Universidad Santo Tomás
Facultad de Ingeniería Electrónica

Diseño de robot cartesiano para procesos de segmentación y fenotipado de plantas con visión artificial

Trabajo de grado
para optar por el título de ingeniero electrónico

YEISON MIGUEL RODRÍGUEZ CRUZ
WILLIAM DAVID TELLEZ SALAMANCA

Director: Ing. Camilo Ernesto Pardo Beainy Esp. Msc. Phd. (st) ,
Co-director: Ing. Edgar Andrés Gutiérrez Cáceres Esp. Msc. Phd. (st) ,

Tunja, 2022

Universidad Santo Tomás

Universidad Santo Tomás
Facultad de Ingeniería Electrónica

Diseño de robot cartesiano para procesos de segmentación y fenotipado de plantas con visión artificial

Trabajo de grado

para optar por el título de ingeniero electrónico

YEISON MIGUEL RODRÍGUEZ CRUZ
WILLIAM DAVID TELLEZ SALAMANCA

Director: Ing. Camilo Ernesto Pardo Beainy Esp. Msc. Phd. (st) ,
Co-director: Ing. Edgar Andrés Gutiérrez Cáceres Esp. Msc. Phd. (st) ,

Aprobado por Fecha.

Camilo Ernesto Pardo Beainy Docente tutor	Edgar Andrés Gutiérrez Cáceres Docente cotutor
Primer Lector Docente	Segundo Lector Docente

Tunja, 2022

Universidad Santo Tomás
Facultad de Ingeniería Electrónica

Yeison Miguel Rodríguez Cruz
William David Tellez Salamanca, 2022.

. . . . . .
Yeison Miguel Rodríguez
Cruz
William David Tellez
Salamanca

Resumen

En la actualidad los avances que se han tenido en la parte de visión por computadora y a su vez tecnologías de aprendizaje automático han permitido que grandes científicos que estudian las plantas transformen sus técnicas de obtención de fenotipado de las plantas he incorporen unas nuevas formas de obtención haciendo uso de técnicas de visión por computadora, ya que como es una de las ramas más importantes hoy en día, tiene a su vez varias funcionalidades en diferentes sectores, uno de ellos he importante para este proyecto el sector agrícola. El proyecto se basa en la búsqueda de algoritmos de segmentación usando en la actualidad y a su vez el diseño de un robot cartesiano que permita realizar procesos de segmentación y fenotipado de plantas en ambientes controlados. El proyecto se realiza con el fin de hacer procesos detallados en plantas individuales observando diferentes características como lo son el crecimiento de estas plantas en diferentes condiciones de crecimiento, a la par de esto ver el crecimiento, el tamaño y la evolución de sus hojas, sus frutos y su área floral, y así, poder extraer características visuales de las plantas, que ayuden a hacer mediciones o cuantificar rasgos visuales.

Palabras clave

Visión por computadora, agricultura, segmentación y fenotipado, vision artificial,robot cartesiano, algoritmos de segmentación.

William David Tellez Salamanca

Dedicado a mis padres Amanda Salamanca y William Tellez quienes con su amor, cariño, paciencia y esfuerzo me han permitido cumplir hoy el sueño de ser un profesional, gracias por inculcarme el ejemplo de esfuerzo y dedicación para lograr todo lo que me propongo. A mi hermana Sara Tellez por su apoyo y cariño constante durante el transcurso de mi vida. A toda mi familia por que con sus consejos, palabras de aliento y oraciones me acompañaron en este proceso. A mi compañero de trabajo por ser un apoyo y excelente compañero a lo largo de nuestro paso por la universidad.

Yeison Miguel Rodrígues Cruz

Dedico este trabajo de grado a mi madre y a mi padre que han estado para mi todos estos años, siempre juntos, demostrándome cada día como debe ser mundo, con sus valores y su esfuerzo para salir adelante y llegar hasta aquí, a partir de ahora he de demostrar que su sacrificio valió la pena. También agradezco a mis hermanos por su apoyo, mi compañero de trabajo y todas las personas que de una u otra forma nos apoyaron en la realización de este proyecto.

Índice general

Índice de figuras

Índice de tablas

Capítulo 1

Introducción

La agricultura automatizada, la robótica agrícola y agricultura de precisión son un área la cual cada día va creciendo constantemente y a su vez tiene planes a futuro para los agricultores grandes y pequeños generando nuevas oportunidades, gastando mucho menos recursos, con mínima manipulación y menos actividades pesadas a la hora de realizar trabajos convencionales del campo. La implementación de estas nuevas tecnologías hace que las personas exploren y pongan en marcha nuevas ideas que beneficien al campo agrícola. Estas áreas abarcan desde uso de las tecnologías IoT, el uso de visión artificial hasta el punto de usar robots para dar solución a los problemas que se presentan cada día en el campo, todo esto con el fin de que los cultivos y los productos tenga una mejor calidad. En el campo agrícola alguno de los trabajos que se realizan para mantener a flote un cultivo son muy pesados, en algunos cultivos no es suficiente el trabajo de una sola persona, al contrario, se necesita de varias personas para suplir el tiempo en el cultivo, pero sin embargo en ocasiones todo este trabajo y este tiempo invertido no es suficiente, ya que algunas cosechas se pierden a veces por el mal trato de las personas que trabajan en el cultivo.

Por parte del estudio del cultivo como tal se tiene que algunos procesos que se realizan de forma manual tales como el riego correcto de las plantas, las mediadas correctas tanto de la planta en general como de sus partes (flor, fruto, hojas), su constante abonización pueden generar cierto porcentaje de error, ya que se está realizando de manera manual, pero para que no se generen estos errores estos datos que se mencionaron anteriormente requieren ser tomados de manera autónoma, buscando con estos que estos procesos sean más exactos o precisos disminuyendo así el tiempo laboral invertido de manera manual.

Lo que busca esta tesis es dar solución a una problemática que se está presentando a la hora de realizar los procesos de segmentación y fenotipado de las plantas en diferentes cultivos, plantaciones y bancos de plantas en ambientes controlados, haciendo uso de la robótica y la visión artificial, con el fin de ayudar al agricultor a que los procesos de recolección de datos sean menos laboriosos y extensos y a la par que no se presenten esos posibles márgenes de errores debido a los procesos que se llevan a cabo a la hora de hablar de fenotipado de plantas. Lo más cercano que se tiene para dar solución a la problemática que se genera al hablar de fenotipado de plantas es la adaptación de una cámara que hace uso de visión artificial (OAK-D), a un robot cartesiano para que este proceso de fenotipado se realice de manera autónoma y sin margen de error.

1.1. Formulación de preguntas

Por medio del desarrollo de este proyecto planteado el cual lleva como título "Diseño de robot cartesiano para procesos de segmentación y fenotipado de plantas con visión artificial", se busca dar respuestas a las siguientes preguntas:

- ¿Como se puede optimizar el proceso de segmentación y fenotipado en bancos de plantas al hacer uso de un robot cartesiano?

- ¿Mediante que algoritmo o técnica se va desarrollar el proceso de segmentación y fenotipado de plantas, teniendo en cuenta las necesidades del cultivador?

- ¿Cuáles podrían ser las posibles ventajas o desventajas que se presentan a la hora de optimizar el proceso de segmentación y fenotipado de plantas en bancos de plantas extenso?

- ¿Que posibles beneficios se observarían al optimizar el proceso de segmentación y fenotipado de plantas?

1.2. Definición del problema

En diferentes cultivos, plantaciones y bancos de plantas en ambientes controlados se realizan los procesos de segmentación y fenotipado de las plantas (especialmente la altura y el ancho) manualmente [2], esto conlleva a que procesos de recolección de datos sean laboriosos y extensos, a su vez puede presentarse márgenes de errores debido a los proceso que se llevan a cabo a la hora de hablar de fenotipado de plantas y con esto un requerimiento de tiempo considerable, por otra parte las grandes poblaciones de maleza que se puede presentar dentro de estos cultivos y bancos de plantas, pueden interrumpir con el crecimiento y evolución de las mismas [3]. Es allí donde el sistema propuesto da solución a la problemática presente haciendo uso de técnicas de visión artificial para suplir con la adquisición de datos generados de forma manual y a su vez caracterizar las poblaciones de maleza presentes en las plantas, el proyecto se va a llevar a cabo en un pequeño banco de plantas a las cuales se les va a hacer el proceso de segmentación y fenotipado de plantas [4] y a futuro poder implementarlo en cultivos y plantaciones con extensas áreas.

1.3. Delimitación del problema

El sistema de segmentación y fenotipado de plantas acompañado del robot cartesiano se implementará en bancos de plantas en ambientes controlados. En un principio este sistema propuesto se pondrá a prueba en pequeños bancos de plantas, con una posible ampliación en plantaciones con áreas más extensas.
El segmentado de estas plantas se realiza en el momento en que la planta empieza a crecer y se puede apreciar sus hojas (al igual que su área floral y área frutal para la toma de otros datos si se desea) [5].

1.4. Justificación

La mala adquisición de datos de fenotipado y el no conocimiento constante del crecimiento de estas plantas (tanto de sus hojas como de sus frutos), son una causa directa que hace que algunos procesos agrícolas sean más lentos y en algunos casos fallidos [2]. Por tal motivo se plantea un sistema en el cual se realicen procedimientos de segmentación y fenotipado a través de técnicas de visión artificial en un diseño de sistema robótico cartesiano, con el fin de extraer características visuales de las plantas [5–7] , que ayuden a hacer mediciones o cuantificar rasgos visuales mediante métodos no invasivos, planteando una alternativa para el mejoramiento en la toma de decisiones presentes en el proceso de fenotipado de plantas, acelerando procesos de mejora de cultivos a pequeña y gran escala. A fin de alcanzar resultados favorables, se busca incorporar técnicas de visión artificial e inteligencia artificial mediante el uso de software y a la par comunicación con un robot cartesiano para la correcta segmentación y fenotipado de plantas en un ambiente controlado.

1.5. Objetivos

1.5.1. Objetivo general

Diseñar un prototipo de robot cartesiano que permita realizar procesos de fenotipado de plantas en ambientes controlados explorando técnicas de segmentación en visión artificial

1.6. Objetivo Específicos

- Recopilar algoritmos de segmentación y fenotipado de plantas usados en la actualidad, revisando el comportamiento sobre imágenes de plantas apoyados en librerías de OpenCV y PlantCV.
- Dimensionar un robot cartesiano investigando modelos ya construidos y realizando un análisis dinámico y cinemático, que sea capaz de movilizar una cámara a través de un banco de plantas en condiciones ambientales controladas.
- Comprobar el funcionamiento de los algoritmos de segmentación que parezcan ser mas útiles para este proyecto, ejecutándolos en un software basado en lenguaje Python, para que, con ayuda de lo consultado se seleccionen los mas idóneos en la continuación de lo que se planteó en fase postgrado.

Capítulo 2

Marco referencial

2.1. Marco teórico

2.1.1. Segmentación de Imágenes

La segmentación de imágenes permite realizar el proceso de clasificación por píxel en el cual se le asigna una categoría a cada píxel de la imagen analizada. Para el proceso de fenotipado se requiere segmentar los diferentes píxeles que aparecen en la imagen tomada por el robot cartesiano en dos clases: planta (cultivos) y fondo (suelo y residuos). En los últimos tiempos las tecnologías de segmentación que comúnmente son utilizadas para esta finalidad son: segmentación que se basa en índices de color, segmentación basada en umbrales y segmentación que se basa en el aprendizaje [8].

2.1.2. Reconocimiento de objetos

Lo que hoy se conoce como sistemas de visión artificial, se pueden reconocer como imágenes basándas en diferentes aspectos visuales como lo puede ser el color, el tamaño, la forma etc, pero en algunos casos, estos sistemas pueden tener márgenes de error, ya que se genera ruido en la parte de reconocimiento por color de las imágenes. Sin embargo, para el caso de fenotipado se pueden utilizar el reconocimiento ya se por su tamaño, su forma o su color y así evitar o delimitar esos márgenes de error que se presentan.

Figura 2.1: *Reconocimiento de objetos (plantas). Fuente: Autor*

2.1.3. Imagnes RGB y de Profundidad

Al definir que es una imagen se hace referencia básicamente a que es una matriz de píxeles, siendo cada píxel comúnmente una composición de los colores rojo, verde y azul abreviado RGB. El formato RGB es un modelo de color basado en la síntesis aditiva. Con este sistema es posible representar un color mediante la mezcla por adición de los tres colores de luz primarios, que son utilizados por las cámaras digitales RGB y que se pueden almacenar en una imagen en formato .jpg (Joint Photographic Experts Group) [9]. Al unir estos dos ha remodelado la tarea robótica ya que la información de profundidad es invariable a las condiciones de iluminación y color, lo que resultó en una mejor comprensión de la forma y estructura del objeto.

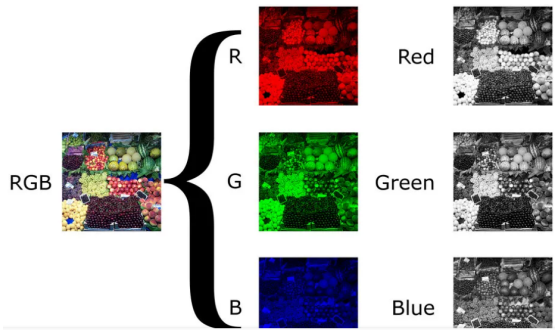

Figura 2.2: *Imagen RGB. Fuente:*

2.1.4. Cámara OAK-D

OpenCV IA Kit-D (OAK-D) es una cámara inteligente la cual cuenta con la capacidad de procesamiento de profundidad. El OAK-D tiene la capacidad de contener dentro de sí IA espacial. La capacidad detener esta IA espacial, es aplicar esta inteligencia artificial en el mundo real no solo para hacer la identificación de objetos sino también entendiendo dónde está en el espacio 3D, en tiempo real. Algunos problemas que se puede presentar en el mundo real se pueden resolver con esta OAK-D. Gracias a la cámara junto con su motor de profundidad estéreo que se encuentra integrado, OAK-D es capaz de realizar la detección de objetos con su cámara RGB de 12MP para identificar el objeto con percepción de profundidad. Casi todo lo que se encuentra en este dispositivo está bajo código abierto de licencia MIT (Massachusetts Institute of Technology). OAK-D está oficialmente afiliado a OpenCV. Tiene dos versiones. (1) ROBLE y (2) ROBLE-D. Estas dos tienen mucho en común, pero la multicámara del OAK-D nos permite obtener una vista 3D estereoscópica. Luxonis creo este dispositivo a la cual le integro inteligencia artificial. [10]

Especificación técnica

- Cámara de 12 MP a color 4K integrada
- 1 MP de par estéreo sincronizado con obturador global integrado
- Profundidad estéreo en tiempo real
- 4 TOPS de Procesamiento Visual
- Sin carga de cómputo en el host
- Cámara de IA espacial con inferencia de varias etapas
- Interfaz USB 3.0 tipo C

- Se puede usar con cualquier sistema operativo host que OpenVINO admita
- 4K / 30fpsCodificación H.265
- JPEG, H.264 y H.265/HEVC codificación

Figura 2.3: *Camara OAK-D. Fuente: [1]*

2.1.5. Robot Cartesiano

Los robots cartesianos son unos de los robots más utilizados en la industria. El robot cartesiano es uno de los robots más simple a la hora de poderlo entender, ya que pude ser fácilmente comprendido por el usuario que es el que lo va a manejar, dado que, como su nombre indica, el esquema de su movimiento se basa en el sistema de ejes cartesianos X-Y-Z. Básicamente este robot dependiendo el uso mueve un carrito o una quía en los ejes XY, por otro lado en el eje Z, mueve un cabezal axialmente hacia arriba y hacia abajo, este cabezal es el que va a sostener la cámara en nuestro caso [11].

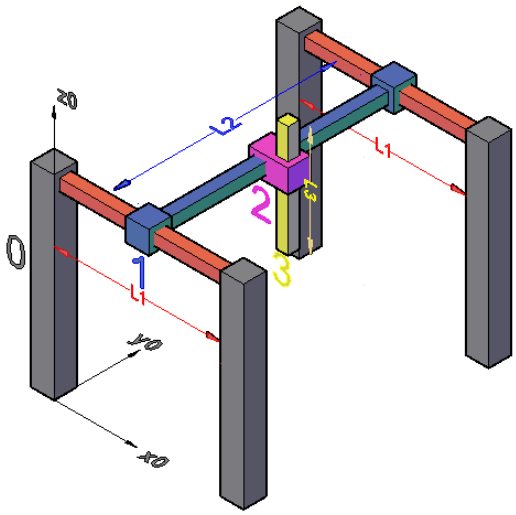

Figura 2.4: *Robot Cartesiano. Fuente: Autor*

2.2. Marco Conceptual

2.2.1. Bancos de plantas

Al realizar el proceso de segmentado de plantas, es fundamental tener presente algunas características físicas como el color del suelo en donde se vayan a poner las plantas, de igual forma el color de las macetas en donde van estas mismas y así, con ayuda del robot cartesiano se realice el proceso de fenotipado deteniéndose en cada planta ubicada en las diferentes macetas y de este modo tomar imágenes RGB y de profundidad individuales, esto con el fin de clasificar las plantas con respecto a su tamaño, su color, su área floral, su área frutal etc [3].

2.3. Marco de Antecedentes

En la actualidad existen algunas investigaciones que pueden ayudar a realizar un estado del arte de diferentes proyectos ya realizados que abarcan los diferentes tópicos que se presentan en el proyecto y a su vez dar paso a lo que se está planteando en el proyecto a realiza:

Según se presenta en [4], se estiman sistemas de mediciones 3D en el fenotipado de plantas, adjuntan un algoritmo segmentación original que permite obtener imágenes de profundidad de la planta desde una vista superior, prueban su algoritmo en diferentes plantas de rosal, yuca y manzano, usan una cámara de bajo costo y bajo peso, sus resultados contribuyen al fenotipado de plantas en ambientes controlados y de campo.

Según se expone por [2], se presenta un algoritmo de segmentación de plantas que extrae características de ancho y alto de una planta, el código es diseñado en una librería de visión por computador de código abierto (OpenCV) y lenguaje Python usando PyCharm como un entorno de desarrollo integrado(IDE), el algoritmo fue usado en plantas de ají con el propósito de estimar su ancho y altura, tuvieron imprecisiones en las medidas de 0.3 – 0.6 cm, también tuvo respuesta bastante buena al predecir tamaños plantas con rasgos geométricos complejos.

Según lo mencionado en [12], se desarrolla un proceso de segmentación y conteo de racimos en un viñedo usando una cámara de profundidad de grado montada en un vehículo agrícola. Ellos generan una reconstrucción de la planta y mediante métodos computacionales estiman el volumen de cada planta que estudian, implementan y evalúan cuatro marcos de aprendizaje profundo, AlexNet, VGG16, VGG19 y GoogleNet, sus métodos propuestos estiman un 91.5 % de precisión en sus pruebas de segmentación de racimos de uva.

Según lo tratado en el artículo [13], se describe el desarrollo de un robot cartesiano con cinco grados de libertad usado para realizar procesos de detección de hojas de cultivo de maíz usando un software llamado “LeafSpec” diseñado por ingenieros de Purdue Ag el cual genera imágenes hiperespectrales de las hojas. Realizaron pruebas en dos genotipos de cultivo y tres grados de tratamientos nitrogenados en ciclos continuos de tiempo, sus resultados se acercan a los obtenidos por un operador manual de este proceso.

Según lo expuesto por [6], tienen como objetivo principal visualizar de manera práctica el funcionamiento de un algoritmo de visión por computador con el fin de catalogar la hojas de fresa sanas y las que están infectadas con algún tipo de enfermedad que no requiere el uso de redes neuronales ni entrenamientos que puedan afectar en la ejecución del algoritmo, realizaron pruebas en entornos de iluminación exterior usando una cámara DLSR normal sin algún lente específico, las pruebas que realizaron sobre la deficiencia de hierro en las horas arrojaron resultados de precisión del 96 % frente al 93 % de un identificador humano.

Según se presenta en [7], proponen una manera automatizada de identificación de enfermedades en varias especies de cultivos, usan patrones binarios locales (LBP) para adquirir determinadas características y una caracterización de la clase. La metodología que usan es diseñar el código de clasificación específico para cada una de las enfermedades que revisaron entre ellas mildiú velloso saludable, mildiú polvoriento y podredumbre negra. Los algoritmos presentados se prueban y va-

rios cultivos teniendo un comportamiento bastante favorable.
Por otro lado, en [8], presentan un estudio de las técnicas de segmentación de plantas obtenidas a través de imágenes. Los criterios de evaluación para tres algoritmos de segmentación (basados en índice de color, umbral, y aprendizaje) fueron el procesamiento de los mismos, el funcionamiento de cada algoritmo con diferentes fondos detrás de la planta y otros parámetros, sus resultados son mostrados mediante graficas que evidencian las ventajas y desventajas de los algoritmos en ciertos criterios

Capítulo 3

Desarrollo del proyecto

3.1. Diseño de robot cartesiano

Para llegar a un robot cartesiano completo se parte de un primer paso, la definición del eslabón 0 que se encuentra representado por la parte rígida del robot que se encuentra fija en el suelo. El eslabón 1 está acoplado prismáticamente al eslabón 0 y a su vez generan un grado de libertad lineal, para este caso se ha elegido L1 el cual genera un desplazamiento a través del eje coordenado X, tal como se muestra en la figura 3.1

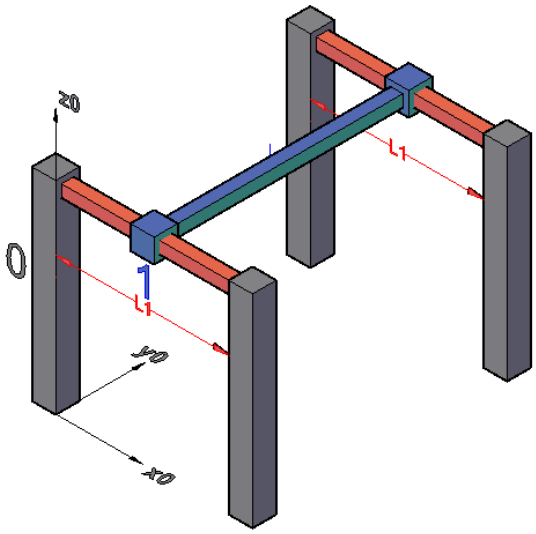

Figura 3.1: *Articulación prismática eslabón 0 y 1.*
Fuente:Autor

Al eslabón 1 se acopla un eslabón 2 mediante articulación prismática, se genera un segundo grado de libertad que ha sido delimitado por L2 el cual realiza un desplazamiento sobre el eje coordenado Y, tal como se muestra en la figura 3.2

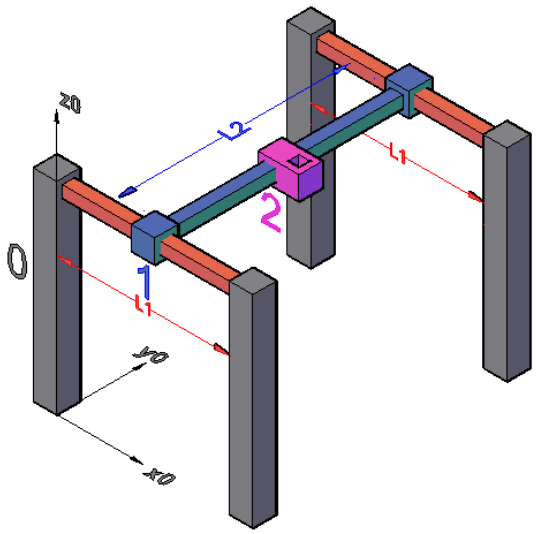

Figura 3.2: *Articulación prismática eslabón 1 y 2.*
Fuente: Autor

El eslabón 2 se encuentra articulado prismáticamente con el eslabón 3 que genera un tercer grado de libertad denotado por L3, este eslabón 3 realiza un desplazamiento lineal a través del eje coordenado Z, así como se muestra en la figura 3.3

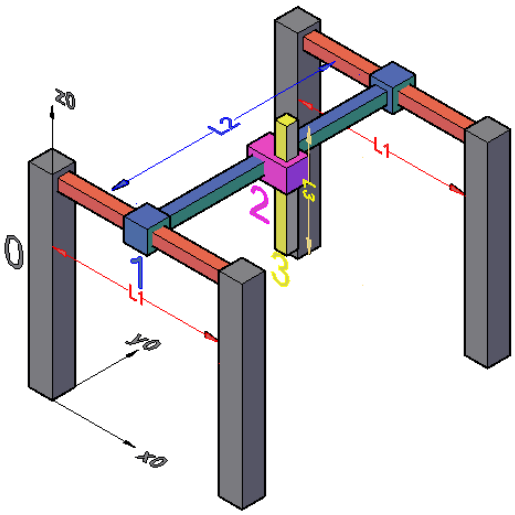

Figura 3.3: *Articulación prismática eslabón 2 y 3*
Fuente: Autor

3.1.1. Cinemática del robot cartesiano

La cinemática de un robot estudia los movimientos que realiza con respecto a un sistema de coordenadas que sirvan como referencia, a través de este estudio se pretende conocer el movimiento espacial que genera el robot, la posición y orientación final con respecto al sistema coordenado y la configuración que debe adoptar el robot para una posición y orientación del extremo final que ya se conoce. Denavit y Hartenber proponen un método para describir y plasmar la geometría espacial de unos elementos que pertenecen a una cadena cinemática, el método reduce la solución de un problema cinemático entre dos elementos rígidos en encontrar una matriz de transformación homogénea de 4x4 que genere una relación entre la localización del extremo del robot con el sistema de coordenadas de su posición base. El método de Denavit y Hartenber consiste en un algoritmo

de 16 pasos, en pocas palabras, el algoritmo indica como deben enumerarse y señalarse el número de eslabones, los ángulos que generan entre ellos, la rotación sobre si mismos (en caso de tener un movimiento giratorio en ligar de lineal), las distancias máximas que puede recorrer cada eslabón, entre otras cosas. Para cada eslabón se genera una matriz de 4x4 que se muestra en la eq. 3.1 que, al multiplicarse entre sí, se genera una matriz de transformación que relaciona la base con el extremo del robot como se muestra en la eq. 3.2. [14]

$$A_i^{i-1} = \begin{pmatrix} cos(\theta_i) & -cos(\alpha_i) & sen(\alpha_i)sen(\theta_i) & a_i cos(\theta_i) \\ sen(\theta_i) & cos(\alpha_i)cos(\theta_i) & -sen(\alpha_i)cos(\theta_i) & a_i sen(\theta_i) \\ 0 & sen(\alpha_i) & cos(\alpha_i) & d_i \\ 0 & 0 & 0 & 1 \end{pmatrix} \tag{3.1}$$

Matriz de transformación Denavit y Hartenber

$$T = ({A_1}^0 * {A_2}^1 * {A_3}^2 * \dots {A_n}^{n-1}) \tag{3.2}$$

Matriz de transformación base a extremo del robot

donde:

- θi: Es el ángulo formado por los ejes Xi-1 y Xi medido en un plano perpendicular al eje Zi-1. Este es el valor variable en la articulacion rotatoria.

