UNIVERSIDAD TÉCNICA DEL NORTE

Facultad de Ingeniería en Ciencias Aplicadas

Carrera de Software

ESTUDIO DE LENGUAJES DE PROGRAMACIÓN ORIENTADOS A GPU PARA LA CREACIÓN DE UN PROTOTIPO DE PROCESAMIENTO DE IMÁGENES DE ACTIVIDADES AGRÍCOLAS.

Trabajo de grado previo a la obtención del título de Ingeniero en Software

Autor:

Cristian Daniel Guerra Ruiz

Director:

Ing. Marco Remigio Pusdá Chulde MSc.

Ibarra - Ecuador 2022

UNIVERSIDAD TÉCNICA DEL NORTE

BIBLIOTECA UNIVERSITARIA

AUTORIZACIÓN DE USO Y PUBLICACIÓN

A FAVOR DE LA UNIVERSIDAD TÉCNICA DEL NORTE

IDENTIFICACIÓN DE LA OBRA

En cumplimiento del Art. 144 de la Ley de Educación Superior, hago la entrega del presentetrabajo a la Universidad Técnica del Norte para que sea publicado en el Repositorio Digital Institucional, para lo cual pongo a disposición la siguiente información:

DATOS DE CONTACTO			
CÉDULA DE IDENTIDAD:	172552764-0		
APELLIDOS Y NOMBRES:	CRISTIAN DANIEL GUERRA RUIZ		
DIRECCIÓN:	OTAVALO, URB. MARÍA JOSÉ, CASA 114		
EMAIL:	cdguerrar@utn.edu.ec		
TELÉFONO FIJO:	062 930 418	**TELÉFONO MÓVIL:**	0969716527

DATOS DE LA OBRA	
TÍTULO:	"ESTUDIO DE LENGUAJES DE PROGRAMACIÓN ORIENTADOS A GPU PARA LA CREACIÓN DE UN PROTOTIPO DE PROCESAMIENTO DE IMÁGENES DE ACTIVIDADES AGRÍCOLAS"
AUTOR:	CRISTIAN DANIEL GUERRA RUIZ
FECHA:	01/11/2022
PROGRAMA:	■ PREGRADO ☐ POSGRADO
TÍTULO POR EL QUE OPTA:	INGENIERO EN SOFTWARE
ASESOR /DIRECTOR:	ING. MSC. MARCO PUSDÁ

CONSTANCIAS

El autor manifiesta que la obra objeto de la presente autorización es original y se la desarrolló,sin violar derechos de autor de terceros, por lo tanto, la obra es original y que es el titular delos derechos patrimoniales, por lo que asume la responsabilidad sobre el contenido de la misma y saldrá en defensa de la Universidad en caso de reclamación por parte de terceros.

Ibarra, al 1 día del mes de noviembre de 2022

EL AUTOR:

Nombre: Daniel Guerra Ruiz
Cédula: 172552764-0

CERTIFICACIÓN DEL DIRECTOR DEL TRABAJO DE GRADO

UNIVERSIDAD TÉCNICA DEL NORTE
FACULTAD DE INGENIERÍA EN CIENCIAS APLICADAS

CERTIFICACIÓN DEL ASESOR

Certifico que la Tesis previa a la obtención del título de Ingeniera en Software con el tema: "ESTUDIO DE LENGUAJES DE PROGRAMACIÓN ORIENTADOS A GPU PARA LA CREACIÓN DE UN PROTOTIPO DE PROCESAMIENTO DE IMÁGENES DE ACTIVIDADES AGRÍCOLAS" ha sido desarrollada y terminada en su totalidad por el Sr. Cristian Daniel Guerra Ruiz, con cédula de identidad Nro. 172552764-0 bajo mi supervisión para lo cual firmo en constancia.

0401200951
MARCO REMIGIO
PUSDA CHULDE

Firmado digitalmente por 0401200951 MARCO REMIGIO PUSDA CHULDE
Fecha: 2022.11.01 11:27:33 -05'00'

Msc. Marco Pusdá
DIRECTOR DE TESIS

DEDICATORIA

El presente trabajo de titulación se lo dedico a mi mamá, Wilma Ruiz, quien siempre estuvo a mi lado, viendo que no me falte nada necesario para seguir con la universidad y siempre siendo mi psicóloga la cual me escuchó y me animó a seguir adelante en mi vida. A mi papá, Ernesto Guerra, quien me dio y forjó el carácter, al igual que la personalidad para acabar la carrera y crecer como persona y profesional, quien me impartió sus conocimientos, siendo yo, de esta forma, un alumno más para él. A mi hermano, Andrés Guerra, quien siempre me ayudó con las materias difíciles, quien me guiaba a encontrar la solución a muchos problemas, quien al igual que mi papá me impartió muchos conocimientos y me brindó muchas oportunidades para crecer profesionalmente. A mi novia, Dayana Torres, la cual desde la pandemia ha sido mi fuerza y mi compañía, quien no me ha dejado caer a ningún instante, al igual que no me ha dejado rendirme y siempre impulsándome a ser mejor, como persona y como profesional. A mis amigos, quienes han hecho de mi etapa en la universidad algo muy bonito y que quedará guardado en mi mente, con ellos pasamos muy buenos y duros momentos dentro de las aulas y laboratorios, pero nos mantuvimos unidos a pesar de todo, siempre ayudándonos.

AGRADECIMIENTO

Agradezco a mis papás por siempre permanecer a mi lado, quienes a pesar de muy duros momentos que tuvimos que vivir siempre han sido mis pilares de vida y están para mi cuando los necesito.

Agradezco a mi hermano por ayudarme tanto en la Universidad como en la vida misma, siendo siempre tan apegado a mí y haciéndome crecer como persona y profesional, por todas las oportunidades que me brindó y me ayudó.

Agradezco a mi novia por ser y darme de su fuerza para culminar la carrera, por estar para mí en todo momento, ayudándome, escuchándome, siendo mi compañera de vida.

Agradezco a mis amigos y compañeros por toda la ayuda que me brindaron, por toda esa amistad que forjamos, por permanecer unidos y por crear muchos recuerdos junto a mí.

Agradezco a mi tutor del presente trabajo de titulación por toda la guía y el conocimiento impartido tanto en las aulas como en esta etapa final de la carrera, de igual manera a mis asesores por su tiempo y experiencia que compartieron conmigo.

TABLA DE CONTENIDO

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

Resumen

La tecnología avanza y sirve de herramienta para más áreas de la industria, en el caso de la agricultura, esta se apoya en muchas herramientas que facilitan su trabajo y progreso, una de estas herramientas es el procesamiento de imágenes, el cual ayuda con la detección de maleza en las líneas de cultivo. Dicho procesamiento se ha desarrollado en varios lenguajes de programación como Matlab siendo de los más conocidos, Python siendo uno de los que más terreno está ganando en los últimos años, CUDA como una herramienta insignia de la empresa NVIDIA, entre otros lenguajes.

El presente trabajo de titulación tiene como objetivos el identificar el lenguaje tendencia entre los investigadores que desarrollan aplicaciones para agricultura de precisión. Todo esto se logra por medio de un estudio de mapeo sistemático y, a partir de sus resultados demostrar por medio de un prototipo el uso de un lenguaje de los utilizados en el estudio (Python). El prototipo se basa en los fundamentos de la ingeniería del cuerpo del conocimiento de la ingeniería del software (SWEBOK).

Palabras clave: GPU, Lenguajes de programación, Procesamiento de imágenes, detección de malezas, agricultura de precisión, Prototipo.

Abstract

Technology advances and serves as a tool for more areas of industry, in the case of agriculture, it is supported by many tools that facilitate its work and progress, one of these tools is the image processing, which helps with weed detection in the crop rows. Such processing has been developed in several programming languages such as Matlab the most known being, Python being one of the most gaining grounds in the last few years, CUDA as a signature tool of NVIDIA, among other languages.

The objectives of this degree project are to identify which language is trending among researchers who develop this type of applications. All this is achieved by applying a systematic mapping study, and, from its results demonstrate by prototyping the use of this identified language. The prototype is based on the engineering fundamentals of the software engineering body of knowledge (SWEBOK).

Keywords: GPU, Programming languages, Image processing, weed detection, precision agriculture, Prototype.

INTRODUCCIÓN

Antecedentes

El procesamiento de imágenes de actividades agrícolas requiere de un proceso rápido y eficiente, esto ha venido siendo un problema de mucha dificultad para trabajar solamente con la Unidad de Procesamiento Central (CPU), aunque es cierto que la potencia de procesamiento de las CPU ha aumentado a lo largo de los años, pero se ha quedado atrás para la gran demanda de muchas aplicaciones. (HajiRassouliha et al., 2018).

Es por esto, que han venido surgiendo varios lenguajes de programación orientados al uso de la Unidad de procesamiento gráfico (GPU) y es dificultoso saber cuál de todos estos lenguajes es el más utilizado y el más adecuado para dicho procesamiento.

Situación Actual

Actualmente las soluciones que brindan para el procesamiento de imágenes de actividades agrícolas están enfocadas en su mayoría en el uso de la CPU, por lo que se quiere cambiar este enfoque hacia el uso de la GPU, la cual tiene un mejor rendimiento en el procesamiento de imágenes gracias a la cantidad de unidades lógicas y aritméticas que sirven de ayuda para llevar a cabo las operaciones de modelado y renderizado tridimensionales. (Castaño-Díez et al., 2008). Y no se han dado un alto índice de estudios o indagaciones literarias para saber que lenguaje de programación orientado a la GPU es el adecuado.

Planteamiento del problema

Debido a la existencia de varios lenguajes de programación orientados a la GPU y el desconocimiento de los mismo, existen un alto grado de limitada aplicabilidad de dichos lenguajes para el procesamiento de imágenes de actividades agrícolas enfocados al uso de la GPU (ver Fig. 1).

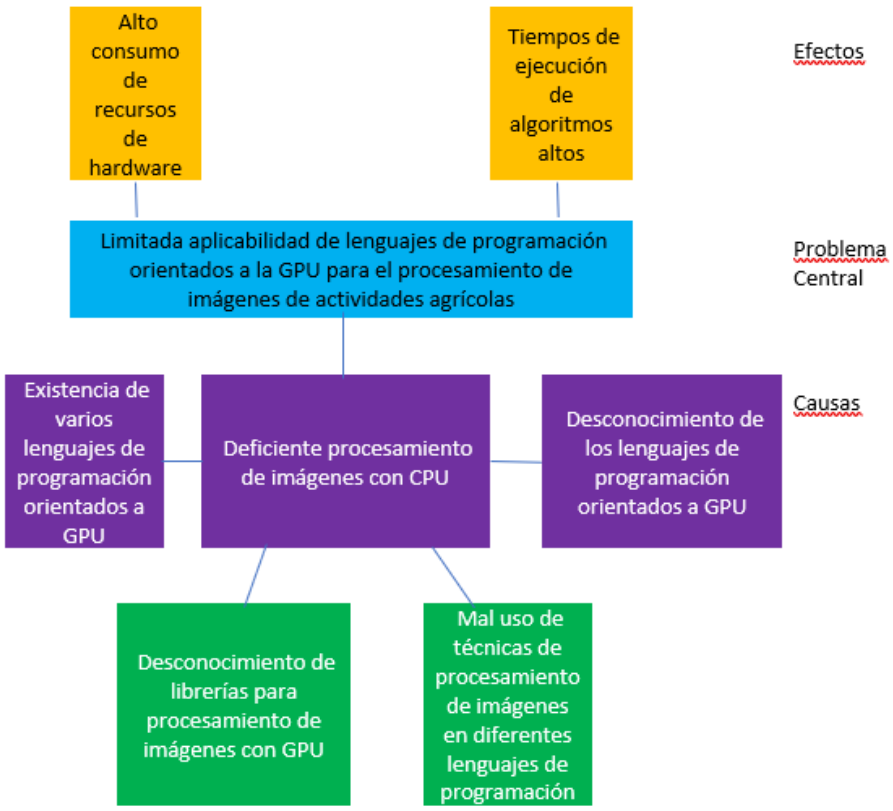


Fig. 1 Árbol de problemas.

