequeños.

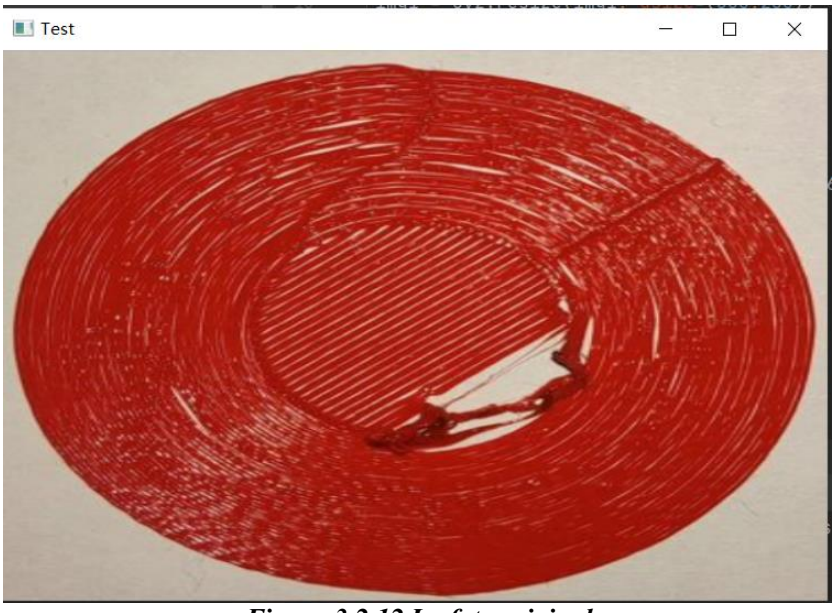



***Figura 3.2.12 La foto original***

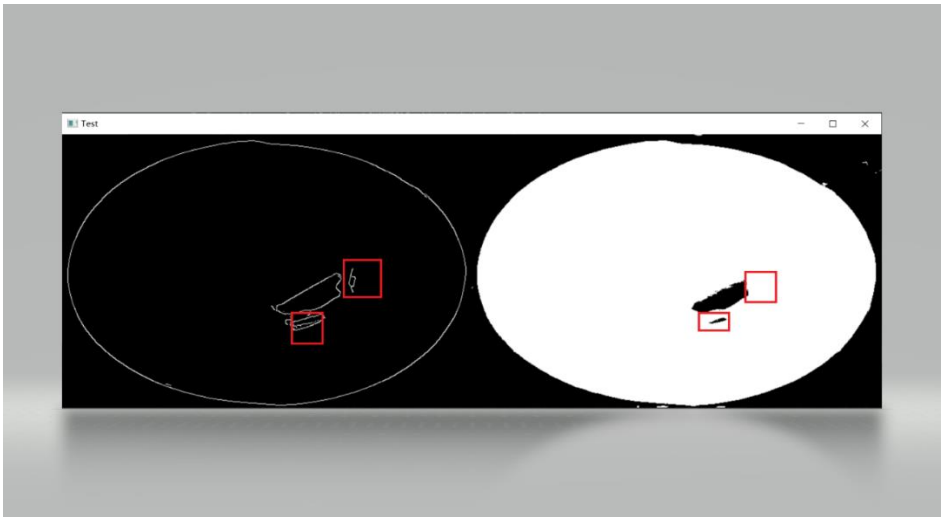



***Figura 3.2.13 Comparación de la función Canny y la función HSV***

De acuerdo con la comparación anterior, se puede concluir que la función *Canny* es la más eficiente para la extracción del contorno de la imagen, y la función *Canny* solo extrae los bordes del contorno, y HSV conducirá a la pérdida de información de la imagen (algunos defectos de impresión no se corrigen con éxito). Por lo tanto, en el siguiente trabajo se utilizará preferentemente la función de *Canny* para extraer contornos.

El siguiente es el problema del umbral de la función de *Canny*. Como se describe en la sección anterior, el umbral de la función *Canny* hace que se filtren algunos bordes. Como se muestra en las Figuras 3.2.14, 3.2.15, 3.2.16 y 3.2.17, los umbrales [50, 100], [50, 150], [50, 200] y [100, 200] se utilizan de izquierda a derecha. justo en la imagen, respectivamente.

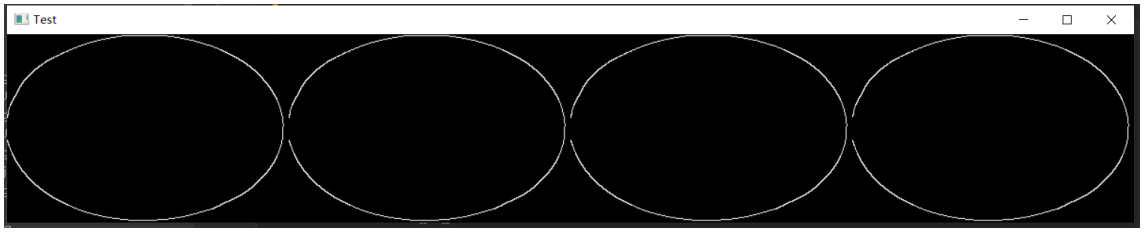



***Figura 3.2.14 Foto 1***

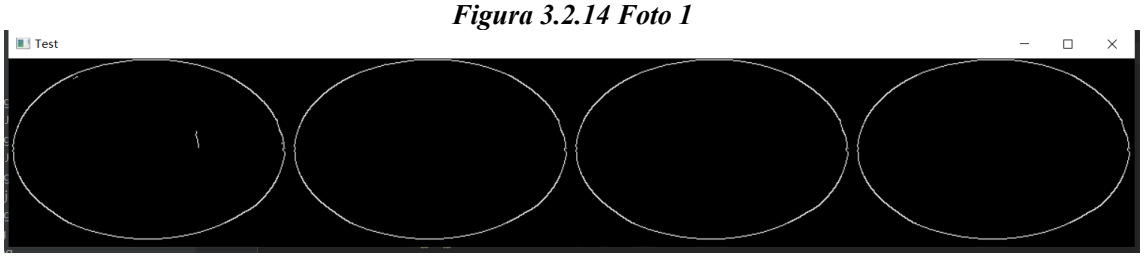



***Figura 3.2.15 Foto 2***

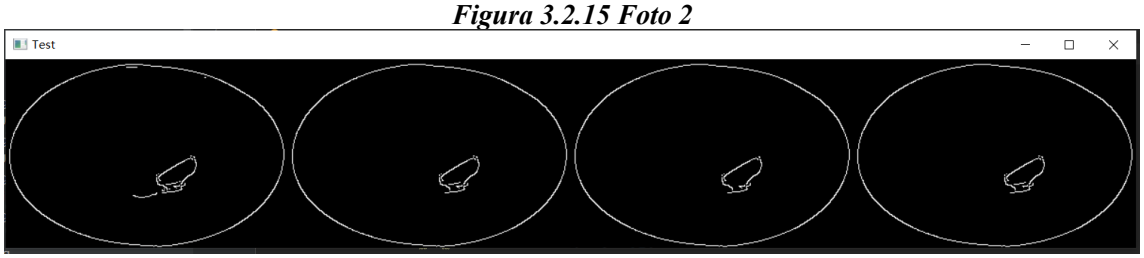



***Figura 3.2.16 Foto 3***

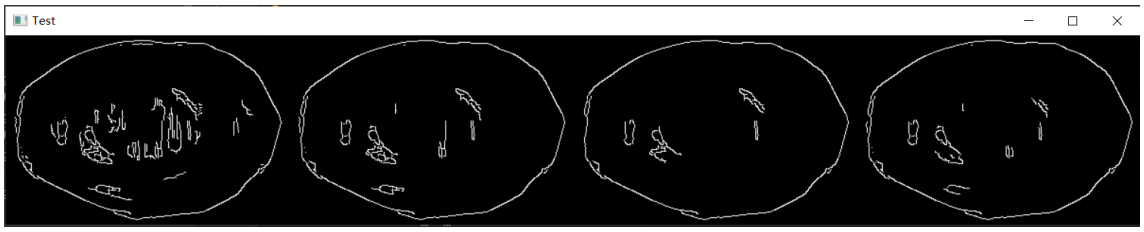



***Figura 3.2.17 Foto 4***

Como se muestra en las Figuras 3.2.14, 3.2.15, 3.2.16 y 3.2.17, para conservar más características de borde para facilitar el trabajo posterior, el umbral superior no debe ajustarse demasiado alto. Al mismo tiempo, un valor de umbral superior demasiado bajo hará que algunos ruidos se detecten erróneamente como bordes. Al mismo tiempo, si el umbral inferior es demasiado alto, algunos bordes no se pueden detectar, y si el umbral inferior es demasiado bajo, el resultado de la detección se verá afectado por el ruido. Combinando el rendimiento de diferentes umbrales, la configuración de umbral de [50, 150] funciona bien en todos los aspectos.

## 3.3 Detección y coincidencia de puntos característicos

Como se describe en la sección anterior, la ampliación de la imagen, la rotación, etc. ocurren con frecuencia en la escena de este trabajo. y como se muestra en la figura 2.2.1 y 2.2.2, el algoritmo ORB funciona mejor que SIFT y SURF cuando la imagen está ampliada o la imagen tiene ruido. Por lo tanto, este trabajo utilizará el algoritmo ORB como algoritmo de detección de puntos característicos de la imagen.

Y para la coincidencia de puntos característicos*, BFmatcher* es muy útil*. BFmatcher* solo puede hacer coincidir los puntos característicos de dos imágenes, y solo se conoce los puntos coincidentes no indica cuán similares son las imágenes. Si desea cuantificar el grado de coincidencia de dos imágenes a través de las matemáticas, debe contar la proporción de puntos de características coincidentes con éxito en todas las características.

$$La\ similitud = \frac{El\ número\ de\ puntos\ de\ características\ coincidentes}{El\ número\ de\ todos\ \ puntos\ de\ características}$$

***Formula 3.3.1 La similitud***

Después de obtener la similitud de las imágenes, la similitud se puede usar para medir si dos imágenes son similares. Si la similitud de la imagen es muy alta, prueba que las dos imágenes son muy similares y existe una gran posibilidad de que no haya fallos de impresión.

Si la similitud es baja, prueba que las dos imágenes son definitivamente dos imágenes y existe una mayor posibilidad de fallos de impresión. ¿Y qué similitud se utiliza como punto de división entre la similitud y la disimilitud de la imagen? Se deben considerar varios factores y se pueden cambiar de acuerdo con diferentes escenarios. Los factores a considerar son:

1. De acuerdo con los datos de la sección de 2.2, el límite superior del grado de coincidencia del punto característico calculado por el algoritmo ORB es generalmente de alrededor del 50 %, por lo que el punto de división de similitud no debe exceder el 50 %. De lo contrario, causará el problema de que las imágenes calificadas no pueden pasar la prueba de comparación.

2. *BFmatcher* generará coincidencias falsas, que representan una pequeña proporción, pero tendrán un cierto impacto en el resultado final. Por lo tanto, el punto de corte de similitud no debe establecerse en un valor más bajo.

Como se muestra en la figura 3.3.1, la similitud utilizada para la medición se puede establecer de acuerdo con los datos existentes. Los datos existentes se obtienen en la sección 4.4

| Imagen | Número de coincidencias exitosas | Número total de coincidencias | Similitud |
|---|---|---|---|
| Foto 1 | 99 | 255 | 39% |
| Foto 2 | 91 | 216 | 42% |
| Foto 3 | 30 | 416 | 7.2% |
| Foto 4 | 4 | 500 | 0.8% |

***Figura 3.3.1 Similitud de diferentes imágenes y dibujos de prototipos.***

En los datos anteriores, las fotos 1 (Figura 3.1.1) y 2 (Figura 3.1.2) son imágenes con impresión 3D normal, y las fotos 3 (Figura 3.1.3) y 4 (Figura 3.1.4) son imágenes con impresión 3D defectuosa. Es obvio que hay una gran diferencia en la similitud entre los dos, y debido a las fallas en las imágenes impresas de la Foto 3 y la Foto 4, hay más contornos en las imágenes, lo que significa que hay más puntos característicos en las imágenes.

Basándose en los datos de esta tabla y los resultados discutidos en el capítulo anterior, se puede usar un punto de corte del 35 % para la similitud. Para imágenes con una similitud superior al 35 %, se puede considerar como un resultado de salida de imagen PASS; para imágenes con una similitud inferior al 35 %, se puede considerar como una impresión de imagen fallida y el resultado de salida es NO PASS.

## 3.4 Pasos de resolución y la configuración

En esta sección, se resumirán un conjunto de procedimientos de trabajo y resultados esperados con base en el soporte de datos proporcionado en las secciones anteriores. Y la configuración de datos relevante se proporciona al final de esta sección.

### 1. Tomar la foto de impresión

Como se muestra en la figura 3.4.1, se tomará una foto directamente desde arriba de la imagen impresa para usarla en el trabajo posterior. Esta foto se utilizará como base para juzgar si la impresión es defectuosa.