- di: La distancia a lo largo del eje Zi-1 desde el origen del sistema de coordenadas (i-1) hasta la intersección del eje Zi-1 y el eje Xi. Este es un parámetro variable en las juntas prismáticas.

- ai: En el caso de una junta rotatoria, es la distancia a lo largo del eje Xi desde la intersección del eje Zi-1 con el eje Xi hasta el origen del i-ésimo sistema. En el caso de juntas prismáticas, se calcula como la distancia más corta entre los ejes Zi-1 y Zi.

- αi: El ángulo de separación del eje Zi-1 del eje Zi, medido usando la regla de la mano derecha en el plano perpendicular al eje Xi.

Modelo cinemático directo Robot cartesiano

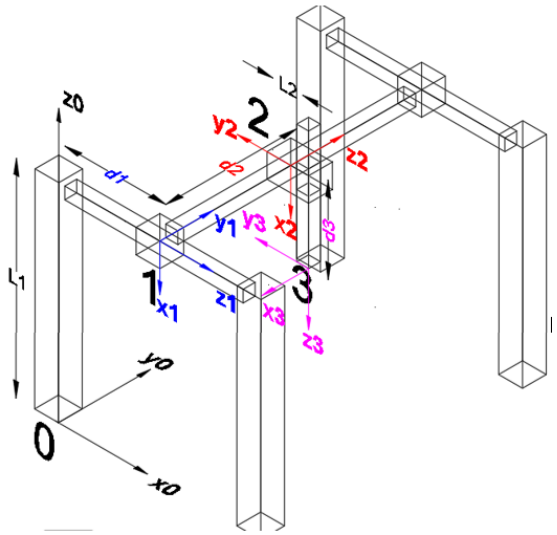

Figura 3.4: *Modelo cinemático directo Robot cartesiano Fuente: Autor*

Articulación	θ**i**	**di**	α**i**	**ai-1**
1	0	l1	-90	d1
2	0	d2	90	0
3	0	-d3	-90	l2

Tabla 3.1: *Parámetros Denavit-Hartenberg para Robot Cartesiano.*

Matriz 0A1 de transformación para eslabón 1 visto desde 0

$$0A1 = \begin{pmatrix} 1 & 0 & 0 & d1 \\ 0 & 0 & 1 & 0 \\ 0 & -1 & 0 & l1 \\ 0 & 0 & 0 & 1 \end{pmatrix} \tag{3.3}$$

Matriz 1A2 de transformación para eslabón 2 visto desde 1

$$1A2 = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 1 & 0 & d2 \\ 0 & 0 & 0 & 1 \end{pmatrix} \tag{3.4}$$

Matriz 2A3 de transformación para eslabón 3 visto desde 2

$$2A3 = \begin{pmatrix} 1 & 0 & 0 & l2 \\ 0 & 0 & 1 & 0 \\ 0 & -1 & 0 & d3 \\ 0 & 0 & 0 & 1 \end{pmatrix} \tag{3.5}$$

Matriz de transformación entre base y extremo

$$0A3 = \begin{pmatrix} 1 & 0 & 0 & d1 + l2 \\ 0 & 0 & 1 & d2 \\ 0 & -1 & 0 & l1 + d3 \\ 0 & 0 & 0 & 1 \end{pmatrix} \tag{3.6}$$

Modelo cinemático inverso Robot cartesiano

El modelo cinemático inverso permite hallar los valores de coordenadas que deben tomar cada uno de las articulaciones del robot, para que el efector se ubique en la posición exacta según una localización espacial especifica. En el modelo cinemático inverso la obtención de las ecuaciones esta altamente ligada a la construcción del robot, para este caso existen tres metodologías para obtener dichas ecuaciones, metodología geométrica, matriz de transformación homogénea y el proceso de desacoplo cinemático. Para este proyecto se decide usar el método geométrico ya que es bastante sencillo de implementar en robots de pocos grados de libertad, ya que solo es necesario realizar un análisis de los movimientos de las articulaciones prismáticas, este proceso ya se ha realizado en el modelo cinemático directo dando como resultado la siguiente imagen. [14]

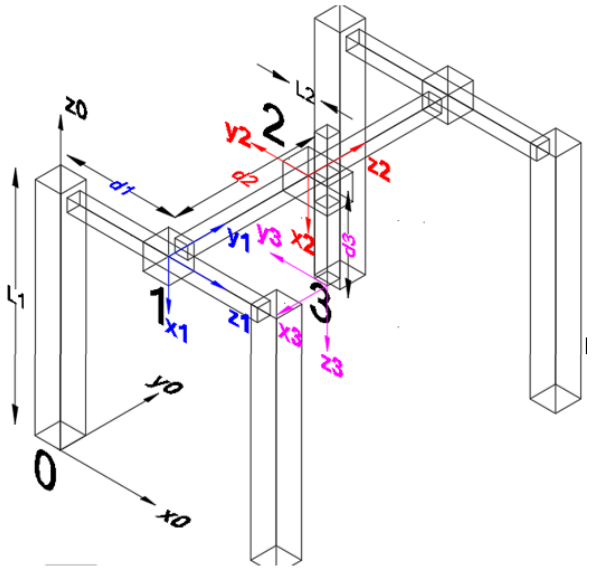

Figura 3.5: *Modelo cinemático inverso Robot cartesiano*
Fuente: autor

Se realiza un análisis de las **tres articulaciones** de manera individual, para la primera se usa como referencia (x1, y1, z1) con respecto al origen (x0, y0, z0), para este caso la articulación es prismática que posee solamente un grado de libertado, la ecuación para establecer la posición de

la **articulación 1** se describe como:

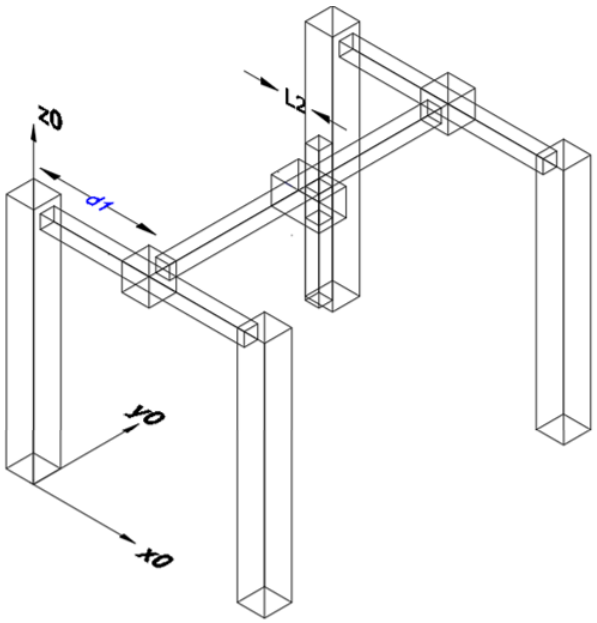

Figura 3.6: *Articulación prismática d1*
Fuente: autor

$$d1 = Px - l2 \tag{3.7}$$

Ecuación 3.7 para el desplazamiento articulación 1

donde :

d1 = Posición de la articulación 1
Px = Desplazamiento sobre el eje X
l2 = Distancia entre los ejes X3 y X3 son respecto a Y1

La articulación 2 la cual tiene como referencia (x2, y2, z2) también es tipo prismática con un grado de libertad y su posicionamiento se realiza a través del eje Y1, la ecuación que determina el desplazamiento de la articulación 2 es la siguiente:

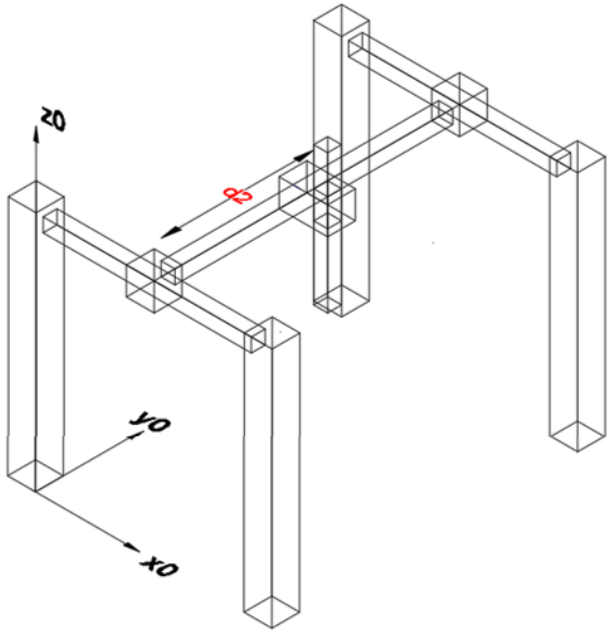

Figura 3.7: *Articulación prismática d2*
Fuente: autor

$$d2 = Py \tag{3.8}$$

Ecuación 3.8 para el desplazamiento articulación 2

donde:

d2 = Posición de la articulación 2
Py = Desplazamiento sobre el eje Y

El valor d2 y Py es igual dado que se generan un desplazamiento sobre el mismo eje.

La articulación 2 la cual tiene como referencia (x3, y3, z3), es una articulación prismática igual que las anteriores con un grado de libertad que se genera a través del eje Z3 y que se genera coincidencia con Z0, pero el desplazamiento se genera en dirección opuesta, la ecuación que describe el comportamiento de **la articulación 3** está dada por:

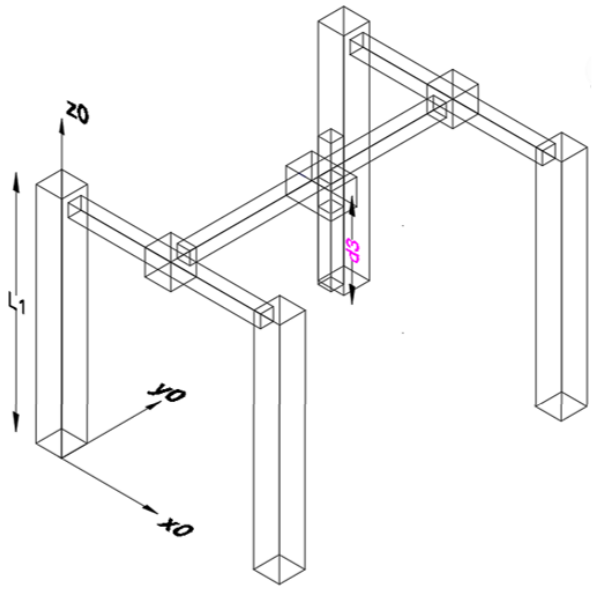

Figura 3.8: *Articulación prismática d3*
Fuente: autor

$$d3 = Pz - l1 \tag{3.9}$$

Ecuación 3.9 para el desplazamiento articulación 3

donde:

d2 = Posición de la articulación 3
Pz = Desplazamiento sobre el eje Z
l1 = Distancia entre X0 y Z1 sobre el eje Z0

De las anteriores ecuaciones (3.7, 3.8, 3.9) se despeja Px, Py y Pz, dando las siguientes ecuaciones:

$$Px = d1 + l2 \tag{3.10}$$

$$Py = d2 \tag{3.11}$$

$$Pz = d3 + l1 \tag{3.12}$$

Si se compara con la matriz de transformación 3.13 entre base y extremo se aprecia que los valores de la cuarta columna corresponden a la posición X, Y, Z que se acaban de hallar (ecuaciones 3.10, 3.11, 3.12), con esto se comprueba que el análisis cinemático inverso se ha realizado de manera correcta.

$$A_1{}^0 = \begin{pmatrix} 1 & 0 & 0 & \boxed{d1 + l2} \\ 0 & 0 & 1 & \boxed{d2} \\ 0 & -1 & 0 & \boxed{l1 + d3} \\ 0 & 0 & 0 & 1 \end{pmatrix} \tag{3.13}$$

La comprobación del análisis cinemático realizado y algunos ejemplos se pueden observar desde la figura 4.3 hasta 4.11"

3.1.2. Dinámica del Robot cartesiano

La dinámica se encarga de relacionar las fuerzas y el movimiento que se genera sobre un cuerpo, en ese sentido se establece un modelo dinámico sobre el robot cartesiano mediante un modelo matemático que enlaza: - Localización del robot y ubicación de cada una de las articulaciones tanto en el extremo como en el origen. - Las fuerzas aplicadas sobre cada una de las articulaciones - Las dimensiones del robot que establecen longitudes de las articulaciones y masas de cada una. A medida que se aumenta el número de articulaciones, incrementa la dificultad del diseño del modelo dinámico, y al ser diseñado normalmente a partir de ecuaciones diferenciales hace que obtener el modelo dinámico de un robot sea uno de los procesos más complejos en la robótica. Este diseño es importante para definir ciertas características para construir el robot, entre ellas se incluyen: el dimensionamiento de las articulaciones y de los actuadores que lo manipulan, la estructura mecánica y el movimiento del robot [14].

En este caso es bueno resaltar que en la mayoría de las aplicaciones reales de lo que hoy se conoce como robótica, las cargas manejadas no son suficientes como para generar deformaciones en los eslabones del robot, por lo tanto, se realiza el cálculo del modelo dinámico considerando las barras como rígidas. La obtención del modelo dinámico de un robot con una estructura mecánica rígida se basa en el planteamiento de lo que viene a ser el equilibrio de fuerzas establecido en la segunda ley de Newton, o su equivalente para movimientos de rotación, la denominada ley de Euler:

$$\begin{aligned} \sum F &= m * a \\ \sum T &= Iw' + wx(Iw) \end{aligned} \tag{3.14}$$

En la actualidad existen diversos algoritmos computacionales para solucionar el modelo dinámico de un robot, pero a la par numerosos investigadores han desarrollado alternativas basadas en la mecánica Newtoniana y Lagrangiana, estos investigadores lo hacen con el fin de obtener modelos manejables por los sistemas de cálculo de una manera más eficiente. Los métodos que más se utilizan hoy en día para la adquisición del modelo dinámico de un robot son, el algoritmo de Newton-Euler, Lagrange-Euler, Gibbs Apple y Trabajos Virtuales. Pero para el desarrollo de este proyecto se tomará como referencia el algoritmo de Newton-Euler, ya que su nivel de complejidad es menor en su desarrollo matemático y a su vez el nivel de eficiencia es mayor que los otros métodos nombrados anteriormente.

Formulación de Newton-Euler para el Modelo Dinámico de un Robot

Este algoritmo (Newton-Euler) se basa en operaciones escalares y vectoriales entre cantidades vectoriales y el producto de matrices vectoriales, y es más eficiente que las operaciones matriciales que implican fórmulas lagrangianas. De hecho, el orden de complejidad aritmética de esta fórmula es (n), lo que indica que depende directamente del número de grados de libertad del robot. El desarrollo de este algoritmo se basa en los siguientes pasos:

- **N-E1:** Como primero se tiene la asignación a cada eslabón de un sistema de referencias de acuerdo con las normas o lo planteado por Denavit-Hartenberg.
- **N-E2:** Obtención de las matrices de rotación Ri y su inversa (Ri-1).

- **N-E3:** Establecer las condiciones iniciales del sistema
  W0: Velocidad angular=[0,0,0]
  W'0: Aceleración angular=[0,0,0]
  V0: Velocidad lineal=[0,0,0]
  V'0: Aceleración lineal=[gx,gy,gz]

  En este caso mientras que la base del robot no esté en movimiento el W0, W'0 y V0 van a ser nulos.

- **N-E4:** Obtener la velocidad angular del sistema
- **N-E5:** Obtener la aceleración angular del sistema
- **N-E6:** Obtener la aceleración lineal del sistema
- **N-E7:** Obtener la aceleración lineal del centro de gravedad del eslabón i
- **N-E8:** Obtener la fuerza ejercida sobre el eslabón i
- **N-E9:** Obtener el par ejercido sobre el eslabón i
- **N-E10:** Obtener la fuerza o par aplicado a la articulación i

En este caso solo se deja mencionado el proceso a seguir para la obtención de la dinámica del robot cartesiano ya que el robot que se plantea no va a cargar una masa muy pesada en el eslabón del eje z.

3.2. Algoritmos de segmentación de imágenes en plantas

En la actualidad se encuentran disponibles librerias de python para uso libre especializadas en el tratamiento de imagenes por computador (OpenCv) y tratamiento de imagenes de plantas (PlantCv), los algoritmos de visión mencionados a continuacion pueden ser encontrados en el siguiente sitio web [15]. Entre estas librerias se encuentran presentes dos procesos principales para el analisis de imagenes de cultivos que pueden ser de gran ayuda para este proyecto, los cuales son:

3.2.1. Segmentación de objetos (detección/aislamiento)

Es un proceso que se enfoca principalmente en la detección de varios objetos dentro de una imagen y el aislamiento de dichos objetos específicos del resto de la imagen. El segmentado y aislamiento de plantas puede abordarse de las siguientes maneras en PlantCV. Dentro de esta sección de segmentación de objeto se tienen diferentes tipos como:

- **Umbralización binaria:** Permite extraer características binarias de una imagen en escala de grises. Como su nombre lo indica se pueden obtener dos tipos de imágenes, una clara y una oscura aplicando este método. La umbralización binaria se puede aplicar usando la siguiente función: **plantcv.threshold.binary(gray_img, threshold, max_value, object_type="light")** el cual tiene cuatro parámetros:
  - **gray_img:** Datos de imagen en escala de grises.
  - **threshold:** Valor de umbral binario a aplicar (0 - 255).
  - **max_value:** Valor a aplicar por encima del umbral (255 = blanco).

- **object_type:** En este parámetro se define si se quiere obtener la imagen clara con fondo oscuro, o la imagen oscura con fondo claro.

En la figura4.16 y 4.17 se muestra la ejecución del proceso de umbralización binaria sobre una planta y su resultado.

- **Umbral adaptativo gaussiano:** Crea una imagen binaria de forma automática de una imagen en escala de grises. La umbralización adaptativa gaussiana se puede aplicar usando la siguiente función: **plantcv.threshold.gaussian (gray_img, max_value, object_type="light")** el cual tiene tres parámetros:

  - **gray_img:** Datos de imagen en escala de grises.
  - **max_value:** Valor a aplicar por encima del umbral (255 = blanco).
  - **object_type:** En este parámetro se define si se quiere obtener la imagen clara con fondo oscuro, o la imagen oscura con fondo claro.

  En la figura 4.19 y 4.20 se muestra la ejecución del proceso de umbralización adaptativa gaussiana sobre una planta y su resultado.

- **Umbral adaptativo medio:** Permite crear una imagen binaria automáticamente a partir de una imagen en escala de grises. La umbralización adaptativa media se puede aplicar usando la siguiente función: **plantcv.threshold.mean(gray_img, max_value, object_type="light")** el cual tiene tres parámetros:

  - **gray_img:** Datos de imagen en escala de grises.
  - **max_value:** Valor a aplicar por encima del umbral (255 = blanco).
  - **object_type:** En este parámetro se define si se quiere obtener la imagen clara con fondo oscuro, o la imagen oscura con fondo claro.

  En la figura 4.22 y 4.23 se muestra la ejecución del proceso de umbralización adaptativa media sobre una planta y su resultado.

- **Umbral automático de Otsu:**

  Permite crear una imagen binaria automáticamente a partir de una imagen en escala de grises. La umbralización automática de Otsu se puede aplicar usando la siguiente función: **plantcv.threshold.otsu(gray_img, max_value, object_type="light")** el cual tiene tres parámetros:

  - **gray_img:** Datos de imagen en escala de grises.
  - **max_value:** Valor a aplicar por encima del umbral (255 = blanco).
  - **object_type:** En este parámetro se define si se quiere obtener la imagen clara con fondo oscuro, o la imagen oscura con fondo claro.

  En la figura 4.25 y 4.26 se muestra la ejecución del proceso de umbralización automática de Otsu sobre una planta y su resultado.

- **Umbral automático triangular:** Permite crear una imagen binaria automáticamente a partir de una imagen en escala de grises. La umbralización automática triangular se puede aplicar usando la siguiente función: **plantcv.threshold.triangle (gray_img, max_value, object_type="light", xstep=1)** el cual tiene cuatro parámetros:

  - **gray_img:** Datos de imagen en escala de grises.
  - **max_value:** Valor a aplicar por encima del umbral (255 = blanco).

- **object_type:** En este parámetro se define si se quiere obtener la imagen clara con fondo oscuro, o la imagen oscura con fondo claro.
- **xstep=1:** Valor para desplazarse en el eje X para describir los puntos y calcular la distancia, empezar en 1 predeterminado e ir cambiando.

En la figura 4.28 y 4.29 se muestra la ejecución del proceso de umbralización automática triangular sobre una planta y su resultado.

Normalización de imágenes: En este proceso se aplica un balance de blancos que contribuye en la disminución de la diferencia entre algunas plantas por alteraciones comunes de ilumina-ción, además que facilita la aplicación de otros algoritmos posteriores como puede ser el umbral. Adicionalmente a este proceso se puede agregar una referencia de color especifica que funcione como base para normalizar una o varias imágenes, este agregado se conoce como corrección de color. El balance de blancos es expresado por la siguiente función: **plantcv.white_balance(img, mode='hist', roi=None)** y tiene 3 parámetros:

- **Img:** datos de la imagen RGB o en escala de grises.
- **Mode:** "hist" es método predeterminado, se calcula un histograma para toda la imagen o el ROI especifico, el contenedor con mayor cantidad de pixeles se usa como referencia para cambiar los valores de la imagen. Si se usa "max" se define el pixel con valor máximo como referencia.
- **OI:** región de interés (x, y, width, height), si no se proporciona se usará toda la imagen como ROI de manera predeterminada.

En la figura 4.13 y 4.14 se muestra la ejecución del proceso de normalización en un banco de plantas y su resultado.

Después del umbral y la sustracción de fondo, es posible que exista algo de ruido (puntos que no forman un objeto como tal), para reducir este ruido se pueden aplicar los siguientes algoritmos:

- **Relleno:** Permite aplicar relleno a partir de una imagen binaria con un desenfoque medio o mascara. El relleno se puede aplicar usando la siguiente función: **plantcv.fill(bin_img, size)** el cual tiene dos parámetros.
  - **bin_img:** Datos de imagen en escala de grises
  - **size:** Tamaño mínimo de los objetos, los mas pequeños son los que se rellenan.

  En la figura 4.31 y 4.32 se muestra la ejecución del proceso de relleno sobre una planta y su resultado.
- **Desenfoque mediano:** Permite aplicar un filtro mediano a partir de una imagen en escala de grises. El desenfoque mediano se puede aplicar usando la siguiente función: **plantcv.median_blur (gray_img, ksize)** el cual tiene dos parámetros.
  - **gray_img:** Datos de imagen en escala de grises.
  - **ksize:** Tamaño del núcleo, entero o tupla.

  En la figura 4.34 y 4.35 se muestra la ejecución del proceso de desenfoque mediano sobre una planta y su resultado.

- **Desenfoque gaussiano:** Permite integrar un filtro de desenfoque gaussiano a una imagen, aplica el valor medio al pixel central de un núcleo. El desenfoque gaussiano se puede aplicar usando la siguiente función: **plantcv.gaussian_blur(img, ksize, sigma_x=0, sigma_y=None)** el cual tiene cuatro parámetros.

  - **Img:** Datos de la imagen RGB o en escala de grises.
  - **ksize:** Tamaño del núcleo, entero o tupla.
  - **sigma_x:** Desviación estándar en el eje X. Cero es predeterminado.
  - **sigma_y:** Desviación estándar en el eje Y. Si Y es ninguno toma el mismo valor de X.

  En la figura 4.37 y 4.38 se muestra la ejecución del proceso de desenfoque gaussiano sobre una planta y su resultado.

- **Región de interés:** Este método permite aislar el objeto que se desea estudiar del resto de la imagen. Primero se encuentran todos los objetos dentro de la imagen, luego se define la región de interés y posteriormente se comprueba si hay objetos dentro de la región. Para encontrar los objetos se hace uso de la siguiente función: **plantcv.find_objects(img, mask)** que tiene dos parámetros:

  - **img:** Datos de la imagen RGB o en escala de grises.
  - **mask:** Mascara binaria usada para detectar contornos.

  Para definir la regio de interés (ROI) se usa la siguiente función: **plantcv.roi.rectangle(img, x, y, h, w)** que tiene los siguiente cuatro parámetros:

  - **img:** Datos de la imagen RGB o en escala de grises.
  - **x:** Posición en X de la imagen.
  - **y:** Posición en Y de la imagen.
  - **h:** Altura del rectángulo.
  - **w:** Ancho del rectángulo.

  Y para encontrar los objetos dentro de la región ya creada se usa la siguiente función: **plantcv.roi_objects(img, roi_contour, roi_hierarchy, object_contour, obj_hierarchy, roi_type='partial')** que tiene los siguientes seis parámetros:

  - **img:** datos de la imagen RGB o en escala de grises.
  - **roi_contour:** Contorno creado.
  - **roi_hierarchy:** Jerarquía del contorno creado.
  - **object_contour:** Objetos hallados en la función de encontrar objetos.
  - **obj_hierarchy:** Jerarquía de los objetos hallados en la función de encontrar objetos.

  En la figura 4.40 y 4.41 se muestra la ejecución del proceso para encontrar objetos dentro de una región de interés y su resultado.

- **Otros tipos de región de interés (ROI)**

  - **Región de interés circular:** Permite crear una región circular sobre un objeto en una imagen usando la siguiente función: **plantcv.roi.circle(img, x, y, r)** que tiene los siguientes cuatro parámetros: img – Datos de imagen RGB o en escala de grises.
    - **x:** Posición en X.

- **y:** Posición en Y.
- **r:** Radio circular.