Fuente: Propia

Objetivos

Objetivo General

Estudiar lenguajes de programación orientados a GPU para la creación de un prototipo de procesamiento de imágenes de actividades agrícolas.

Objetivos Específicos

- Aplicar una revisión de la literatura en base a un estudio de mapeo sistemático para descubrir el lenguaje de programación orientado a la GPU más utilizado.
- Desarrollar un prototipo para el procesamiento de imágenes de actividades agrícolas en base al lenguaje resultante de la revisión literaria.
- Validar los resultados del presente trabajo mediante los fundamentos de ingeniería del cuerpo del conocimiento SWEBOK.

Alcance y metodología

Alcance

Se realizó una revisión literaria de treinta artículos referentes al procesamiento de imágenes de actividades agrícolas, la búsqueda de los artículos se la define con los siguientes parámetros: un periodo de 5 años atrás del año actual, idioma inglés, en base a cadenas de búsqueda escritas en las bases de datos bibliográficas de la Universidad Técnica del Norte. Se realizó una indagación donde se identificó el lenguaje de programación orientado a la GPU más utilizado en la revisión de los artículos, de esta forma se procedió al desarrollo de un prototipo de aplicación de escritorio con dicho lenguaje resultante para la detección de malezas en imágenes de cultivos agrícolas de la zona 1 de Ecuador, se trabajó con una Tarjeta Gráfica NVIDIA GeForce RTX 2060 adicionada a la placa madre de una laptop modelo ASUS ROG Strix G531G con un procesador Intel Core i7-9750H con 16GB de memoria RAM, el conjunto (data set) de imágenes utilizado en dicho prototipo fue proporcionado por el tutor del presente trabajo de titulación (ver Fig. 2). Para finalizar se validó los resultados del algoritmo con imágenes con los siguientes parámetros: luminosidad, tamaño y resolución.

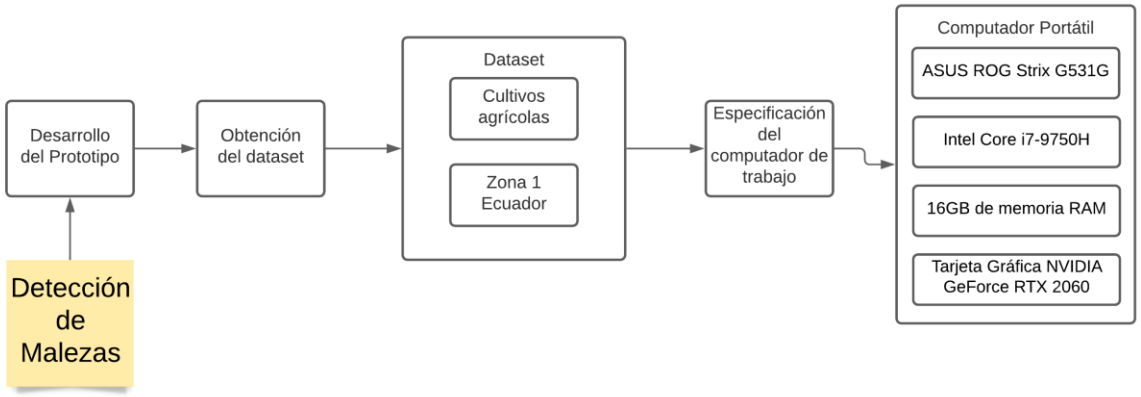


Fig. 2. Alcance tecnológico.

Fuente: Propia

Metodología

Para el presente trabajo se ocupó una investigación documental, la cual realiza una indagación de todo tipo de documentos escritos respecto a un determinado tema. (Rizo, 2015). Se realizó un estudio de mapeo sistemático para el proceso de indagación literaria, este estudio es una técnica particular para la revisión de la literatura, empleado para investigar respondiendo ciertas preguntas. (Corona y Montoya, 2018). De esta forma se identifica el

lenguaje de programación orientado a GPU más utilizado en artículos científicos sobre procesamiento de imágenes de actividades agrícolas. Para la creación del prototipo se lo realizó con una metodología de desarrollo de Software en Cascada, este es un modelo secuencial que nos permite organizar de mejor manera todas las actividades por todo el ciclo de vida del software y es el modelo más fácil de entender e implementar. (Papadopoulos, 2015). También se basó en los fundamentos de ingeniería del cuerpo del conocimiento de la ingeniería de Software (SWEBOK). Todo este proceso se lo muestra gráficamente (ver Fig. 3).

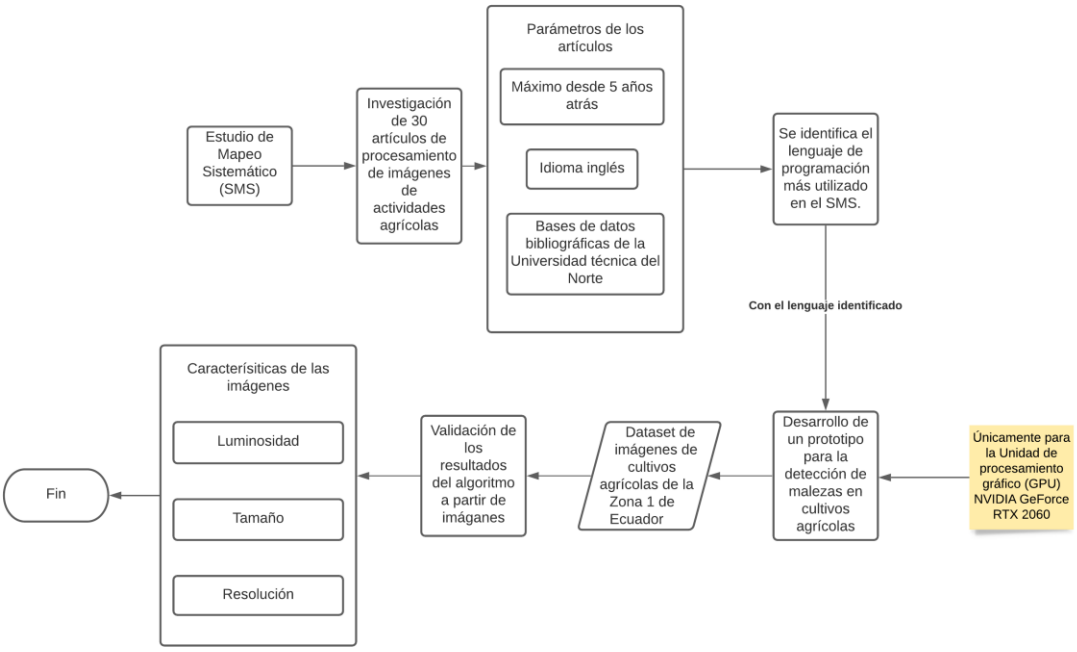


Fig. 3. Diagrama de flujo del proceso.

Fuente: Propia

Justificación y Riesgos

Justificación

Ecuador está muy atrasado tanto en conocimiento como en tecnología, el acceso a la tecnología es limitado y de un precio muy alto, por lo previamente mencionado, este trabajo de titulación se enfoca en el Objetivo de Desarrollo Sostenible 9, el Objetivo de Desarrollo 5 del Plan Nacional toda una vida, estos objetivos proponen avances tecnológicos a través de inversión en investigación e innovación científica y de esta manera impulsar la competitividad y el crecimiento económico.

Justificación Tecnológica

Actualmente la tecnología ha avanzado muy rápido y la industria agrícola ha venido implementando sistemas de automatización para que esta prospere y entregue todo su potencial. (Morales et al., 2015). Debido a la falta de conocimiento de las bondades de la GPU y la variedad de lenguajes de programación orientados a la misma, se presenta una mala aplicabilidad de estos sistemas de automatización en actividades agrícolas.

Riesgos

Se realizó un análisis de posibles riesgos que pueden afectar la realización del presente trabajo de titulación (ver Fig. 4), de la misma forma se pensó en posibles soluciones para los mismos.

Riesgo 1: Daño de la tarjeta gráfica

Riesgo 2: Mala aplicabilidad del Estudio de Mapeo Sistemático

Riesgo 3: Mala calidad de la data set de imágenes

Riesgo 4: No contar con un dron para la obtención de imágenes

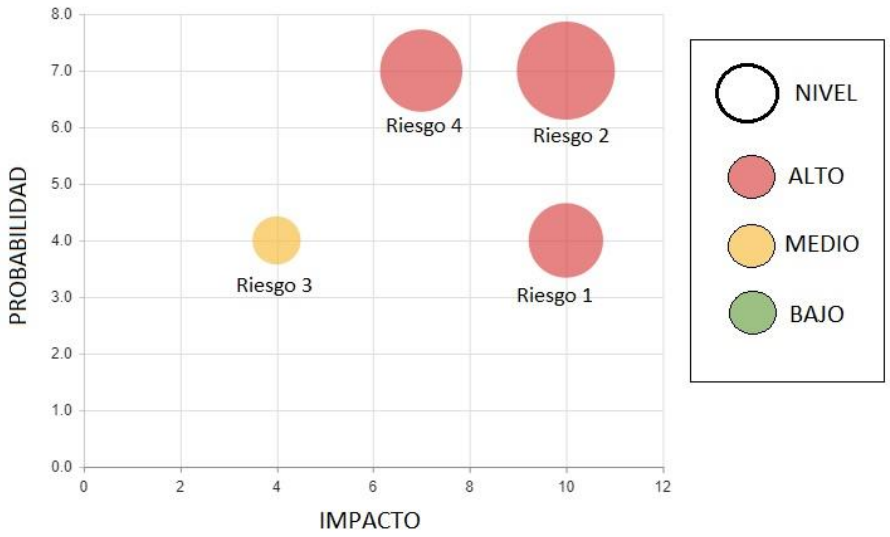


Fig. 4. Matriz de riesgos.

Fuente: Propia

Daño de la tarjeta gráfica: Para minimizar este riesgo se cuenta con otra tarjeta gráfica y un computador de escritorio, para no quedar estancado en el proyecto.

Mala aplicabilidad del Estudio de Mapeo Sistemático: Para no tender a realizar un mal estudio se debe tener mucha bibliografía sobre la aplicabilidad de este.

Mala calidad de la data set de imágenes: Para evitar tener problemas con las imágenes para probar el prototipo se debe obtener un data set de imágenes de una muy buena calidad por medio de un dron.

No contar con un dron para la obtención de imágenes: Para minimizar este riesgo podemos contar con un conjunto de imágenes obtenidas por una cámara fotográfica.

CAPÍTULO 1

1. Marco Teórico

1.1. Definición de conceptos

En este subcapítulo se explica los conceptos necesarios para la comprensión del presente trabajo de grado y así lograr que el lector obtenga un buen entendimiento de este.

1.1.1. Revisión sistemática de la literatura (SLR)

La revisión sistemática de literatura, conocida en el idioma inglés como Systematic Literature Review (SLR), es una herramienta que permite evaluar e identificar información relevante que logre responder una pregunta de investigación propuesta, la SLR es vista como un estudio secundario y difiere de las revisiones tradicionales por ser planificadas. (B. Kitchenham, 2007).

Las SLRs consumen mucho tiempo y esta puede ser una de sus desventajas al momento de comenzar una investigación, se debe tomar en cuenta mucho la metodología que realizará y el esfuerzo que esta requerirá. SI bien los temas de investigación de una SLR son referentes a Software, este tipo de revisión puede ser aplicada a otras disciplinas. (B. Kitchenham, 2007). En la Fig. 5 se muestra el proceso que se debe llevar a cabo para realizar una adecuada SLR.

Fig. 5. Proceso de una SLR.
Fuente: (Genero et al., 2014)

1.1.2. Estudio de mapeo sistemático (SMS)

El estudio de mapeo sistemático se lo conoce en el idioma inglés como Systematic Maping Study (SMS), es un estudio secundario que complementa a las SLRs y su metodología se basa en estas, su principal objetivo es dar una visión más amplia sobre un tema de ingeniería de software de interés, es común que se utilice un SMS para para identificar tendencias. (B. A. Kitchenham et al., 2011).

En muchas revisiones literarias es muy común que se pueda confundir una SLR con una SMS y por eso es preciso describir las diferencias entre estas 2, la Tabla 1 nos muestra dichas diferencias existentes.