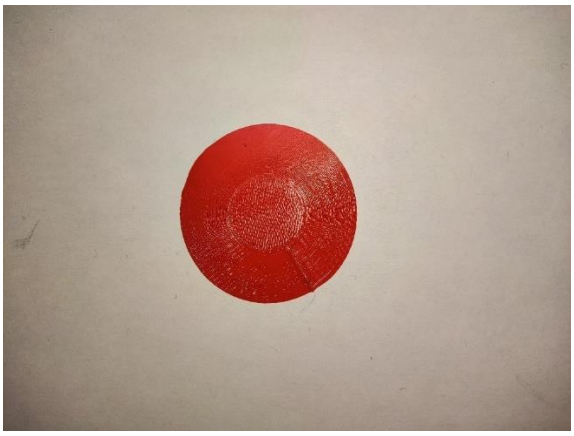


*Figura 3.4.1 El ejemplo de la foto de la imagen impresa*

### 2. Encontrar el tamaño de la foto

Como se muestra en la figura 3.4.2, la foto debe recortarse para evitar que el ruido de fondo interfiera con áreas de la imagen impresa.

*Figura 3.4.2 El ejemplo del recorte el tamaño de la imagen impresa*

### 3. Utilizar el filtro mediano para eliminar el ruido

Como se muestra en la figura 3.4.3, se utiliza el filtrado mediano para filtrar la mayoría ruido hace que la imagen sea adecuada para el trabajo posterior. El valor del área de convolución utilizado es 13.


***Figura 3.4.3 El ejemplo del filtro mediano para eliminar el ruido***

**4. Utilizar la función *Canny* para extraer los contornos de la imagen**

Como se muestra en la figura 3.4.4, se utiliza la función *Canny* para extraer los contornos de la imagen. De acuerdo con la Sección 3.2.4, el umbral [50, 150] se utiliza como la configuración de la función Canny. Por debajo de este umbral, la función Canny puede extraer la mayoría de los contornos de la imagen y se ve menos afectada por el ruido.

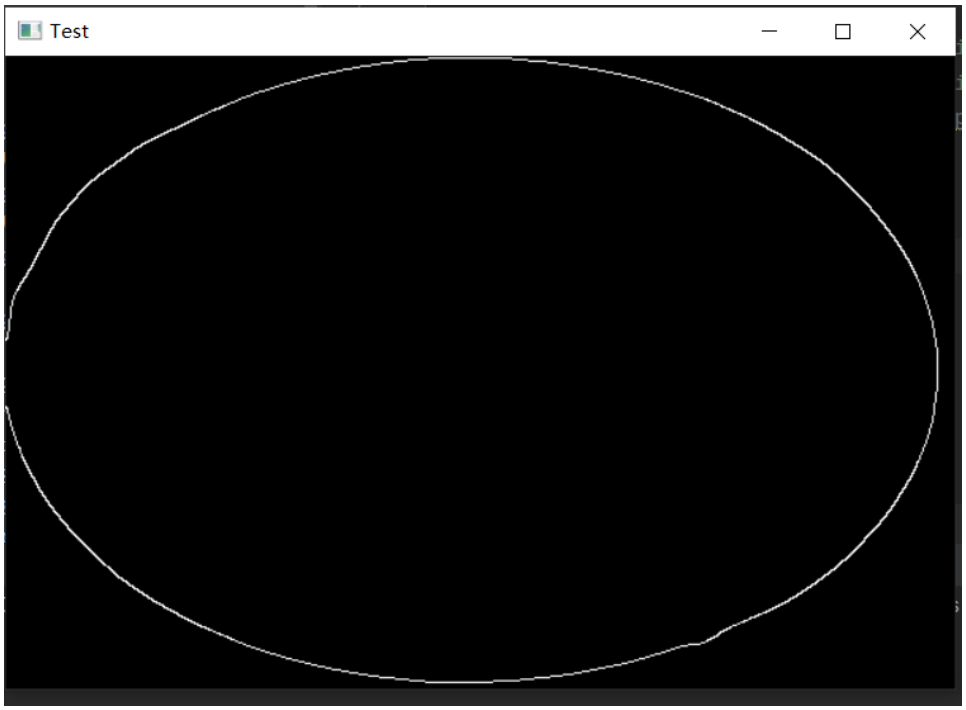



***Figura 3.4.4 El ejemplo de la función de Canny***

**5. Se calcula la similitud y se determina si hay defecto**

Como se muestra en la figura 3.4.5, se utiliza la función de ORB para determinar los puntos característicos de la foto de imágenes impresas y dibujos de prototipos. A la

izquierda está la imagen impresa, y a la derecha está el contorno del diseño del prototipo.

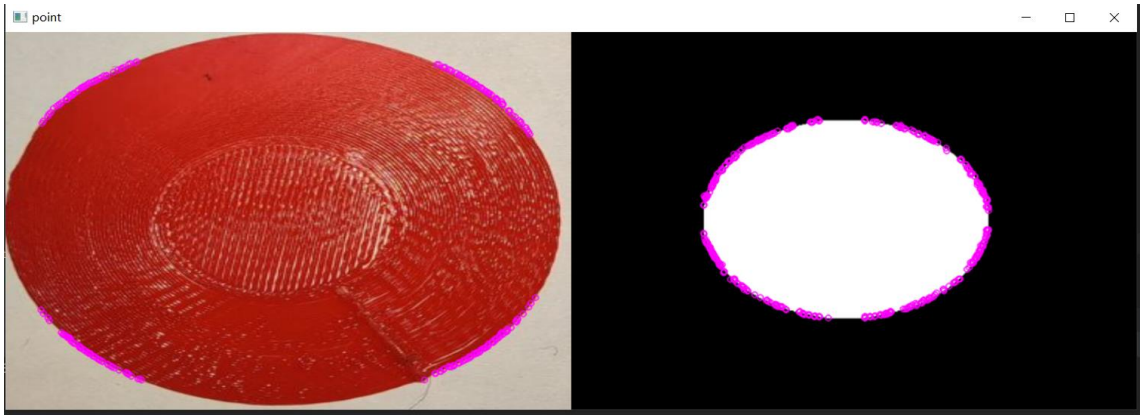



***Figura 3.4.5 El ejemplo de la función de ORB***

Como se muestra en la Figura 3.4.6, se utiliza la función BFmatcher para hacer coincidir los puntos característicos de las dos imágenes. Y según la similitud, se juzga si la imagen impresa es defectuosa o no. De acuerdo con los datos de la Sección 3.3, se utiliza una similitud del 35 % como criterio de evaluación. Por encima de este estándar, la impresión se considera libre de defectos y por debajo de este estándar, la impresión se considera defectuosa.

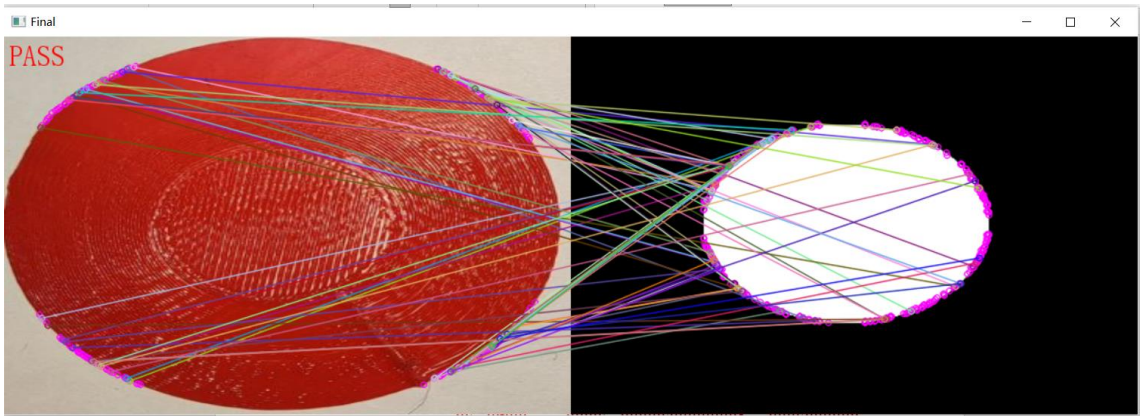



***Figura 3.4.6 El ejemplo de la determinación si hay defecto***

Todas las configuraciones y funciones utilizadas se enumeran en la Tabla 3.4.1.

| Los nombres de las funciónes | Los nombres de las configuraciónes | Los datos |
|---|---|---|
| El tamaño de la foto | El tamaño de la foto | 600*400 pixeles |
| El filtro mediano | El área de convolución | 13 |
| *Canny* | [MinVal, MaxVal] | [50, 150] |
| Determinación de defecto | La similitud | 35% |

***Tabla 3.4.1 las configuraciones y funciones utilizadas***

# 4. Resultado

Las secciones anteriores proporcionan una solución completa con las fotos de ejemplo. A continuación, se describe la solución y se presentarán los resultados obtenidos.

## 4.1 Posición de la cámara

Se va a utilizar una solución de visión artificial pura para detectar errores en la impresión 3D. La ubicación donde se tomó la foto tendrá un cierto impacto en el resultado de la detección, se necesita planificar la posición de la cámara razonablemente de acuerdo con las características de la impresión 3D.

Como se muestra en la figura 4.1.1, en primer lugar, la mayoría de las impresiones 3D se realizan en capas independientes del prototipo, la impresora 3D siempre dividirá el diseño en varias capas, comenzando desde la capa inferior e imprimiendo hasta la capa superior. Siempre que la impresión sea correcta y sin errores, cada capa debe imprimir el patrón plano correcto.

***Figura 4.1.1 Esquema de impresión 3D***

Debido a que la impresión 3D se realiza capa por capa, la mejor opción es la detección directamente desde arriba de la impresora ya que se puede fotografiar y detectar el efecto de cada nivel de impresión y mostrar los resultados.

El problema de ubicar la cámara desde arriba es que a medida que avanza la impresión 3D, el prototipo impreso irá teniendo altura gradualmente, y la distancia a la cámara también será menor. Luego, la imagen del prototipo tomada aparecerá más grande debido a la perspectiva, pero el dibujo de diseño no tendrá el efecto correspondiente.

¿Afectará esto a la detección de puntos característicos?

La respuesta es que afectará, pero usar el algoritmo ORB no tiene ningún efecto. Porque ORB tiene invariancia de escala e invariancia rotacional. Esto significa que, en

teoría, el escalado y la rotación de la imagen no tendrán ningún efecto en la detección de puntos característicos.

Como se muestra en la figura 4.1.2, debido a que la impresora 3D ejecuta los comandos de impresión capa por capa, se debe intentar tomar una foto de la imagen impresa en este ángulo, para que se pueda observar más claramente la situación de impresión de cada capa.

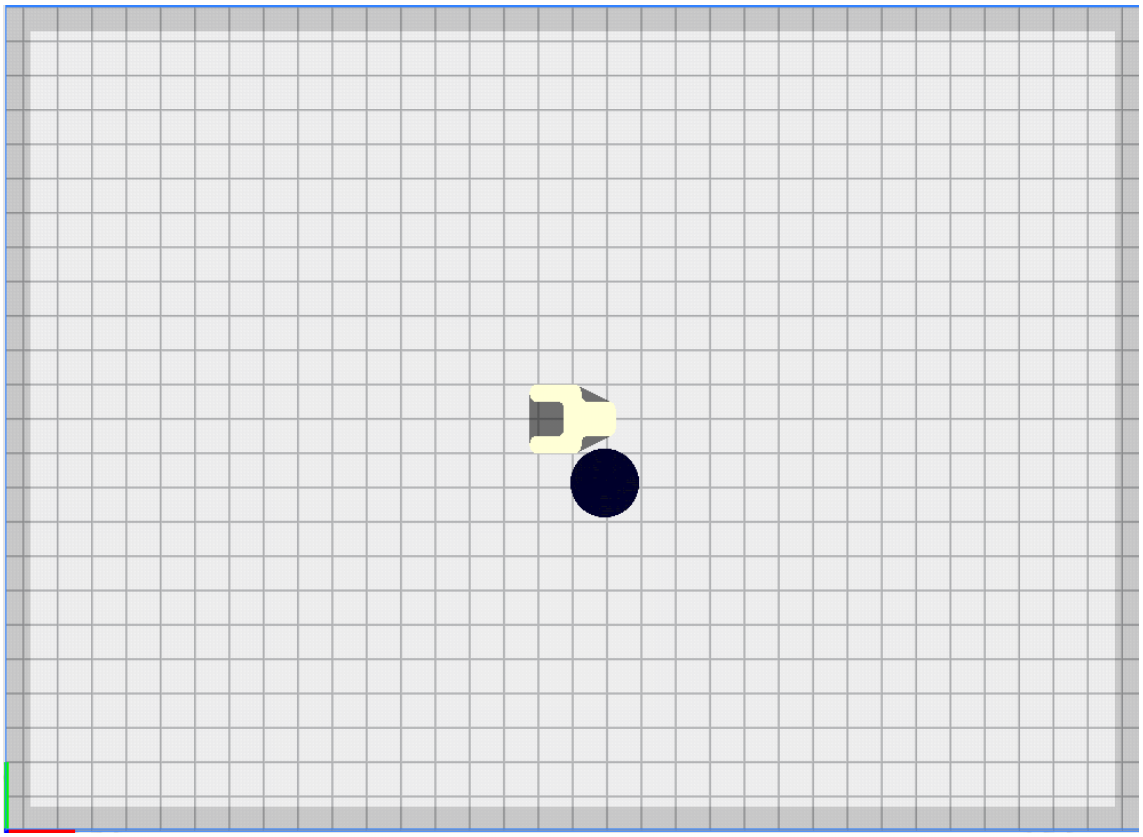


***Figura 4.1.2 La detección desde arriba de la impresora 3D***

## 4.2 Imágenes de proceso

En esta sección, se presentará el proceso de preprocesamiento y los resultados para cada imagen diferente. Como se muestra en las figuras 4.2.1, 4.2.2, 4.2.3 y 4.2.4, se completan el filtrado medio de la imagen para eliminar el ruido.