- **Región de interés elíptica:** Permite crear una región elíptica sobre un objeto en una imagen usando la siguiente función: **plantcv.roi.ellipse(img, x, y, r1, r2, angle)** que tiene los siguientes seis parámetros:
  - **img:** Datos de imagen RGB o en escala de grises.
  - **x:** Posición en X.
  - **y:** Posición en Y.
  - **r1:** Los dos radios más pequeños de la elipse.
  - **r2:** Los dos radios mas grandes de la elipse.
  - **angle:** Angulo de rotación de la elipse.
- **Región de interés poligonal:** Permite crear una región poligonal sobre un objeto en una imagen usando la siguiente función: **plantcv.roi.custom(img, vertices)** que requiere dos parámetros:
  - **img:** Datos de imagen RGB o en escala de grises.
  - **vertices:** Lista de coordenadas de los vértices que componen el polígono.
- **Regiones de interés múltiples:** Permite crear múltiples regiones de interés sobre una imagen, para esto se puede asignar un numero de filas y de columnas que diseña una matriz de ROIs, adicionalmente se asigna el espaciado entre ellas y las coordenadas de origen de la primera región, o también se pueden señalar las coordenadas individuales de cada región que desee crear. Para crear múltiples regiones de interés se puede usar la siguiente función: **plantcv.roi.multi(img, coord, radius, spacing=None, nrows=None, ncols=None)** que tiene los siguientes seis parámetros:
  - **img:** Datos de imagen RGB o en escala de grises.
  - **coord:** Tupla con dos elementos que definen el centro de ROI en la esquina superior izquierda.
  - **radius:** Radio de ROI.
  - **spacing:** Tupla con dos elementos que define el espaciado horizontal y vertical entre ROIs.
  - **nrows:** Número de filas en la matriz de ROIs.
  - **ncols:** Número de columnas en la matriz de ROIs.

  La figura 4.43 y 4.44 se muestra la ejecución de las anteriores funciones para crear regiones de interés y su resultado se muestran en la imagen.

3.2.2. Análisis de objetos

Es un proceso que se enfoca principalmente en el análisis individual de un objeto, extrayendo características propias del objeto como pueden ser: el color, área de interés, análisis térmico, análisis espectral, etc, dentro de este proceso se tiene:

- **Analizar color:** Esta función permite extraer características para espacios de color (rgb, hsv y lab). El histograma de color se puede obtener usando el siguiente comando: **plantcv.analyze_color(rgb_img, mask, hist_plot_type=None, label="default")** el cual tiene cuatro parámetros:
  - **rgb_img:** Datos de imagen en espacio de color rgb.
  - **mask:** Mascara binaria de los contornos seleccionados.
  - **hist_plot_type:** Tipo de espacio de color para realizar el histograma (rgb, hsv, lab, all)

- **label:** parámetro opcional, cambia el nombre de la variable de las observaciones registradas (predeterminado: "default").

En la figura 4.46 y 4.47 se muestra la ejecución del proceso para realizar un análisis de color a través de un histograma y su resultado.

- **Detección canny Edge:** Es un proceso crea una imagen binaria de los bordes a partir de un filtro canny el cual se puede obtener con el siguiente comando: **plantcv.canny_edge_detect(img, mask=None, sigma=1.0, low_thresh=None, high_thresh=None, thickness=1, mask_color=None, use_quantiles=False)** el cual tiene los siguientes parámetros:

  - **img:** datos de la imagen RGB o en escala de grises
  - **mask:** Mascara para restringir la ejecución de canny a un espacio determinado, imagen binaria.
  - **slow_thresh:** Límite interior del umbral de histéresis establecido en 10 % de la imagen.
  - **high_thresh:** Límite superior del umbral de histéresis establecido en 20 % de la imagen.
  - **thickness:** Grosor de los bordes, predeterminado en 1.
  - **mask_color:** Color de la máscara (ninguno, blanco o negro).
  - **use_quantiles:** Si es verdadero tratara los limites de los umbrales como cuantiles de la imagen de magnitud de borde no absolutos.

  En la figura 4.49 y 4.50 se muestra la ejecución del proceso para realizar detección canny Edge a través de un histograma de color y su resultado.

3.2.3. Visualización

- **Esqueleto:** Es un proceso que genera un esqueleto obtenido de una máscara respectiva de la imagen. La función que permite realizar crear el esqueleto de una planta es el siguiente: **plantcv.morphology.skeletonize(mask)** y tiene el siguiente parámetro:

  - **Mask:** Máscara binaria de la planta.

  La ejecución de la función para obtener el esqueleto de una planta y su resultado se muestran en la figura 4.52 y 4.53.

- **Prune:** Este proceso permite crear una imagen esqueleto y segmentarla en sus diferentes partes, la función para crear un prune es el siguiente: **plantcv.morphology.prune(skel_img, size=0, mask=None)** y tiene los siguientes tres parámetros:

  - **skel_img:** Imagen esqueleto (salida de la función para crear un esqueleto).
  - **size:** Las piezas más pequeñas que el valor ingresado son removidas del esqueleto (default = 0).
  - **mask:** Máscara binaria (opcional), si se agrega, las imágenes generadas se superponen a la máscara.

  La ejecución del proceso de prune y su resultado se muestran en la figura 4.55 y 4.56.

- **Longitud de segmentos:** Este proceso permite establecer las longitudes geodésicas de cada uno de los segmentos encontrados con la función Prune. Para obtener estas longitudes se puede usar la siguiente función: **plantcv.morphology.segment_path_length (segmented_img, objects, label="default")** que tiene los siguientes tres parámetros:

- **segmented_img:** Imagen segmentada o esqueleto.
- **objects:** Objetos segmentados del esqueleto (Prune).
- **label:** Parámetro opcional que modifica el nombre de las variables (predeterminado = "default").

La ejecución del proceso de medir las longitudes de los segmentos de un esqueleto y su resultado se muestran en la figura 4.58 y 4.59.

- **Ordenar segmentos:** Este proceso permite categorizar los segmentos en dos grandes grupos: hojas y otros objetos. Para clasificar los segmentos se puede usar la siguiente función: **plantcv.morphology.segment_sort (skel_img, objects, mask=Ninguno, first_stem=True)** que tiene los siguientes cuatro parámetros:
  - **skel_img:** Imagen esqueleto (salida de la función para crear un esqueleto).
  - **objects:** Objetos segmentados del esqueleto (Prune).
  - **mask:** Máscara binaria (opcional), si se agrega, las imágenes generadas se superponen a la máscara.
  - **first_stem:** Cuando es verdadero el segmento inferior se clasifica como tallo, si es falso el algoritmo se aplica a cada segmento.

  La ejecución del algoritmo de clasificación de segmentos y su resultado se muestran en la figura 4.61 y 4.62.

- **Rellenar segmentos:** Este proceso permite rellenar la mascara con el color correspondiente de cada uno de los segmentos. Para rellenar la mascara del color de los objetos segmentados se puede usar la siguiente función: **plantcv.morphology.fill_segments(mask, objects, stem_objects=None, label="default")** el cual tiene los siguientes cuatro parámetros:
  - **mask:** Máscara binaria.
  - **objects:** Objetos segmentados del esqueleto (Prune).
  - **stem_objects:** Este parámetro permite combinar los segmentos del tallo para formar uno solo.
  - **label:** Parámetro opcional que altera el nombre de la variable de los registros observados (predeterminado = "default")

  La ejecución del relleno de segmentos y su resultado se muestran en la figura 4.64 y 4.65.

- **Análisis de forma:** Este proceso permite extraer algunas características numéricas de la forma de un objeto. Para analizar la forma de un objeto se puede hacer uso de la siguiente función: plantcv.analyze_object(img, obj, mask, label="default") el cual tiene los siguientes cuatro parámetros:
  - **Img:** Imagen RGB o en escala de grises.
  - **obj:** Contorno de objeto u objetos agrupados.
  - **mask:** Máscara binaria.
  - **label:** Parámetro opcional que modifica el nombre de la variable con los datos observados (predeterminado = "default").

  La ejecución del proceso de análisis de forma y su resultado se muestran en la figura 4.67 y 4.68.

Capítulo 4

Análisis de Resultados

4.1. Diseño de robot cartesiano

Para realizar la simulación del robot cartesiano se hizo uso de la aplicación de RoboDK en el cual se armo un robot cartesiano con sus tres respectivos ejes (x,y,z) como se ve en la figura 4.1, para así poder generar una posible trayectoria (de manera virtual) que va a cumplir el robot, y así, realizar el fenotipado de las plantas presentes en el banco.

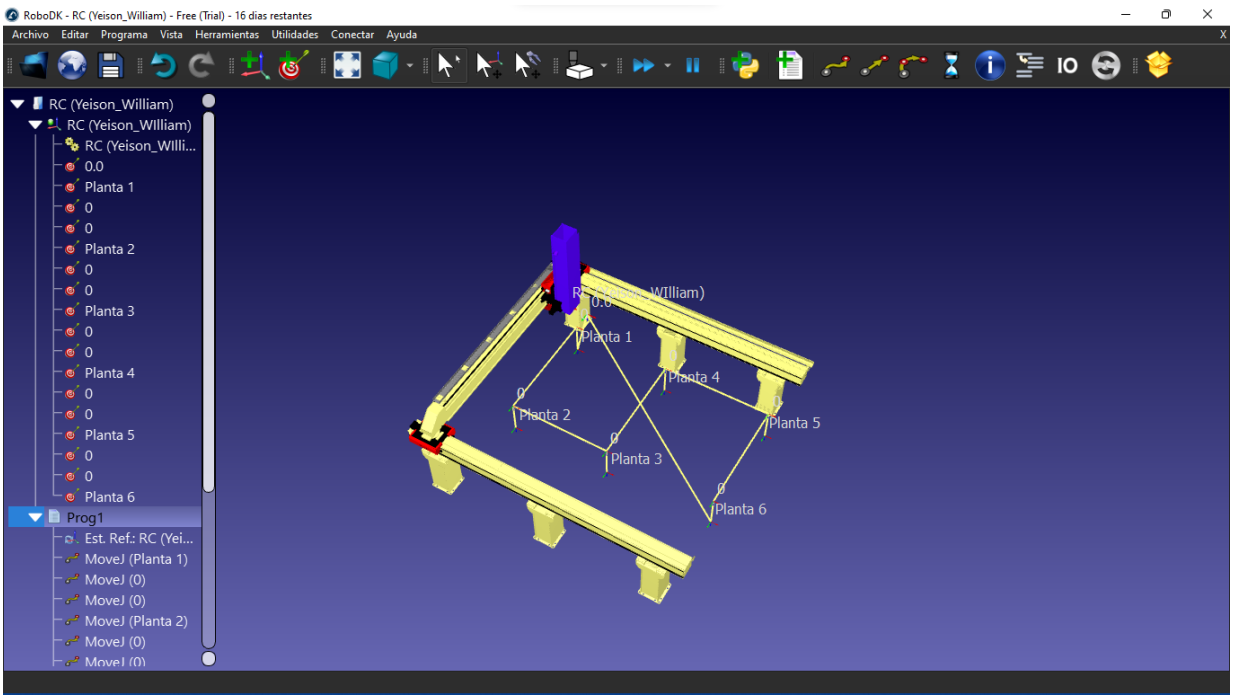

Figura 4.1: *Simulación de Robot cartesiano en RoboDK. Fuente: Autor*

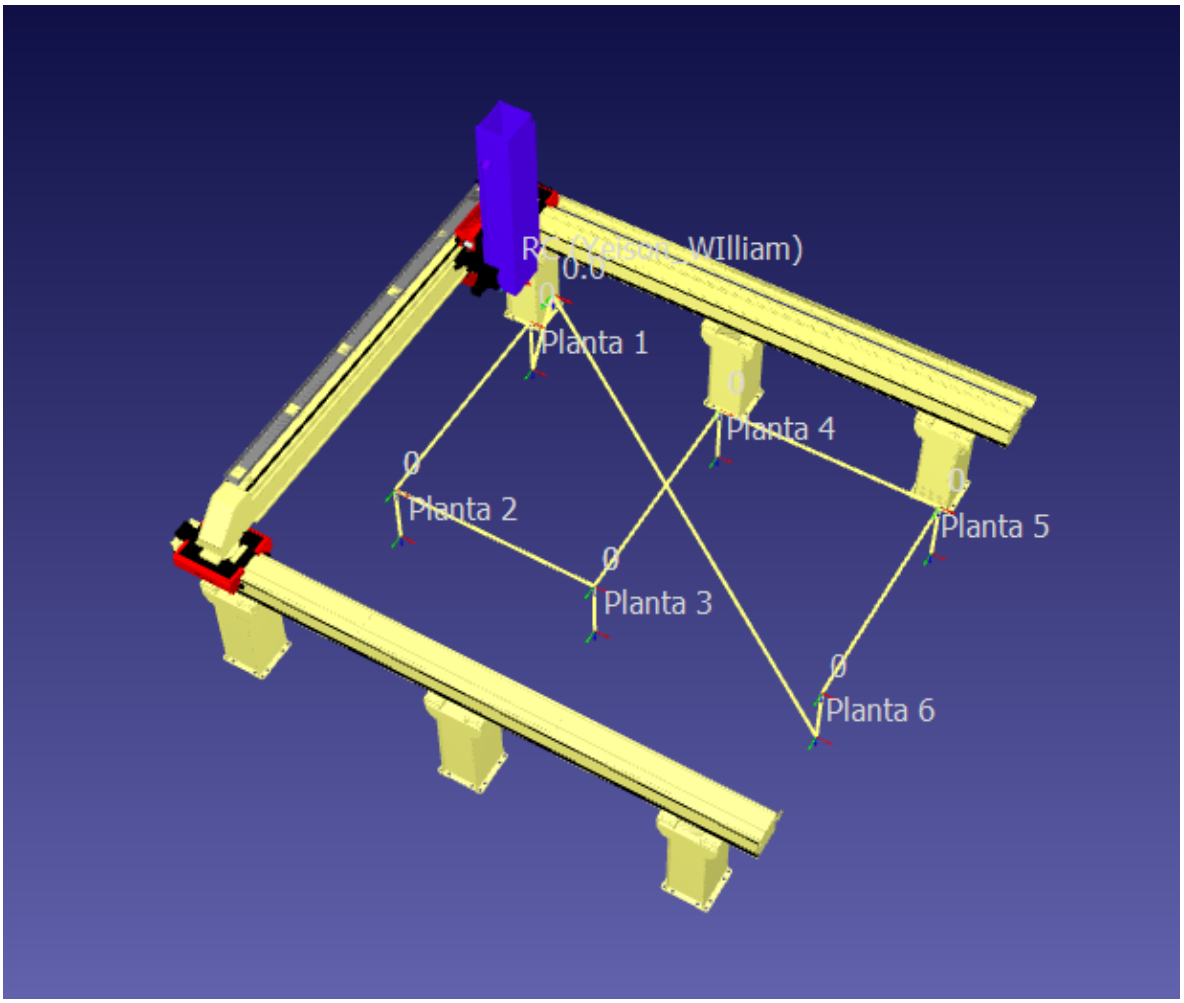

Figura 4.2: *Simulación de la trayectoria del Robot cartesiano. Fuente: Autor*

En la figura 4.2 se ve la simulación de una posible trayectoria que va a cumplir el robot cartesiano, en esta trayectoria el robot cruza por las diferentes materas en las cuales van a estar situadas diferentes variedades de plantas (definidos en la simulación como planta 1, planta 2, etc) para tomar las respectivas muestras y así poder generar el fenotipado de las plantas propuestas en el banco.

4.2. Verificacion del planteamiento del modelo cinemático del robot cartesiano a partir del método de Denavit y Hartenber

A continuación se realiza una simulación de las matrices y las transformaciones resultantes asignando los parámetros de diseño correspondientes, y así poder visualizar la posición del efector final, es decir la posición final a la que se va a dirigir el robot:

Ejemplo 1
L1 = 80 cm
L2 = -20 cm
D1 = 40 cm
D2 = 40 cm
D3 = 30 cm

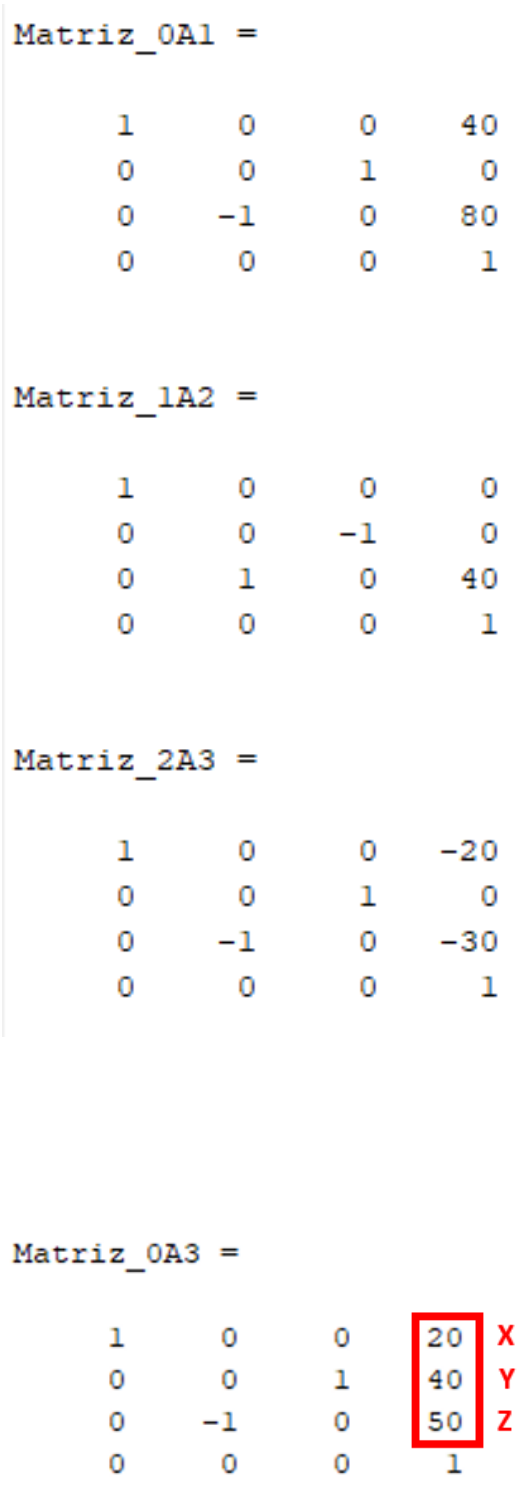

Figura 4.3: *Matrices y transformación homogénea. Fuente: Autor*

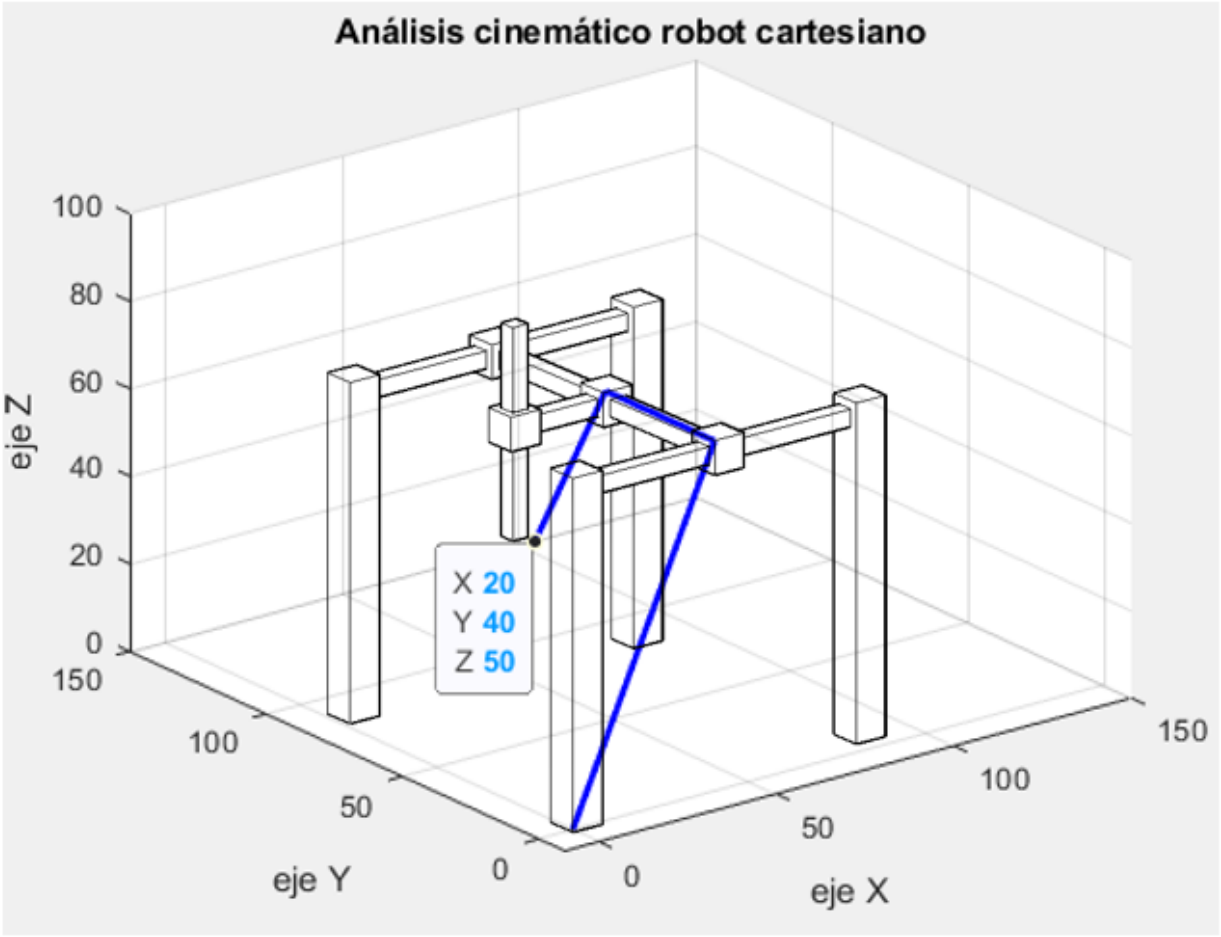

Figura 4.4: *Simulación análisis cinemático. Fuente: Autor*

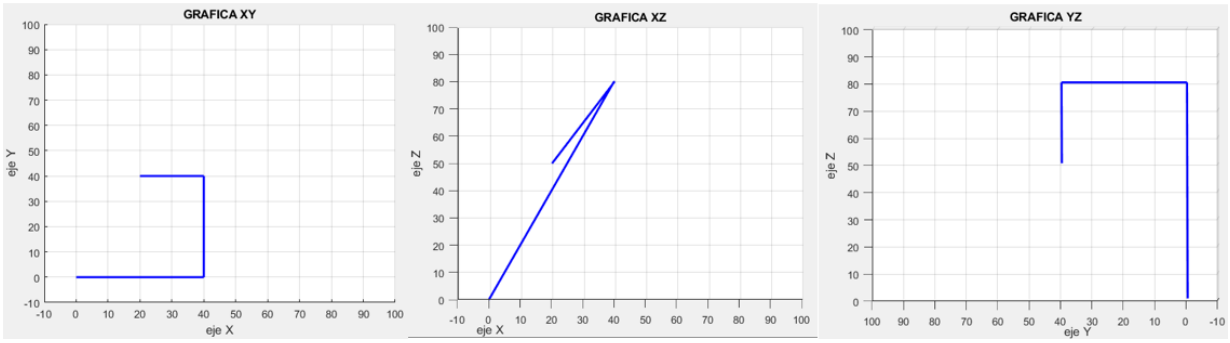

Figura 4.5: *Simulación análisis cinemático vista desde tres perspectivas diferentes. Fuente: Autor*

Ejemplo 2

L1 = 80 cm
L2 = 40 cm
D1 = 20 cm
D2 = 20 cm
D3 = 40 cm

```
Matriz_0A1 =

     1     0     0    20
     0     0     1     0
     0    -1     0    80
     0     0     0     1


Matriz_1A2 =

     1     0     0     0
     0     0    -1     0
     0     1     0    20
     0     0     0     1


Matriz_2A3 =

     1     0     0    40
     0     0     1     0
     0    -1     0   -40
     0     0     0     1
```

```
Matriz_0A3 =

     1     0     0    60   X
     0     0     1    20   Y
     0    -1     0    40   Z
     0     0     0     1
```

Figura 4.6: *Matrices y transformación homogénea. Fuente: Autor*

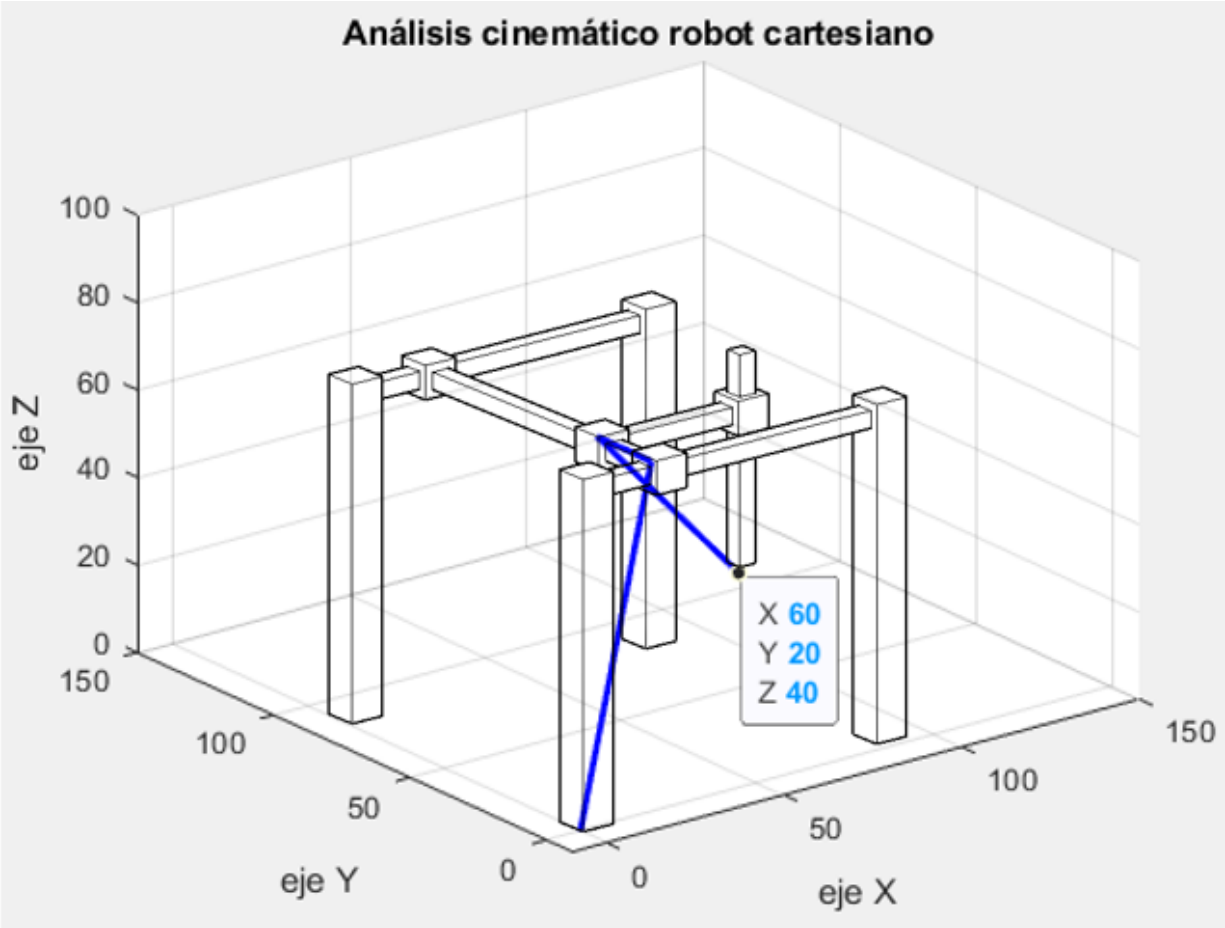

Figura 4.7: *Simulación análisis cinemático. Fuente: Autor*

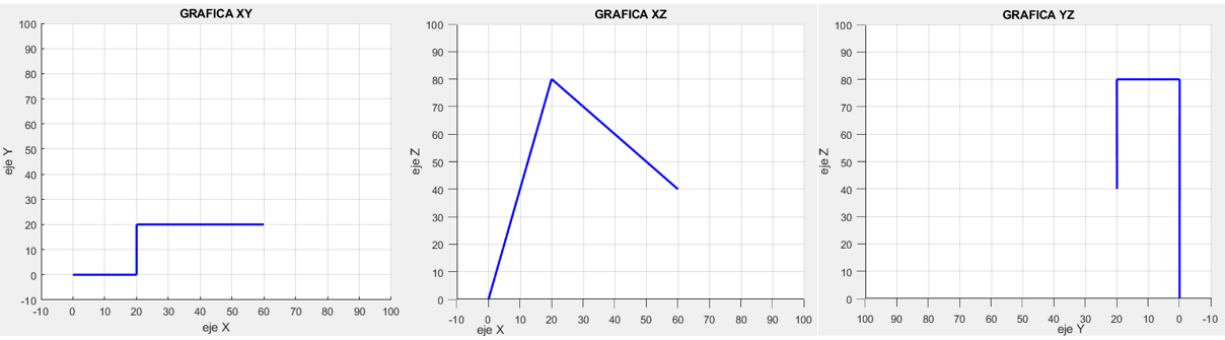

Figura 4.8: *Simulación análisis cinemático vista desde tres perspectivas diferentes. Fuente: Autor*