TABLA 1. Diferencias entre SMS y SLR.

Elementos	**SMS**	**SLR**
Objetivos	Clasificación y análisis temático de la literatura sobre un tema específico de la ingeniería de software	Identificar las mejores prácticas con respecto a procedimientos, tecnologías, métodos o herramientas específicas, mediante la agregación de información obtenida a partir de estudios empíricos.
Pregunta de investigación	Genérica, relacionada con tendencias de investigación, como, por ejemplo: qué investigadores, cuántos estudios, qué tipo de estudios, etc.	Específica, relacionada con resultados de estudios empíricos, como, por ejemplo: ¿Es mejor el método/tecnología A que la B?
Proceso de búsqueda	Definido por el área de estudio o de interés.	Definido por la pregunta de investigación la cual identifica la tecnología específica que está siendo investigada.
Alcance	Amplio – se incluyen todos los artículos sobre un área de interés, pero	Centrado – sólo se incluyen artículos que contengan estudios empíricos

	sólo se extrae de ellos datos para clasificarlos	relacionados con las preguntas de investigación y se extrae de ellos información detallada sobre los resultados obtenidos en cada uno de ellos
Requisitos de la estrategia de búsqueda	A menudo menos estricta si sólo se buscan tendencias, por ejemplo, se puede buscar sólo en un conjunto específico de publicaciones, limitándolas a artículos de revistas, o limitándolas a una o dos bibliotecas digitales	Extremadamente exigente– se deben encontrar todos los artículos relevantes. Generalmente además de buscar en las fuentes establecidas, puede ser necesario buscar en las referencias de los estudios primarios seleccionados o consultar a los expertos para incluir el mayor número de artículos posible
Evaluación de la calidad	No es esencial. Al incluir tanto estudios teóricos como empíricos de cualquier tipo, suele ser muy difícil definir un mecanismo de evaluación	Es importante asegurarse de que los resultados se basan en la evidencia de mejor calidad
Resultados	Un conjunto de artículos relacionados con un área de interés clasificados en una serie de dimensiones, especificando el número total de artículos en cada dimensión	Se agregan los resultados de los estudios empíricos para contestar a las preguntas de investigación

Fuente: (Genero et al., 2014)

1.1.3. Unidad de procesamiento gráfico (GPU)

La GPU o Unidad de Procesamiento Gráfico es un hardware de aceleración de procesamiento gráfico, la cual fue diseñada para satisfacer la gran cantidad de cálculos en algoritmos de una complejidad muy alta, estos aceleradores también tienen la ventaja de poder utilizarse en dispositivos que no cuenten con sistemas computacionales. (HajiRassouliha et al., 2018).

1.1.4. Lenguajes de Programación orientados a GPU

Con la mejora notable de rendimiento gracias a la implementación de la GPU se han dado a conocer varias tecnologías y lenguajes de programación orientados a este hardware como son:

CUDA

Es una plataforma de desarrollo para las GPGPU o Computación de propósito general en unidades de procesamiento gráfico creada por la Corporación NVIDIA enfocada en la computación y programación paralela. (Saahithyan & Suthakar, 2017). En la Fig. 6 se presenta el modelo de programación paralela en el que se basa CUDA.

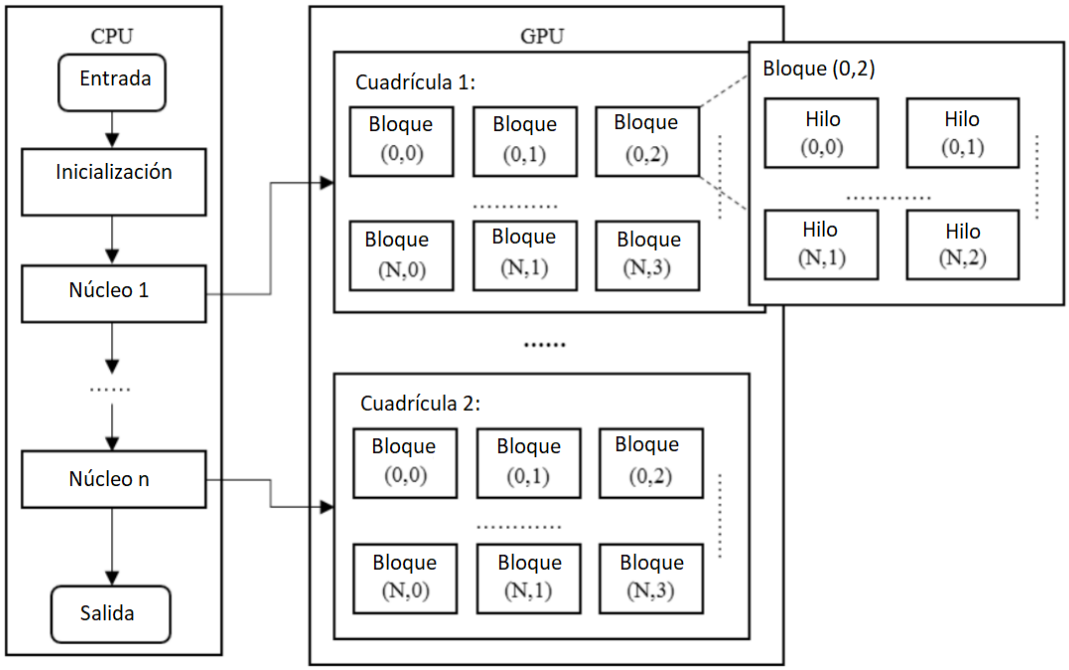


Fig. 6. Modelo de programación de CUDA.

Fuente: (Xiang et al., 2017)

Python

Es un lenguaje de tipado dinámico e interpretado, fácil de aprender y de utilizar, esto hace que una variable pueda tener cualquier tipo de dato y no necesita ser compilado para ser ejecutado. (Nolasco, 2018). En la Fig. 7 se presenta el modelo de programación en el que se basa Python.

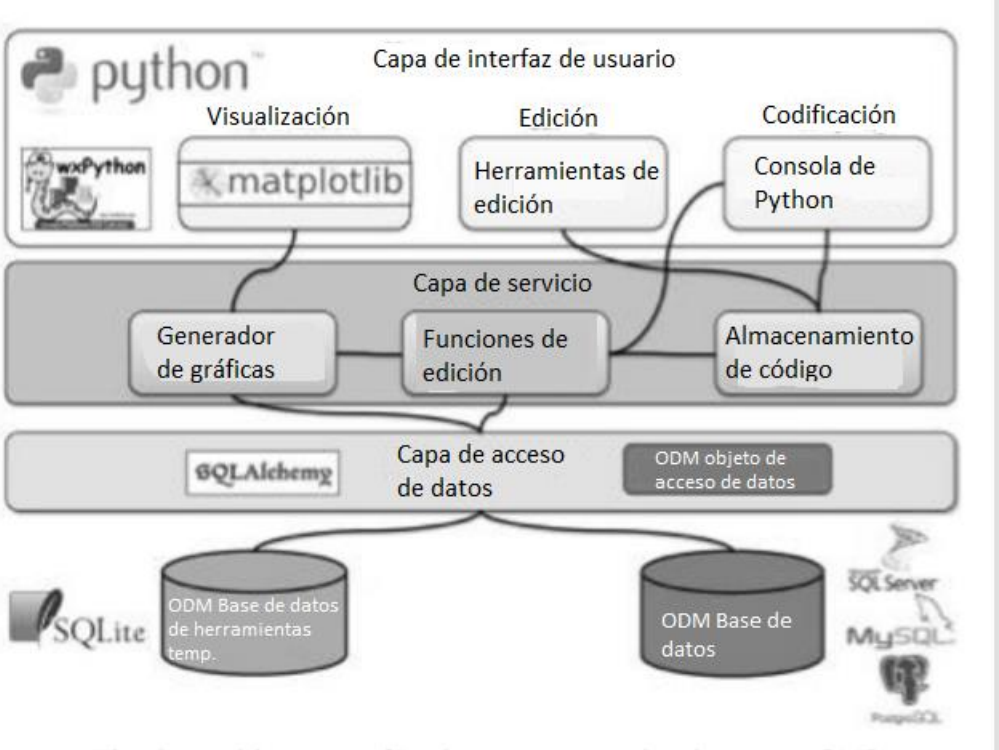


Fig. 7. Modelo de programación de Python.

Fuente: (Lainjo, 2019)

Matlab

Es una plataforma de desarrollo para realizar cálculos numéricos y analizar datos y desarrollar algoritmos con un lenguaje propio, esta plataforma permite realizar gráficos en dos y tres dimensiones. (García y Rodríguez, 2020). En la Fig. 8 se presenta la arquitectura de programación paralela en la que se basa Matlab.

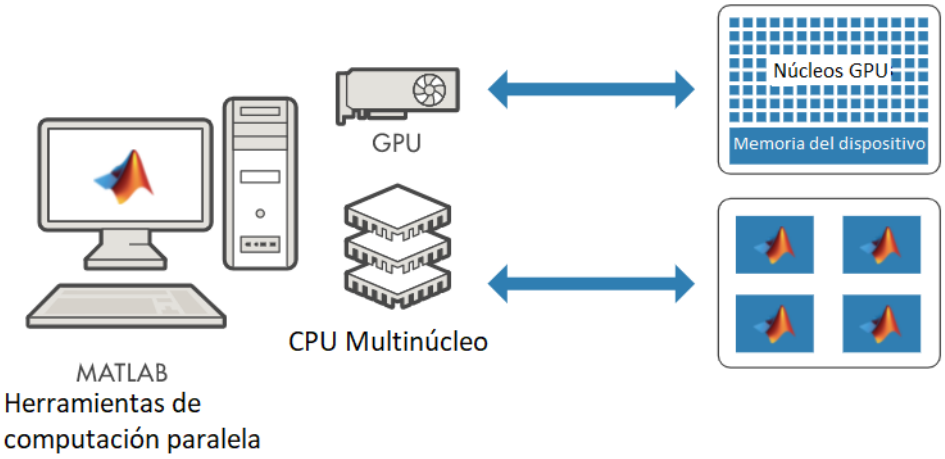


Fig. 8. Arquitectura de programación paralela Matlab.

Fuente: (MathWorks, 2022)

1.1.5. Inteligencia artificial (IA)

La Inteligencia Artificial pretende simular le comportamiento humano y su forma de razonar, esta ciencia entrena a los sistemas informáticos para que pueda tener habilidades humanas como el aprendizaje o la toma de decisiones. (C. Zhang & Lu, 2021).

La IA ha venido evolucionando de manera muy acelerada, abriéndose paso por diferentes áreas más allá de la ingeniería y de los sistemas computacionales como la medicina, la agricultura, entre otras. Justamente hace 20 años expertos y científicos no creían en la idea de que una máquina pueda pensar por sí misma sin la ayuda de un individuo que la controle, la IA apenas se la reconocía como una disciplina o materia de la ingeniería. (Wooldridge, 2020).

1.1.6. Procesamiento de imágenes

El Procesamiento de imágenes es una técnica que realiza ciertas transformaciones a las imágenes para lograr identificar información útil.(Zhang et al., 2022). Esta técnica de preprocesamiento es un campo que se ha visto evolucionar y avanzar de manera muy rápida en la innovación de datos y que se desarrolla en los bastos campos de la información. (Xu, 2020). La Fig. 9 presenta los pasos fundamentales para realizar un procesamiento de imágenes.

Fig. 9. Pasos fundamentales del procesamiento de imágenes.

Fuente: (Gonzalez & Woods, 2018)

1.1.7. Agricultura de precisión

La agricultura describe a un grupo de actividades y técnicas que consisten en utilizar el entorno que rodea un punto geográfico para producir comida o alimentos, pues bien, estas actividades son consideradas como la mayor fuente de empleo en una gran cantidad de países. Pero con el crecimiento de la población, la agricultura ha venido necesitando la ayuda e integración de la tecnología o lo digital. (Coulibaly et al., 2022).