*Figura 4.2.1 El resultado de filtrado medio – foto 1*

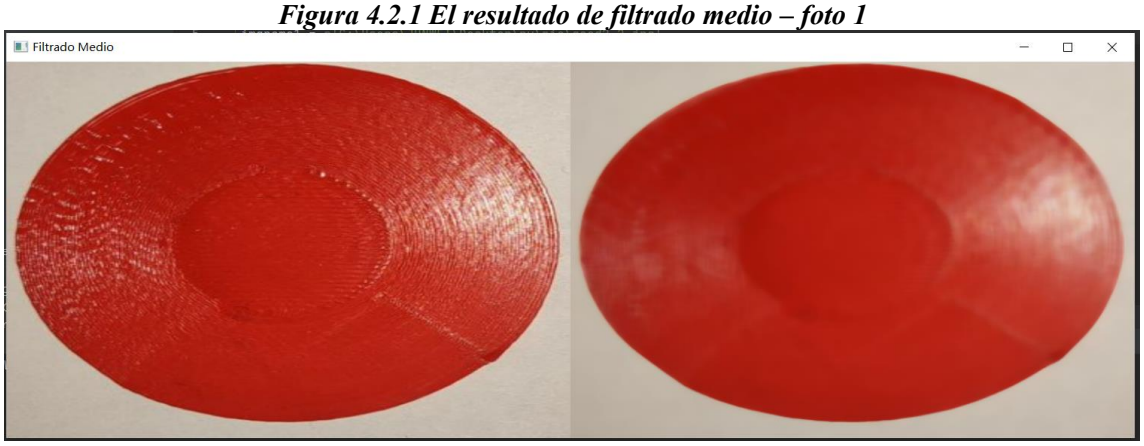



*Figura 4.2.2 El resultado de filtrado medio – foto 2*

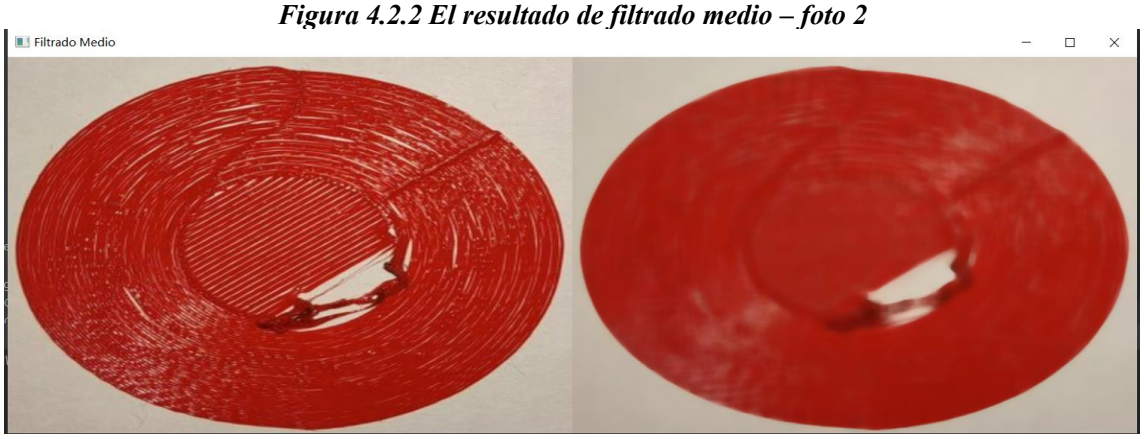



*Figura 4.2.3 El resultado de filtrado medio – foto 3*

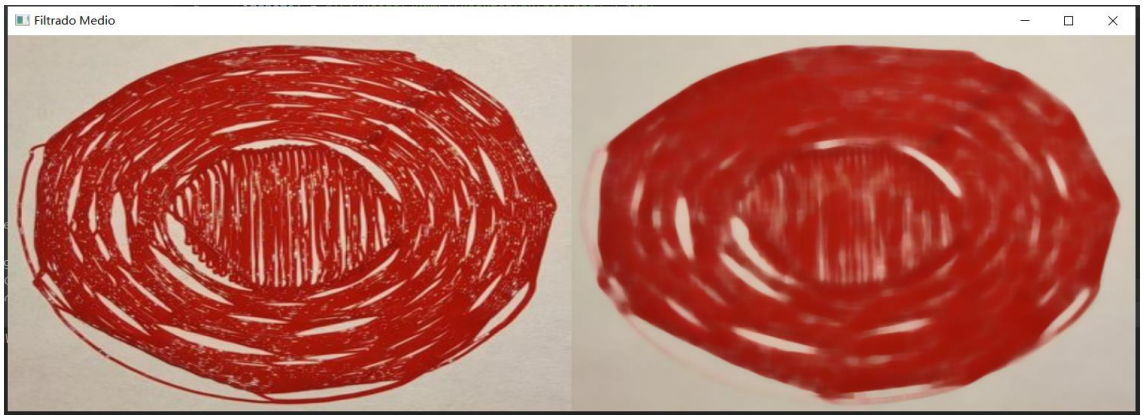



***Figura 4.2.4 El resultado de filtrado medio – foto 4***

Como se describe en la sección 3.4, el valor del área de convolución utilizado es 13. A continuación, se puede realizar el procedimiento de extracción del contorno con la imagen procesada.

Como se muestra en las figuras 4.2.5, 4.2.6, 4.2.7 y 4.2.8, se utiliza la función *Canny* y su umbral se establece en [50, 150] para extraer los contornos de la imagen.

La imagen 4.2.9 extrae el contorno del diseño del prototipo. Se puede ver que la imagen es bastante estándar, con poco o ningún ruido, excepto por algunas líneas indicativas en el fondo. Entonces no se necesita hacer demasiado procesamiento previo de la imagen para poder extraer el contorno del diseño.