Ejemplo 3

L1 = 80 cm
L2 = -40 cm
D1 = 60 cm
D2 = 60 cm
D3 = 40 cm

```
Matriz_0A1 =

     1     0     0    60
     0     0     1     0
     0    -1     0    80
     0     0     0     1


Matriz_1A2 =

     1     0     0     0
     0     0    -1     0
     0     1     0    60
     0     0     0     1


Matriz_2A3 =

     1     0     0   -40
     0     0     1     0
     0    -1     0   -40
     0     0     0     1
```

```
Matriz_0A3 =

     1     0     0    20   X
     0     0     1    60   Y
     0    -1     0    40   Z
     0     0     0     1
```

Figura 4.9: *Matrices y transformación homogénea. Fuente: Autor*

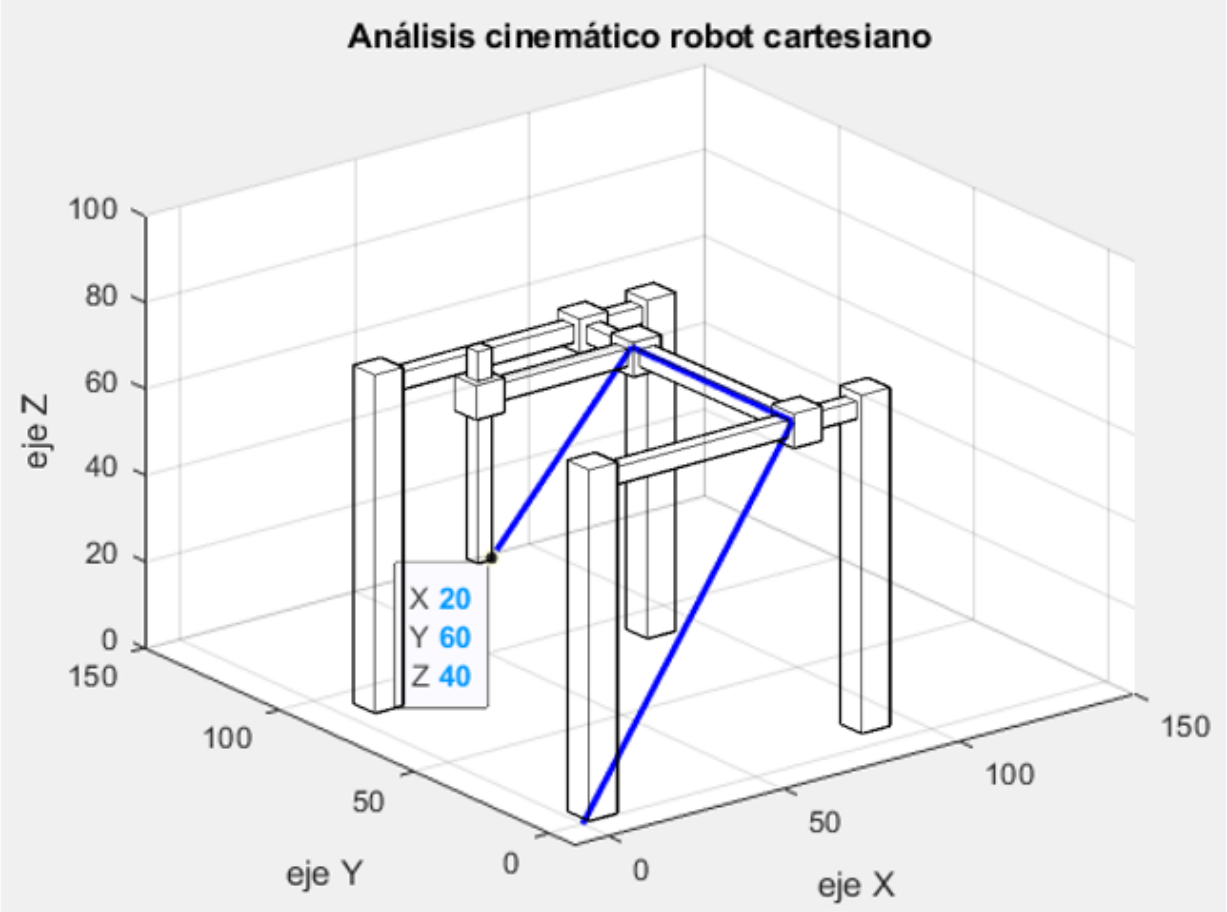

Figura 4.10: *Simulación análisis cinemático. Fuente: Autor*

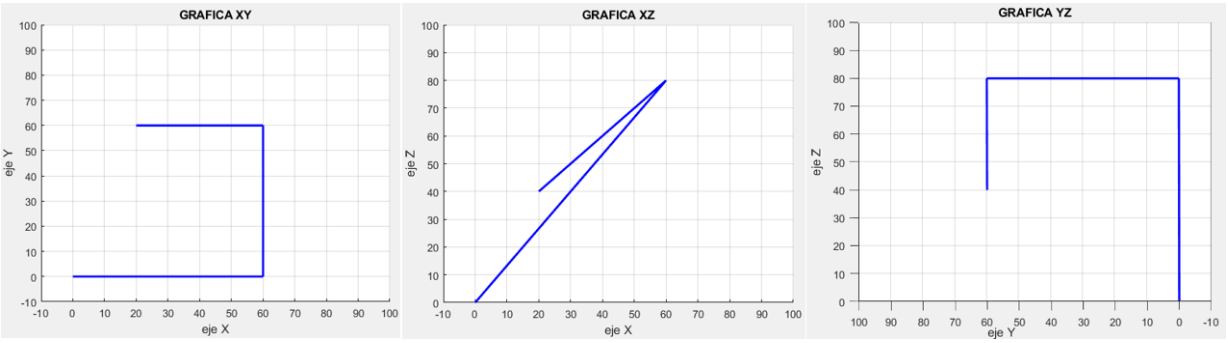

Figura 4.11: *Simulación análisis cinemático vista desde tres perspectivas diferentes. Fuente: Autor*

4.3. Comprobación de los algoritmos de segmentación de imágenes en plantas

Teniendo la investigación correspondiente y vista en la parte de desarrollo del proyecto sobre los algoritmos de segmentación de imágenes que se esta utilizando hoy en día, se procede a comprobar el funcionamiento de cada uno de ellos, para así seleccionar el algoritmo que más nos favorece y poder realizar el fenotipado de las plantas.

4.3.1. Normalización de imágenes

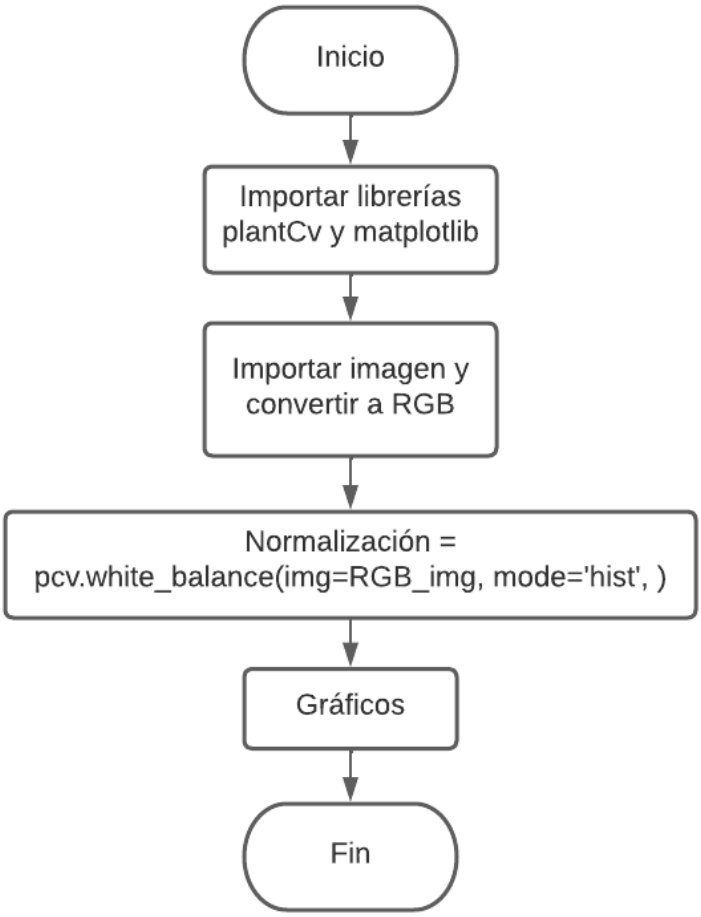

Figura 4.12: *Diagrama de flujo de la normalización de imágenes. Fuente: Autor*

Normalización de Imágenes

In [89]:
```
import cv2
from plantcv import plantcv as pcv
from matplotlib import pyplot as plt

#IMAGEN
img = cv2.imread("Banco1.jfif")
RGB_img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)

#define subplots size
fig, ax = plt.subplots(2, 2, figsize=(10,7))
fig.tight_layout()


#INICIO ALGORITMO
# Set global debug behavior to None (default), "print" (to file), or "plot" (Jupyter Notebooks or X11)
pcv.params.debug = "none"

# Corrects image based on color standard and stores output as corrected_img
corrected_img = pcv.white_balance(img=RGB_img, mode='hist', )

#FIN ALGORITMO

#GRAFICOS
plt.subplot(1, 2, 1)
plt.imshow(RGB_img)
plt.title('Original RGB')
plt.axis('off')


plt.subplot(1, 2, 2)
plt.imshow(corrected_img)
plt.title('Imagen normalizada')
plt.axis('off')
plt.show()
```

Figura 4.13: *Código Normalización de Imágenes. Fuente: Autor*

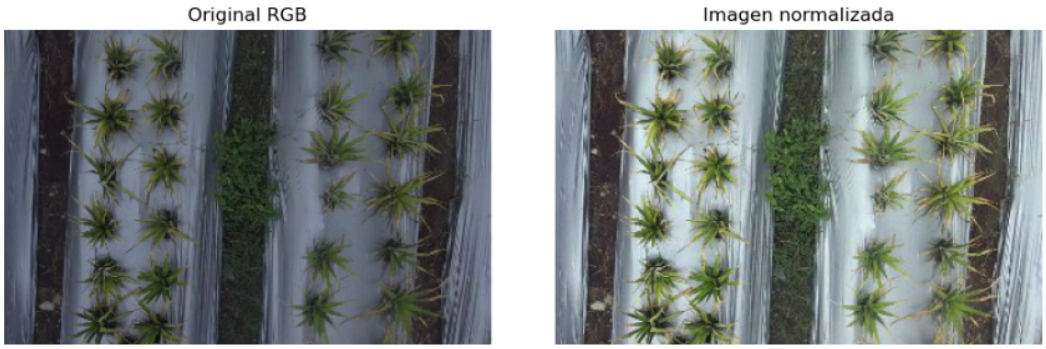

Figura 4.14: *Imagen Normalizada de una planta. Fuente: Autor*

En la figura 4.14 se muestra la ejecución del proceso de normalización en un banco de plantas y su resultado, en este algoritmo se hizo un balance de blancos que contribuye en la disminución de la diferencia entre algunas plantas por alteraciones comunes de iluminación. Esta normalización en lo que viene a ser el procesamiento de imágenes es usada básicamente para cambiar la parte del nivel de intensidad de los píxeles, y así, obtener un mejor contraste en las imágenes que tienen poco contraste provocado por el deslumbramiento. Este algoritmo de normalización de imagen tiene unas etapas o argumentos por así decirlo, que lleva a la generación de la imagen deseada: **El primer argumento o etapa** de esta función viene a ser la imagen de origen que se quiere normalizar, **como segunda etapa** es la imagen de destino, en la cual se crea una imagen de salida con el tamaño o las dimensiones que se deseen, **en el tercer argumento** se tiene el valor inferior del rango en el que se quiere normalizar la imagen, **en la cuarta etapa** se tiene ahora el valor superior del rango en el que se quiere normalizar la imagen, ya en **la quinta etapa o argumento** se hace uso del tipo de normalización que se desea utilizar (puede ser cv2.NORM_INF, cv2.NORM_L1 y cv2.NORM_MINMAX), en este quinto argumento cada tipo tiene sus

propias formulas para calcular la normalización, **en el sexto argumento o etapa** se utiliza mas que todo para fijar el tipo de datos que se desea para la imagen de salida y por ultimo **en el séptimo argumento** se usa mas que todo para generar una máscara, esta etapa es muy util ya que al generar esta mascara se puede normalizar la parte de la imagen que se desea en vez de normalizar toda la imagen completa.

4.3.2. Métodos de segmentación de objetos

Umbralización binaria

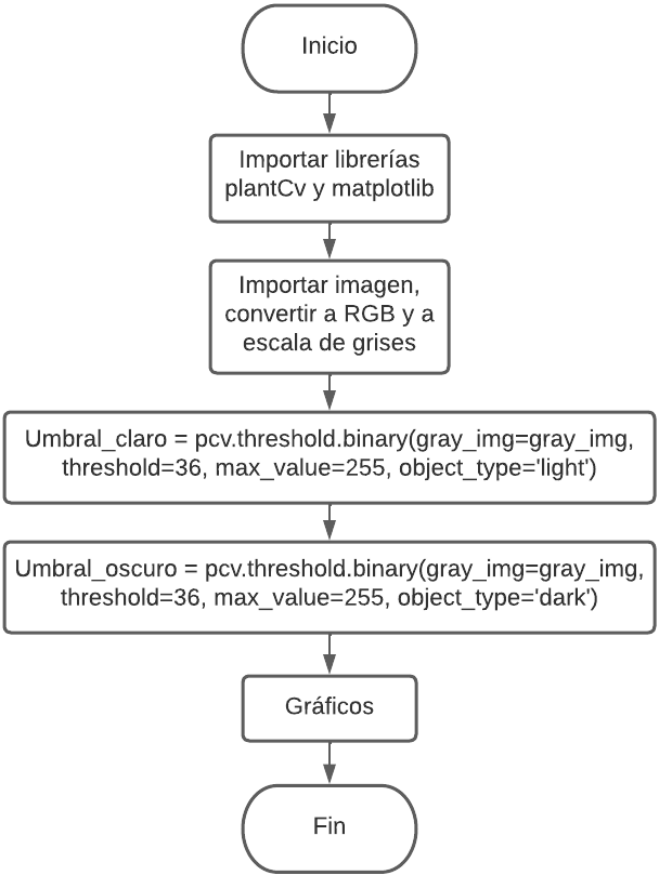

Figura 4.15: *Diagrama de flujo de la umbralización binaria. Fuente: Autor*

In [73]:
```python
import cv2
from plantcv import plantcv as pcv
from matplotlib import pyplot as plt

#IMAGEN
img1 = cv2.imread("original_image.jpg")
RGB_img = cv2.cvtColor(img1, cv2.COLOR_BGR2RGB)

# Saturation ('s') channel is output
gray_img = pcv.rgb2gray_hsv(rgb_img=RGB_img, channel='s')

#define subplots size
fig, ax = plt.subplots(2, 2, figsize=(10,7))
fig.tight_layout()


#INICIO ALGORITMO

# Set global debug behavior to None (default), "print" (to file),
# or "plot" (Jupyter Notebooks or X11)

pcv.params.debug = "none"

# Create binary image from a gray image based on threshold values,
# targeting light objects in the image.
threshold_light = pcv.threshold.binary(gray_img=gray_img, threshold=36, max_value=255, object_type='light')

# Create binary image from a gray image based on threshold values,
# targeting dark objects in the image.
threshold_dark = pcv.threshold.binary(gray_img=gray_img, threshold=36, max_value=255, object_type='dark')

#FIN ALGORITMO
```

```python
#GRAFICOS
plt.subplot(2, 2, 1)
plt.imshow(RGB_img)
plt.title('Original RGB')
plt.axis('off')

plt.subplot(2, 2, 3)
plt.imshow(threshold_light, cmap='gray')
plt.title('Umbral claro')
plt.axis('off')

plt.subplot(2, 2, 4)
plt.imshow(threshold_dark, cmap='gray')
plt.title('Umbral oscuro')
plt.axis('off')

plt.subplot(2, 2, 2)
plt.imshow(gray_img, cmap='gray')
plt.title('Gris (Saturación)')
plt.axis('off')
plt.show()
```

Figura 4.16: *Código Umbralización binaria. Fuente: Autor*

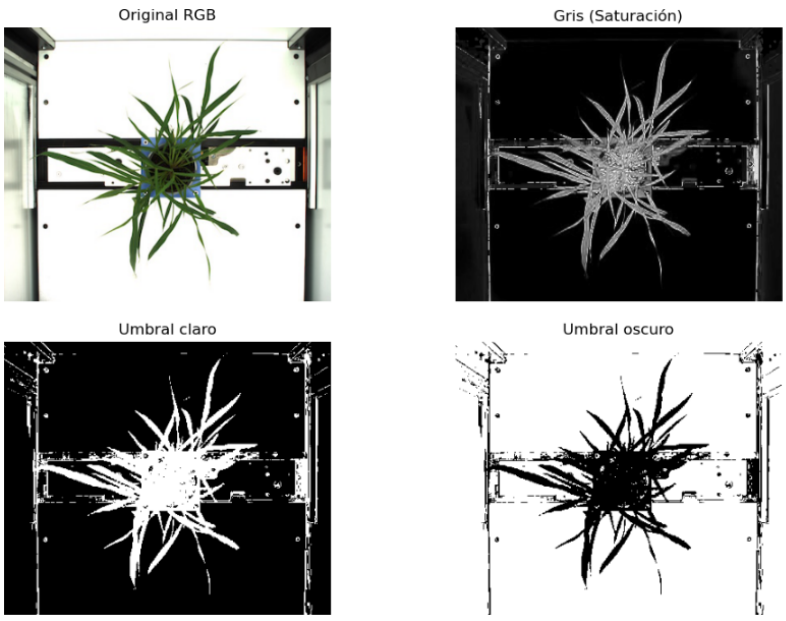

Figura 4.17: *Imagen Umbralizada de Forma Binaria de una Imagen. Fuente: Autor*

En la figura 4.17 se muestra la ejecución del proceso de umbralización binaria sobre una planta y su resultado, en este algoritmo se realizo la extracción de características binarias de la imagen presente en escala de grises, se obtienen dos tipos de imágenes, una clara y una oscura aplicando este método. Para explicar mejor el uso de esta umbralización binaria se tiene que es un proceso en donde los pixeles provenientes de la imagen de origen tienen valores inferiores al umbral que se estableció previamiente, se establecen como pertenecientes al fondo de la imagen, pero si estos pixeles tienen los valores superiores al umbral establecido, son clasificados como pertenecientes al cuadro principal de la imagen, en este caso la planta.

Umbral adaptativo gaussiano

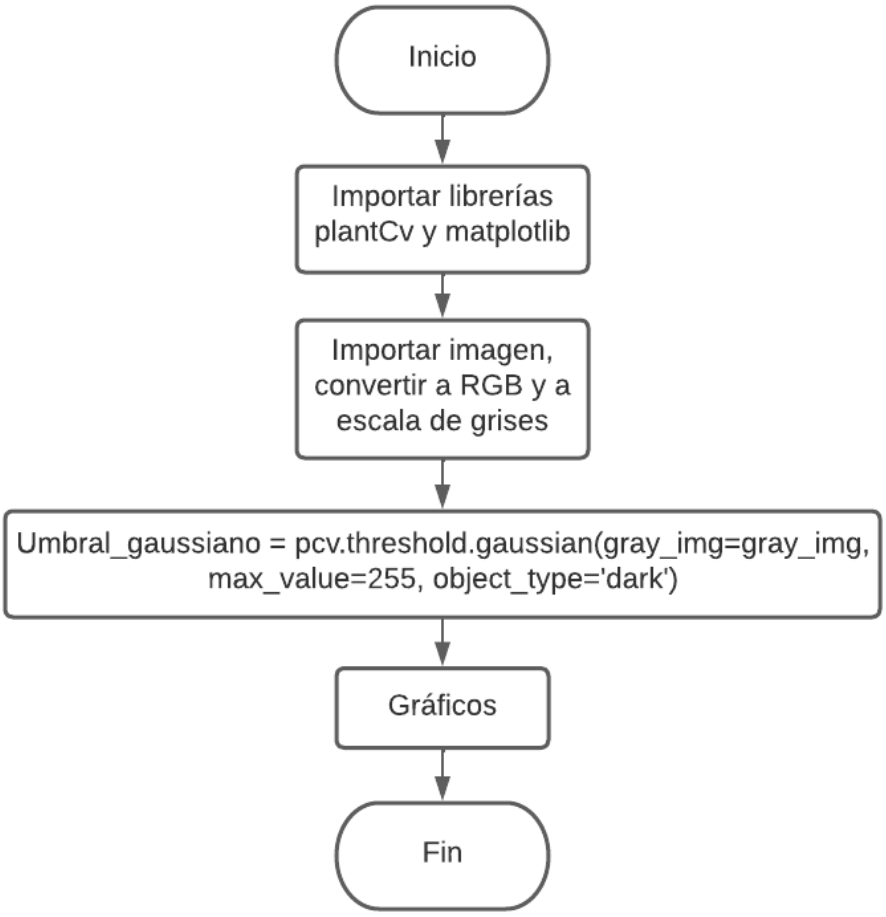

Figura 4.18: *Diagrama de flujo del umbral adaptativo gaussiano . Fuente: Autor*

```python
import cv2
from plantcv import plantcv as pcv
from matplotlib import pyplot as plt

#IMAGEN
img1 = cv2.imread("original_image.jpg")
RGB_img = cv2.cvtColor(img1, cv2.COLOR_BGR2RGB)

#   channel = color subchannel ('l' = lightness, 'a' = green-magenta , 'b' = blue-yellow)
gray_img = pcv.rgb2gray_lab(rgb_img=RGB_img, channel='a')
#define subplots size
fig, ax = plt.subplots(1, 3, figsize=(10,7))
fig.tight_layout()


#INICIO ALGORITMO
pcv.params.debug = "none"

# Create binary image from a gray image based
threshold_gaussian = pcv.threshold.gaussian(gray_img=gray_img, max_value=255, object_type='dark')

#FIN ALGORITMO

#GRAFICOS
plt.subplot(1, 3, 1)
plt.imshow(RGB_img)
plt.title('Original RGB')
plt.axis('off')

plt.subplot(1, 3, 2)
plt.imshow(gray_img, cmap='gray')
plt.title('Gris (magenta-verde)')
plt.axis('off')

plt.subplot(1, 3, 3)
plt.imshow(threshold_gaussian, cmap='gray')
plt.title('Umbral gaussiano')
plt.axis('off')
plt.show()
```

Figura 4.19: *Código Umbral adaptativo gaussiano . Fuente: Autor*

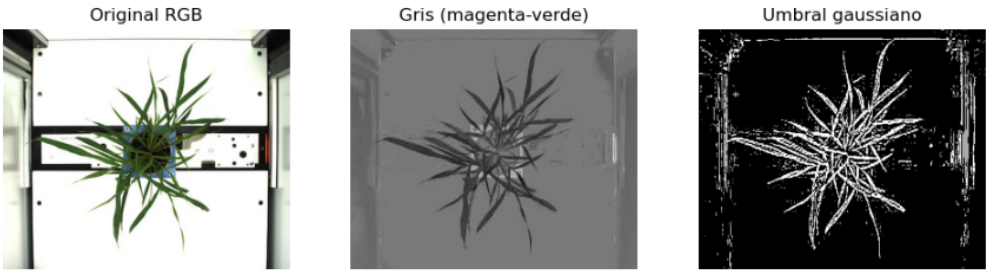

Figura 4.20: *Imagen Haciendo uso de Umbral Adaptativo Gaussiano. Fuente: Autor*

En la figura 4.20 se muestra la ejecución del proceso de umbralización adaptativa gaussiana sobre una planta y su resultado, en este algoritmo se creo una imagen binaria de forma automática de una imagen en escala de grises, por otro lado en este umbral adaptativo gaussiano se asigno un valor de umbral que se calcula o que varía con base a las características del entorno que se esta evaluando (en este caso la imagen de una planta), esto permitió segmentar la imagen para observar si el fondo tenia distintos niveles de grises o iluminación no uniforme y corregirlos.