La agricultura de precisión se basa en la unión y aplicación de tecnologías de la información y comunicación en el manejo y gestión de actividades agrícolas, esto con el objetivo de modularizar las mejores prácticas agrícolas y así realizar un control adecuado de los procesos agrícolas y la optimización de la producción y la intervención humana. (Coulibaly et al., 2022).

También se considera a la agricultura de precisión como un método que adopta varias tecnologías de la información con el fin de incrementar la calidad, cantidad, tiempo y ubicación de la producción agrícola, pero aun cuando su objetivo es reducir los costos de algunos procesos, este método puede llegar a ser muy costoso, por lo que no se lo ejecuta en muchos campos o lugares donde no tienen la suficiente estabilidad económica o disponibilidad de cierta tecnología. (Cui et al., 2022).

Existen los UAV (vehículos aéreos no tripulados) los cuales sirven como herramienta para el monitoreo de cultivos en gran cantidad, de igual manera existen los UGV (vehículos terrestres no tripulados) como herramienta de inspección de cultivos para otro tipo de aplicaciones, dichas herramientas cumplen con ciertas funciones en general para la agricultura de precisión como la recopilación y análisis de datos, y a partir de estos poder tomar decisiones. (Njoroge et al., 2018).

1.1.8. Aplicaciones agrícolas por medio de imágenes de UAV

Se define a los UAVs como vehículos aéreos que no son tripulados, es decir que se controlan remotamente por medio de aplicaciones y que estos pueden llevar consigo varias cámaras de tipo multiespectral o hiperespectral, y estas sirven para la recolección de imágenes aéreas. (Radoglou-Grammatikis et al., 2020). La Fig. 10 muestra algunos tipos de UAVs.

Fig. 10. Tipos de UAVs

Fuente: (Radoglou-Grammatikis et al., 2020)

Por medio de los UAVs se desarrollan los UAS, los cuales son sistemas aéreos no tripulados y son de gran ayuda para la agricultura de precisión debido a: técnicas de bajo costo en monitoreo del medio ambiente, técnicas de alta resolución espacial y temporal, y en técnicas de adquisición de imágenes. Pues gracias a estas técnicas, los UAVs sirven de

soporte para los agricultores en la adquisición de datos y la toma de decisiones a partir de los mismos. (Boursianis et al., 2022).

Una aplicación de los UAVs en la agricultura es la pulverización de cultivos, esta aplicación fue introducida en los años 90 en Japón, en ese tiempo se utilizaban helicópteros para botar pesticidas en los campos de cultivos, con la evolución de la tecnología ahora se utilizan UAVs adaptados con tanques de pesticidas de hasta 10 litros de capacidad para realizar el mismo trabajo. (Radoglou-Grammatikis et al., 2020).

Según Radoglou-Grammatikis et al. (2020) describe aplicaciones de los UAVs en la agricultura de precisión, las cuales son:

- Uso de sistemas GPS.
- Uso de sistemas GIS.
- Mapeo de producción.
- Cartografía de suelos.
- Tecnologías de sistemas remotos

1.1.9. Detección y manejo de malezas

A partir del mapeo de producción de la agricultura de precisión surge la aplicación de detección y manejo de malezas de los cultivos. Las malezas se definen como plantas o vegetación no deseada que puede causar problemas en los cultivos, hasta matarlos por su necesidad de agua y alimentación, de esta forma se interpone en el crecimiento adecuado de las plantas que son deseadas. Uno de los mayores problemas que existen por las malezas es la cosecha de cultivos, esto se debe a que para el manejo de malezas se utilizan pesticidas o químicos que se deshacen de estos, pero estos pesticidas se los rocía por todo el cultivo, haciendo que todas las plantas pierdan su calidad, a más del gran daño ambiental que se genera. Pues gracias al mapeo de malezas se pueden detectar las plantas no deseadas por medio de UAVs o UGVs y así saber que planta requieres ser excluida y cual no. (Tsouros et al., 2019).

1.2. Estudio de mapeo sistemático

Se realizó la revisión literaria (SMS) previa al desarrollo de un prototipo propuesto para el procesamiento de análisis de actividades agrícolas. Para el desarrollo del SMS se siguió el proceso de la Figura 5 de una revisión sistemática de literatura.

1.2.1. Planificar el SMS

Identificar la necesidad de la revisión

Este estudio de mapeo sistemático tiene como objetivo identificar la tendencia de los lenguajes de programación orientados a GPU que se usan en algoritmos de procesamiento de imágenes, y también, ¿Qué aplicaciones se han hecho? En lo referente al procesamiento de imágenes de actividades agrícolas.

Formular las preguntas de revisión

Se formularon las preguntas de investigación de la presente SMS y se las describe en la tabla 2 junto a su objetivo.

TABLA 2. Preguntas de Investigación.

Nro.	Pregunta de Investigación	Objetivo
P1	¿Cuáles son los lenguajes de programación orientados a GPU más utilizados en el procesamiento de imágenes?	Identificar el lenguaje de programación más utilizado en software de procesamiento de imágenes.
P2	¿Qué tipo de aplicaciones se han desarrollado en actividades agrícolas mediante procesamiento de imágenes con GPU?	Descubrir los tipos de aplicaciones que se han desarrollado en actividades agrícolas mediante procesamiento de imágenes con GPU.

Fuente: Propia

Definir el protocolo de la revisión

Antecedentes

La evolución de la IA hizo que esta se divida en un vasto abanico de materias de estudio, una de estas es la visión por computador y su procesamiento de imágenes, esta tecnología puede automatizar el proceso de clasificación y monitoreo de objetos o seres vivos, disminuyendo el agotamiento que conllevaba realizar dicha actividad por un humano. Dicha técnica de tratado de imágenes ayuda al ser humano a recolectar bastante información útil que sus ojos no puedan detectar a simple vista y por eso muestra un gran potencial para la eliminación de algunas técnicas o tareas de investigación fastidiosas. (Condorí et al., 2020). Esta es una de las razones por la cual la industria agrícola ha venido buscando dichos beneficios computacionales que desea mejorar aspectos de seguridad y productividad de la

agricultura, lidiando temas de aumento poblacional, disminución de tierras para cultivo o disponibilidad de agua. (Franco et al., 2020). En los últimos años el procesamiento de imágenes de actividades agrícolas ha presentado una gran demanda computacional para ser llevadas a cabo en el menor tiempo posible, por esta razón se ha venido observando el gran aumento en el rendimiento del hardware de aceleración de procesamiento gráfico. Inicialmente se realizaba el procesamiento de imágenes con algoritmos ejecutados con la CPU o Unidad de Procesamiento Central, estos algoritmos han ido evolucionando y mostrando una mayor demanda de procesamiento, aunque es cierto que la potencia de procesamiento de las CPU ha aumentado a lo largo de los años, pero se ha quedado atrás para la gran demanda de muchas aplicaciones. (HajiRassouliha et al., 2018). Es ahí cuando entra la GPU, la cual tiene un mejor rendimiento en el procesamiento de imágenes gracias a la cantidad de unidades lógicas y aritméticas que sirven de ayuda para llevar a cabo las operaciones de modelado y renderizado tridimensionales. (Castaño-Díez et al., 2008). Con esta mejora notable de rendimiento gracias a la implementación de la GPU se han dado a conocer varias tecnologías para la creación de algoritmos para el procesamiento de imágenes como son CUDA, Matlab, Python, entre otras, cada una mostrando diferentes beneficios para quien las utilice, sin embargo, es importante saber la tendencia de estos lenguajes de programación que tienen los desarrolladores a escoger para realizar un trabajo adecuado a futuro.

Preguntas de investigación

Con el presente SMS se responde a las preguntas escritas en la tabla 2.

Estrategia de búsqueda

Se realizó la búsqueda de artículos de investigación en las siguientes bases bibliográficas: Science Direct, Taylor & Francis e IEEE Xplorer. La búsqueda está comprendida en un periodo entre 2017 y 2021, es decir los últimos cinco años respecto al año actual.

Se hizo una selección de término principales y alternativos para comenzar con la búsqueda y se los muestra en la tabla 3, todos los términos están en idioma inglés debido a que este es el idioma universal para lo referente a investigación y necesitamos encontrar publicaciones en dicho idioma.

TABLA 3. Términos Principales de Búsqueda.

Términos Principales	Términos Alternativos
GPU	GPU OR Graphics Processing Unit
Programming languages	Programming Languages OR Programming OR Computer Programming OR Software Programming
Image Processing	Image Processing OR Digital Image Processing
Agriculture	Agriculture OR Agronomy

Fuente: Propia

A partir de los términos indicados en la tabla 3 se definieron diferentes cadenas de búsqueda para poder ser utilizadas en cada fuente bibliográfica propuesta y así poder obtener una mejor extracción y cantidad de artículos, estas cadenas se muestran en la tabla 4.

TABLA 4. Cadenas de Búsqueda.

Fuente Bibliográfica	Cadena de búsqueda
Science Direct	(GPU OR Graphics Processing Unit) AND (Programming Languages) AND (Image Processing OR Digital Image Processing) AND (Agriculture)
Taylor & Francis	(Image) AND (processing) AND (gpu) AND (programming) AND (languages) AND (agriculture)
IEEE Xplorer	(GPU) AND (Image Processing) AND (Agriculture)

Fuente: Propia

La búsqueda se la realizó en las fuentes bibliográficas descritas y con las cadenas formadas previamente, se tomaron en cuenta los títulos y los resúmenes de cada publicación.

Criterios de selección de estudios

Se seleccionaron artículos en idioma inglés que hagan referencia a procesamiento de imágenes de actividades agrícolas con lenguajes de programación orientados a GPU y que hayan sido publicados entre los años 2017 y 2021 en congresos o conferencias, y en revistas indexadas.

Procedimiento para la selección de estudios

La selección de estudios se la realizó por medio de la aplicación de un método de selección de estudios que se utiliza en las revisiones literarias llamado: criterios de inclusión y exclusión (Inclusion/Exclusion Criteria). Los criterios de inclusión son todos los elementos que tiene un artículo para ser tomado en cuenta, mientras que los criterios de exclusión son todos los parámetros que presenta un estudio para no ser tomado en cuenta, estos criterios se describen en la sección previa de “Criterios de selección de estudio”. Para esto se leyeron los resúmenes de cada artículo y así se identificó si cada uno cumplía con los criterios establecidos y de esta manera cumplir con la revisión. Si después de leer el resumen existen dudas del método de inclusión o exclusión se procede a leer el artículo completo.

Listas de comprobación y procedimiento para la evaluación de la calidad de los estudios

Como primer filtro para la selección de los artículos se tomará como criterio que cada estudio se lo encuentre en congresos, conferencias o revistas indexadas. Como segundo filtro se tomó en cuenta los cuartiles en los que se encuentran publicadas las revistas indexadas, siendo Q1 y Q2.

Estrategia para la extracción de datos

Se realizó un matriz de metadatos como formulario de extracción de datos de cada artículo de investigación, esta matriz se divide en dos partes, la primera que muestra los metadatos como: título, autores, año, tipo de publicación, etc. Y la segunda parte consta de las respuestas que dan cada estudio a las preguntas de investigación propuestas en la tabla 2 por medio de dimensiones que sirven para la clasificación como se muestra en la tabla 5, gracias a la realización de esta segunda parte se conforma un esquema de clasificación de los artículos.

TABLA 5. Esquema de Clasificación.

Dimensiones	**Categorías**
Lenguaje de Programación	Python, Matlab, Lenguaje C, Java, CUDA, Otros.
Tipo de Aplicación que se ha desarrollado	Precision Agriculture, Image Processing, Computer Vision.

Fuente: Propia

Síntesis de los datos extraídos

Se tomó en cuenta el número o porcentajes de artículos que correspondan a cada dimensión y esta síntesis se la expone en tablas o gráficos para lograr responder una a una las preguntas de investigación haciendo relación con su dimensión correspondiente.

Estrategia de divulgación

El resultado de este SMS se lo describe en el presente trabajo de titulación y sirve de base para el desarrollo de un prototipo con el lenguaje de programación identificado en la revisión literaria propuesta, el mismo que se muestra en el capítulo dos de este texto.

Validar el protocolo de la revisión

El protocolo fue revisado por el tutor del presente trabajo de titulación y los respectivos asesores, para de esta manera asegurar el cumplimiento del objetivo de este SMS.