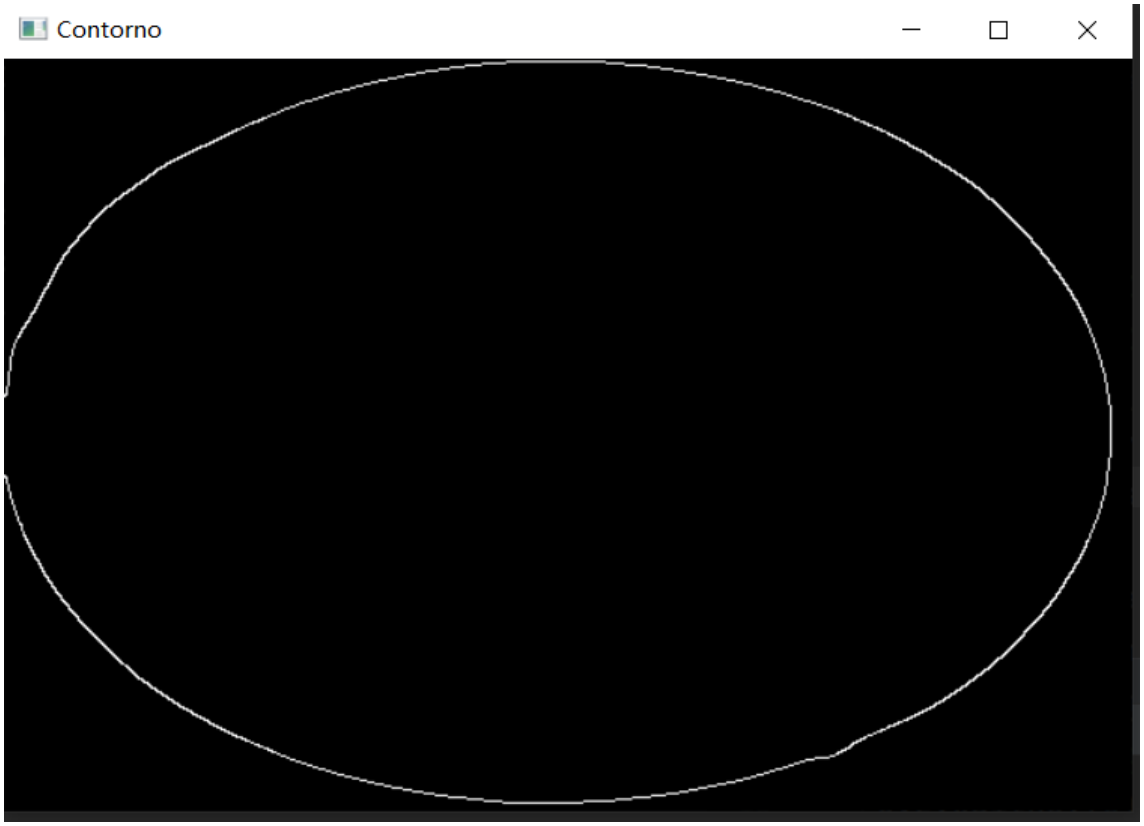



***Figura 4.2.5 El resultado de Canny – foto 1***

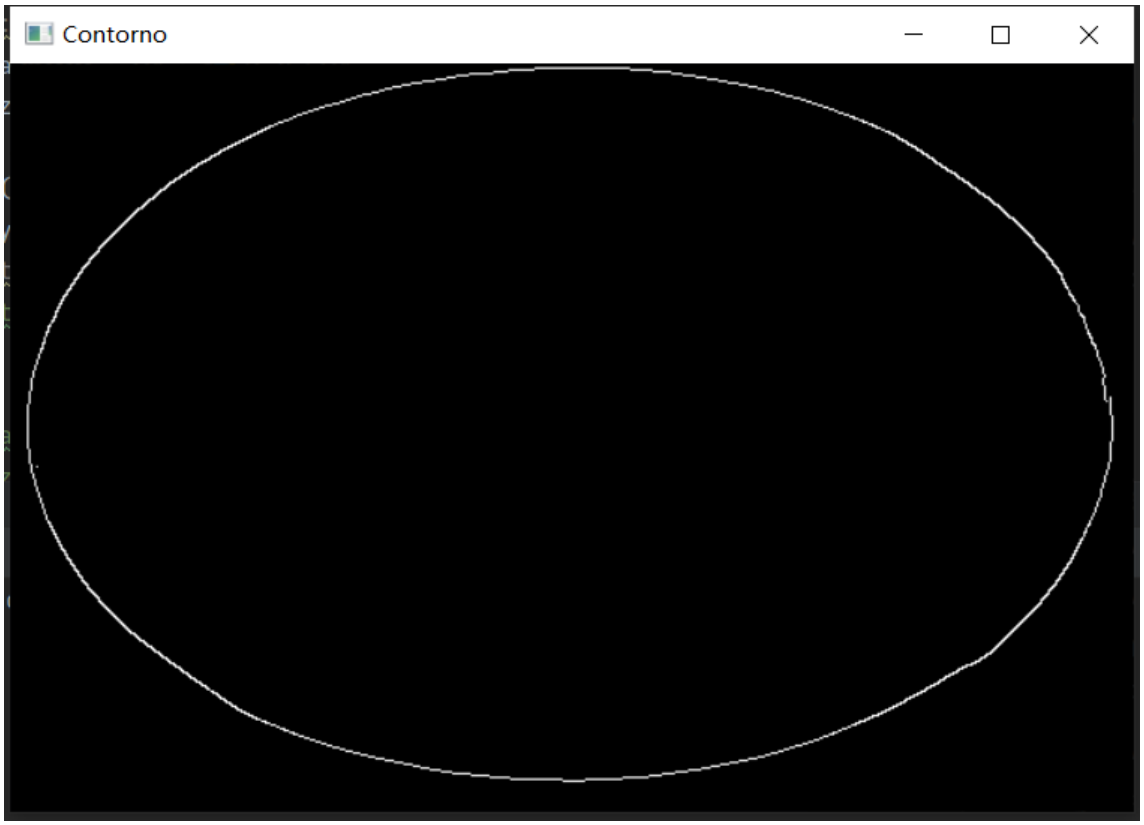



*Figura 4.2.6 El resultado de Canny – foto 2*

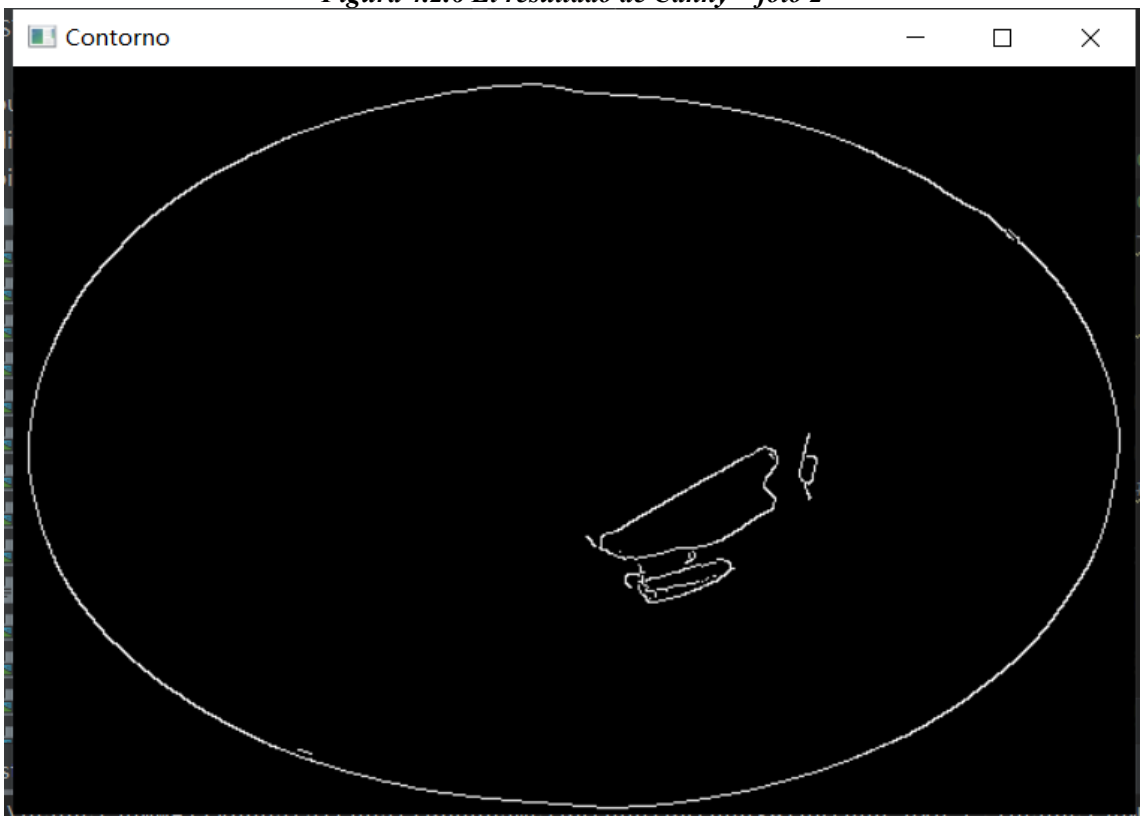



*Figura 4.2.7 El resultado de Canny – foto 3*

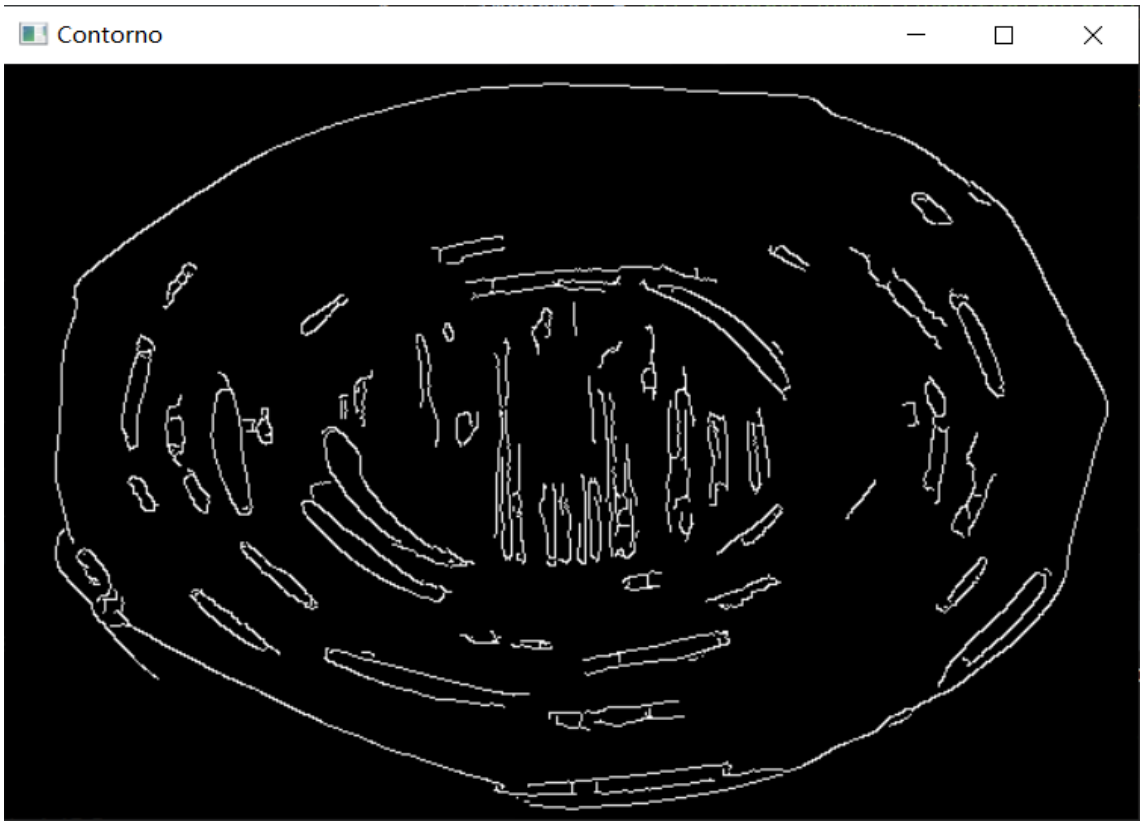



*Figura 4.2.8 El resultado de Canny – foto 4*

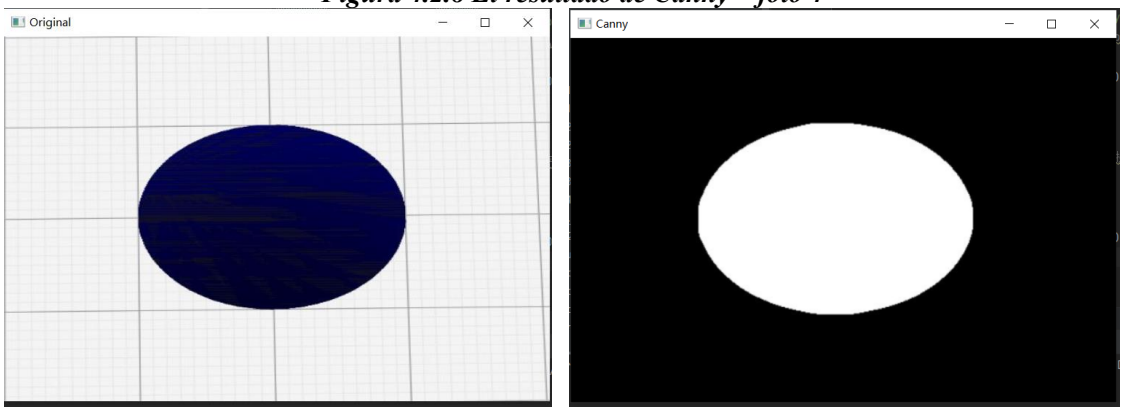



*Figura 4.2.9 El resultado de Canny – Diseño del prototipo*

## 4.3 Detección de puntos característicos

El próximo trabajo será mucho más fácil, ahora se ha extraído los contornos de imagen nítidos y confiables. A continuación, se debe usar el algoritmo ORB para detectar puntos clave de la imagen.

Idealmente, los puntos clave de la imagen deben distribuirse a lo largo de los bordes de la imagen. Si la imagen impresa contiene los defectos, los defectos también deben tener los puntos clave correspondientes.

Primero, se extrae los puntos característicos del dibujo de diseño del prototipo. Se puede detectar los puntos clave de la imagen utilizando la función ORB descrita anteriormente. Y dibuja los puntos clave en la imagen original como se muestra en la figura 4.3.1.

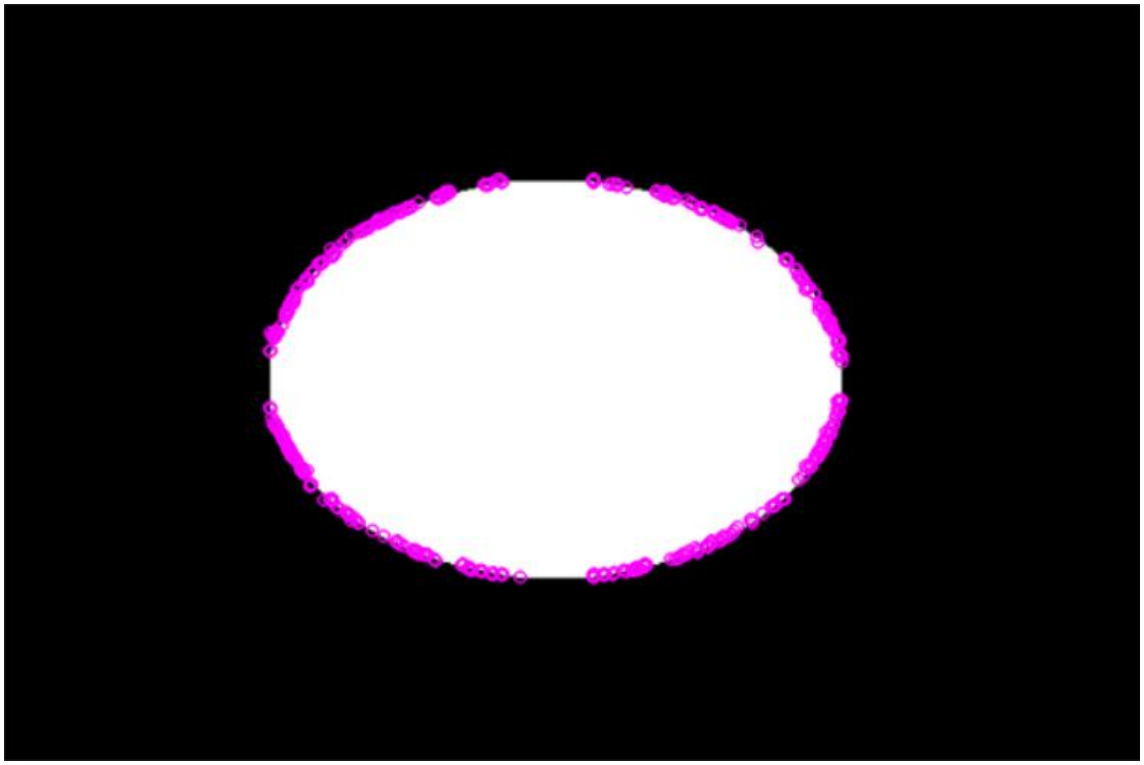


***Figura 4.3.1 Los puntos clave del Diseño del prototipo***

Se puede ver que los puntos clave calculados por ORB se distribuyen en los bordes del contorno de la imagen. De la misma manera, se puede realizar el mismo procesamiento sobre las fotos de las imágenes impresas como se muestra en las figuras de 4.3.2, 4.3.3, 4.3.4 y 4.3.5.