Umbral adaptativo medio

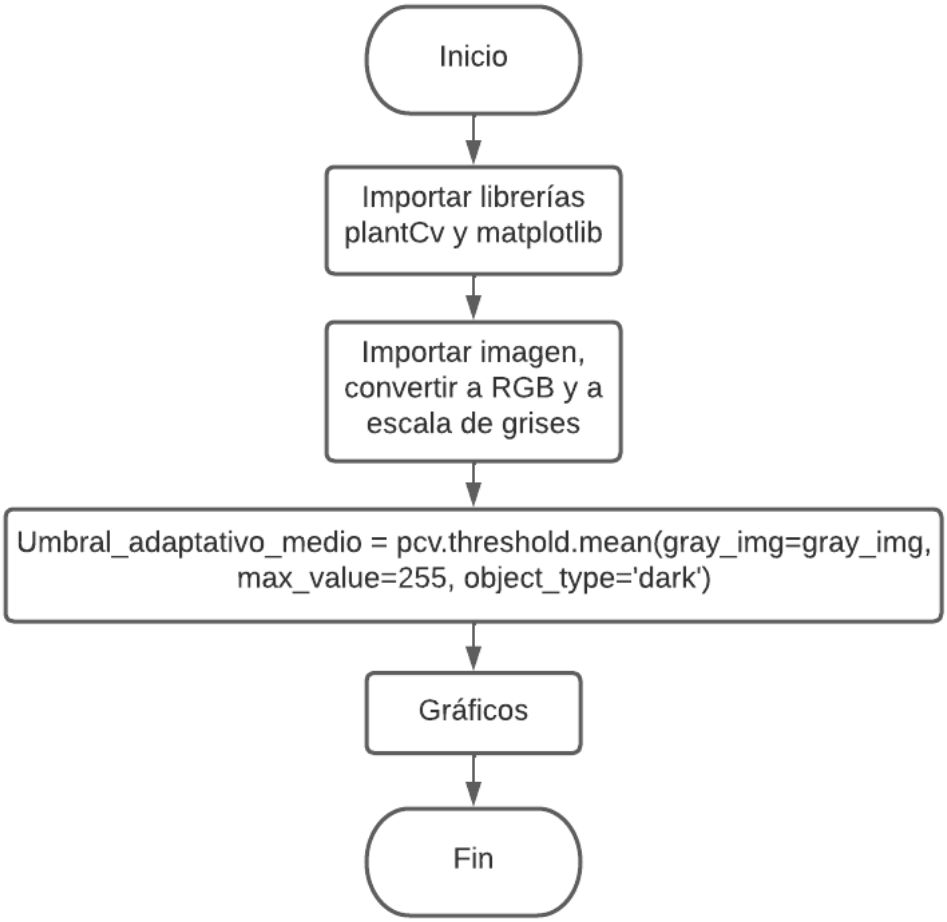

Figura 4.21: *Diagrama de flujo del umbral adaptativo medio . Fuente: Autor*

```python
import cv2
from plantcv import plantcv as pcv
from matplotlib import pyplot as plt

#IMAGEN
img1 = cv2.imread("original_image.jpg")
RGB_img = cv2.cvtColor(img1, cv2.COLOR_BGR2RGB)
#   channel = color subchannel ('l' = lightness, 'a' = green-magenta , 'b' = blue-yellow)
gray_img = pcv.rgb2gray_lab(rgb_img=RGB_img, channel='a')
#define subplots size
fig, ax = plt.subplots(1, 3, figsize=(10,7))
fig.tight_layout()

#INICIO ALGORITMO
pcv.params.debug = " none"

# Create binary image from a gray image based
threshold_mean = pcv.threshold.mean(gray_img=gray_img, max_value=255, object_type='dark')
#FIN ALGORITMO

#GRAFICOS
plt.subplot(1, 3, 1)
plt.imshow(RGB_img)
plt.title('Original RGB')
plt.axis('off')

plt.subplot(1, 3, 2)
plt.imshow(gray_img, cmap='gray')
plt.title('Gris (magenta-verde)')
plt.axis('off')

plt.subplot(1, 3, 3)
plt.imshow(threshold_mean, cmap='gray')
plt.title('Umbral adaptativo medio')
plt.axis('off')
plt.show()
```

Figura 4.22: *Código Umbral adaptativo medio . Fuente: Autor*

Figura 4.23: *Imagen Haciendo uso de Umbral adaptativo medio. Fuente: Autor*

En la figura 4.23 se muestra la ejecución del proceso de umbralización adaptativa media sobre una planta y su resultado, esta umbralizacion permite crear una imagen binaria automáticamente a partir de una imagen en escala de grises, por otro lado en este umbral adaptativo medio se asigno un valor de umbral que se calcula o que varía con base a las características del entorno que se esta evaluando (en este caso la imagen de una planta), esto permitió segmentar la imagen para observar si el fondo tenia distintos niveles de grises o iluminación no uniforme y corregirlos.

Umbral automático de Otsu

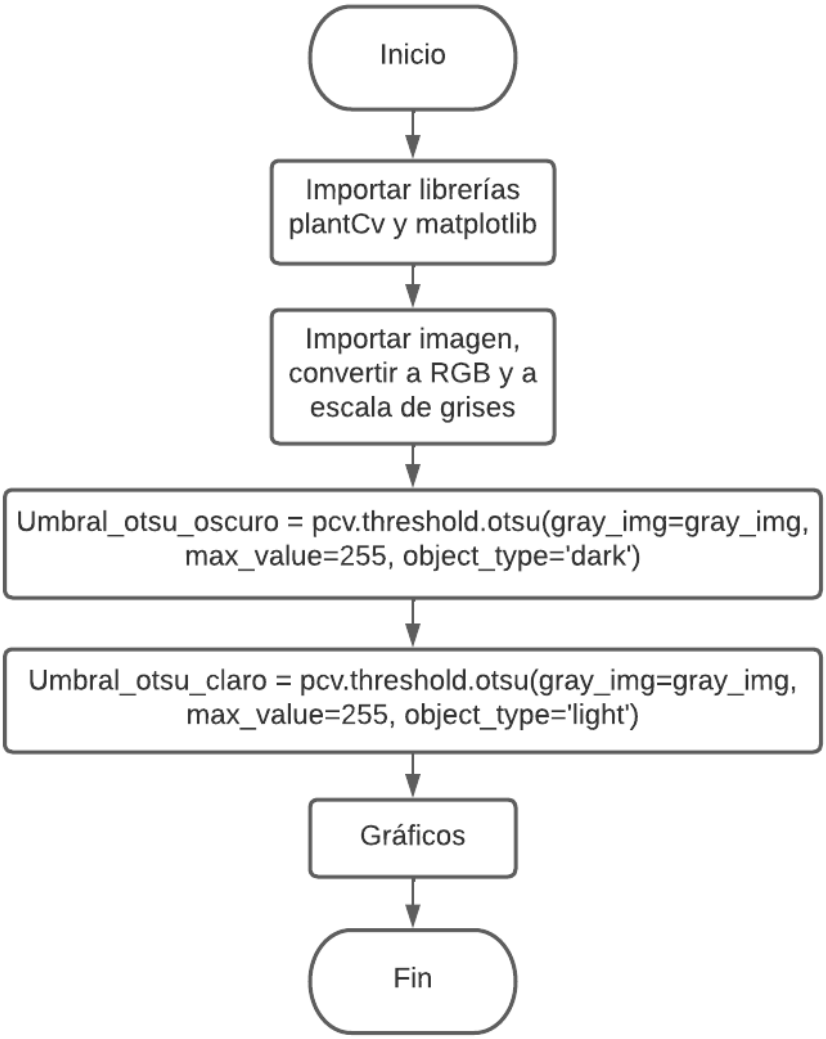

Figura 4.24: *Diagrama de flujo de la umbralización automática de Otsu. Fuente: Autor*

```python
import cv2
from plantcv import plantcv as pcv
from matplotlib import pyplot as plt

#IMAGEN
img1 = cv2.imread("original_image.jpg")
RGB_img = cv2.cvtColor(img1, cv2.COLOR_BGR2RGB)
#   channel = color subchannel ('L' = lightness, 'a' = green-magenta , 'b' = blue-yellow)
gray_img = pcv.rgb2gray_lab(rgb_img=RGB_img, channel='a')
#define subplots size
fig, ax = plt.subplots(1, 3, figsize=(10,7))
fig.tight_layout()

#INICIO ALGORITMO
pcv.params.debug = "none"
# Create binary image from a gray image based on threshold values.
# Targeting dark objects in the image.
threshold_dark = pcv.threshold.otsu(gray_img=gray_img, max_value=255, object_type='dark')

# Create binary image from a gray image based on threshold values.
# Targeting light objects in the image.
threshold_light = pcv.threshold.otsu(gray_img=gray_img, max_value=255, object_type='light')
#FIN ALGORITMO

#GRAFICOS
plt.subplot(2, 2, 1)
plt.imshow(RGB_img)
plt.title('Original RGB')
plt.axis('off')

plt.subplot(2, 2, 3)
plt.imshow(threshold_light, cmap='gray')
plt.title('Umbral claro')
plt.axis('off')

plt.subplot(2, 2, 4)
plt.imshow(threshold_dark, cmap='gray')
plt.title('Umbral oscuro')
plt.axis('off')

plt.subplot(2, 2, 2)
plt.imshow(gray_img, cmap='gray')
plt.title('Gris (magenta-verde)')
plt.axis('off')
plt.show()
```

Figura 4.25: *Código umbralización automática de Otsu. Fuente: Autor*

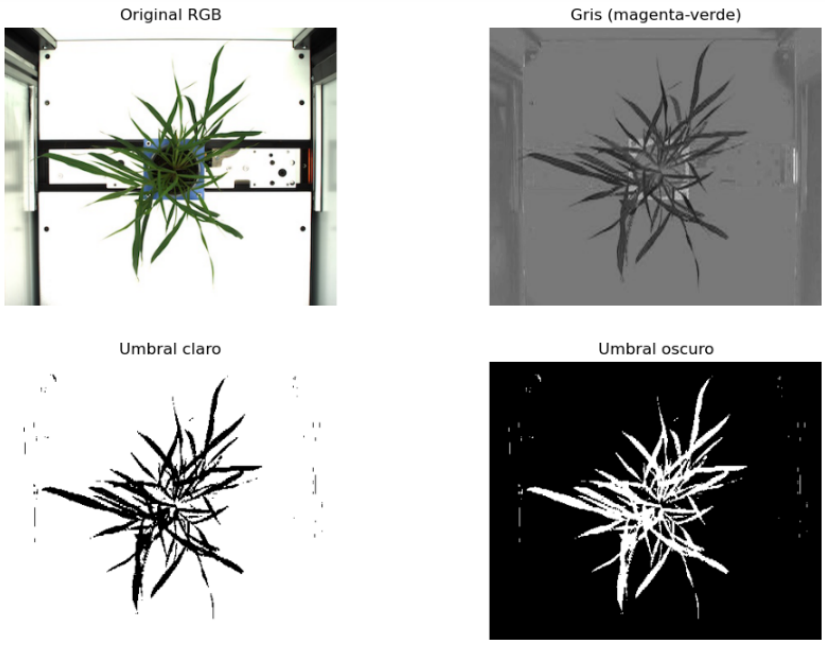

Figura 4.26: *Imagen Haciendo uso de umbralización automática de Otsu. Fuente: Autor*

En la figura 4.26 se muestra la ejecución del proceso de umbralización automática de Otsu sobre una planta y su resultado, esta permite crear una imagen binaria automáticamente a partir de una imagen en escala de grises. Lo que se quiere con este método de Otsu básicamente es calcular el valor del umbral de tal forma que la dispersión o división que se presenta dentro de cada segmento sea lo más pequeña posible, pero que a la par la dispersión entre segmentos diferentes sea lo más grande posible.

Umbral automático triangular

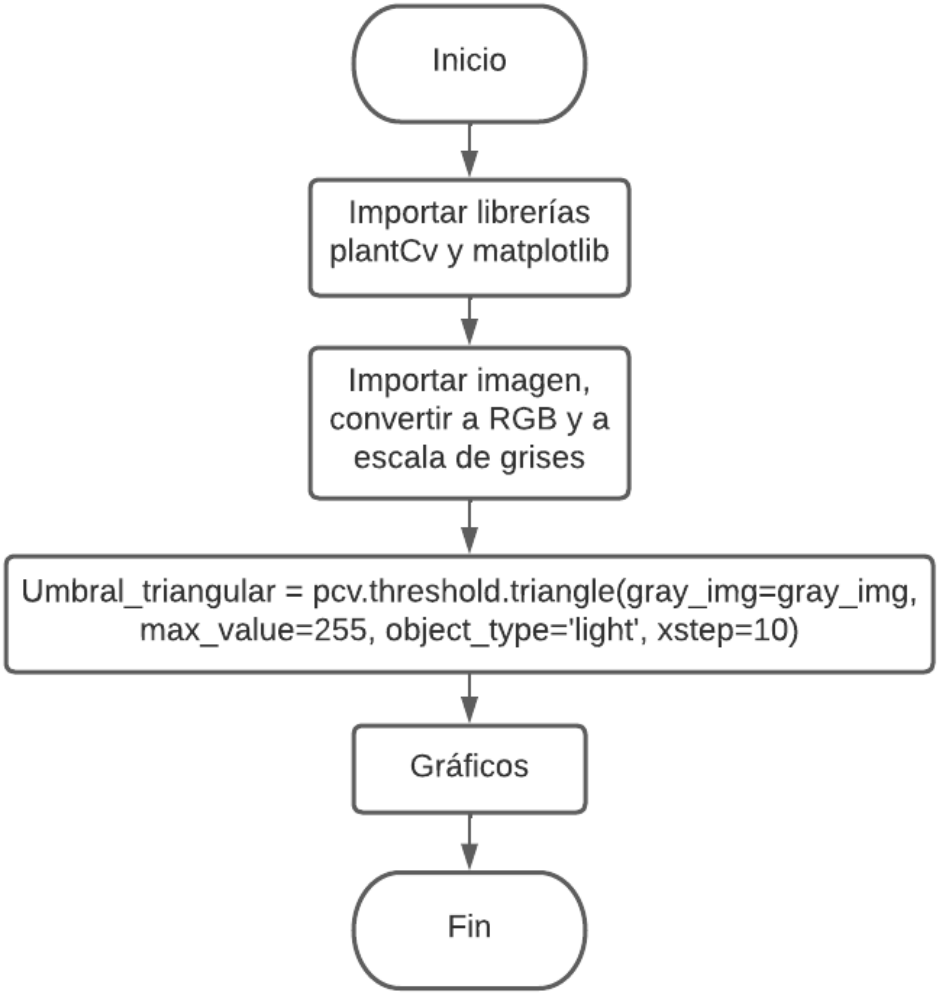

Figura 4.27: *Diagrama de flujo del umbral automático triangular. Fuente: Autor*

```python
import cv2
from plantcv import plantcv as pcv
from matplotlib import pyplot as plt

#IMAGEN
img1 = cv2.imread("original_image.jpg")
RGB_img = cv2.cvtColor(img1, cv2.COLOR_BGR2RGB)
#   channel = color subchannel ('l' = lightness, 'a' = green-magenta , 'b' = blue-yellow)
gray_img = pcv.rgb2gray_lab(rgb_img=RGB_img, channel='a')
#define subplots size
fig, ax = plt.subplots(1, 3, figsize=(10,7))
fig.tight_layout()

#INICIO ALGORITMO
pcv.params.debug = "none"

# Create binary image from a gray image based
thresholded = pcv.threshold.triangle(gray_img=gray_img, max_value=255, object_type='light', xstep=10)


#FIN ALGORITMO


#GRAFICOS
plt.subplot(1, 3, 1)
plt.imshow(RGB_img)
plt.title('Original RGB')
plt.axis('off')

plt.subplot(1, 3, 3)
plt.imshow(thresholded, cmap='gray')
plt.title('Umbral automático triangular')
plt.axis('off')


plt.subplot(1, 3, 2)
plt.imshow(gray_img, cmap='gray')
plt.title('Gris (magenta-verde)')
plt.axis('off')
plt.show()
```

Figura 4.28: *Código umbral automático triangular. Fuente: Autor*

Figura 4.29: *Imagen Haciendo uso de umbral automático triangular. Fuente: Autor*

En la figura 4.29 se muestra la ejecución del proceso de umbralización automática triangular sobre una planta y su resultado, esta permite crear una imagen binaria automáticamente a partir de una imagen en escala de grises.

4.3.3. Reducción de ruido

Después de que se realice un umbral y una sustracción de fondo, es posible que exista algo de ruido (puntos que no forman un objeto como tal), para que se pueda reducir esto se aplican los siguientes algoritmos:

Relleno

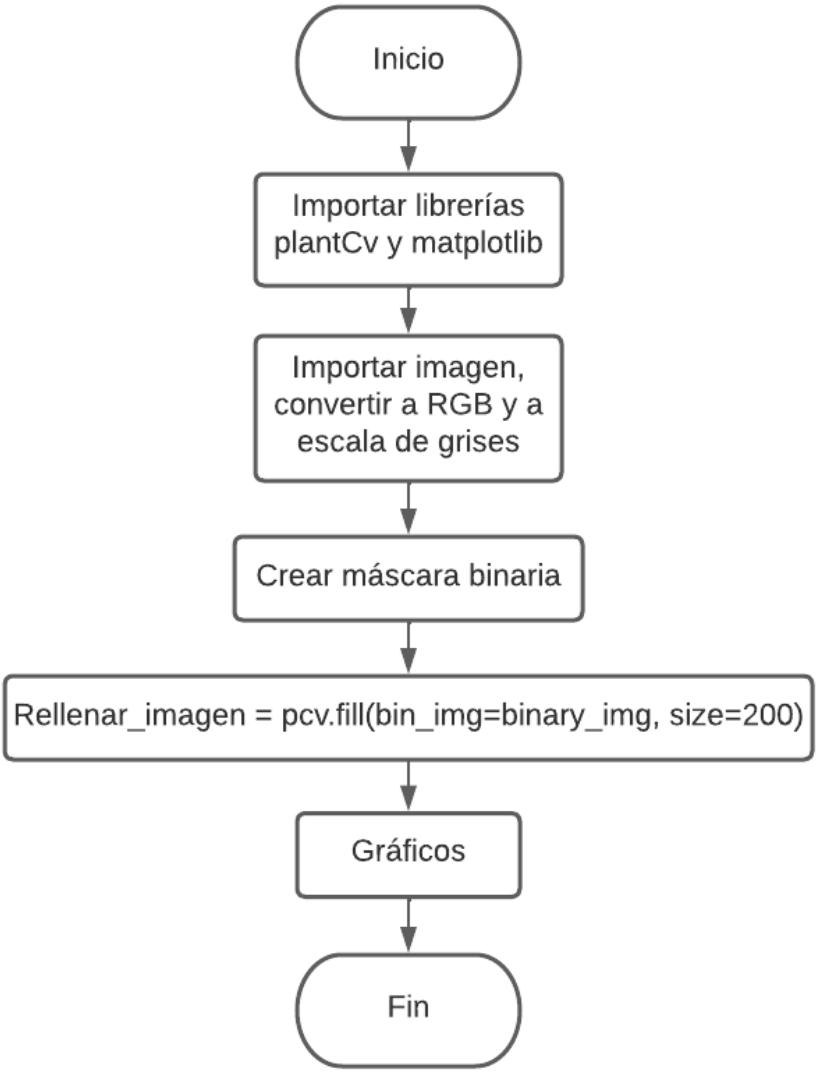

Figura 4.30: *Diagrama de flujo de Relleno. Fuente: Autor*

```python
import cv2
from plantcv import plantcv as pcv
from matplotlib import pyplot as plt

#IMAGEN
img1 = cv2.imread("original_image2.jpg")
RGB_img = cv2.cvtColor(img1, cv2.COLOR_BGR2RGB)
gray_img = cv2.cvtColor(img1, cv2.COLOR_BGR2GRAY)

#define subplots size
fig, ax = plt.subplots(1, 3, figsize=(10,7))
fig.tight_layout()

#INICIO ALGORITMO
pcv.params.debug = "none"

# Apply fill to a binary image that has had a median blur applied.
# Image mask is the same binary image with median blur.
mask = pcv.threshold.binary(gray_img=gray_img, threshold=150, max_value=255, object_type='dark')
binary_img = pcv.median_blur(gray_img=mask, ksize=5)
fill_image = pcv.fill(bin_img=binary_img, size=200)
#FIN ALGORITMO

#GRAFICOS
plt.subplot(1, 3, 1)
plt.imshow(mask, cmap='gray')
plt.title('Imagen Binaria')
plt.axis('off')

plt.subplot(1, 3, 2)
plt.imshow(binary_img, cmap='gray')
plt.title('Imagen binaria con desenfoque medio')
plt.axis('off')

plt.subplot(1, 3, 3)
plt.imshow(fill_image, cmap='gray')
plt.title('Imagen rellenada')
plt.axis('off')
plt.show()
```

Figura 4.31: *Código Relleno. Fuente: Autor*

Figura 4.32: *Imagen Haciendo uso de Relleno. Fuente: Autor*

En la figura 4.32 se muestra la ejecución del proceso de relleno sobre una planta y su resultado, este permite aplicar relleno a partir de una imagen binaria con un desenfoque medio o mascara, gracias a la función size se rellenan píxeles de algunas partes de la imagen que no se completaron, es decir que los píxeles mas pequeños de las partes de la imagen que no se completaron son los que se rellenan completándolos del color correspondiente (para este caso en es cala de grises).

Desenfoque mediano

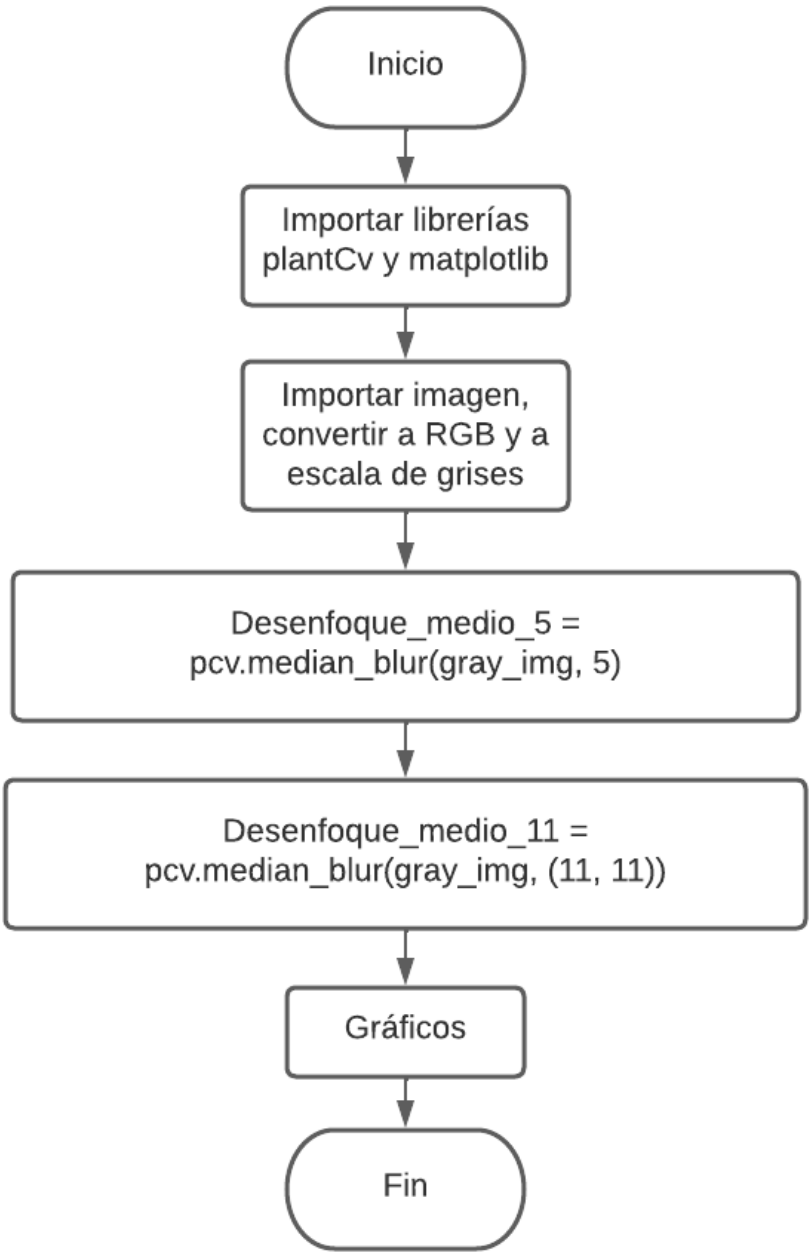

Figura 4.33: *Diagrama de flujo del desenfoque mediano. Fuente: Autor*

```python
import cv2
from plantcv import plantcv as pcv
from matplotlib import pyplot as plt

#IMAGEN
img1 = cv2.imread("original_image2.jpg")
RGB_img = cv2.cvtColor(img1, cv2.COLOR_BGR2RGB)
gray_img = pcv.rgb2gray(rgb_img=RGB_img)

#define subplots size
fig, ax = plt.subplots(1, 3, figsize=(10,7))
fig.tight_layout()

#INICIO ALGORITMO
pcv.params.debug = "none"
# Apply median blur to a binary image that has been previously thresholded.
blur_5 = pcv.median_blur(gray_img, 5)
# Apply median blur to a binary image that has been previously thresholded.
blur_11 = pcv.median_blur(gray_img, (11, 11))
#FIN ALGORITMO

#GRAFICOS
plt.subplot(1, 3, 1)
plt.imshow(gray_img, cmap='gray')
plt.title('Original Gris')
plt.axis('off')


plt.subplot(1, 3, 2)
plt.imshow(blur_5,cmap='gray')
plt.title('Desenfoque medio (ksize = 5)')
plt.axis('off')

plt.subplot(1, 3, 3)
plt.imshow(blur_11, cmap='gray')
plt.title('Desenfoque medio (ksize = (11,11))')
plt.axis('off')
plt.show()
```