1.2.2. Realizar el SMS

Identificar la investigación relevante

Para la realización del presente SMS se encontraron 74 artículos cuyo año de publicación está comprendido entre 2017 y 2021. Algunas de las fuentes bibliográficas presentaban limitaciones al momento de realizar la búsqueda de los estudios primarios con la cadena de búsqueda propuesta en la primera fase del proceso del SMS en la sección de la estrategia de búsqueda, por lo que se tuvo que modificar dicha cadena para cada fuente como se lo muestra en la tabla 4 para poder obtener los mismos resultados que se obtuvieron con la cadena original. Para identificar los estudios primarios se analizaron los títulos y resúmenes de cada uno de ellos, y de cada fuente se guardó los metadatos que contienen: título, autores, año de publicación, palabras claves, etc.

Seleccionar los estudios primarios

De los 74 artículos encontrados, se leyó el resumen de cada uno y se procedió a eliminar los que no tenían nada que ver con el procesamiento de imágenes de actividades agrícolas con lenguajes de programación orientados a la GPU, si no se encontraba información relevante en los resúmenes se aplicaba la inclusión y exclusión leyendo el texto completo, realizado lo previo quedaron 40 artículos del primer filtro de selección. Para el segundo y último filtro se revisó el cuartil en el que se ubican las revistas de investigación en las que se publicaron los estudios encontrados, seleccionando los que se encontraban en revistas de cuartil Q1 y Q2, quedando una cantidad de 30 artículos finales.

Evaluar la calidad de los estudios primarios

Para la evaluación de los estudios seleccionados se tomó en cuenta los cuartiles en los que se encontraban las revistas que publicaron dichos artículos, tal cual como se describió en la sección anterior, siendo los cuartiles Q1 y Q2 los criterios de evaluación, de la misma manera se realizó una matriz de metadatos en la cual se describió los datos importantes de cada artículo como son: autores, año de publicación, título, etc. Y en esta matriz también se describía la respuesta a las preguntas de investigación que nos otorgaron cada uno de los estudios seleccionados.

Extraer los datos relevantes

En este punto se comenzó con la extracción de metadatos de los 30 artículos que pasaron los dos filtros de selección, para la clasificación de los estudios se basó en el esquema presentado en la Tabla 5. Todos los artículos seleccionados y los metadatos extraídos de cada estudio primario se encuentran almacenados en una carpeta de OneDrive que se encuentra en una URL en el Anexo A.

Sintetizar los datos extraídos

En esta sección se presenta una síntesis de la información registrada en la matriz de metadatos, la cual se presenta como el formulario de extracción de datos, de esta forma se respondieron las preguntas de investigación mostradas en la Tabla 2. Se presenta también datos cuantitativos en tablas y gráficos para un mejor entendimiento, y se procedió con la explicación e interpretación de los resultados obtenidos.

Lenguaje de Programación (PI1)

De acuerdo con los resultados que se muestran en la Tabla 6, que corresponden a la primera pregunta de investigación: ¿Cuáles son los lenguajes de programación orientados a GPU más utilizados en el procesamiento de imágenes?, se determina que el lenguaje de programación más utilizado en los artículos seleccionados es el lenguaje Python (43%), seguido del lenguaje Matlab (23%) siendo estos dos los más reconocidos en el área de la IA. También se observa la presencia de los lenguajes C (7%) y Java (7%). Los lenguajes que menos utilizan son: TensorFlow (3%), OpenGL Shading Language (3%), PHP (3%) y CUDA (3%). Por último, nos encontramos con una unión de lenguajes que son Matlab + Python (3%) y Java + C++ (3%).

TABLA 6. Resultados por lenguajes de programación.

Lenguajes de Programación	Cantidad	Porcentaje
Python	14	47%
Matlab	7	23%
C	2	7%
Java	2	7%
OpenGL Shading Language	1	3%
PHP	1	3%
CUDA	1	3%
Matlab + Pyhton	1	3%
Java + C++	1	3%
Total	30	100%

Fuente: Propia

Tipo de aplicación que se ha desarrollado (PI2)

En la Tabla 7 podemos apreciar los resultados para la segunda pregunta de investigación: ¿Qué tipo de aplicaciones se han desarrollado en actividades agrícolas mediante procesamiento de imágenes con GPU?, de los cuales podemos concluir que los tipos de aplicación que se han desarrollado en cada estudio primario seleccionado son: “Detección y clasificación de plantas por sus características” (40%) siendo este el tipo de aplicación más desarrollada por los investigadores de estos artículos, seguido de “Recolección de datos de plantas para toma de decisiones” (30%) , en tercer lugar se encuentra “Detección de plagas, enfermedades y malezas en los cultivos” (23%) y con menos frecuencia se encuentra “Detección de líneas de cultivo” (10%).

TABLA 7. Resultados por aplicación desarrollada.

Tipo de aplicación que se ha desarrollado	Cantidad	Porcentaje
Recolección de datos de plantas para toma de decisiones	9	30%
Detección de plagas, enfermedades y malezas en los cultivos	7	23%
Detección y clasificación de plantas por sus características	12	40%
Detección de líneas de cultivo	2	7%
Total	30	100%

Fuente: Propia

1.2.3. Reportar el SMS

Redactar el informe de la revisión

Como informe de la revisión literaria realizada se toma en cuenta toda la sección “Estudio de mapeo sistemático” (1.2) del presente trabajo, en el cual se redactó todo el proceso realizado del SMS propuesto y, de la misma forma, los resultados de la revisión se los redacta en la sección “Obtención y filtración de resultados” (1.3).

1.3. Obtención y filtración de resultados

1.3.1. Interpretación de Resultados

En esta sección se interpretan los resultados cuantitativos descritos anteriormente para dar una mejor explicación y respuesta a las preguntas de investigación. Dichos resultados se los expone en el capítulo 3 de resultados del presente documento.

1.3.2. Resultados Adicionales

Estos resultados son muestra de información extra que se consiguió al realizar el SMS y se los describe en el Capítulo 3 de resultados del presente trabajo de titulación.

CAPÍTULO 2

2. Desarrollo

2.1. Levantamiento de requisitos

Para un adecuado desarrollo de software se debe tener un conocimiento bien claro de lo que se debe construir, y de lo que realmente quiere nuestro cliente o potencial cliente. Para poder cumplir y satisfacer esas necesidades existe un proceso llamado levantamiento o licitación de requisitos, con esto el desarrollador sabrá exactamente lo que se debe hacer y lo que el cliente demanda para su software, y así, evitar en su mayoría, errores y cambios que se pueden obtener. Un requisito es una declaración que expresa una necesidad que conlleva condiciones expuestas por la parte interesada, estos requisitos vienen en distintas categorías que se las puede clasificar en varias dificultades. (ISO et al., 2011).

La persona encargada de levantar los requisitos que tiene el cliente debe tomar en cuenta ciertas características o puntos que ayudarán al buen desarrollo del sistema y al buen entendimiento de este por parte del programador. En la Tabla 8 se muestran las características que cada requerimiento individual debe cumplir:

TABLA 8. Características de un requisito individual.

Característica	Descripción
Necesario	El requisito describe una característica o factor de calidad que, si este se remueve, el sistema mostrará un fallo o deficiencia.
Implementación Gratis	El requisito expresa lo que se debe hacer mas no como se debe hacer o implementar.
Inequívoco	El requisito establecido solo debe interpretarse en un solo sentido.
Consistente	El requisito no tiene ningún conflicto con otros requerimientos.
Completo	El requisito no necesitará ampliarse debido a que la descripción de este satisface en su totalidad la necesidad del cliente.

Singular	Solo se debe declarar un solo requisito y no hacer uso de conjunciones.
Factible	El requisito debe ser alcanzable, sin necesidad de utilizar una mayor tecnología y con riesgos aceptables.
Trazable	Al requisito se lo puede rastrear desde su origen hasta su implementación.
Verificable	El requisito se podrá verificar en el sistema demostrando que cumple con la necesidad del cliente.

Fuente: (ISO et al., 2011)

También se debe tomar en cuenta que algunas características es mejor adoptarlas a un conjunto o grupo de requerimientos, en la Tabla 9 se muestran dichas características que nos ayudan a llegar a soluciones más factibles:

TABLA 9. Características del grupo de requisitos.

Característica	**Descripción**
Completo	Los requisitos no necesitarán ampliarse debido a que contienen todo lo pertinente a la definición del sistema.
Consistente	El grupo de requisitos no debe mostrar requerimientos individuales contradictorios.
Asequible	Todo el grupo de requisitos debe satisfacerse con una solución factible.
Delimitado	Los requisitos deben mantener el enfoque de la solución pretendida sin salirse de lo necesario para satisfacer al cliente.

Fuente: (ISO et al., 2011)

Los requisitos se dividen en dos tipos, los requisitos funcionales y los requisitos no funcionales.

Requisitos Funcionales

Los requisitos funcionales son servicios fundamentales que otorga el software, los cuales muestran cambios en el sistema dependiendo la interacción de entradas específicas. (Alashqar, 2022).

Requisitos No Funcionales

Los requisitos no funcionales son aquellos requisitos del sistema, tanto software como hardware, que indican como se debe desarrollar, dar soporte o la forma en que se debe poner en producción, y para estos se debe tener en cuenta la calidad del software, el entorno en el cual se va a desplegar y los procesos que intervienen en el desarrollo y mantenimiento del producto de software. (Alashqar, 2022).

Si bien es cierto, los requisitos funcionales son considerados los más fundamentales para el desarrollo y buen funcionamiento del software, los requisitos no funcionales a veces pueden llegar a ser más importantes que los funcionales. (Alashqar, 2022).

La licitación de requisitos se la define como un proceso interactivo entre la parte interesada y quien realiza la licitación, es por esto que existen diversas técnicas para identificar requisitos, algunas de estas se las describe en la Tabla 10.

TABLA 10. Técnicas de identificación de requisitos.

Técnicas
Lluvia de ideas o Brainstorming.
Entrevistas o cuestionarios.
Observación del entorno de trabajo (Patrones).
Revisión de la documentación técnica.
Análisis de mercado.
Modelamiento, simulaciones, prototipos.
Benchmarking de procesos y sistemas.
Técnicas de análisis organizacional.

Fuente: (ISO et al., 2011)

Como primer paso para la construcción del prototipo, y como el más fundamental, se levantaron los requisitos funcionales y no funcionales y mediante estos se desarrolló un prototipo que nos permite procesar imágenes de líneas de cultivos para la detección de malezas.

Para el levantamiento de requisitos se recurrió al uso de una entrevista con la parte interesada, se le realizaron ciertas preguntas que nos ayudaron con información importante para el desarrollo, esta entrevista se presenta en el Anexo B.

Se utilizó el formato de historia de usuario que se muestra en la Tabla 11 para la definición de los requisitos tanto funcionales como no funcionales del prototipo.

TABLA 11. Formato de historia de usuario.

Historia de Usuario		
Código	**Nombre**	**Prioridad**
RF-XXX / RNF-XXX (Número de requisito)	Nombre del requisito funcional o no funcional.	Baja/Media/Alta
Descripción	Descripción sobre el requisito funcional o no funcional.	
Proceso	Proceso por el cual se cumplirá con lo descrito en el requisito funcional o no funcional.	
Nota Aquí se describe una observación a tomar en cuenta para este requisito funcional o no funcional.		

Fuente: (Criollo, 2021)

2.1.1. Requisitos funcionales

A partir de la entrevista que se encuentra en el Anexo B, se identificaron los requisitos funcionales para el desarrollo del prototipo, estos requisitos se los describe en las Tablas 12 – 15 como historias de usuario.

TABLA 12. Requisito funcional 1. Ingreso de imagen.

Historia de Usuario		
Código	**Nombre**	**Prioridad**
RF-001	Ingreso de imagen	Alta
Descripción	Como usuario requiero de una vista en la cual se ingrese la imagen a procesar.	
Proceso	Se mostrará la vista principal en la cual el usuario de clic a un botón para que se abra una venta en la cual debe escoger la imagen que desea y al dar clic en aceptar esta se procesará.	
Nota		

Fuente: Propia

TABLA 13. Requisito funcional 2. Detección de malezas.