*Figura 4.3.2 Los puntos claves de la foto 1*

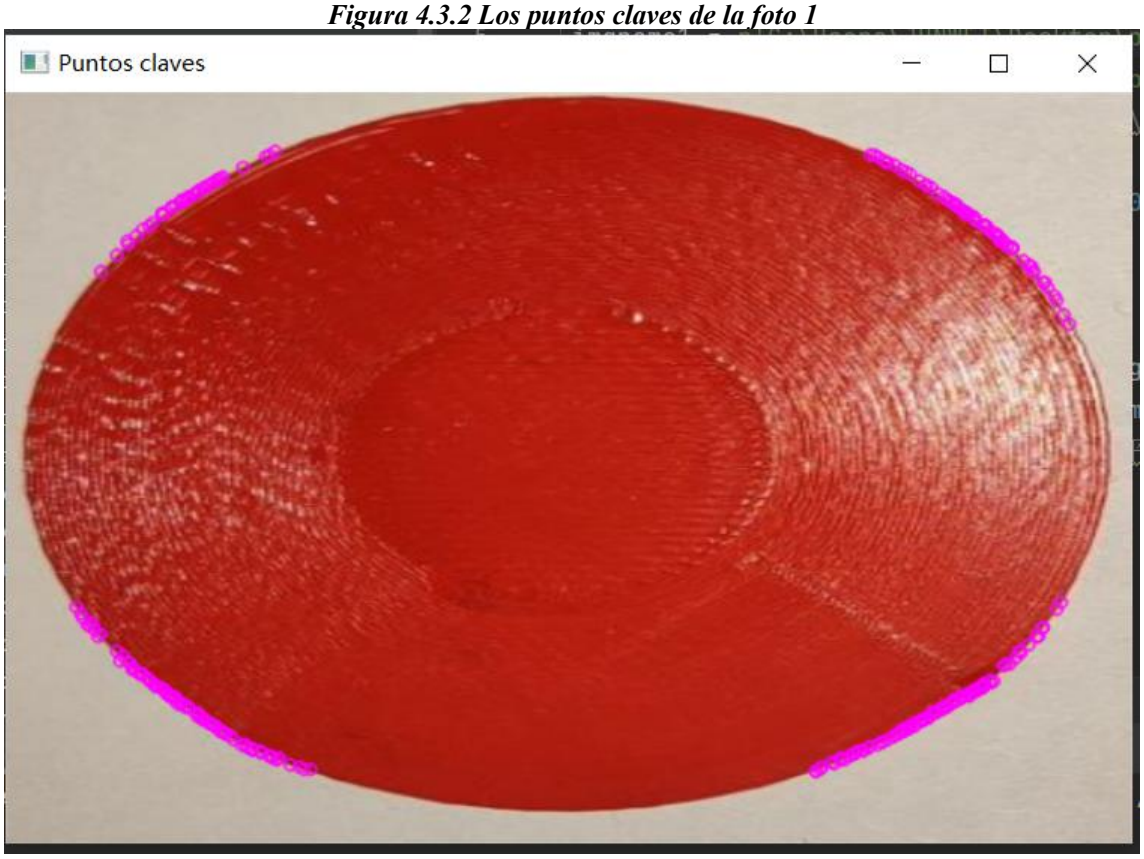



*Figura 4.3.3 Los puntos claves de la foto 2*

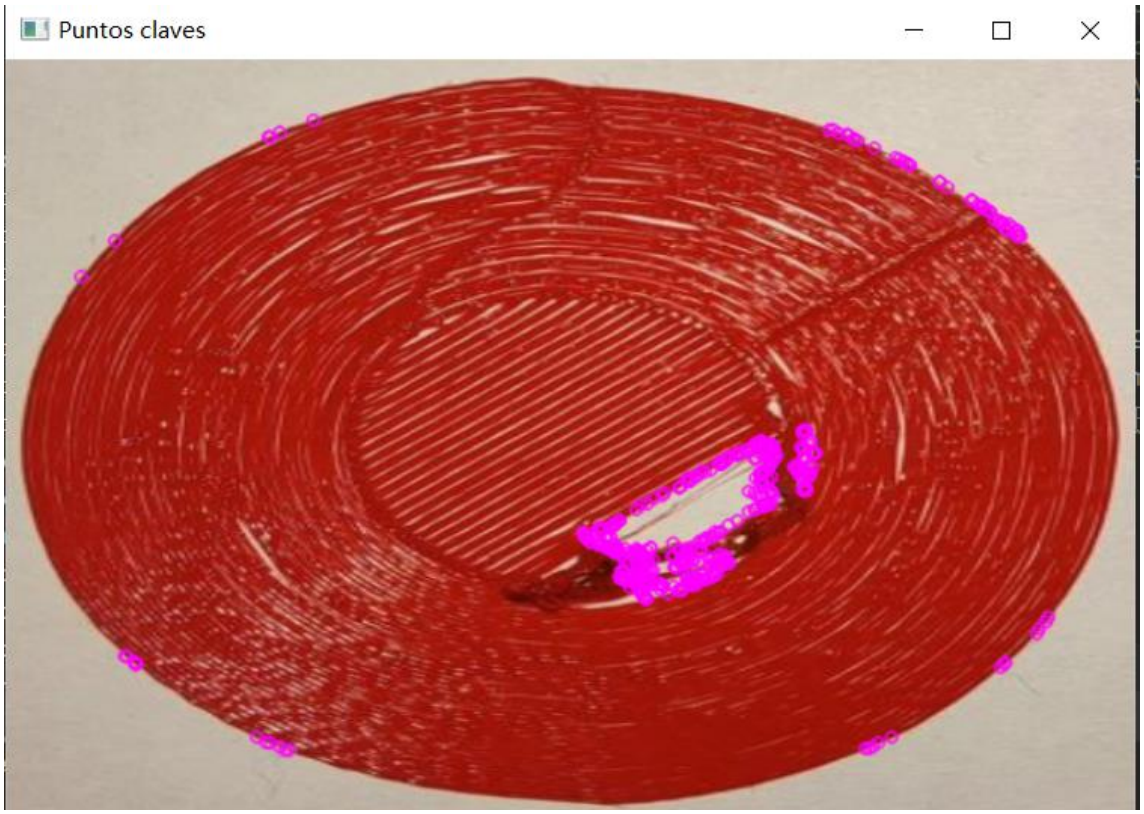



*Figura 4.3.4 Los puntos claves de la foto 3*

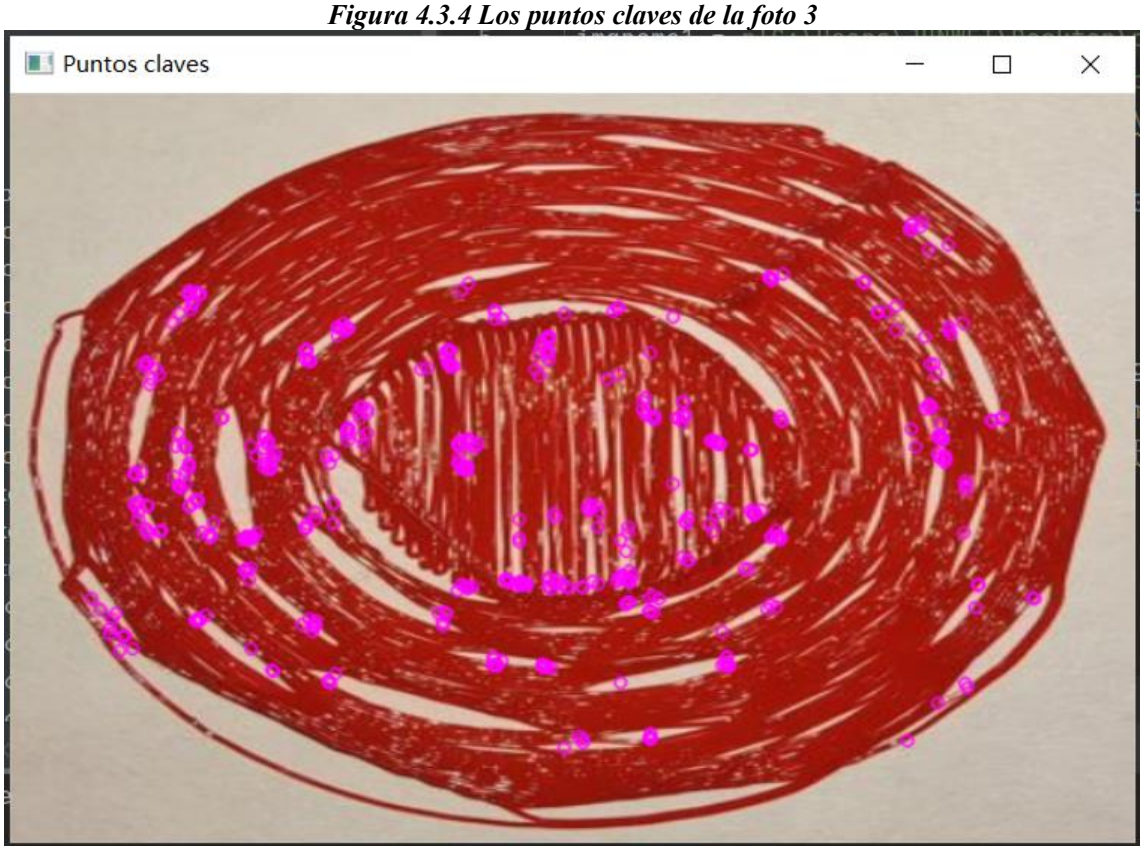



*Figura 4.3.5 Los puntos claves de la foto 4*

Se puede observar que el algoritmo ORB tiene mayor precisión para los defectos. Como se muestra en la figura 4.3.5, los defectos más pequeños de la figura también son detectados por el algoritmo ORB y marcados con puntos característicos.

## 4.4 Coincidencia de puntos característicos

En el trabajo se compara y genera los resultados en esta sección. A lo largo del trabajo se ha discutido cómo funciona BFmatcher en el trabajo anterior y, afortunadamente, hay funciones relacionadas en la biblioteca OpenCV que se pueden usar. Esta función es cv2.BFmatcher(). Esta elección se basa en encontrar los dos puntos característicos más similares en el diseño del prototipo para cada punto característico de la foto.

Debido a que el uso de BFmatcher es propenso a coincidencias falsas, para excluir coincidencias falsas. Si y solo si la similitud de los puntos de características más similares (en BFmatcher, la similitud se expresa por la distancia) es mucho mayor que los siguientes puntos de características similares, se considerará una coincidencia exitosa. Esto elimina la gran mayoría de coincidencias falsas, pero no elimina las coincidencias falsas totalmente como se muestra en la figura 4.4.1 y 4.4.2.

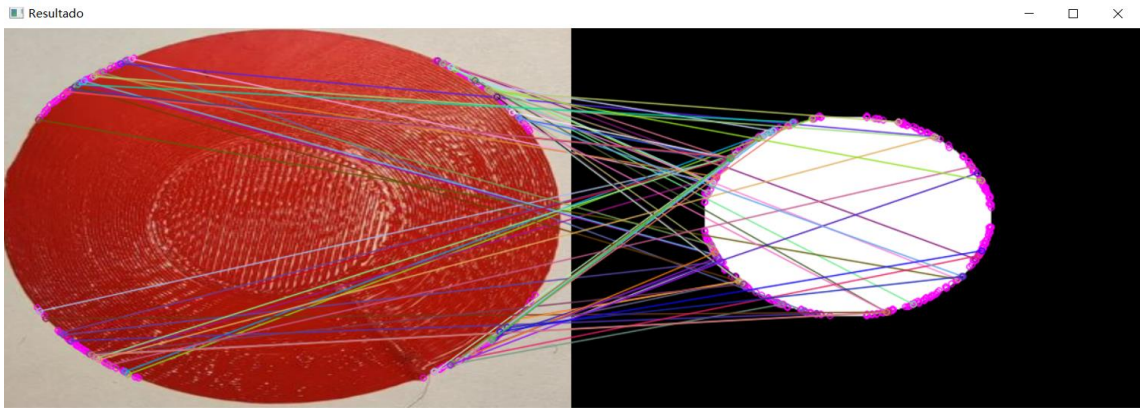



***Figura 4.4.1 El resultado de buena imagen impresa***

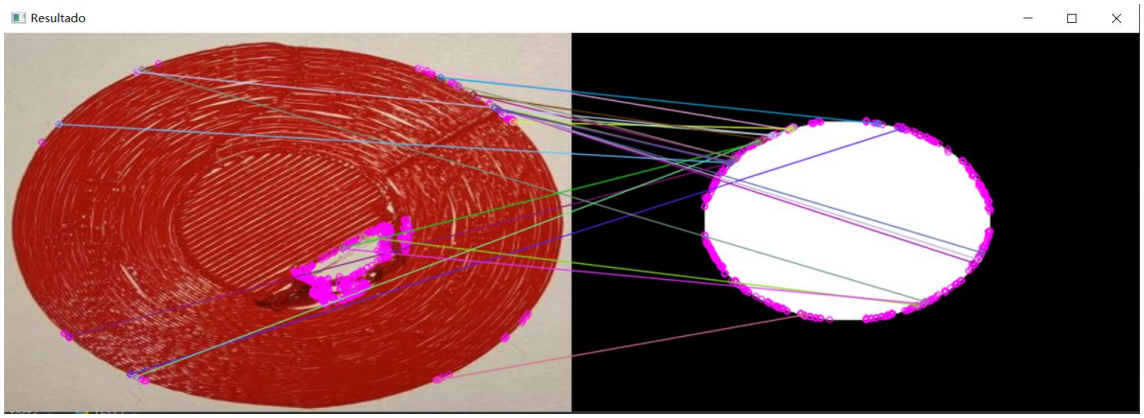



***Figura 4.4.2 El resultado de mala imagen impresa***

Para las imágenes mal impresas, sus bordes exteriores aún combinan bastante bien con el diseño. La siguiente tarea que se debe realizar es buscar generar la conclusión de si la impresión es exitosa o no de acuerdo con el resultado de la coincidencia. Según las secciones de 3.3 y 3.4,

La similitud entre la imagen impresa y el diseño del prototipo se puede calcular utilizando la fórmula 3.3.1. Y de acuerdo con los resultados del cálculo, se cuenta la similitud de cada foto y, finalmente, se usa la similitud del 35 % como estándar de juicio para determinar si la imagen tiene defectos. Como se muestra en la Figura 4.4.3, 4.4.4, 4.4.5 y 4.4.6, el uso de este criterio puede distinguir con mayor precisión entre la impresión normal de imágenes y la impresión de imágenes defectuosas.