Figura 4.34: *Código Desenfoque mediano. Fuente: Autor*

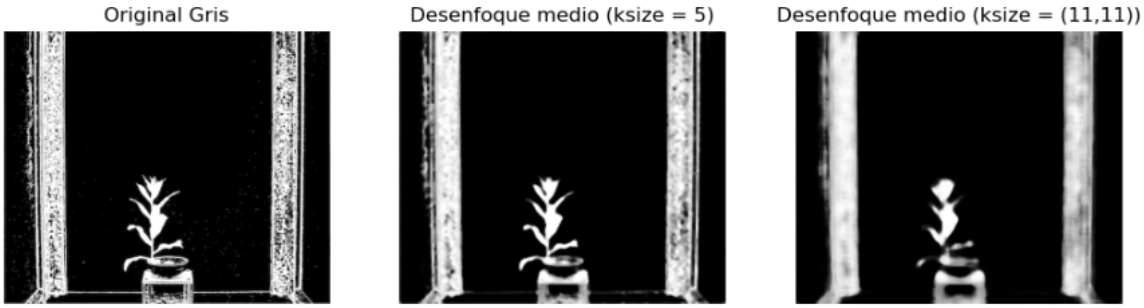

Figura 4.35: *Imagen Haciendo uso de Desenfoque mediano. Fuente: Autor*

En la figura 4.35 se muestra la ejecución del proceso de desenfoque mediano sobre una planta y su resultado, este algoritmo permite aplicar un filtro mediano a partir de una imagen en escala de grises, lo que realiza este filtro medio es lo que hace referencia a una técnica de filtrado digital, que básicamente es eliminar el ruido de una imagen, en este caso eliminar el ruido de la imagen de la planta, este filtrado medio es utilizado mas que todo en el procesamiento de imágenes digitales ya que en diferentes condiciones, conserva los bordes y elimina el ruido que se presenta en la imagen, este filtrado mediano es uno de los mejores algoritmos para para poder eliminar el llamado ruido de sal y pimienta, este ruido hace referencia a la distorsión de algunos píxeles que se presentan en forma de ruido blanco y ruido negro de ahí el nombre de sal y pimienta

Desenfoque gaussiano

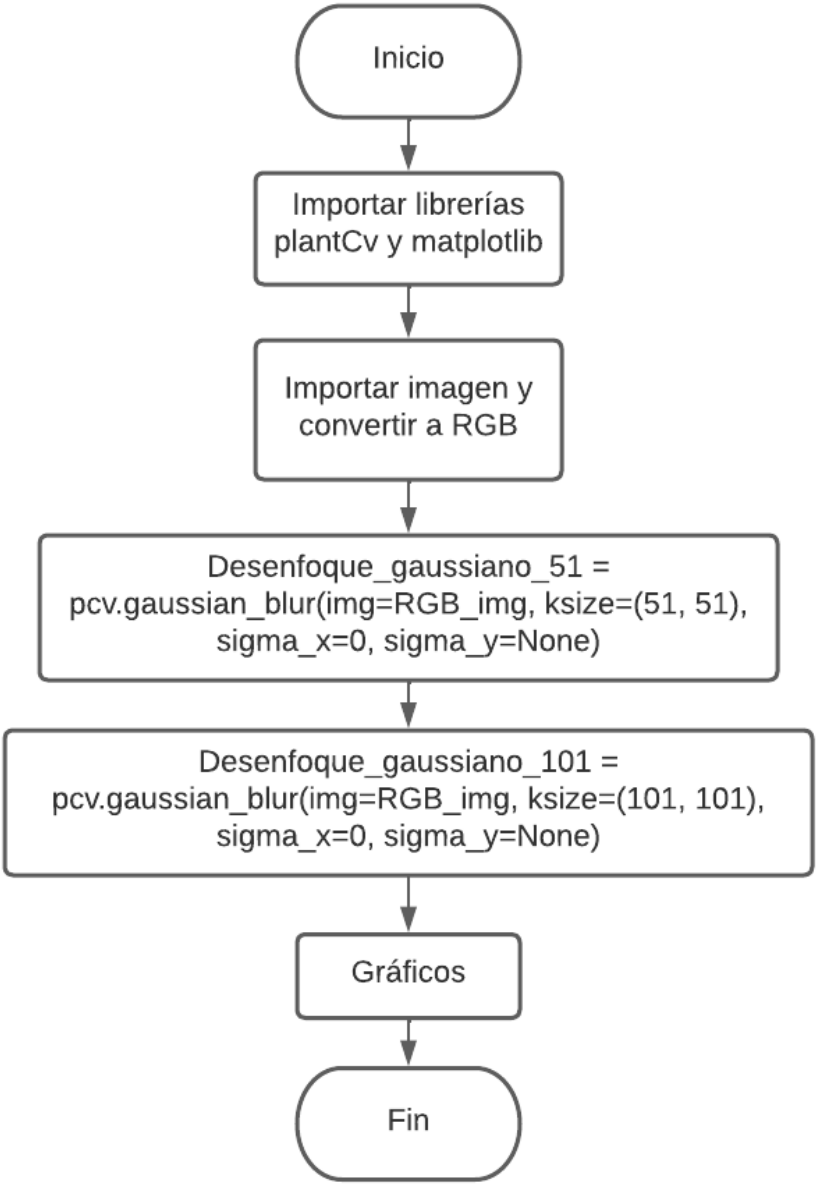

Figura 4.36: *Diagrama de flujo desenfoque gaussiano. Fuente: Autor*

```python
import cv2
from plantcv import plantcv as pcv
from matplotlib import pyplot as plt

#IMAGEN
img1 = cv2.imread("original_image2.jpg")
RGB_img = cv2.cvtColor(img1, cv2.COLOR_BGR2RGB)

#define subplots size
fig, ax = plt.subplots(1, 3, figsize=(10,7))
fig.tight_layout()

#INICIO ALGORITMO
pcv.params.debug = "none"
# Apply gaussian blur to a binary image that has been previously thresholded.
gaussian_img = pcv.gaussian_blur(img=RGB_img, ksize=(51, 51), sigma_x=0, sigma_y=None)

# Apply gaussian blur to a binary image that has been previously thresholded.
gaussian_img1 = pcv.gaussian_blur(img=RGB_img, ksize=(101, 101), sigma_x=0, sigma_y=None)

#FIN ALGORITMO

#GRAFICOS
plt.subplot(1, 3, 1)
plt.imshow(RGB_img)
plt.title('Original RGB')
plt.axis('off')


plt.subplot(1, 3, 2)
plt.imshow(gaussian_img)
plt.title('Desenfoque gaussiano(ksize=(51,51)')
plt.axis('off')

plt.subplot(1, 3, 3)
plt.imshow(gaussian_img1)
plt.title(' Desenfoque gaussiano(ksize=(101,101))')
plt.axis('off')
plt.show()
```

Figura 4.37: *Código desenfoque gaussiano. Fuente: Autor*

Figura 4.38: *Imagen Haciendo uso de desenfoque gaussiano. Fuente: Autor*

En la figura 4.38 se muestra la ejecución del proceso de desenfoque gaussiano sobre una planta y su resultado, este algoritmo permite integrar un filtro de desenfoque gaussiano a una imagen, y este aplica el valor medio al píxel central de un núcleo, lo que hace este desenfoque es mezclar suave o ligeramente los colores de los píxeles que sean vecinos el uno al otro en la imagen (en este caso en escala de grises), lo que genera que la imagen pierda algunos detalles en este caso la nitidez y la claridad, por otra parte los bordes que están presentes en la imagen se ven afectados ya que se genera un efecto parecido a cuando un fotógrafo toma un foto desenfocada.

Región de interés

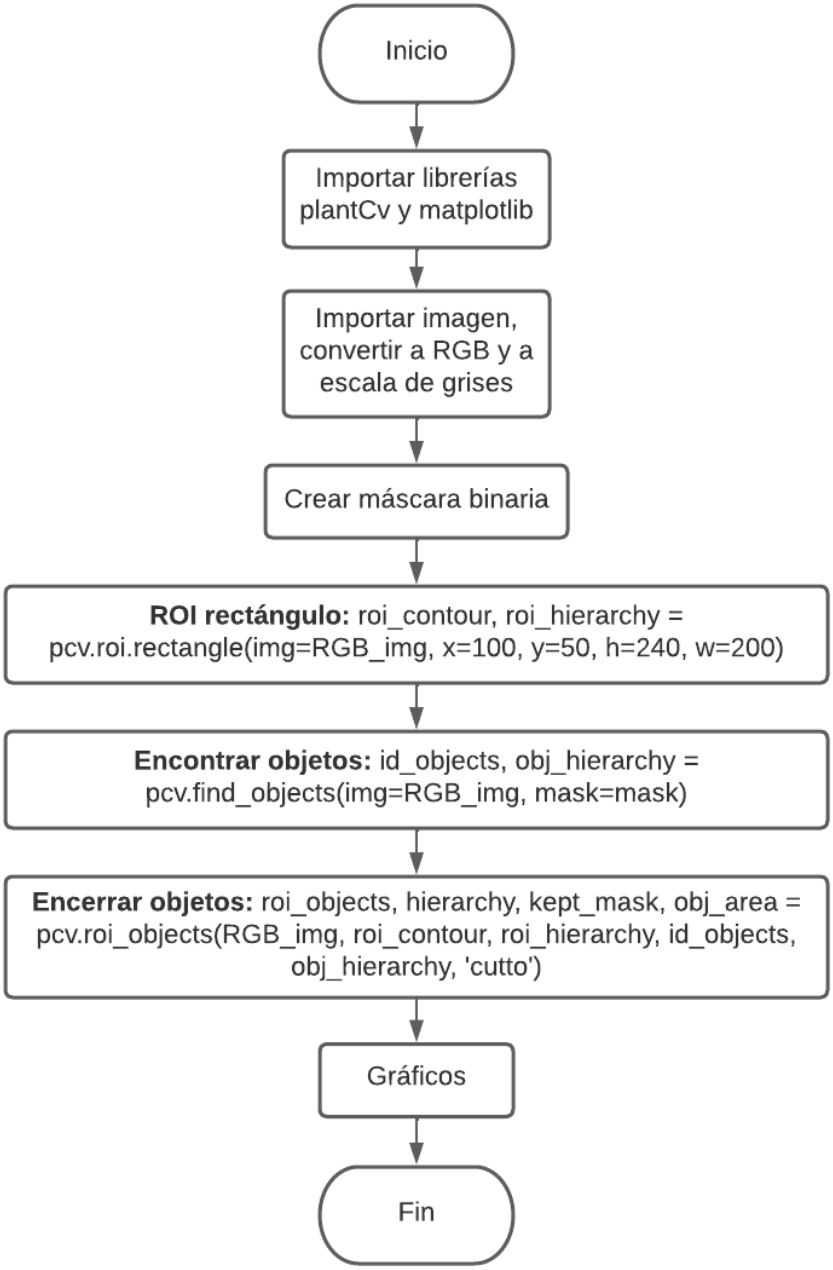

Figura 4.39: *Diagrama de flujo región de interés. Fuente: Autor*

```python
import cv2
from plantcv import plantcv as pcv
from matplotlib import pyplot as plt

#IMAGEN
img1 = cv2.imread("original_image.jpg")
RGB_img = cv2.cvtColor(img1, cv2.COLOR_BGR2RGB)
gray_img = cv2.cvtColor(img1, cv2.COLOR_BGR2GRAY)

#define subplots size
#fig, ax = plt.subplots(1, 3, figsize=(10,7))
#fig.tight_layout()


#INICIO ALGORITMO
pcv.params.debug = "plot"
mask = pcv.threshold.binary(gray_img=gray_img, threshold=150, max_value=255, object_type='dark')

roi_contour, roi_hierarchy = pcv.roi.rectangle(img=RGB_img, x=100, y=50, h=240, w=200)

# Identify objects (plant material) in an image, all objects regardless of
# hierarchy are filled (e.g. holes between leaves).
id_objects, obj_hierarchy = pcv.find_objects(img=RGB_img, mask=mask)

# ROI objects allows the user to define if objects partially inside ROI are included or if objects are cut to ROI.
roi_objects, hierarchy, kept_mask, obj_area = pcv.roi_objects(RGB_img, roi_contour, roi_hierarchy,id_objects, obj_hierarchy, 'par
# Define region of interest in an image, there is a futher function 'ROI Objects' that allows the user to define if you want to
roi_objects, hierarchy, kept_mask, obj_area = pcv.roi_objects(RGB_img, roi_contour, roi_hierarchy, id_objects, obj_hierarchy, 'cu

#FIN ALGORITMO
```

Figura 4.40: *Código región de interés. Fuente: Autor*

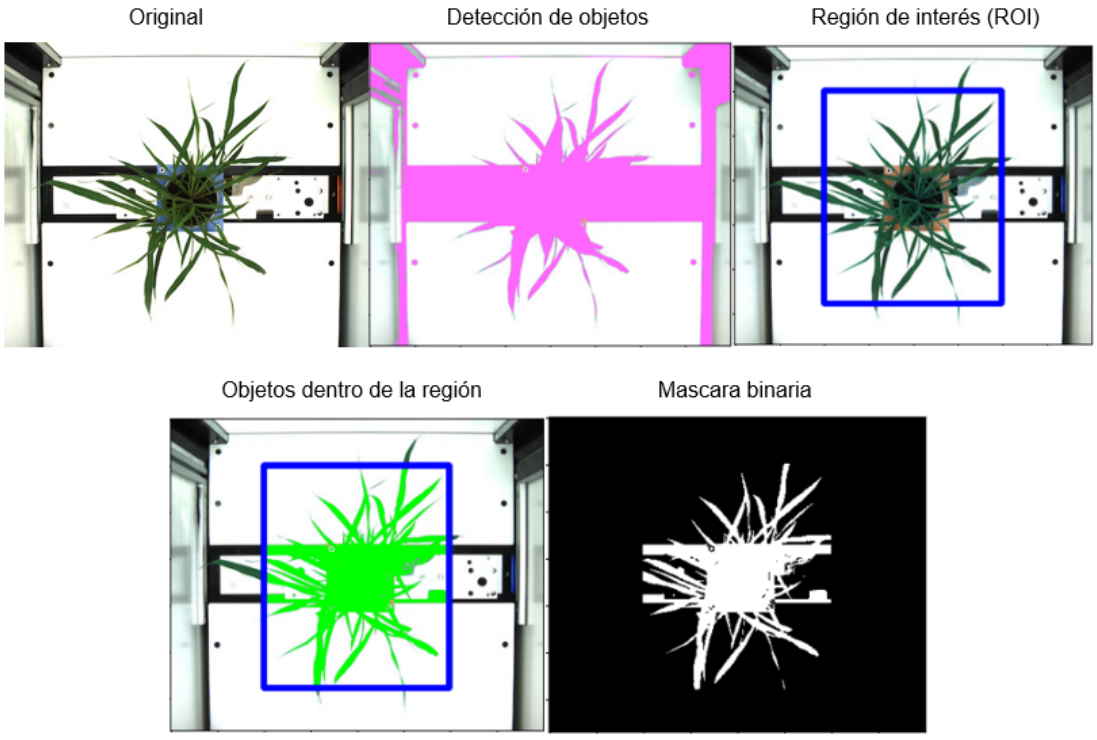

Figura 4.41: *Imagen Haciendo uso de región de interés. Fuente: Autor*

En la figura 4.41 se muestra la ejecución del proceso para encontrar objetos dentro de una región de interés y su resultado, este método o algoritmo permite aislar el objeto que se desea estudiar del resto de la imagen. Primero se encuentran todos los objetos dentro de la imagen, luego se define la región de interés y posteriormente se comprueba si hay objetos dentro de la región.

4.3.4. Otros tipos de región de interés (ROI)

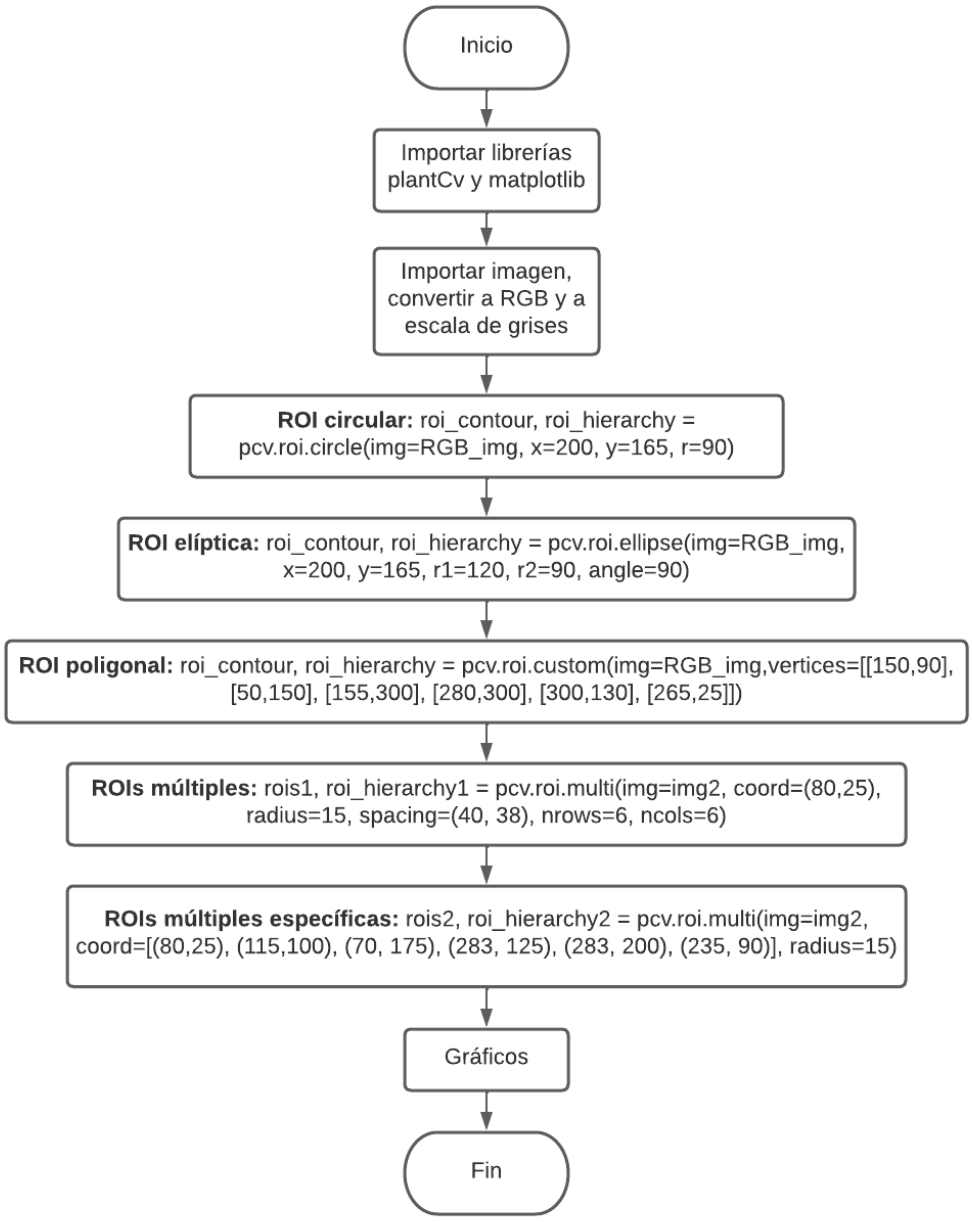

Figura 4.42: *Diagrama de flujo de otros tipos de región de interés (ROI). Fuente: Autor*

```python
import cv2
from plantcv import plantcv as pcv
from matplotlib import pyplot as plt

#IMAGEN
img1 = cv2.imread("original_image.jpg")
img2 = cv2.imread("Banco1.png")
RGB_img = cv2.cvtColor(img1, cv2.COLOR_BGR2RGB)
gray_img = cv2.cvtColor(img1, cv2.COLOR_BGR2GRAY)

#define subplots size
#fig, ax = plt.subplots(1, 3, figsize=(10,7))
#fig.tight_layout()

#INICIO ALGORITMO
pcv.params.debug = "plot"

# ROI CIRCULAR
roi_contour, roi_hierarchy = pcv.roi.circle(img=RGB_img, x=200, y=165, r=90)

# ROI ELIPTICA
roi_contour, roi_hierarchy = pcv.roi.ellipse(img=RGB_img, x=200, y=165, r1=120, r2=90, angle=90)

# ROI POLIGONAL
roi_contour, roi_hierarchy = pcv.roi.custom(img=RGB_img,vertices=[[150,90], [50,150], [155,300], [280,300], [300,130], [265,25]]

# ROIs MULTIPLES
rois1, roi_hierarchy1 = pcv.roi.multi(img=img2, coord=(80,25), radius=15,
                                      spacing=(40, 38), nrows=6, ncols=6)

# Specify a list of coordinates of desired ROIs
rois2, roi_hierarchy2 = pcv.roi.multi(img=img2, coord=[(80,25), (115,100), (70, 175), (283, 125), (283, 200), (235, 90)], radius=
```

Figura 4.43: *Código de otros tipos de región de interés (ROI). Fuente: Autor*

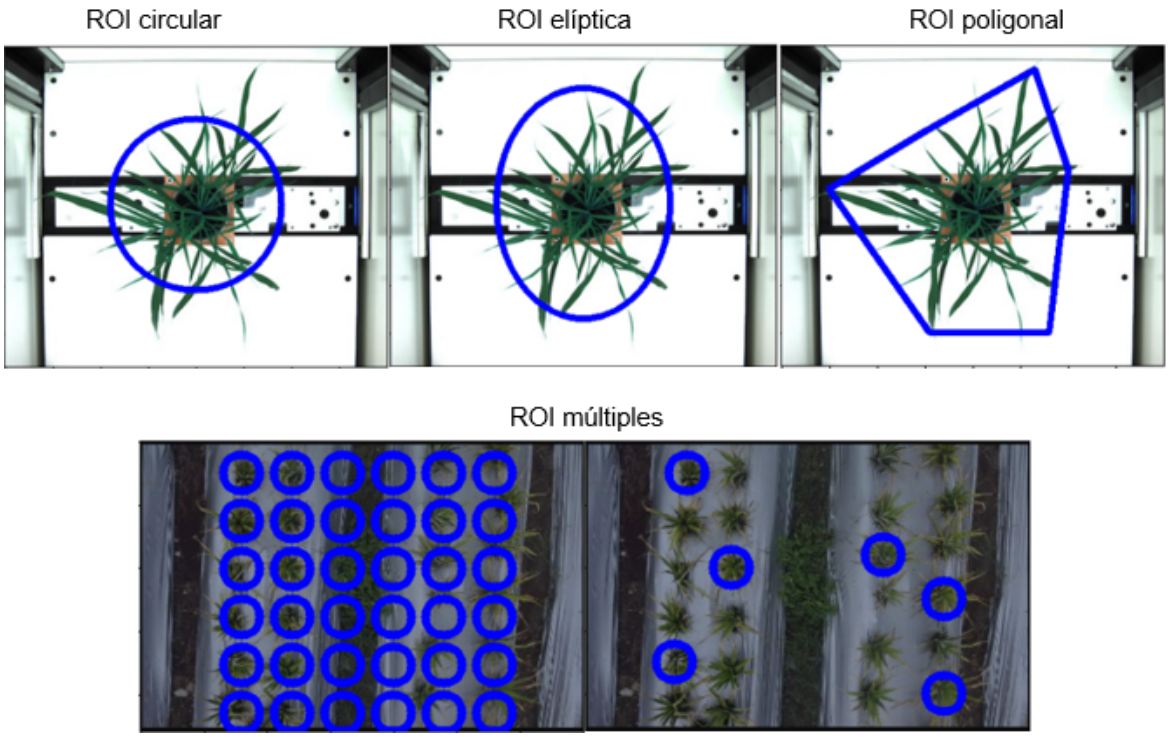

Figura 4.44: *Imagen haciendo uso de otros tipos de región de interés (ROI). Fuente: Autor*

Dentro de este se encontraron algoritmos que pueden ayudar a la detección y segmentación de objetos, los cuales son:

Como primero se tiene la **Región de interés circular**, este algoritmo permite crear una región circular sobre un objeto en una imagen.
Como segundo se tiene la **Región de interés elíptica**, la cual permite crear una región elíptica sobre un objeto en una imagen.
Como tercero se tiene la **Región de interés poligonal**, este algoritmo permite crear una región poligonal sobre un objeto en una imagen.
Y como cuarto se tiene la **Regiones de interés múltiples**, el cual permite crear múltiples regiones de interés sobre una imagen.

Dentro del código presente en la figura 4.43 se realizaron los 4 algoritmos, para posteriormente en la figura 4.44 visualizar las funciones para crear regiones de interés explicadas anteriormente, este método o algoritmo permite aislar el objeto que se desea estudiar del resto de la imagen. Primero se encuentran todos los objetos dentro de la imagen, luego se define la región de interés (para este caso alguna de las 4 mencionadas anteriormente) y posteriormente se comprueba si hay objetos dentro de la región.