Historia de Usuario		
Código	**Nombre**	**Prioridad**
RF-002	Detección de malezas	Alta
Descripción	Como usuario requiero de la detección de malezas de líneas de cultivo mediante visión por computador.	
Proceso	Una vez ingresada la imagen, el algoritmo procesará la imagen y dará como resultado la detección de malezas en cuadrados rojos con etiquetas de “maleza”, todo esto mediante visión por computador.	
Nota		

Fuente: Propia

TABLA 14. Requisito funcional 3. Formato de la imagen.

Historia de Usuario		
Código	**Nombre**	**Prioridad**
RF-003	Formato de la imagen	Alta
Descripción	Como usuario requiero que la imagen a procesar sea solo de formato JPG.	
Proceso	En la vista principal, al elegir la imagen a procesar se validará que la extensión de la imagen sea solo JPG.	
Nota		

Fuente: Propia

TABLA 15. Requisito funcional 4. Dimensiones de la imagen.

Historia de Usuario		
Código	**Nombre**	**Prioridad**
RF-004	Dimensiones de la imagen	Media
Descripción	Como usuario requiero que la imagen a procesar tenga las dimensiones de 960px * 540px.	
Proceso	La imagen ingresada se redimensionará a las siguientes dimensiones: 960px * 540px, tanto para la anchura como para la altura respectivamente, y de esta manera poder procesarla de mejor manera.	
Nota		

Fuente: Propia

2.1.2. Requisitos no funcionales

A partir de la entrevista que se encuentra en el Anexo B, se identificaron los requisitos no funcionales para el desarrollo del prototipo, estos requisitos se los describe en las Tablas 16 – 18 como historias de usuario.

TABLA 16. Requisito no funcional 1. Lenguaje de programación.

Historia de Usuario		
Código	**Nombre**	**Prioridad**
RNF-001	Lenguaje de programación	Alta
Descripción	El lenguaje de programación a usarse para el desarrollo del algoritmo es Python.	
Proceso	El lenguaje de programación Python es el escogido debido al resultado que nos muestra el mapeo de estudio sistemático y con este se desarrollará el algoritmo.	
Nota		

Fuente: Propia

TABLA 17*. Requisito no funcional 2. Altura de la imagen.*

Historia de Usuario		
Código	**Nombre**	**Prioridad**
RNF-002	Altura de la imagen	Alta
Descripción	La altura desde donde se tomó la fotografía hasta la superficie debe ser de 15 metros.	
Proceso	Las imágenes que ingresan al algoritmo a ser procesadas tendrán una altura de 15 metros desde donde se tomó la fotografía hasta la superficie donde se encuentran las líneas de cultivo.	
Nota		

Fuente: Propia

TABLA 18*. Requisito no funcional 3. Tipo de cultivo a analizar.*

Historia de Usuario		
Código	**Nombre**	**Prioridad**
RNF-003	Tipo de cultivo a analizar	Alta
Descripción	Los cultivos que se muestran en las imágenes deben ser solo de maíz.	
Proceso	Las imágenes que ingresan al algoritmo a ser procesadas deben ser solo de maizales o cultivos de maíz.	
Nota		

Fuente: Propia

2.2. Diseño del prototipo

En la etapa de diseño se definió toda la estructura y flujo del algoritmo, esto se lo realiza para disminuir o evitar la mayor cantidad de errores posibles que puedan aparecer al codificar o en su caso, implementar dicho algoritmo.

El diseño se lo describe como el proceso por el cual se define la arquitectura, las interfaces y demás características de un producto de software o sistema informático. Al resultado de todo este proceso se lo considera como diseño de software. (Bourque et al., 2014).

En esta etapa se formularon varias ideas de posibles soluciones que satisfagan los requisitos levantados y todo el proceso que el algoritmo debe cumplir, de entre todas esas ideas se escogió una que es la que se desarrolló e implementó.

Algoritmo

Se define como una serie o lista de pasos bien definidos la cual nos permite solucionar un problema en específico. Pasa desde una entrada (estado inicial), y después de cumplir operaciones propuestas, este llega a una salida (estado final). (Juganaru, 2014).

Un algoritmo es como una receta para hornear un pastel, se necesita de ciertos ingredientes y sobre todo se deben cumplir una lista de pasos para lograr hornear el pastel

que se quiere. Los algoritmos se pueden representar como: Lenguaje natural, pseudocódigo, diagramas de flujo y programas o software. Cada representación previamente descrita se la realiza dependiendo del grado o nivel de descripción del algoritmo. (Juganaru, 2014).

Los niveles de descripción son:

- Descripción de alto nivel.
- Descripción formal.
- Implementación

Para este último, primero se deben cumplir con descripciones de alto nivel o formales.

Todo algoritmo debe pasar de un lenguaje natural a un lenguaje de programación para poder ser ejecutado en un programa o software. Para realizar esta transformación se debe cumplir la siguiente serie de pasos:

- Edición
- Compilación
- Enlazado

La edición hace referencia al escoger un editor de texto de nuestra preferencia para posteriormente escribir código con el lenguaje de programación deseado. Seguido de esto viene la compilación, en este paso dicho código se transforma en instrucciones que logra entender el computador. Por último, el enlazado, aquí se crea un programa o un ejecutable por medio de librerías y código fuente, este es el resultado de todo el proceso de transformación. (Juganaru, 2014).

Todo algoritmo tiende a tener errores antes o después de la compilación, y debido a esto se puede volver al paso de edición, para resolver cualquier problema presentado en el código fuente y seguir con el proceso hasta tener el ejecutable, dicho proceso se lo presenta en la Fig. 11.

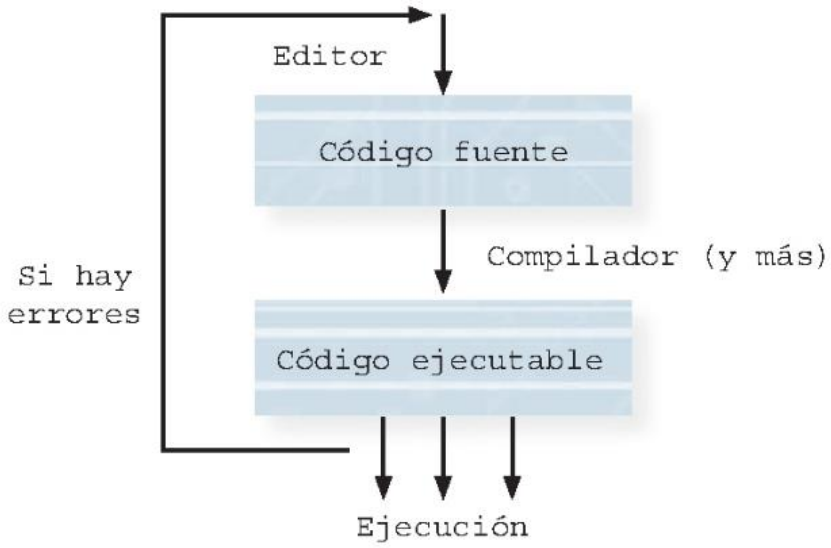


Fig. 11. Transformación de un programa.

Fuente: (Juganaru, 2014)

Diagrama de flujo

Antes de pasar a la codificación del algoritmo, primero se debe realizar un diagrama de flujo del mismo, estos diagramas son un intermediario entre el lenguaje natural y el lenguaje de máquina. Un diagrama tiene la ventaja de expresar la información de mejor manera y con mucha claridad, este se lee de arriba hacia abajo, haciendo que cada tarea forme una lista de pasos a realizar. (Juganaru, 2014).

Un diagrama de flujo debe cumplir con ciertas reglas, las cuales son:

- El diagrama debe mostrar flechas que indican el sentido de la lectura.
- Debe comenzar y terminar con componentes de inicio y fin respectivamente.
- Las entradas y salidas de datos se representan con un paralelogramo.
- Las tomas de decisión o condicionales se los representa con rombos.

Previo al desarrollo del prototipo planteado, se realizó un diagrama de flujo del algoritmo que sirvió con la detección de malezas en líneas de cultivo, el mencionado diagrama se lo presenta en la Fig. 12.

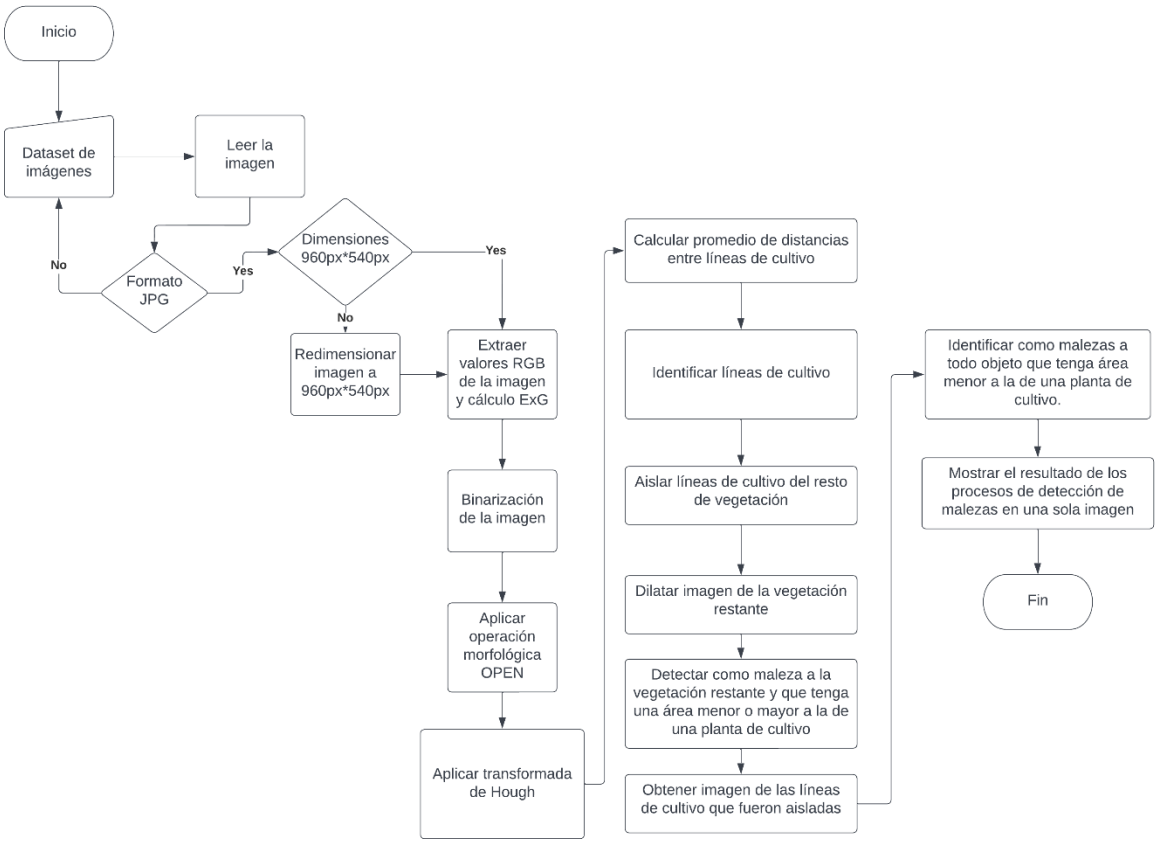


Fig. 12. Diagrama de flujo del algoritmo.

Fuente: Propia

Como se puede visualizar en la Fig. 12, después de dar inicio al diagrama se siguió la siguiente lista de pasos:

- Ingresa una imagen al programa.
- El programa lee la imagen.
- El programa verifica el formato de la imagen, si es un formato JPG puede seguir el flujo, caso contrario pide una nueva imagen.
- El programa verifica las dimensiones de la imagen, si las dimensiones son 960px * 540px puede seguir el flujo, caso contrario redimensiona la imagen.
- Se extraen los valores de los canales RGB de la imagen y se procede a hacer un cálculo “ExG” con los mismos.
- Se binariza la imagen resultante del cálculo anterior con el método de OTSU.
- Se reduce el ruido de la imagen binarizada.
- Se aplica la transformada de Hough para obtener los puntos de coordenadas de las líneas a identificar.
- Se calcula el promedio de las distancias entre los puntos obtenidos.