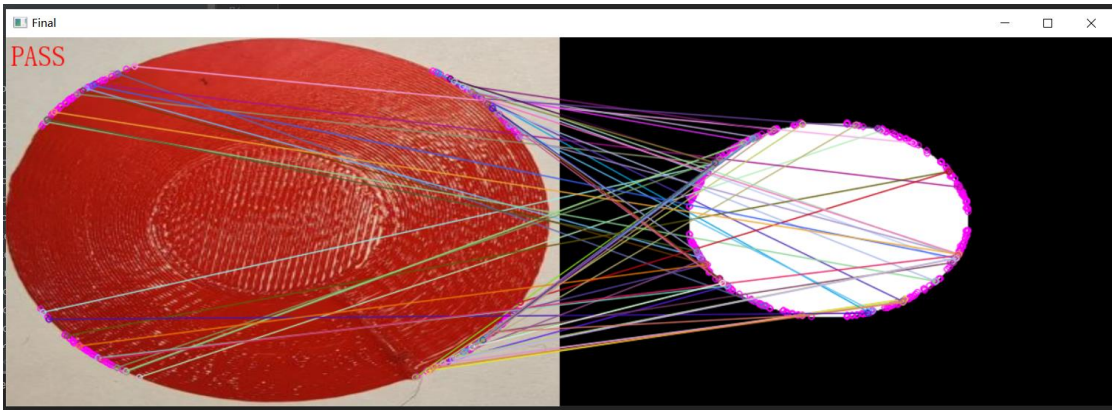



*Figura 4.4.3 Resultado 1-Pass*

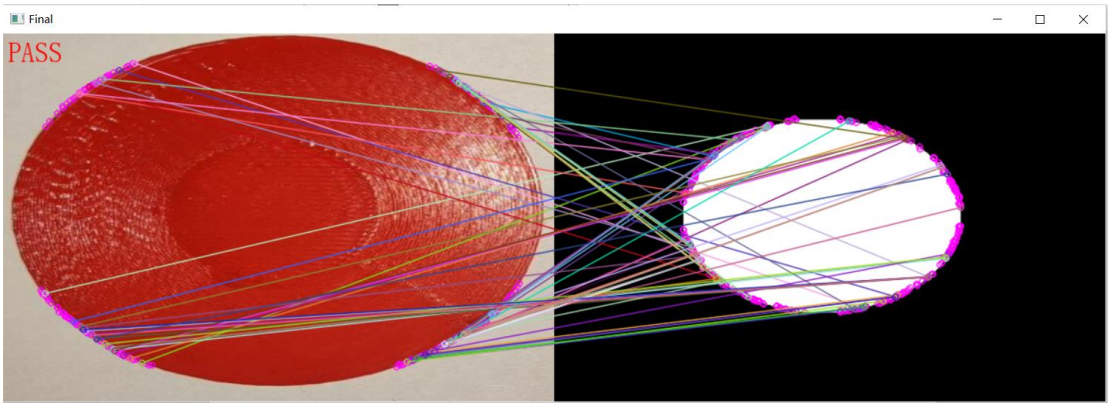



*Figura 4.4.4 Resultado 2-Pass*

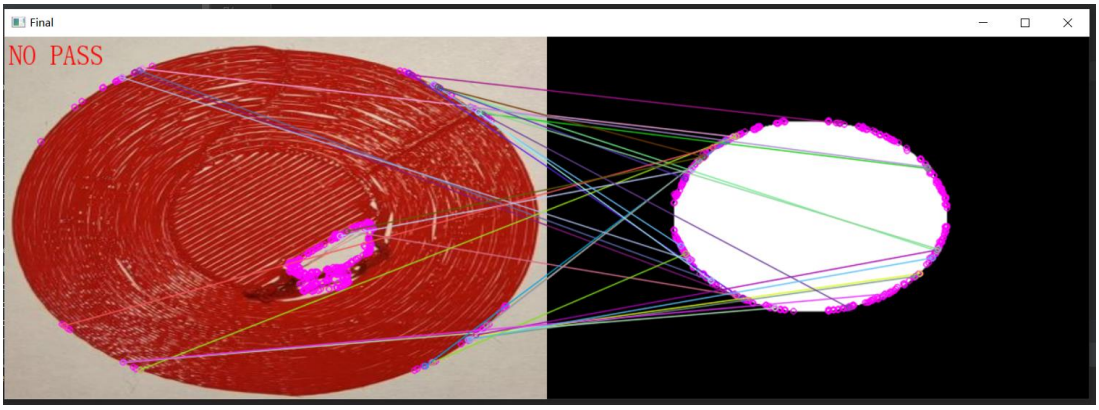



*Figura 4.4.5 Resultado 3- No Pass*

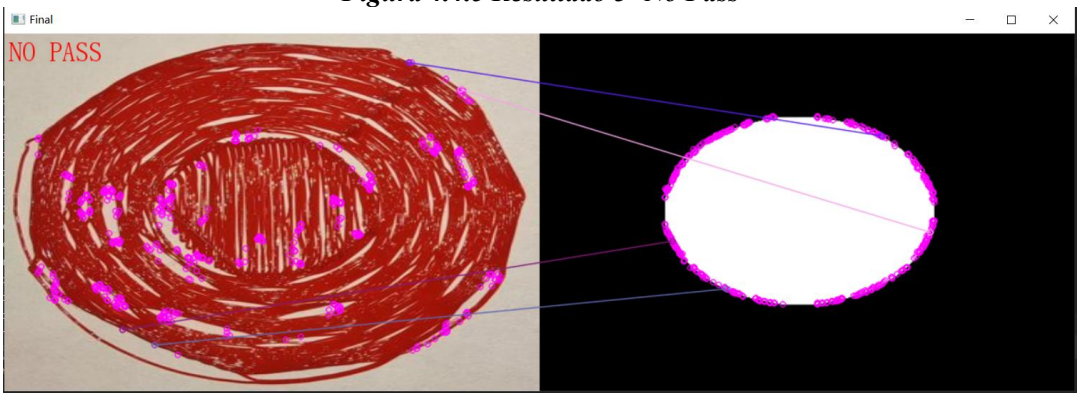



*Figura 4.4.6 Resultado 4- No Pass*

## 4.5 Tiempo de ejecución

Para este trabajo, el tiempo de ejecución también es un elemento indicador importante. Esto determina si el resultado de toda la ejecución del programa es de tiempo real y efectivo. Como se muestra en la tabla 4.5.1, se registra el tiempo de ejecución de todo el procedimiento para procesar 4 fotos diferentes, y cada foto se proba 3 veces y se registra.

| Imagen | Primer experimento (s) | Segundo experimento (s) | Tercero experimento (s) | Tiempo promedio (s) |
|---|---|---|---|---|
| Foto 1 | 0.663 | 0.615 | 0.755 | 0.678 |
| Foto 2 | 0.673 | 0.660 | 0.714 | 0.682 |
| Foto 3 | 0.628 | 0.798 | 0.776 | 0.734 |
| Foto 4 | 0.789 | 0.737 | 0.749 | 0.758 |

***Tabla 4.5.1 Los tiempos de ejecución***

El tiempo de ejecución del programa está relacionado con la configuración de la computadora utilizada para la ejecución. La CPU de la computadora utilizada para completar esta parte de la prueba funciona a una frecuencia de 2.1Ghz y el modelo es AMD Ryzen 5 3500U, que es una CPU con un rendimiento relativamente normal. Si reemplaza la herramienta de ejecución con una configuración más alta, obtendrá un tiempo de ejecución del programa más corto.

A partir de la tabla anterior 4.5.1, este programa puede garantizar resultados precisos en 1 segundo cada vez que se ejecuta, y puede retroalimentar los resultados en tiempo real.

## 4.6 Verificación de otro escenario

Se ha comprobado la viabilidad de las ideas y procedimientos del trabajo en el apartado anterior. A continuación, se utilizará el mismo procedimiento para experimentar con otros escenarios. Y algunas escenas tienen menos ruido de fondo, para ello se trata de usar solo la función *Canny* para extraer los contornos de los bordes de la imagen sin utilizar el filtro mediano para eliminar el ruido de fondo. El resto de la configuración es igual que en el apartado anterior.

A continuación, se muestran los resultados del trabajo con la figura 4.5.1, 4.5.2, 4.5.3, 4.5.4, 4.5.5 y 4.5.6.