4.3.5. Análisis de objetos

Analizar color

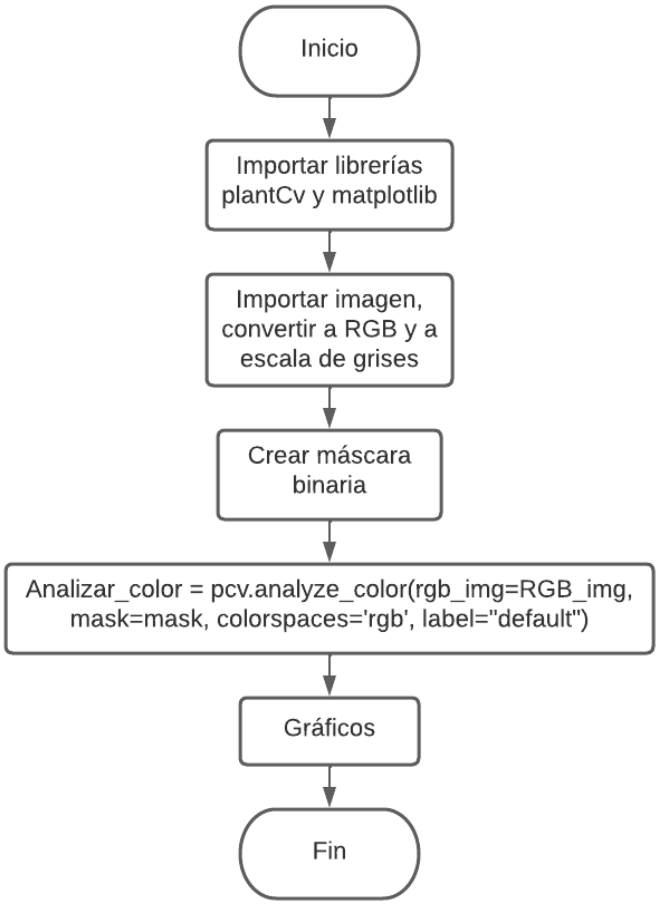

Figura 4.45: *Diagrama de flujo para el análisis de color. Fuente: Autor*

```python
import cv2
from plantcv import plantcv as pcv
from matplotlib import pyplot as plt

#IMAGEN
img1 = cv2.imread("original_image.jpg")
RGB_img = cv2.cvtColor(img1, cv2.COLOR_BGR2RGB)
gray_img = cv2.cvtColor(img1, cv2.COLOR_BGR2GRAY)


#INICIO ALGORITMO
pcv.params.debug = "plot"
mask = pcv.threshold.binary(gray_img=gray_img, threshold=180, max_value=255, object_type='dark')

# Analyze Color
analysis_image = pcv.analyze_color(rgb_img=RGB_img, mask=mask, colorspaces='rgb', label="default")

# Access data stored out from analyze_color
hue_circular_mean = pcv.outputs.observations['default']['hue_circular_mean']['value']
```

Figura 4.46: *Código para el análisis de color. Fuente: Autor*

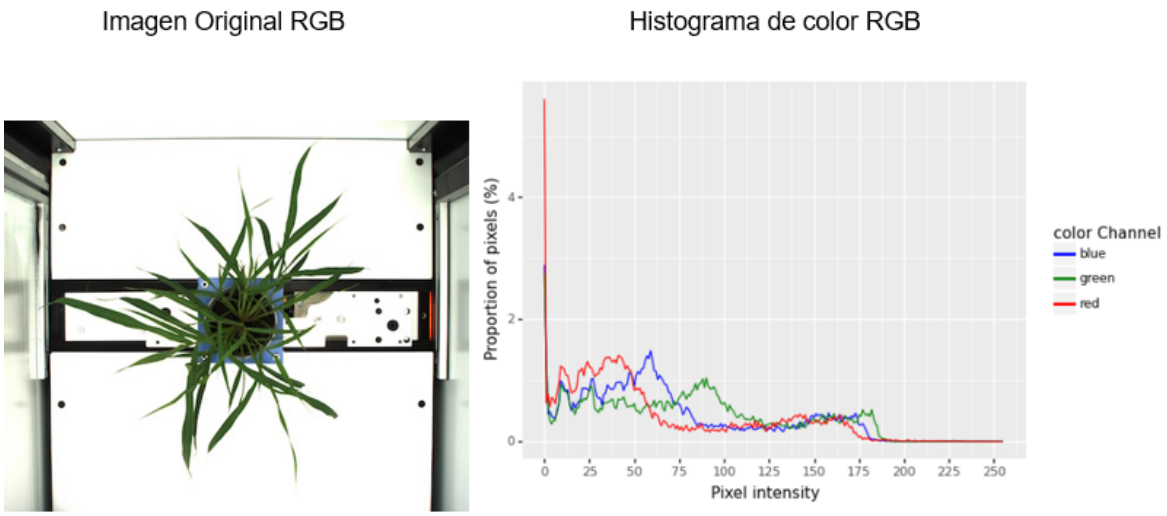

Figura 4.47: *Imagen haciendo uso de análisis de color. Fuente: Autor*

En la figugra 4.47 se muestra la ejecución del proceso para realizar un análisis de color a través de un histograma y su resultado, esta función permite extraer características para espacios de color (rgb, hsv y lab), en este algoritmo se genera un histograma de color que básicamente va a contabilizar el número de veces que el valor de un píxel ocurre en la imagen, y a su vez la cantidad de veces que un color se presenta en la imagen, es decir, que uniendo estas dos vamos a tener un histograma de la cantidad de veces que se repite un color por píxel en la imagen.

Detección canny Edge

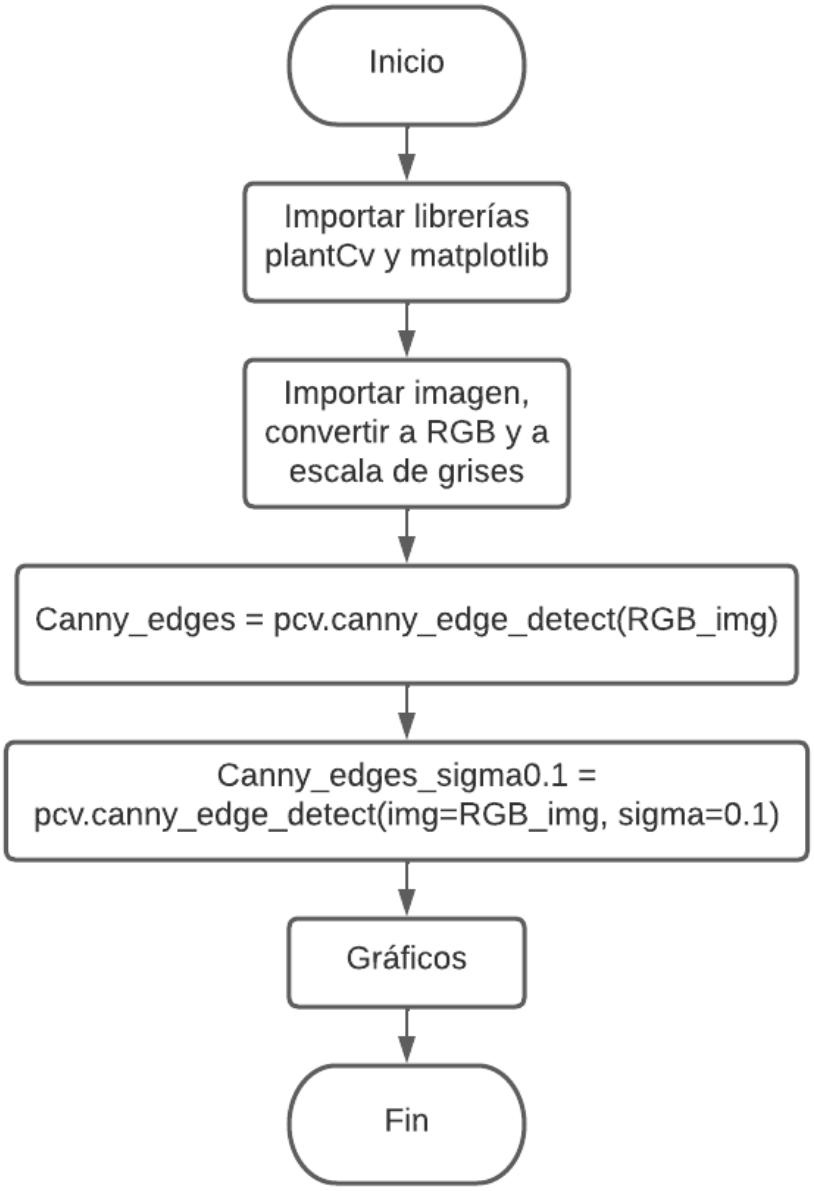

Figura 4.48: *Diagrama de flujo para detección canny Edge. Fuente: Autor*

```python
import cv2
from plantcv import plantcv as pcv
from matplotlib import pyplot as plt

#IMAGEN
img1 = cv2.imread("original_image.jpg")
RGB_img = cv2.cvtColor(img1, cv2.COLOR_BGR2RGB)
gray_img = cv2.cvtColor(img1, cv2.COLOR_BGR2GRAY)

#define subplots size
fig, ax = plt.subplots(1, 3, figsize=(10,7))
fig.tight_layout()

#INICIO ALGORITMO
pcv.params.debug = "none"
# Create binary image of edges.
edges = pcv.canny_edge_detect(RGB_img)

# Lower sigma value to pick up more edges
edges2 = pcv.canny_edge_detect(img=RGB_img, sigma=0.1)

#GRAFICOS
plt.subplot(1, 3, 1)
plt.imshow(RGB_img)
plt.title('Original RGB')
plt.axis('off')


plt.subplot(1, 3, 2)
plt.imshow(edges, cmap='gray')
plt.title('Canny edge')
plt.axis('off')

plt.subplot(1, 3, 3)
plt.imshow(edges2, cmap='gray')
plt.title('Canny edge, sigma 0.1')
plt.axis('off')
plt.show()
```

Figura 4.49: *Código para detección canny Edge. Fuente: Autor*

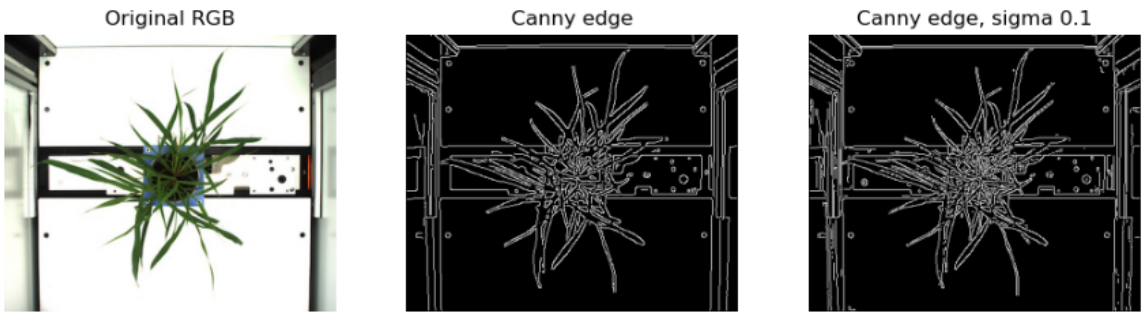

Figura 4.50: *Imagen haciendo uso de la detección canny Edge. Fuente: Autor*

En la figura 4.50 se muestra la ejecución del proceso para realizar detección canny Edge a través de un histograma de color y su resultado, este proceso permite crea una imagen binaria de los bordes a partir de un filtro canny, esta técnica se utilizada mas que todo para la detección de bordes en las imágenes y básicamente lo que permite esta detección de bordes es la detección de objetos que se encuentran presentes en la imágenes, es decir, lo que hace este algoritmo es simplificar la imagen hasta el punto de que solo se dibuje el borde de los objetos que se encuentran en la imagen que en este caso el objeto es la planta.

4.3.6. Visualización

Esqueleto

Figura 4.51: *Diagrama de flujo para esqueleto. Fuente: Autor*

```python
import cv2
from plantcv import plantcv as pcv
from matplotlib import pyplot as plt

#IMAGEN
img1 = cv2.imread("mask_image.jpg")
gray_img = cv2.cvtColor(img1, cv2.COLOR_BGR2GRAY)

#define subplots size
#fig, ax = plt.subplots(1, 3, figsize=(10,7))
#fig.tight_layout()

#INICIO ALGORITMO
pcv.params.debug = "plot"
mask = pcv.threshold.binary(gray_img=gray_img, threshold=180, max_value=255, object_type='light')

skeleton = pcv.morphology.skeletonize(mask=mask)
```

Figura 4.52: *Código para esqueleto. Fuente: Autor*

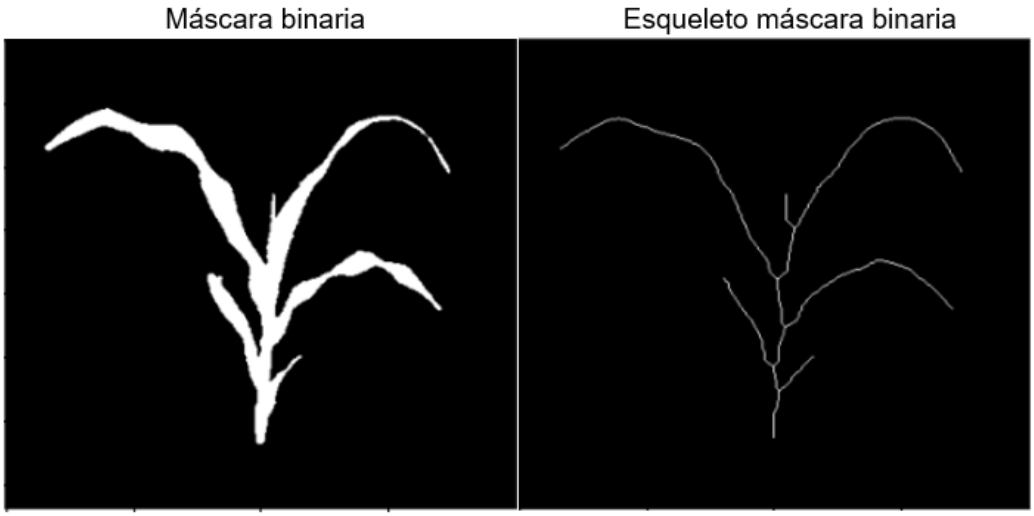

Figura 4.53: *Imagen haciendo uso de esqueleto. Fuente: Autor*

En la figura 4.53 se muestra la ejecución de la función para obtener el esqueleto de una planta y su resultado, esta función es un proceso que genera un esqueleto obtenido de una máscara respectiva de la imagen, es decir, que se genera un esqueleto basado en la forma original del objeto que esta presente en la imagen (la planta).

Prune

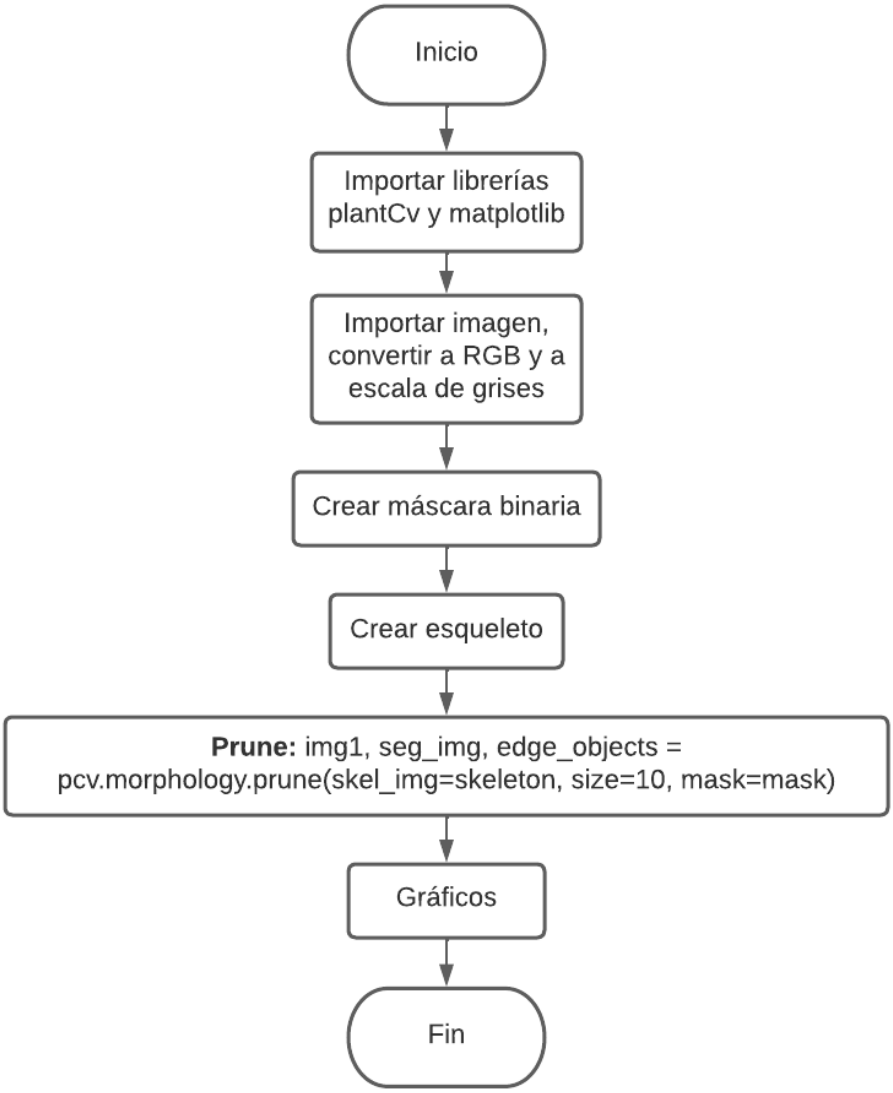

Figura 4.54: *Diagrama de flujo para Prune. Fuente: Autor*

```python
import cv2
from plantcv import plantcv as pcv
from matplotlib import pyplot as plt

#IMAGEN
img1 = cv2.imread("mask_image.jpg")
gray_img = cv2.cvtColor(img1, cv2.COLOR_BGR2GRAY)

#define subplots size
#fig, ax = plt.subplots(1, 3, figsize=(10,7))
#fig.tight_layout()

#INICIO ALGORITMO
pcv.params.debug = "plot"
mask = pcv.threshold.binary(gray_img=gray_img, threshold=180, max_value=255, object_type='light')
#Esqueleto
skeleton = pcv.morphology.skeletonize(mask=mask)
#Prune
img1, seg_img, edge_objects = pcv.morphology.prune(skel_img=skeleton, size=10, mask=mask)
#FIN ALGORITMO
```

Figura 4.55: *Código para Prune. Fuente: Autor*

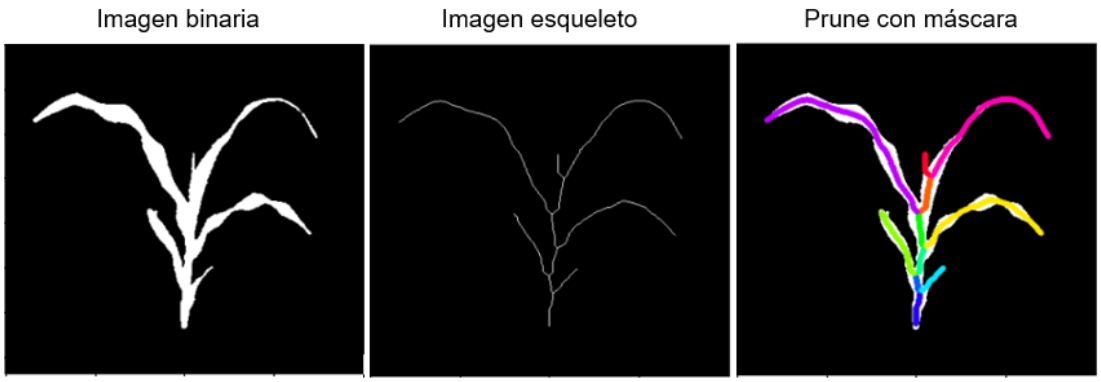

Figura 4.56: *Imagen haciendo uso de Prune. Fuente: Autor*

En la figura 4.56 se muestra la ejecución de la función para obtener el prune y su resultado, este proceso permite crear una imagen esqueleto y segmentarla en sus diferentes partes, en este caso las partes serian diferenciadas entre lo que es el tallo de la planta y sus diferentes hojas.

Longitud de segmentos

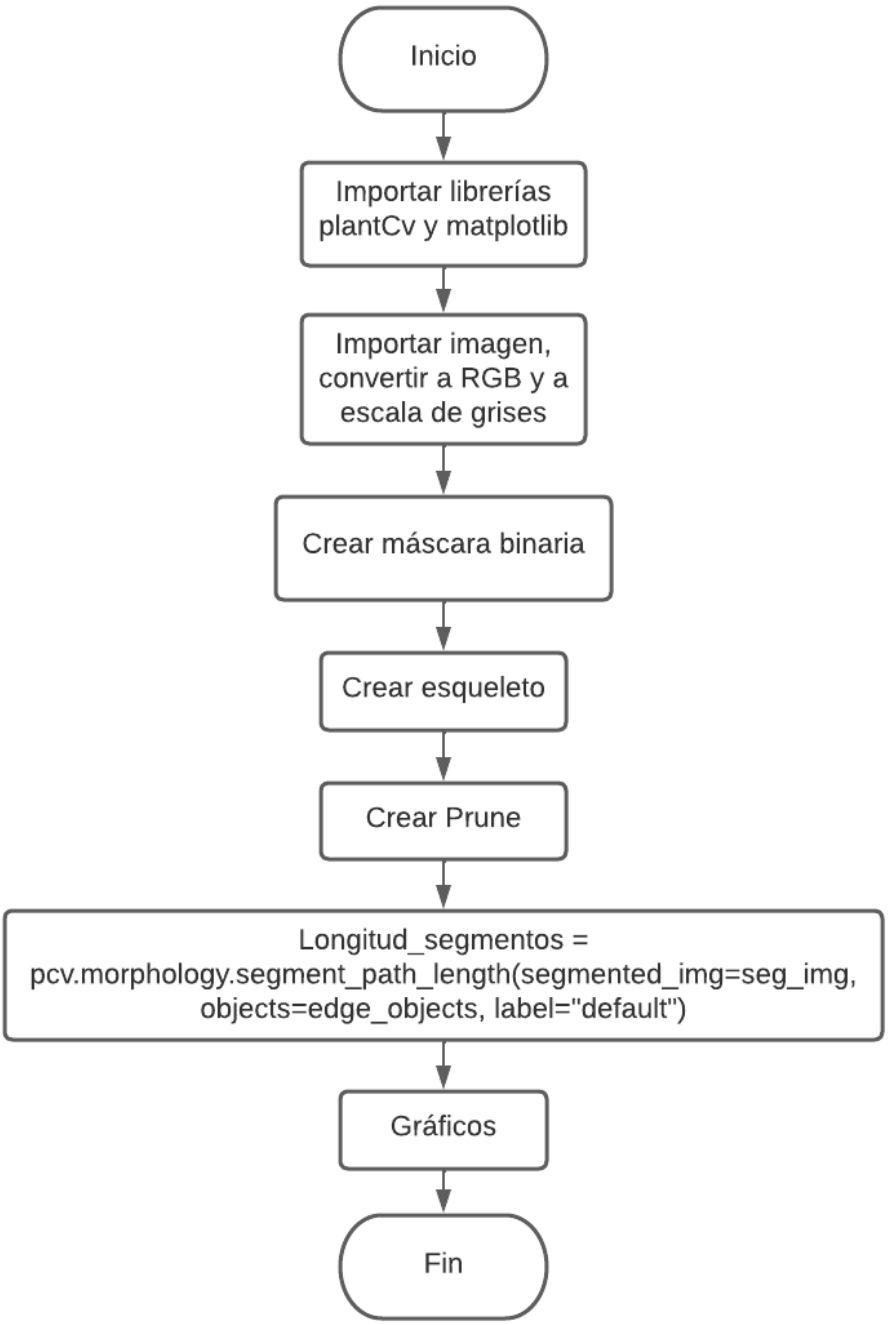

Figura 4.57: *Diagrama de flujo para la función longitud de segmentos. Fuente: Autor*

```python
import cv2
from plantcv import plantcv as pcv
from matplotlib import pyplot as plt

#IMAGEN
img1 = cv2.imread("mask_image.jpg")
gray_img = cv2.cvtColor(img1, cv2.COLOR_BGR2GRAY)

#define subplots size
#fig, ax = plt.subplots(1, 3, figsize=(10,7))
#fig.tight_layout()

#INICIO ALGORITMO
pcv.params.debug = "plot"
mask = pcv.threshold.binary(gray_img=gray_img, threshold=180, max_value=255, object_type='light')
#Esqueleto
skeleton = pcv.morphology.skeletonize(mask=mask)
#Prune
img1, seg_img, edge_objects = pcv.morphology.prune(skel_img=skeleton, size=10, mask=mask)

# longitudes de los segmentos
labeled_img = pcv.morphology.segment_path_length(segmented_img=seg_img, objects=edge_objects, label="default")
# Access data stored out from segment_path_length
path_lengths = pcv.outputs.observations['default']['segment_path_length']['value']

#FIN ALGORITMO
```

Figura 4.58: *Código para la función longitud de segmentos. Fuente: Autor*

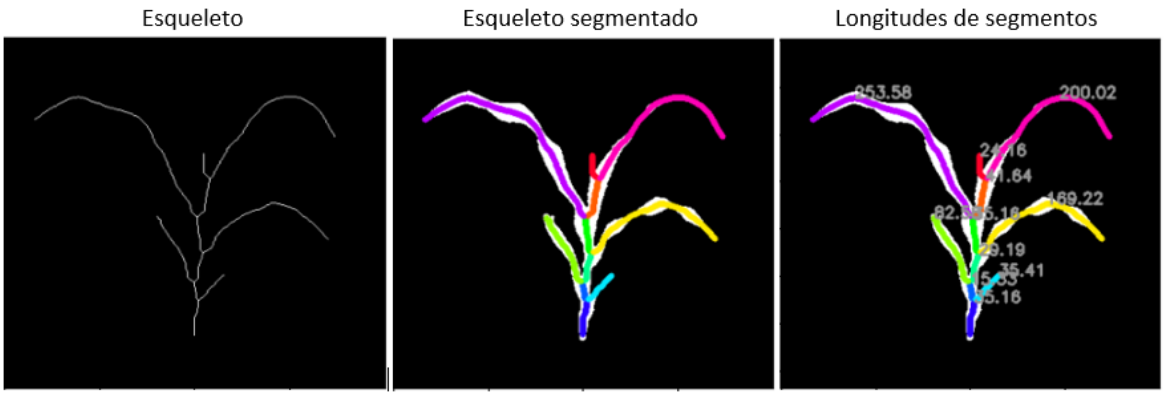

Figura 4.59: *Imagen haciendo uso de la función longitud de segmentos. Fuente: Autor*

En la figura 4.59 se muestra la ejecución de la función para obtener la longitud de segmentos de un esqueleto y su resultado, este proceso permite establecer las longitudes geodésicas de cada uno de los segmentos encontrados con la función Prune, es decir, la longitud que tiene cada una de las partes seleccionadas en la función prune que en este caso como anteriormente se menciono seria la longitud de lo que viene a ser el tallo y las diferentes hojas que conforman esta planta.