- Se identifican las líneas representativas (líneas de cultivo) de la imagen con los valores obtenidos anteriormente y que cumplan con el promedio de la distancia entre líneas.
- Se aíslan las líneas de cultivo identificadas del resto de la vegetación de la imagen.
- Se aplica dilatación a la imagen de la vegetación restante.
- A la vegetación restante en la imagen se la identifica como maleza.
- Obtener una imagen con las líneas de cultivo que fueron aisladas previamente.
- En la misma imagen en la que se detectó la maleza anteriormente, se procede a identificar nuevos objetos como malezas con la condición de que sus áreas sean menores a las de una planta de cultivo.
- A la maleza identificada en las tareas anteriores se la muestra en una imagen final como resultado del algoritmo.

2.3. Desarrollo del prototipo

Para cumplir con los requisitos levantados se desarrolló el siguiente algoritmo que nos permite detectar malezas en las líneas de cultivo en imágenes que ingresan y pasan por dicho algoritmo.

En esta etapa de desarrollo se utilizaron librerías que facilitan con visualizaciones, conversiones y cálculos matemáticos que impliquen a la imagen ingresada al programa. Las librerías utilizadas son las siguientes:

- Cv2 de OpenCV, esta librería es de código abierto y contiene una gran variedad de algoritmos de visión por computador, y está basada en lenguaje C++. (OpenCV, 2022). La Fig. 13 muestra el proceso de lectura de imágenes con dicha librería.

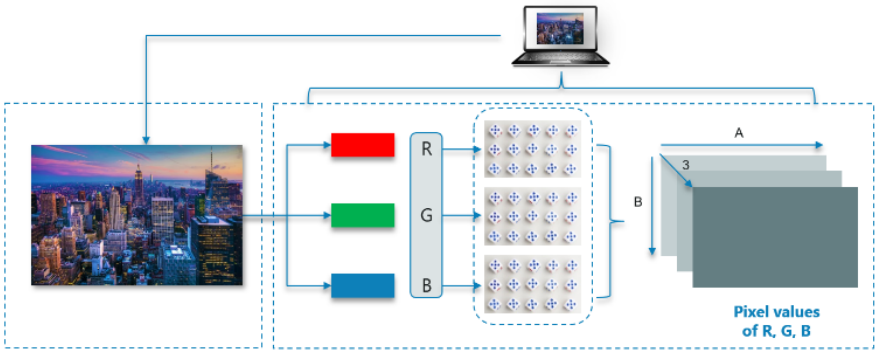


Fig. 13. Proceso de lectura de imagen con OpenCV.

Fuente: (Rao, 2021).

- Numpy, es una librería que brinda objetos de arreglos multidimensionales, así como derivados de dicho objeto y una gran rapidez en cálculos y operaciones con arreglos. Numpy es considerada una librería fundamental para la computación científica. (NumPy, 2022). En la Fig. 14 se muestran los usos que se le puede dar a dicha librería.

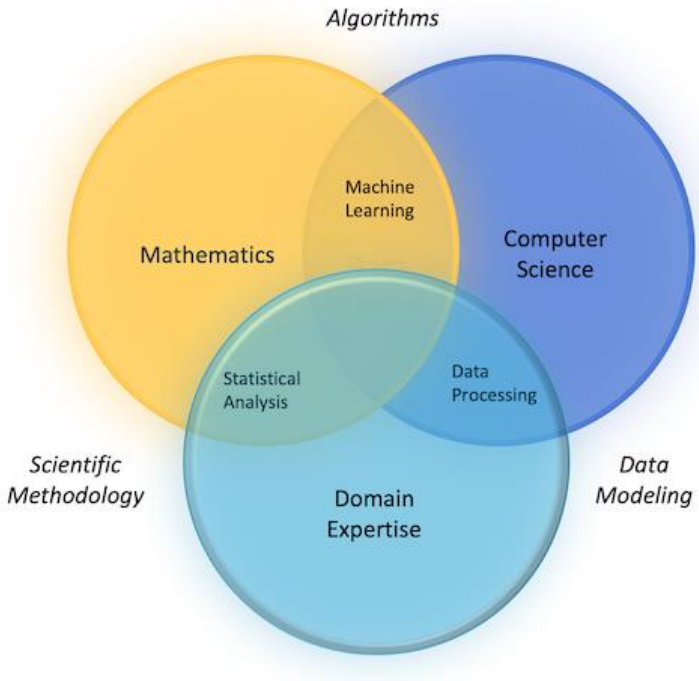


Fig. 14. Usos de la librería NumPy.

Fuente: (NumPy, 2022).

- skimage.morphology de scikit-image, esta es una librería utilizada para procesamiento de imágenes, su módulo morphology nos brinda las funciones dilation y disk que se utilizaron en el algoritmo de detección de malezas en líneas de cultivo. (scikit-image, 2022). En la Fig. 15 se muestran algunas de las funcionalidades que nos brinda dicha librería.

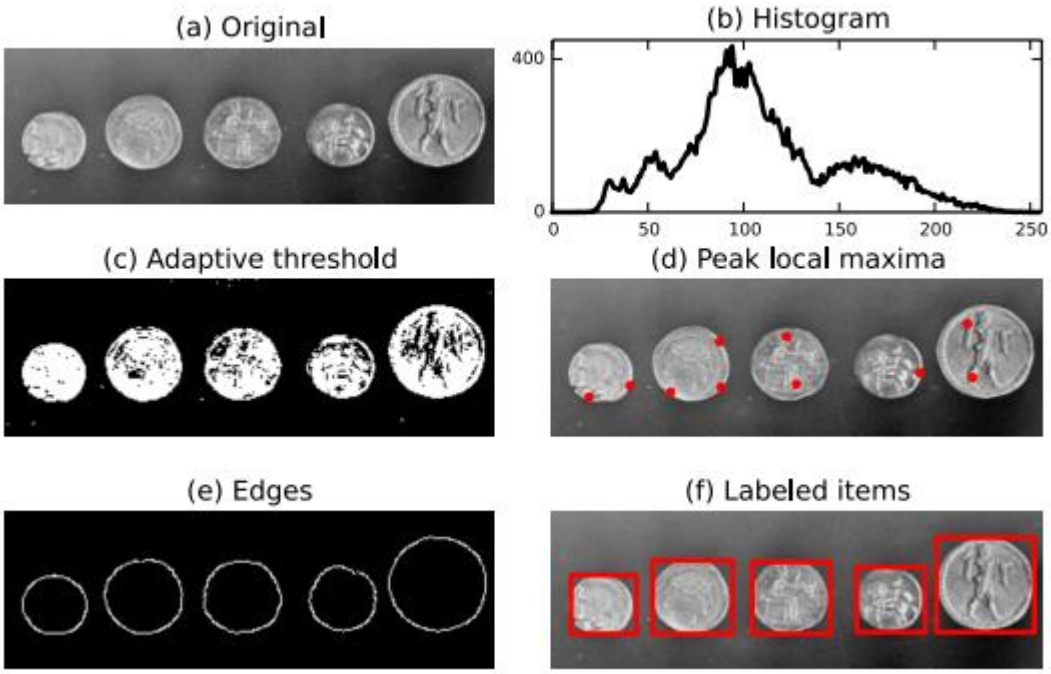


Fig. 15. Funcionalidades de la librería scikit-image.

Fuente:(van der Walt et al., 2014).

- copy de Python, esta librería es desarrollada por Python como una de sus librerías estándar y se la utiliza para realizar una copia a un objeto o un arreglo. (Python, 2022).
- os de Python, es una librería desarrollada por Python la cual es una herramienta para extraer la raíz y la extensión del “path” de una imagen gracias a las funcionalidades del sistema operativo. (Python, 2022).
- tkinter de Python, librería que es de gran ayuda para el desarrollo de interfaces gráficas dentro de Python con el kit de herramientas de TK, esta es interfaz por defecto de dicho lenguaje. (Python, 2022).

La Fig. 16 muestra ciertas características del lenguaje de programación Python.

Fig. 16. Características de Python.

Fuente: (Mahnken, 2021).

- PIL de Pillow, esta librería fue desarrollada para poder utilizar características de procesamiento de imágenes en el algoritmo, de ahí salen sus siglas PIL (Python Imaging Library). (Pillow, 2022). En la Fig. 17 se muestra el flujo de procesamiento de imágenes con dicha librería.

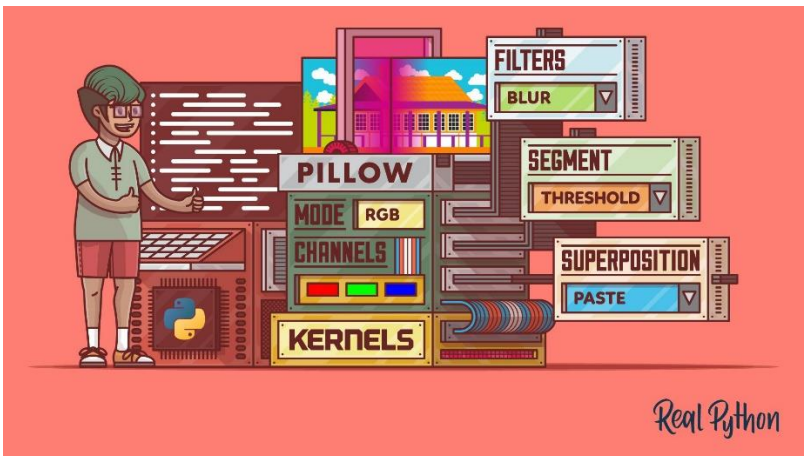


Fig. 17. Flujo de procesamiento de imágenes con Pillow.

Fuente: (Gruppetta, 2022).

La manera en que se deben importar las librerías es la siguiente:

```
# Librerías para GUI
from tkinter import *
from tkinter import filedialog
from PIL import Image, ImageTk
import imutils


# Librerías para procesamiento de imágenes
import cv2
import numpy as np;
from skimage.morphology import dilation, disk;
import copy
import os
```

Tal y como se explicó en la etapa de diseño, el algoritmo debe seguir los siguientes pasos:

Ingreso, lectura y verificaciones de una imagen al programa

Primero se desarrolló una interfaz gráfica para el prototipo la cual contiene un botón que abre el explorador de archivos para elegir la imagen que se desee procesar en el algoritmo, la imagen seleccionada primero debe pasar por dos validaciones antes de continuar con el flujo del algoritmo.