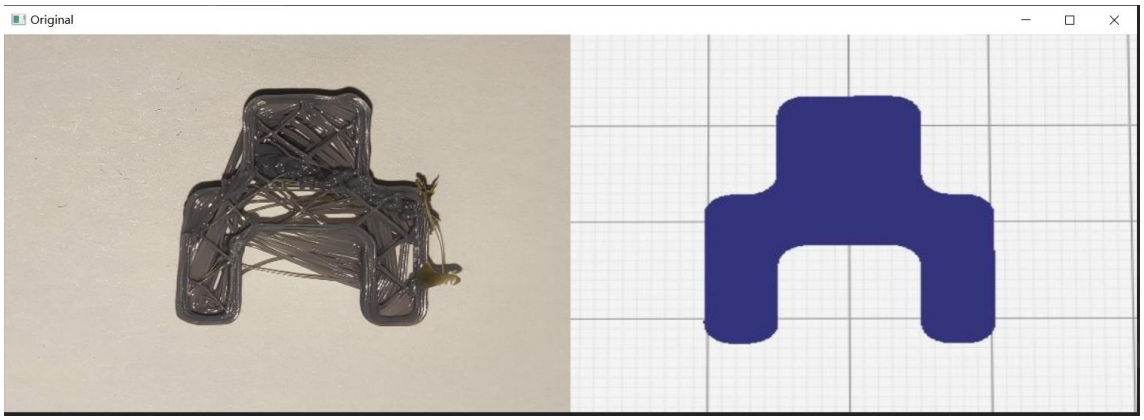



*Figura 4.6.1 La comparación de fotos 1 impresas y diseño.*

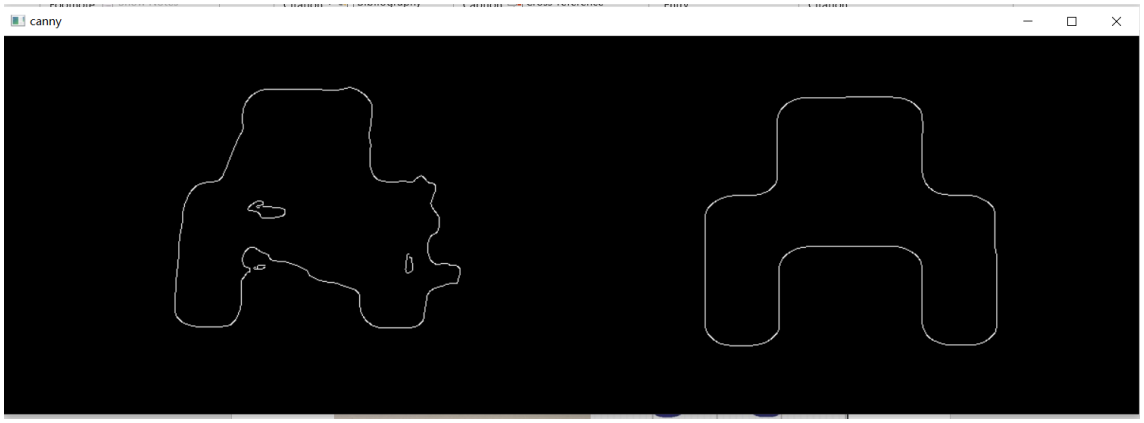



*Figura 4.6.2 La comparación Canny de fotos 1 impresas y diseño.*

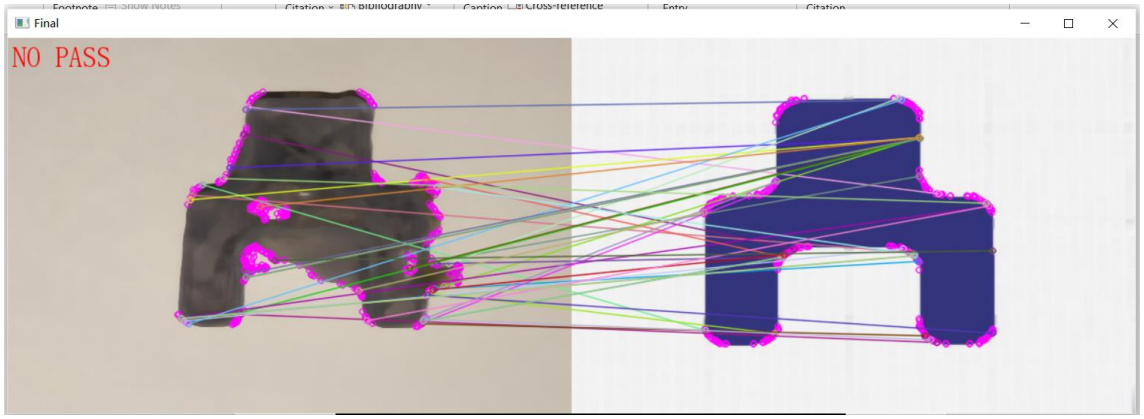



*Figura 4.6.3 El resultado 5- No Pass*

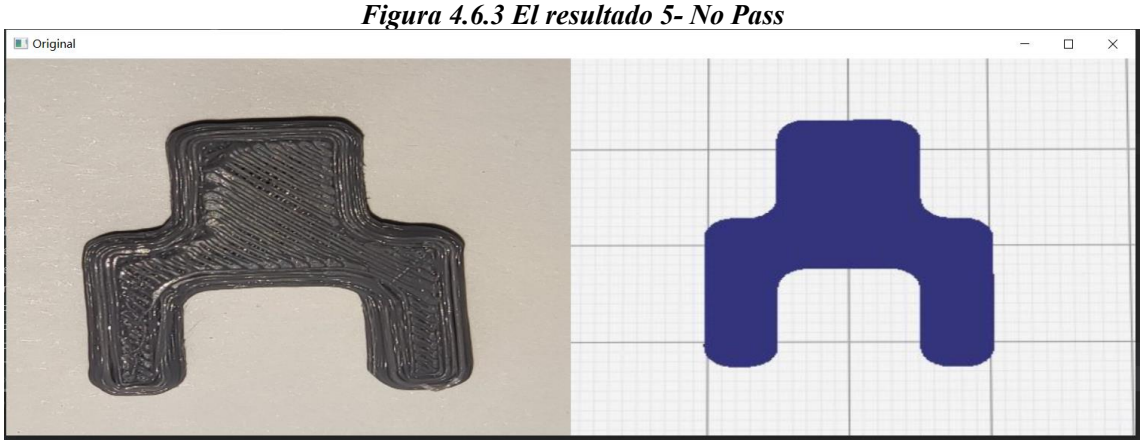



*Figura 4.6.4 La comparación de fotos 2 impresas y diseño.*

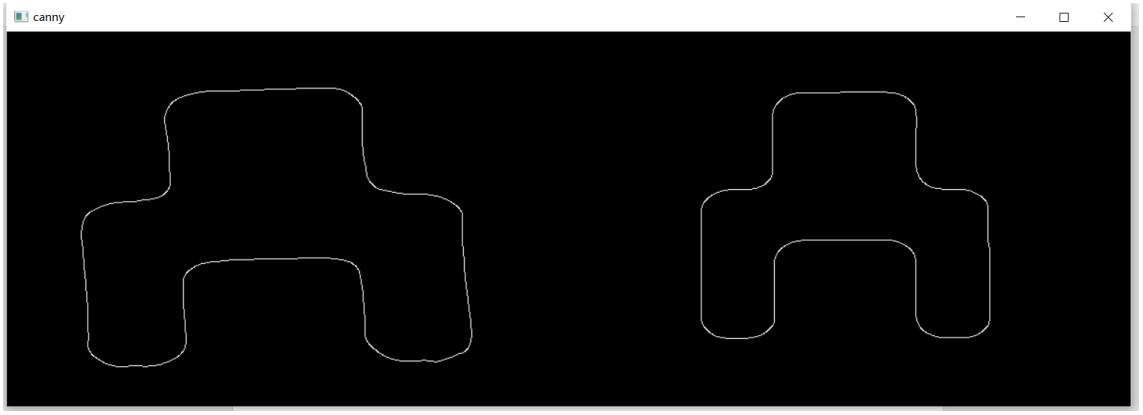



*Figura 4.6.5 La comparación Canny de fotos 2 impresas y diseño.*

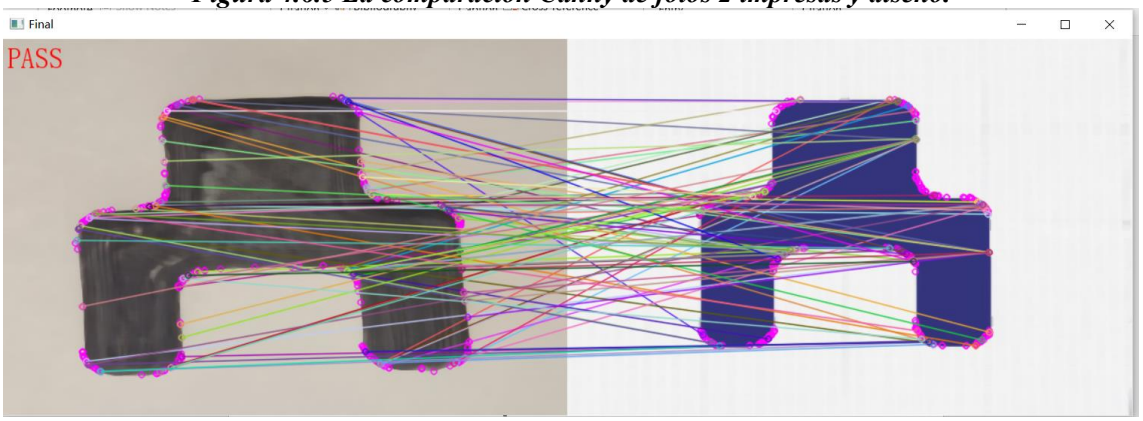



*Figura 4.6.6 El resultado - Pass*

En este escenario, se demuestra que también es una buena opción usar directamente la función *Canny* para extraer el contorno de la imagen en el caso de poco ruido. Pero si se usa la función *Canny* directamente bajo condiciones de iluminación complejas, es muy probable que los contornos extraídos falten parcialmente, como la siguiente figura 4.5.7:

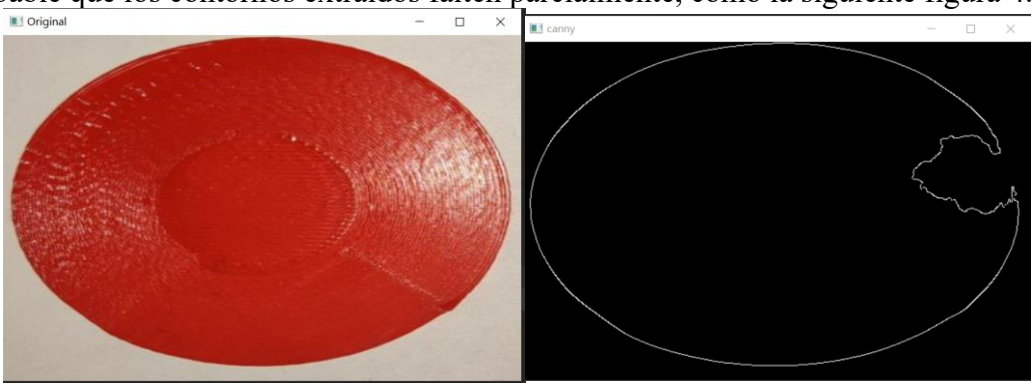



*Figura 4.6.7 Los contornos extraídos faltan parcialmente*

Por lo tanto, se puede ajustar el proceso de todo el programa según el entorno de impresión y los materiales de impresión. Si se agregan o eliminan algunos pasos, puede acelerar en gran medida la detección. Por supuesto, también podemos adoptar el enfoque más conservador, ajustar el brillo en cualquier condición, usando HSV o *Canny* para extracción contornos, etc.

# 5. Conclusiones y Trabajos futuros

## 5.1 Conclusiones

En líneas generales se puede decir que el algoritmo desarrollado, basado en los métodos de extracción de características ORB, es una alternativa interesante para la detección de fallos en imágenes.

En base a los resultados presentados en el trabajo total, se pueden extraer las siguientes conclusiones:

- Los puntos característicos de la imagen se extraen mediante el algoritmo ORB y se usa *BFmatcher* para comparar los puntos característicos. Tiene gran valor práctico cuantificar la similitud de las imágenes calculando la tasa de éxito de la comparación. Es innegable que, para los fallos de impresión más pequeños, aún pueden ocurrir condiciones indetectables. El algoritmo ORB es un algoritmo de de puntos característicos muy sensible. Para los defectos muy pequeños, el algoritmo ORB también puede marcar con los puntos característicos, pero para eliminar el ruido de fondo de la imagen, necesitamos usar un filtro mediano para procesar la imagen. Esta parte del proceso elimina el ruido y también hace que los bordes del contorno de la imagen se vuelvan borrosos.
  Se puede reducir el área de convolución del filtro mediano, pero reducir el área de convolución también reducirá el efecto de filtrado sobre el ruido. Entonces, para defectos de impresión más pequeños, debe usar una cámara mejor para tomar una imagen con menos ruido, también es posible reducir el área de convolución o no adaptarse al filtro mediano para procesar la imagen. De esta manera, se puede obtener un mejor efecto de detección.

- En términos generales, el contorno de la imagen se puede obtener utilizando la función *Canny* y la función HSV. Pero como la comparación realizada en la Sección 3.2, la función *Canny* es mucho mejor que la función HSV para extraer los bordes del contorno de la imagen. Por lo tanto, se recomienda utilizar primero la función *Canny* para la extracción de contornos. La función HSV también puede completar el trabajo de extracción de contorno, pero es fácil recibir interferencias de ruido. Además, cuando el color de la imagen impresa está cerca del color de fondo, dado que el principio de HSV para la extracción de contornos es filtrar píxeles con diferentes tonos, puede haber un problema de que el contorno no se pueda extraer con éxito. Además, para la situación de detección con condiciones de iluminación complejas, es mejor cambiar directamente las condiciones de impresión que usar más métodos de procesamiento de imágenes. Lo mejor es apagar el flash de la cámara para evitar una mayor introducción de iluminación más compleja.

- El juicio de similitud final es un proceso de juicio cuantitativo. En las Secciones 3.3, se analiza en detalle cómo calcular la similitud y establecer las condiciones

de similitud para juzgar el éxito de la impresión. En este trabajo, se ha utilizado la similitud 35% como umbral para admitir una imagen como satisfactoria (PASS). Si existen requisitos más altos para la precisión de impresión, el umbral se puede aumentar aún más. Sin embargo, no se recomienda aumentar el umbral a más del 40%, ya que teniendo en cuenta la influencia del ruido y la precisión de la comparación, a menos que las dos fotos comparadas sean exactamente las mismas dos fotos, difícilmente habrá una similitud de más del 40%. Dado que los errores de impresión tienen un gran impacto en la detección de puntos característicos, lo que a su vez cambia la similitud. En la comparación, las imágenes casi sin errores de imprenta son más del 10 % similares, por lo que no es necesario establecer un umbral de similitud demasiado alto. De lo contrario, es muy probable que se produzca un error de apreciación, y la imagen que se imprimió correctamente se considera un error de impresión.

Como conclusión se puede decir que se han cumplido todos los objetivos establecidos.

En el trabajo se evalúan varias medidas de procesamiento de imágenes y se aplican a imágenes reales a través de combinaciones razonables. Se procesan imágenes ruidosas en contornos de imagen limpios. Y a través de la detección de características y la comparación de puntos de características, la tecnología de visión artificial juzga con éxito la similitud entre la imagen impresa y el dibujo del diseño del prototipo, y el resultado es una salida adicional.

El algoritmo ORB y el comparador BFmatcher juegan un papel decisivo en el trabajo general. Incluso para fallos relativamente pequeños, el algoritmo ORB puede identificar y marcar puntos característicos con precisión. En comparación con el algoritmo SIFT y el algoritmo SURF, el algoritmo ORB utilizado tiene una alta precisión y un tiempo de ejecución corto, lo que juega un papel crucial en el trabajo.

## 5.2 Trabajos Futuros

El trabajo general ha logrado resultados de comparación de muy alta precisión mediante el uso del algoritmo ORB para extraer puntos característicos de la imagen y comparar puntos característicos de la imagen con *BFmatcher*. Sin embargo, en el trabajo general, personalmente hay algunas ideas y experiencias, que se pueden estduiar más a fondo en el futuro:

- Como uno de los algoritmos centrales del trabajo general, el algoritmo ORB es la clave para que se extraigan los puntos clave de la imagen. Además del algoritmo ORB, también se puede utilizar el algoritmo SIFT (*Scale-Invariant Feature Transform*), el algoritmo SURF (*Speeded Up Robust Features*) y otros algoritmos para extraer los puntos clave de la imagen. Aunque sus estrategias para extraer puntos clave de la imagen son bastante diferentes, en los experimentos, la detección y extracción de puntos clave y el posterior trabajo de comparación tienen un impacto relativamente pequeño. Pero diferentes algoritmos tienen un gran impacto en la velocidad de ejecución general del programa. Sin embargo, mejorar aún más la estabilidad y la eficiencia de ejecución del programa reemplazando el algoritmo es una dirección que se puede seguir para la investigación.