Ordenar segmentos

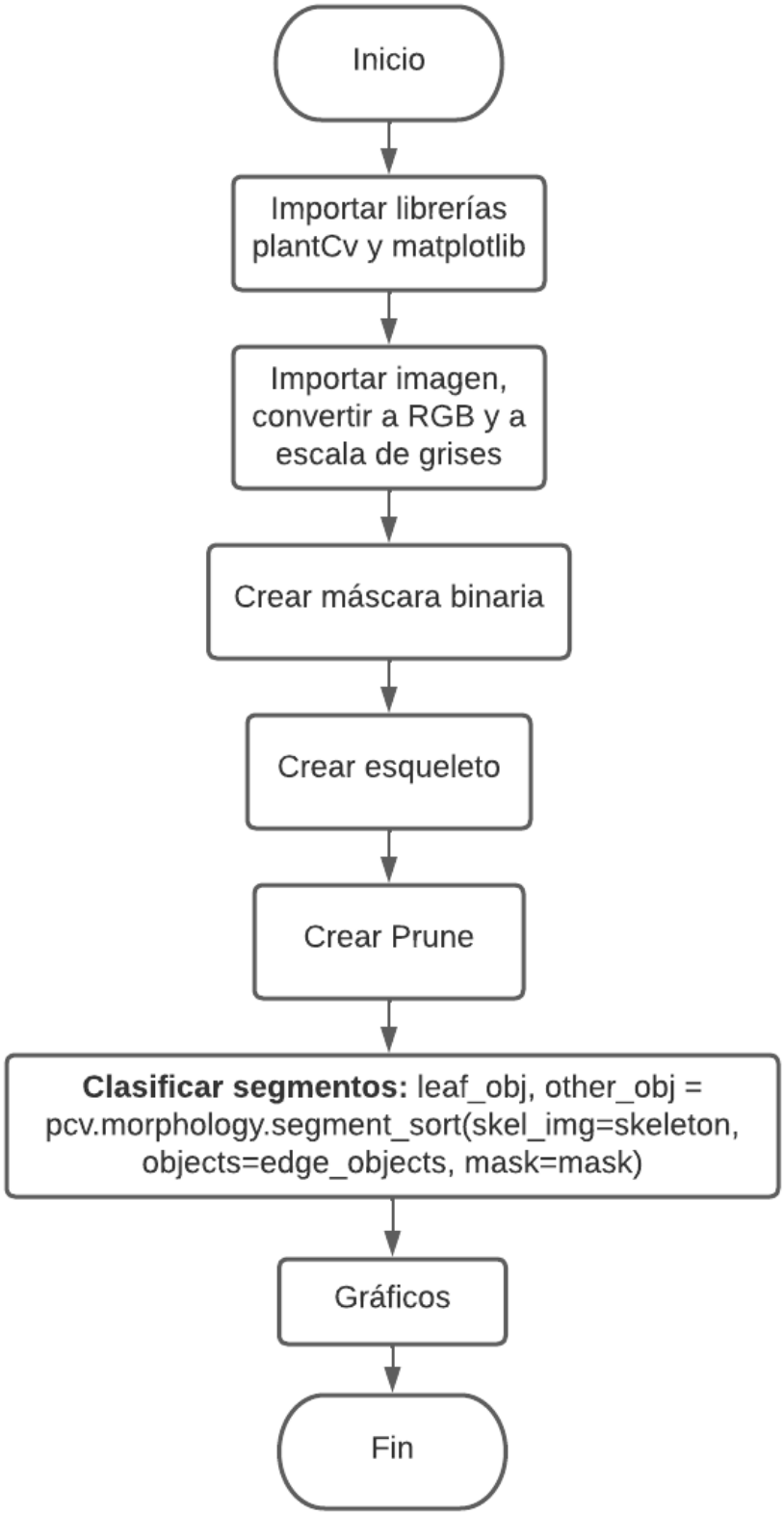

Figura 4.60: *Diagrama de flujo para la función de ordenar segmentos. Fuente: Autor*

```python
import cv2
from plantcv import plantcv as pcv
from matplotlib import pyplot as plt

#IMAGEN
img1 = cv2.imread("mask_image.jpg")
gray_img = cv2.cvtColor(img1, cv2.COLOR_BGR2GRAY)

#define subplots size
#fig, ax = plt.subplots(1, 3, figsize=(10,7))
#fig.tight_layout()

#INICIO ALGORITMO
pcv.params.debug = "plot"
mask = pcv.threshold.binary(gray_img=gray_img, threshold=180, max_value=255, object_type='light')
#Esqueleto
skeleton = pcv.morphology.skeletonize(mask=mask)
#Prune
img1, seg_img, edge_objects = pcv.morphology.prune(skel_img=skeleton, size=10, mask=mask)

#Clasificacion de segmentos
pcv.params.line_thickness = 3

leaf_obj, other_obj = pcv.morphology.segment_sort(skel_img=skeleton, objects=edge_objects)

leaf_obj, other_obj = pcv.morphology.segment_sort(skel_img=skeleton, objects=edge_objects, mask=mask)

#FIN ALGORITMO
```

Figura 4.61: *Código para la función de ordenar segmentos. Fuente: Autor*

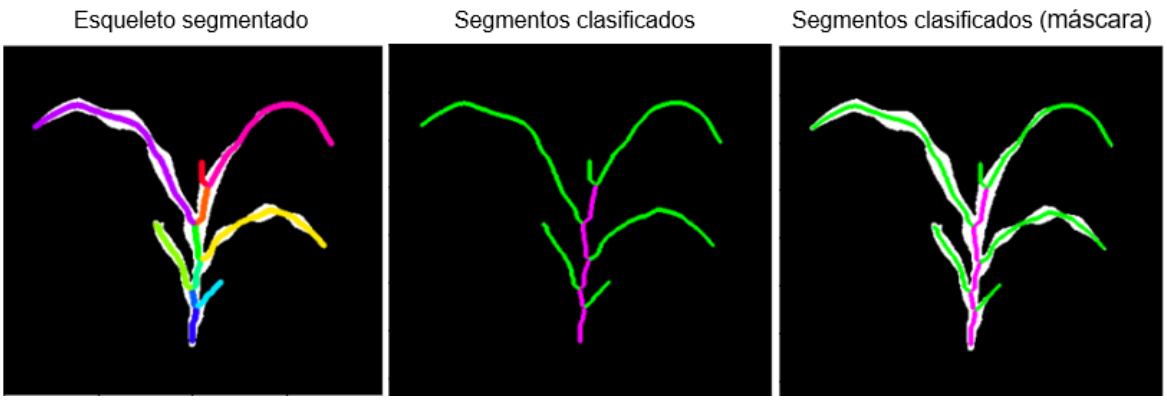

Figura 4.62: *Imagen haciendo uso de la función Ordenar segmentos. Fuente: Autor*

En la figura 4.62 se muestra la ejecución de la función para obtener la clasificación de segmentos de una planta y su resultado, este proceso es muy importante ya que permite categorizar y organizar los segmentos seleccionados el la funcion prune en dos grandes grupos: lo que son las hojas que conforman la planta (lineas de color verde) y otro grupo lo que es el tallo (linea de color morado).

Rellenar segmentos

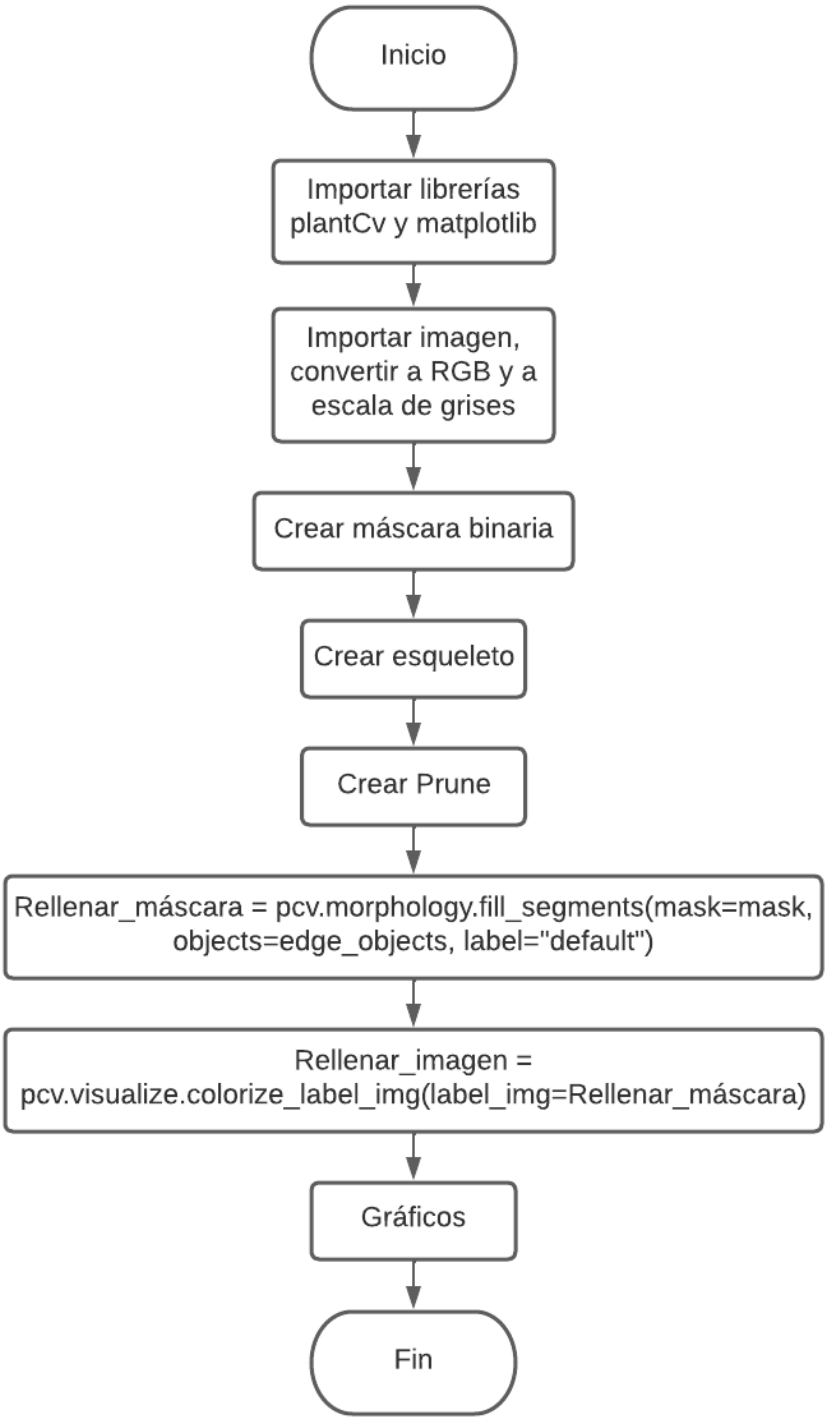

Figura 4.63: *Diagrama de flujo para la función de rellenar segmentos. Fuente: Autor*

```python
import cv2
from plantcv import plantcv as pcv
from matplotlib import pyplot as plt

#IMAGEN
img1 = cv2.imread("mask_image.jpg")
gray_img = cv2.cvtColor(img1, cv2.COLOR_BGR2GRAY)

#define subplots size
#fig, ax = plt.subplots(1, 3, figsize=(10,7))
#fig.tight_layout()

#INICIO ALGORITMO
pcv.params.debug = "plot"
mask = pcv.threshold.binary(gray_img=gray_img, threshold=180, max_value=255, object_type='light')
#Esqueleto
skeleton = pcv.morphology.skeletonize(mask=mask)
#Prune
img1, seg_img, edge_objects = pcv.morphology.prune(skel_img=skeleton, size=10, mask=mask)

#Rellenar segmentos
filled_mask = pcv.morphology.fill_segments(mask=mask, objects=edge_objects, label="default")
# Convert labeled mask to a colorized image
filled_image = pcv.visualize.colorize_label_img(label_img=filled_mask)
# Access data stored out from fill_segments
segments_area = pcv.outputs.observations['default']['segment_area']['value']

#FIN ALGORITMO
```

Figura 4.64: *Código para la función de rellenar segmentos. Fuente: Autor*

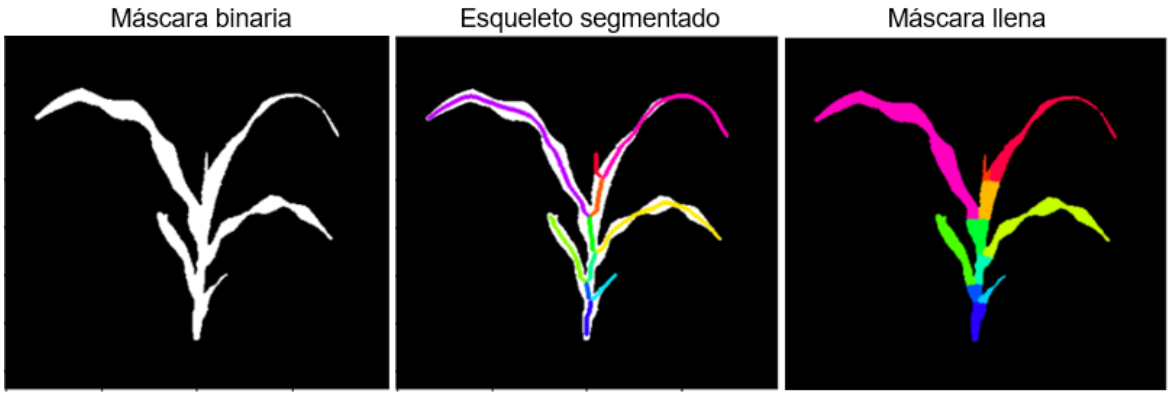

Figura 4.65: *Imagen haciendo uso de la función rellenar segmentos. Fuente: Autor*

En la figura 4.65 se muestra la ejecución de la función para obtener el relleno de segmentos de una planta y su resultado, este proceso permite rellenar la mascara con el color correspondiente de cada uno de los segmentos.

Análisis de forma

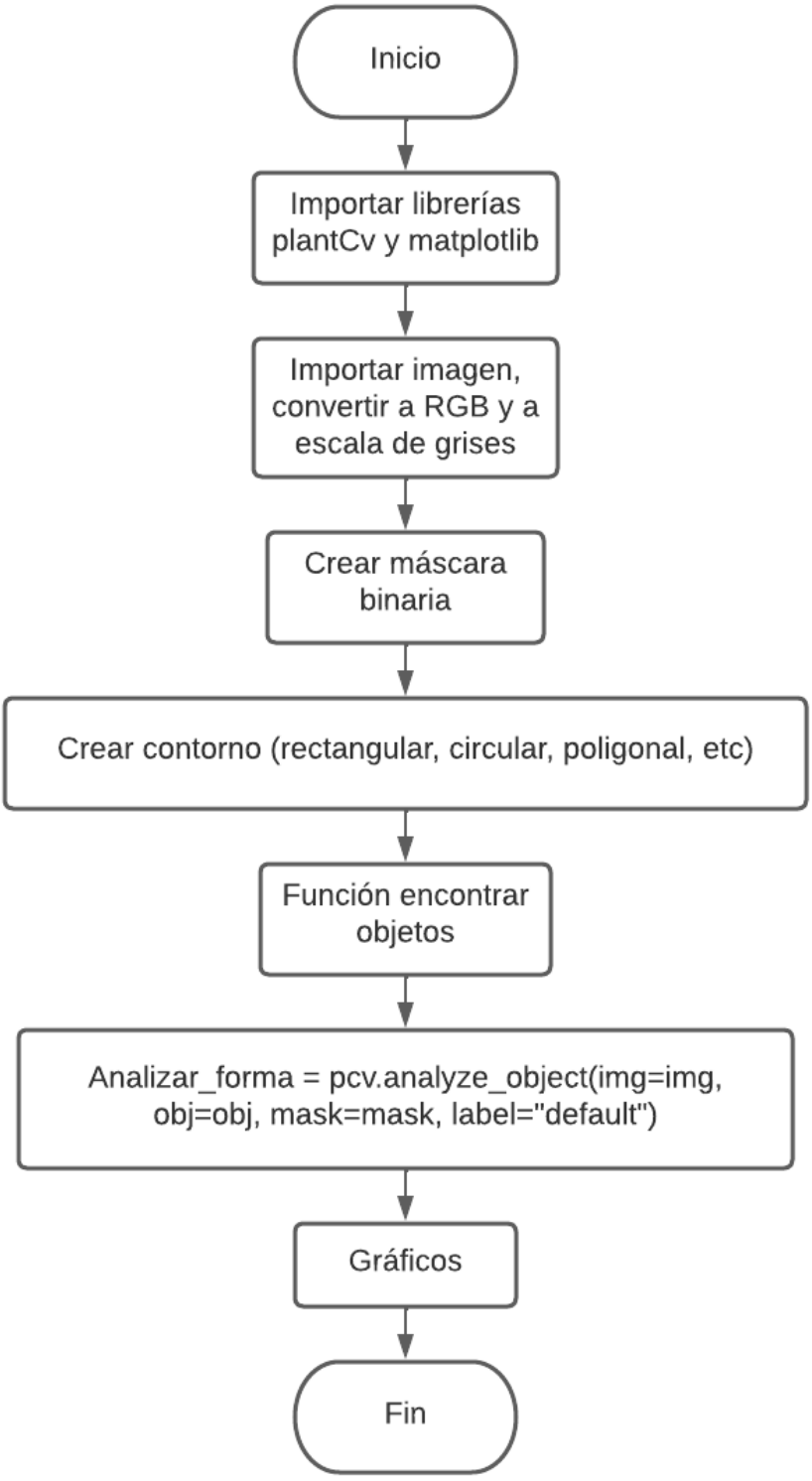

Figura 4.66: *Diagrama de flujo para el análisis de forma. Fuente: Autor*

```python
import cv2
from plantcv import plantcv as pcv
from matplotlib import pyplot as plt

#IMAGEN
img = cv2.imread("mask_image.jpg")
RGB_img = cv2.cvtColor(img1, cv2.COLOR_BGR2RGB)
gray_img = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)

#define subplots size
#fig, ax = plt.subplots(1, 3, figsize=(10,7))
#fig.tight_layout()

#INICIO ALGORITMO
pcv.params.debug = "plot"
#mascara
mask = pcv.threshold.binary(gray_img=gray_img, threshold=180, max_value=255, object_type='light')
#contorno
roi1, roi_hierarchy= pcv.roi.rectangle(img=img, x=50, y=50, h=280, w=300)
#objetos
id_objects, obj_hierarchy = pcv.find_objects(img=img, mask=mask)
roi_objects, hierarchy3, kept_mask, obj_area = pcv.roi_objects(img=img, roi_contour=roi1,
                                                               roi_hierarchy=roi_hierarchy,
                                                               object_contour=id_objects,
                                                               obj_hierarchy=obj_hierarchy,
                                                               roi_type='partial')

obj, mask = pcv.object_composition(img=img, contours=roi_objects, hierarchy=hierarchy3)
#Analizar imagen
analysis_image = pcv.analyze_object(img=img, obj=obj, mask=mask, label="default")

#FIN ALGORITMO
```

Figura 4.67: *Código para el análisis de forma. Fuente: Autor*

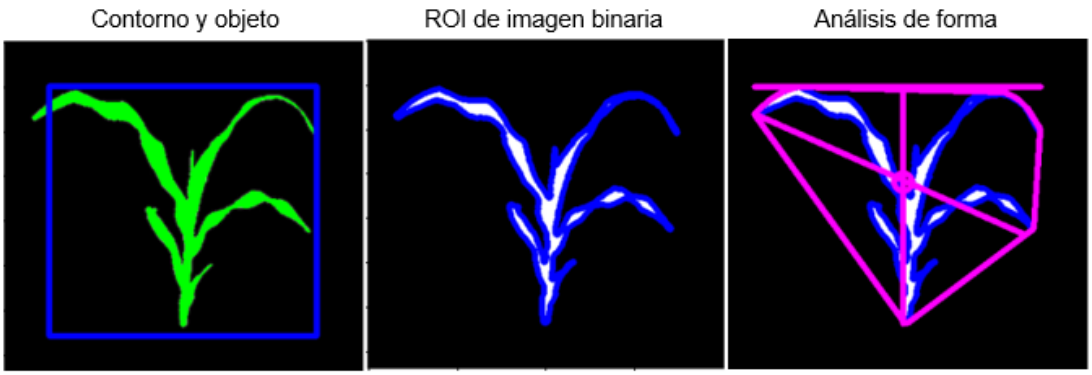

Figura 4.68: *Imagen haciendo uso del análisis de forma. Fuente: Autor*

En la figura 4.68 se muestra la ejecución de la función para obtener el análisis de forma de una planta y su resultado, en este proceso permite extraer algunas características numéricas de la forma de un objeto.

4.4. Conclusiones

Mediante el desarrollo del presente proyecto se lograron comprender y aplicar conceptos de visión artificial en el campo de la agricultura y a su vez de técnicas de robótica para el planteamiento del robot cartesiano.
Como base fundamental del análisis de los algoritmos de visión por computador se había planteado una comparación entre ellos, sin embargo, luego de haber realizado la investigación respectiva de los algoritmos, se pudo concluir que muchos de los algoritmos son herramientas independientes que no son comparables, por ejemplo, la normalización de imágenes, Prune, longitud de segmentos, rellenar segmentos, análisis de forma, etc. Existen también varios algoritmos que sí pueden ser comparados, por ejemplo, umbralización, desenfoque, mascaras, región de interés, etc. Debido a que en la etapa de pregrado se probaron los algoritmos sobre imágenes y no un video en tiempo real sobre un cultivo, es difícil determinar cuales de estos comparables funcionaran con mayor efectividad bajo el espacio coterminal que se desea desarrollar para este proyecto, así que se probaran sobre la cámara OAK-D que estará disponible para este proyecto y se determinara cuales de los comparables son más apropiados de implementar.
El análisis cinemático que se había planteado solamente tenía como propósito el dimensionamiento del robot cartesiano, sin embargo, luego de haber realizado el modelamiento se pudo concluir, que las matrices de transformación resultantes funcionaran como base fundamental del diseño del código del robot que se ha propuesto implementar en el espacio coterminal, las matrices obtenidas son muy similares a las que se usan para programar los robots industriales, además que se facilitara la manipulación del robot debido a que no hay ángulos de rotación en ningún eslabón, por otra parte, se desarrollo la simulación en la cual se puede apreciar una posible trayectoria que va a cumplir el robot cartesiano, en esta trayectoria el robot cruzara por las diferentes materas en las cuales van a estar situadas diferentes variedades de plantas y gracias al algoritmo de detección de objetos que se implementara en la fase de posgrado se tomaran las respectivas muestras y así poder generar el fenotipado de las plantas propuestas en el banco.

4.5. Trabajo futuro

Como trabajo futuro, se tiene la continuación de este proyecto en fase posgrado, en la cual se va a hacer el montaje completo de todo lo mencionado a lo largo de este proyecto, y así tener un sistema completo funcional de un robot cartesiano, que tenga la habilidad de generar el fenotipado de algunas plantas en un ambiente controlado.

Bibliografía

[1] OpenCV, "Opencv ai kit: Oak—d," 2021. [Online]. Available: https://store.opencv.ai/collections/all

[2] C. Gupta, V. Tewari, R. Machavaram, and P. Shrivastava, "An image processing approach for measurement of chili plant height and width under field conditions," *Journal of the Saudi Society of Agricultural Sciences*, 2021.

[3] M. C. L. S. M. Ramos, P., "The benchbot.." 2021. [Online]. Available: https://github.com/precision-sustainable-ag/OpenCV_Competition2021#readme

[4] Y. Chéné, D. Rousseau, P. Lucidarme, J. Bertheloot, V. Caffier, P. Morel, É. Belin, and F. Chapeau-Blondeau, "On the use of depth camera for 3d phenotyping of entire plants," *Computers and Electronics in Agriculture*, vol. 82, pp. 122–127, 2012.

[5] J. D. Sandino, O. L. Ramos-Sandoval, and D. Amaya-Hurtado, "Method for estimating leaf coverage in strawberry plants using digital image processing," *Revista Brasileira de Engenharia Agrícola e Ambiental*, vol. 20, pp. 716–721, 2016.

[6] E. Kiani and T. Mamedov, "Identification of plant disease infection using soft-computing: Application to modern botany," *Procedia computer science*, vol. 120, pp. 893–900, 2017.

[7] X. E. Pantazi, D. Moshou, and A. A. Tamouridou, "Automated leaf disease detection in different crop species through image features analysis and one class classifiers," *Computers and electronics in agriculture*, vol. 156, pp. 96–104, 2019.

[8] E. Hamuda, M. Glavin, and E. Jones, "A survey of image processing techniques for plant extraction and segmentation in the field," *Computers and electronics in agriculture*, vol. 125, pp. 184–199, 2016.

[9] O. Ospina, H. Anzola Vásquez, O. Ayala Duarte, and A. Baracaldo Martínez, "Validación de un algoritmo de procesamiento de imágenes red green blue (rgb), para la estimación de proteína cruda en gramíneas vs la tecnología de espectroscopía de infrarrojo cercano (nirs)," *Revista de Investigaciones Veterinarias del Perú*, vol. 31, no. 2, 2020.

[10] R. Naushad, "Introducción a los kits opencv ai (oak-1 y oak-d)," 2021. [Online]. Available: https://medium.com/swlh/introduction-to-opencv-ai-kits-oak-1-and-oak-d-6cdf8624517

[11] J. C. Puchalt, J. F. Gonzalez-Rojo, A. P. Gómez-Escribano, R. P. Vázquez-Manrique, and A.-J. Sánchez-Salmerón, "Multiview motion tracking based on a cartesian robot to monitor caenorhabditis elegans in standard petri dishes," vol. 12, no. 1, pp. 1–11, 2022.

[12] A. Milella, R. Marani, A. Petitti, and G. Reina, "In-field high throughput grapevine phenotyping with a consumer-grade depth camera," *Computers and electronics in agriculture*, vol. 156, pp. 293–306, 2019.

[13] Z. Chen, J. Wang, T. Wang, Z. Song, Y. Li, Y. Huang, L. Wang, and J. Jin, "Automated in-field leaf-level hyperspectral imaging of corn plants using a cartesian robotic platform," *Computers and Electronics in Agriculture*, vol. 183, p. 105996, 2021.

[14] S. G. Tzafestas, *Introduction to mobile robot control*. Elsevier, 2013.

[15] PlantCV, "Welcome to the documentation for plantcv," 2020. [Online]. Available: https://plantcv.readthedocs.io/en/stable/

[16] J. A. Muñoz Guevara, "Propuesta para la clasificación y paletizado automático de productos en la planta de grupo familia medellín," 2017.

[17] N. L. S. Palomino and U. N. R. Concha, "Técnicas de segmentación en procesamiento digital de imágenes," *Revista de investigación de Sistemas e Informática*, vol. 6, no. 2, pp. 9–16, 2009.

[18] J. Angulo and J. Serra, "Segmentación de imágenes en color utilizando histogramas bivariables en espacios color polares luminancia/saturación/matiz," *Computación y Sistemas*, vol. 8, no. 4, pp. 303–316, 2005.

[19] C. B. Rene, T. M. S. Javier, M. H. German, and R. C. F. Ardul, "Algoritmo para obtención de índice de desempeño, en configuraciones de robots scara, cartesiano y antropomórfico." *Innovaciones en Mecatrónica*, p. 41.

[20] C. HERNÁNDEZ PÉREZ, "Desarrollo de un simulador basado en un robot cartesiano de tres grados de libertad," Tech. Rep., 2015.

[21] F. Pérez Sanz *et al.*, "Desarrollo de un sistema de fenotipado basado en visión artificial para el estudio de la cinética de crecimiento de plantas," 2018.

[22] M. I. Ros Sánchez *et al.*, "Diseño de un sistema inteligente móvil para facilitar procesos de fenotipado en agricultura de precisión," 2020.

[23] M. E. B. NÁJERA, "Control de trayectorias de un robot cartesiano," 2016.

[24] L. Z. Muñoz Sanchez and M. I. Gómez Gómez, "Análisis del impacto de las técnicas de preprocesamiento en la segmentación de imágenes de plantas en ambientes semi-controlados," 2016.

[25] J. González-Esquiva, J. Hernández-Hernández, G. García-Mateos, A. Ruiz-Canales, and J. Molina-Martínez, "Estudio y comparación de técnicas de segmentación por color para la estimación de la fracción de cobertura vegetal," in *II Simposio Nacional de Ingeniería Hortícola, Almería*, 2016.