- Todo el trabajo es esencialmente el trabajo de la visión artificial, o puede entenderse simplemente como una comparación de imágenes. Dado que la impresión 3D solo imprime una pequeña cantidad de prototipos a la vez, también es posible realizar comparaciones manuales además de utilizar los procedimientos de este trabajo. Al mismo tiempo, con el resultado de la sección de 4.5, el programa tiene una alta eficiencia operativa y un gran diseño modular, se pueden calcular los resultados confiables en 1 segundo lo que significa que el programa puede extenderse aún más a la línea de producción para la detección de muestras a alta velocidad. Para la transmisión de alta velocidad en la línea de producción, es necesario mejorar aún más la eficiencia operativa del programa y acortar el tiempo de funcionamiento del programa. Y aumentar la usabilidad del programa, así como aumentar aún más la interfaz de interacción humano-computadora.

- Otra línea de trabajo sería intentar juzgar la similitud de las imágenes a través de la inteligencia artificial. Por ejemplo, aprendizaje profundo, red neuronal, algoritmo genético, etc. para realizar trabajos de aprendizaje y juicio. Las razones por las cuales el método anterior no se adopta en este documento es:

1. Los algoritmos de inteligencia artificial anteriores requieren una gran cantidad de modelos de muestra para el entrenamiento a fin de lograr una alta precisión en las comparaciones en vivo para obtener el mejor rendimiento y resultados. Debido a la falta de datos de entrenamiento tan grandes, no se puede obtener un modelo de entrenamiento de alta disponibilidad y, por lo tanto, no se pueden obtener resultados de comparación de alta precisión.

2. Cuando la impresora 3D realice la tarea de impresión, el resultado impreso tendrá algunos cambios en comparación con el prototipo de diseño. Por ejemplo, el

diseño es un círculo sólido, pero la impresora usa líneas para llenar el círculo al imprimir. Como resultado, el patrón impreso no es exactamente igual al diseño. Las impresoras 3D solo pueden garantizar que los contornos impresos sean los mismos y que los patrones internos sean claros. Pero no se garantiza que sea exactamente igual.

Pero al mismo tiempo, el algoritmo de inteligencia artificial tiene una escalabilidad muy alta. Si podemos resolver los problemas planteados anteriormente y más problemas que se pueden encontrar en la práctica, se pueden obtener resultados de muy alta calidad a través de algoritmos de inteligencia artificial.

- Varias interfaces de entrada y salida se reservan en el programa utilizado en este trabajo, y se puede intentar usar las interfaces para usar varias funciones. Por ejemplo, cuando el programa detecta que la impresión 3D falla, además de mostrar el resultado NOPASS, también envía un correo electrónico para notificar al técnico y así sucesivamente. También se pueden implementar funciones más interesantes.

- Este programa está escrito en Python en su totalidad, y puede intentar escribirlo en diferentes lenguajes de programación y trasladarlo a otras plataformas de trabajo en el futuro.

# 6. Referencias bibliográficas

[1] ORB: an efficient alternative to SIFT or SURF, Autor: Ethan Rublee, Vincent Rabaud, Kurt Konolige, Gary Bradski, Willow Garage, Menlo Park, California, 2011 International Conference on Computer Vision, 6-13 Nov. 2011

[2] Fast Matching of Image Targets based on ORB Feature Points, Autor: Xingxing Lia, Chao Duan, Panpan Yin, Yan Zhi, Departamento de ingeniería electrónica e información, Facultad de Tecnología y Negocios de Guangzhou

[3] Image Matching Using SIFT, SURF, BRIEF and ORB: Performance Comparison for Distorted Images, Ebrahim Karami, Siva Prasad, and Mohamed Shehata, Faculty of Engineering and Applied Sciences, Memorial University, Canada, 7 Oct 2017

[4] Introduction to ORB (Oriented FAST and Rotated BRIEF) https://medium.com/data-breach/introduction-to-orb-oriented-fast-and-rotated-brief-4220e8ec40cf

[5] cv:BFMatcher Class Reference https://docs.opencv.org/4.x/d3/da1/classcv_1_1BFMatcher.html

[6] Feature Matching https://opencv24-python-tutorials.readthedocs.io/en/latest/py_tutorials/py_feature2d/py_matcher/py_matcher.html

[7] Distinctive Image Features from Scale-Invariant Keypoints, D. Lowe, International Journal of Computer Vision, Vol. 60, Issue 2, 2004, pp. 91-110

[8] Fast Approximate Nearest Neighbors with Automatic Algorithm Configuration, M.Muja and D.Lowe VISAPP International Conference on Computer Vision Theory and Applications, 2009

[9] Wikipedia: Algoritmo de *Canny* https://es.wikipedia.org/wiki/Algoritmo_de_*Canny*

[10] *Canny* Edge Detection https://docs.opencv.org/4.x/da/d22/tutorial_py_*Canny*.html

[11] Wikipedia: Impresión 3D https://es.wikipedia.org/wiki/Impresi%C3%B3n_3D

[12] Página Oficial de OpenCV https://opencv.org/

[13] OpenCV: Open-Source Computer Vision Library https://github.com/opencv/opencv

[14] A Computational Approach to Edge Detection, John *Canny*, Artificial Intelligence Laboratory, Massachusetts Institute of Technology, Cambridge, MA, USA, Nov. 1986

[15] Scaling object recognition: Benchmark of current state of the art techniques. M. Aly, P. Welinder, M. Munich, and P. Perona. In First IEEE Workshop on Emergent Issues in Large Amounts of Visual Data (WS-LAVD), IEEE International Conference on Computer Vision (ICCV), September 2009. 6

[16] Wikipedia: Modelo de color HSV https://es.wikipedia.org/wiki/Modelo_de_color_HSV#:~:text=El%20modelo%20HSV%20fue%20creado,no%20que%20HSL%20o%20HSI.

[17] Introducción Opencv (Biblioteca de visión de computadora de código abierto) https://programmerclick.com/article/72162211320/

[18] Multi-target Tracking Algorithm Based on ORB Feature Points Matching. Autor: Peng L, Yanjiang W. Journal of Hunan University (Natural Sciences), 2017

[19] A parallel method for aerial image stitching using ORB feature points, Autor: Wang G, Zhai Z, Xu B, et al 2017 IEEE/ACIS 16th International Conference on Computer and Information Science (ICIS). IEEE Computer Society, 2017

[20] Keypoint Signatures for Fast Learning and Recognition[C]. Autor: Calonder M, Lepetit V, Fua P, et al. European conference on computer vision, 2008

[21] Smoothing Filters https://theailearner.com/2019/05/06/smoothing-filters/

[22] Image Matching Using SIFT, SURF, BRIEF and ORB: Performance Comparison for Distorted Images, Ebrahim Karami, Siva Prasad, and Mohamed Shehata, Faculty of Engineering and Applied Sciences, Memorial University, Canada

[23] A Novel Image Correlation Matching Approach, B.Shan, JMM, vol. 5, no. 3, 2010

[24] Fast SIFT Design for Real-Time Visual Feature Extraction, Liang-Chi Chiu, Tian-Sheuan Chang, Jiun-Yen Chen and N. Chang, IEEE Trans.on Image Process, 2013.

[25] SIFT and SURF Performance Evaluation Against Various Image Deformations on Benchmark Dataset, N. Y. Khan, B. McCane, G. Wyvill, in Proceeding of 2011 International Conference of Digital Image Computing Techniques and Applications.

[26] Scaling object recognition: Benchmark of current state of the art techniques. M. Aly, P. Welinder, M. Munich, and P. Perona. In First IEEE Workshop on Emergent Issues in Large Amounts of Visual Data (WS-LAVD), IEEE Intern

ational Conference on Computer Vision (ICCV), September, 2009.

[27] Surf: Speeded up robust features. H. Bay, T. Tuytelaars, and L. Van Gool. In European Conference on Computer Vision, May 2006.

[28] Multi-image matching using multi-scale oriented patches. M.Brown, S.Winder, and R.Szeliski. In Computer Vision and Pattern Recognition, 2005.

[29] A Bayesian similarity measure for deformable image matching, B.Moghaddam, C.Nastar and A.Pentland, Image and Vision Computing, 2001.

[30] BRIEF: Binary robust independent elementary features, M.Calonder, V.Lepetit, C.Strecha, and P. Fua. In European Conference on Computer, Vision, 2010

[31] Comparison of SIFT and SURF Methods for Use on Hand Gesture Recogniti on based on Depth Map, P. Sykora, P. Kamencay and R.Hudec, AASRI Proc edia, 2014.

# 7. Código de ejecución

TFM2.py

```
import cv2
import numpy as np
from PIL import Image, ImageDraw, ImageFont
import time

def med(img):# El filtrado mediano
    dst = cv2.medianBlur(img, 13)
    return dst

def borde(img):# HSV para contorno
    img_hsv = cv2.cvtColor(img, cv2.COLOR_BGR2HSV)
    lower_red = np.array([0, 0, 0])
    upper_red = np.array([11, 255, 255])
    borde = cv2.inRange(img_hsv, lower_red, upper_red)
    return borde

def compre(img1,img2):# ORB para los puntos claves
    orb = cv2.ORB_create()
    kp1, des1 = orb.detectAndCompute(img1, None)
    kp2, des2 = orb.detectAndCompute(img2, None)
    return kp1,kp2,des1,des2

def match(des1,des2):# BFmatcher
    bf = cv2.BFMatcher()
    matches = bf.knnMatch(des1, des2, k=2)
    # Buscar las parejas mejores
    good = []
    for m, n in matches:
        if m.distance < 0.75 * n.distance:
            good.append([m])
    return good

def output(img,good,all):# Sale el resultado
    if len(good) / len(all) > 0.35:    # 0.3 0.5
        print(len(good))
        print(len(all))
        print(len(good) / len(all))
        print('PASS')
        # Sale el resultado en la imagen
        src1 = Image.fromarray(cv2.cvtColor(img, cv2.COLOR_BGR2RGB))
        font = ImageFont.truetype('simsun.ttc', 30)
```

```
            # Posicion del resultado
            position = (5, 5)
            # El resultado
            str = 'PASS'
            draw = ImageDraw.Draw(src1)
            draw.text(position, str, font=font, fill=(255, 0, 0))
            src1 = cv2.cvtColor(np.asarray(src1), cv2.COLOR_RGB2BGR)
            cv2.imshow("Final", src1)
            cv2.waitKey(0)
        else:
            print('NO PASS')
            print(len(good))
            print(len(all))
            print(len(good) / len(all))
            # Sale el resultado en la imagen
            src2 = Image.fromarray(cv2.cvtColor(img, cv2.COLOR_BGR2RGB))
            font = ImageFont.truetype('simsun.ttc', 30)
            # Posicion del resultado
            position = (5, 5)
            # El resultado
            str = 'NO PASS'
            draw = ImageDraw.Draw(src2)
            draw.text(position, str, font=font, fill=(255, 0, 0))
            src2 = cv2.cvtColor(np.asarray(src2), cv2.COLOR_RGB2BGR)
            cv2.imshow("Final", src2)
            cv2.waitKey(0)



if __name__ == '__main__':
    start = time.time()# Timer
    # Leer las fotos
    path1 = r'C:\Users\JUNWEI\Desktop\py\pic\bad2_2.jpg'
    path2 = r'C:\Users\JUNWEI\Desktop\py\pic\test3.jpg'
    test = cv2.imread(path2)
    # Recorte el size
    test = cv2.resize(test, dsize=(600, 400))
    img1 = cv2.imread(path1)
    img1 = cv2.resize(img1, dsize=(600, 400))
    #cv2.imshow('Original',img1)
    #cv2.waitKey(0)
    dst1 = med(img1) # Pasa a la funcion de filtrado mediano
    #cv2.imshow('Media', dst1)
    #cv2.waitKey(0)
    #borde1 = borde(dst1) # Pasa a la funcion de HSV
    #cv2.imshow('Cotorno', borde1)
    #cv2.waitKey(0)
    canny1 = cv2.Canny(dst1, 50, 150)# Usa la funcion canny para contorno
    canny2 = cv2.Canny(test, 50, 150)# Usa la funcion canny para contorno
```

```
    hmerge = np.hstack((canny1, canny2))    # Poner las imagenes juntos
    cv2.imshow("Canny", hmerge)    # el resultado de canny
    cv2.waitKey(0)
    kp1,kp2,des1,des2 = compre(canny1, canny2)# Los puntos claves de ORB
    #print(len(des1)) # El numero de los puntos claves
    #print(len(des2))
    img2 = cv2.drawKeypoints(img1, kp1, img1, color=(255, 0, 255))# Dibuja los puntos claves
    img3 = cv2.drawKeypoints(test, kp2, test, color=(255, 0, 255))# Dibuja los puntos claves
    hmerge = np.hstack((img2, img3))    # oner las imagenes juntos
    #cv2.imshow("point", hmerge)    # el resultado de ORB
    #cv2.waitKey(0)
    good = match(des1,des2)# el resultado de BFmatcher
    img4 = cv2.drawMatchesKnn(img1, kp1, test, kp2, good, None, flags=2)# Dibuja las parejas buenas
    #cv2.imshow("Resultado", img4)    # El resultado
    #cv2.waitKey(0)
    end = time.time()# Sale el tiempo de ejecucion
    print('Running time: %s Seconds' % (end - start))
    output(img4,good,des1)# Sale el resultado
```