En esta sección el algoritmo lee la imagen que se subió previamente con la librería cv2 y su función imread(). Primero se verificó que la extensión de la imagen sea “.JPG” y así poder continuar con el algoritmo. Segundo, se verificaron las dimensiones de la imagen leída, si estas son iguales a 540 de alto y 960 de ancho pues se continúa con el algoritmo, caso contrario, debido a que las dimensiones pueden ser muy grandes se redimensiona la imagen con la librería cv2 y su función resize() para así poder trabajarla de mejor manera. Se muestra la porción de código que evidencia lo mencionado anteriormente:

```
path_image = filedialog.askopenfilename(filetypes = [
        ("image", ".JPG")
```

```
        ])
    image = cv2.imread(path_image)
    copyImg = copy.copy(image)
root, extension = os.path.splitext(path_image)
    if(extension=='.JPG'):
        # Verificar Dimensiones y redimensionar
        height, width = img.shape[:2]
        if(height!=540 and width!=960):
            copyImg = cv2.resize(copyImg, (960,540))


        deteccionMalezas(copyImg)
    else:
        print("La imagen ingresada no tiene una extensión .JPG")
```

Dentro de la función “deteccionMalezas” realizamos dos copias de la imagen leída con la librería copy y su función copy(), las cuales fueron de mucha ayuda más adelante. Y para finalizar, se muestra la imagen con la librería cv2 y su función imshow(), también ocupamos la función waitKey() para que la imagen que se muestra se suspenda y no desaparezca de la pantalla hasta que no se aplaste una tecla, y finalizamos con la función destroyAllWindows() para que al aplastar dicha tecla se destruyan todas las ventanas que se muestran en ese momento. Lo mencionado anteriormente se lo ejecutó con las siguientes líneas de código:

```
copyImage1 = copy.copy(img)
copyImage2 = copy.copy(img)
cv2.imshow('Imagen ingresada y redimensionada', img)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

Al verificar la extensión y las dimensiones de la imagen se pretende cumplir con los requisitos funcionales 3 y 4 que se muestran en la Tabla 14 y 15. La imagen ingresada se visualiza en la Fig. 18.

Fig. 18. Imagen ingresada y redimensionada.

Fuente: Propia

Extraer canales RGB y cálculo ExG

Se procedió a separar la imagen en los canales RGB (Red, Green, Blue) y después utilizar sus valores para transformar la imagen a escala de grises. Se utilizó el siguiente código para extraer los canales RGB de la imagen:

```
blue, green, red = cv2.split(img)
```

De la librería OpenCV se utilizó la función cv2.split() la cual divide a una imagen en los 3 canales RGB y de esta forma se obtuvieron las siguientes imágenes para el canal red (R) ver Fig. 19, para el canal green (G) ver Fig. 20 y para el canal blue (B) ver Fig. 21.

- Canal red (R)

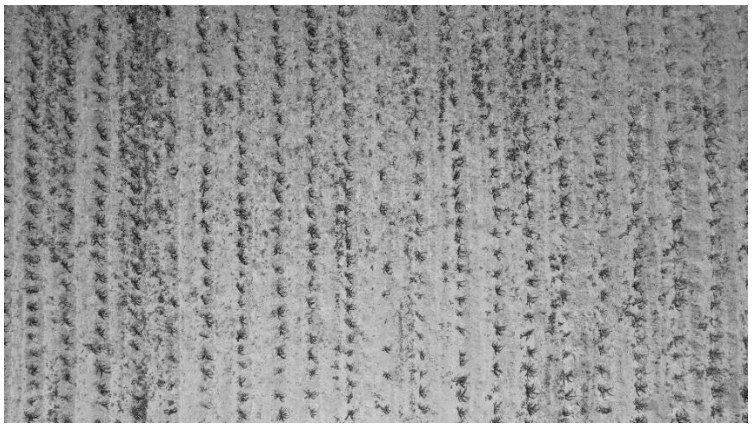


Fig. 19. Canal red.

Fuente: Propia

- Canal green (G)

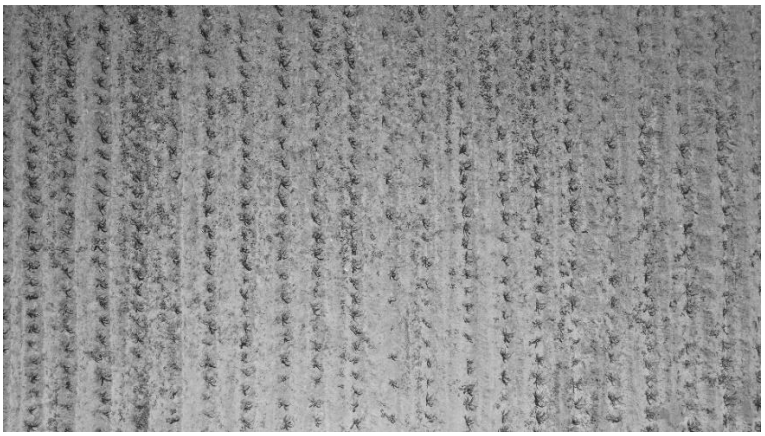


Fig. 20. Canal green.
Fuente: Propia

- Canal blue (B)

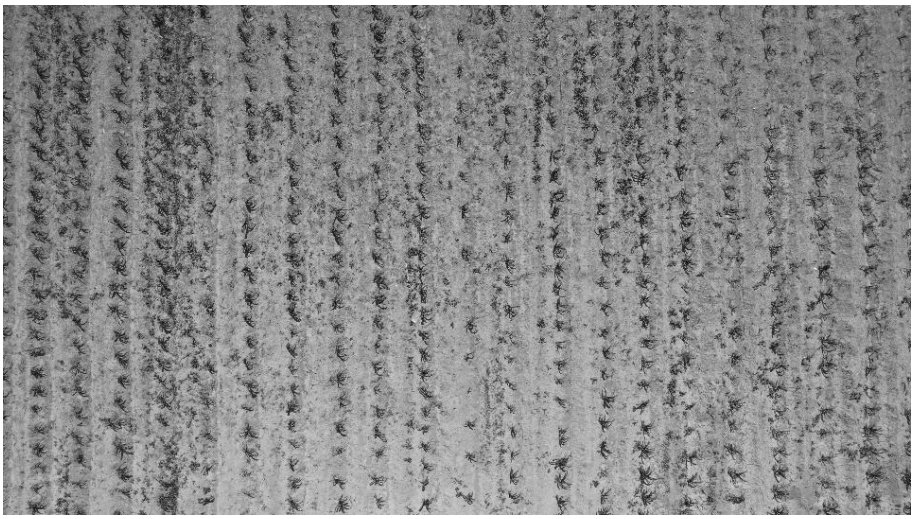


Fig. 21. Canal blue.
Fuente: Propia

Después de haber obtenido cada canal de la imagen se realizó el cálculo “ExG” para obtener la imagen en escala de grises, esto se obtuvo con las siguientes líneas de código:

```
exg = 2*(red)-(green)-(blue)

cv2.imshow("Imagen escala de grises", exg)

cv2.waitKey(0)

cv2.destroyAllWindows()
```

Y se obtuvo una imagen a escala de grises que se visualiza en la Fig. 22 con la cual se siguió con el procesamiento correspondiente.

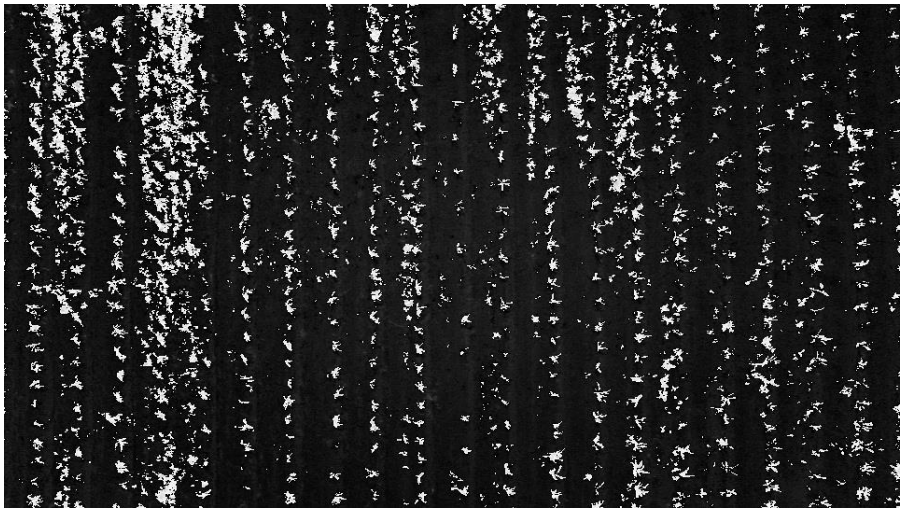


Fig. 22. Imagen escala de grises.
Fuente: Propia

Binarizar la imagen

Una vez obtenida la imagen en escala de grises se realizó el proceso de binarización de la misma por medio del método de OTSU.

Binarizar es el proceso de dividir una imagen en blancos y negros, es decir, este es un método de procesamiento de imágenes, el cual clasifica el valor de cada píxel de la imagen en dos grupos: blanco (255) y negro (0), todo esto ocupando una intensidad de umbral (threshold) específica.

La binarización se calcula de la siguiente manera:

$$B(x,y) = \begin{cases} 0, & if\ I(x,y) < T(x,y) \\ 1, & if\ I(x,y) \geq T(x,y) \end{cases}$$

Donde: $T(x,y)$ es el valor del umbral, $I(x,y)\ \in [0,1]$ es la intensidad del píxel en el punto (x,y) de una imagen (I). (Kim et al., 2021).

La umbralización o Thresholding, es el proceso mediante el cual su identifica un umbral adecuado para realizar una binarización óptima de una imagen, se relacionan las características de gris y también se calculan umbrales de gris de la imagen. (Ma & Yue, 2022). Este proceso se divide en 2 tipos: umbralización global y umbralización local, el tipo global utiliza un solo valor de intensidad y es la mejor opción para imágenes que fácilmente se pueden dividir en blancos y negros, el tipo local utiliza una dimensión específica de la vista (x*x) para realizar una segmentación parcial. (Kim et al., 2021).

El método de OTSU o método de segmentación del umbral de varianza máxima interclase, escoge automáticamente el valor del umbral con una estructura simple y un procesamiento rápido, dicho umbral de segmentación lo escoge por medio de la maximización o minimización de la varianza interclase de la imagen, siendo esta de dimensión $(m\ x\ n)$. (Ma & Yue, 2022).

Para la binarización de la imagen se utilizó la función cv2.threshold() con cv2.THRESH_BINARY como tipo de "thresholding" o umbralización y conjunto a cv2.THRESH_OTSU que nos otorga un valor de "threshold" o umbral automáticamente. La función cv2.threshold() pide de parámetros:

- Primer parámetro, la imagen en escala de grises.
- Segundo parámetro, el valor de umbral, en este algoritmo se utilizó el valor de 0 (cero) debido a que ocupamos el valor que determina la binarización de OTSU con cv2.THRESH_OTSU.
- Tercer parámetro, es el valor máximo de pixeles que superan al umbral, el valor por defecto es 255.
- Cuarto parámetro, es el tipo de umbralización, la librería OpenCV otorga varios tipos, pero en este algoritmo se utilizó cv2.THRESH_BINARY y a esta se le sumó cv2.THRESH_OTSU para obtener un valor de umbral automático.

Para obtener la imagen binarizada se utilizaron las siguientes líneas de código:

```
umbral,threshold =
cv2.threshold(exg,0,255,cv2.THRESH_BINARY+cv2.THRESH_OTSU)
cv2.imshow("Binarizacion con OTSU ",threshold)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

En la Fig. 23 se visualiza la imagen binarizada resultante de haber ejecutado las líneas de código anteriormente descritas.

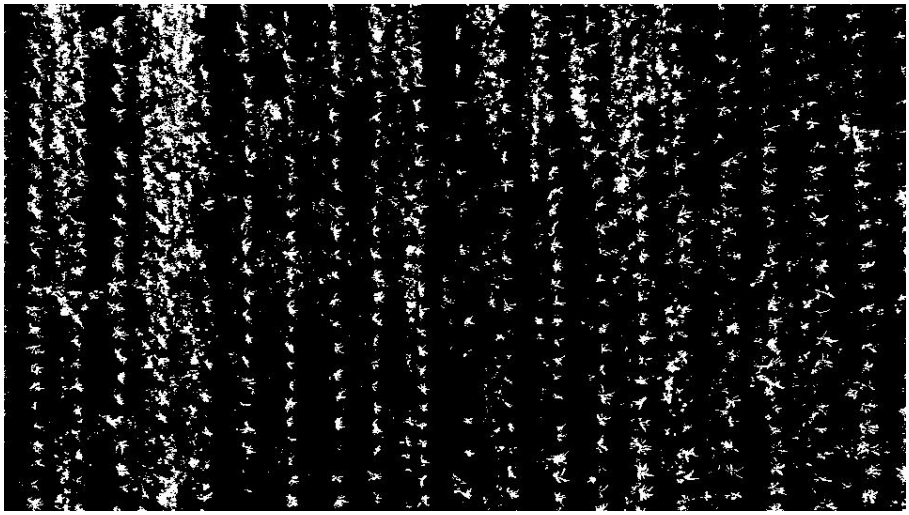


Fig. 23. Binarización con OTSU.

Fuente: Propia

Reducción de ruido de la imagen

Las operaciones morfológicas son algoritmos matemáticos utilizados para el procesamiento de imágenes, estos brindan herramientas que facilitan dicho procesamiento dando mejor calidad a imágenes corruptas, sucias o que no han sido tratadas previamente. (Mardiris & Chatzis, 2016).

Estas operaciones son métodos fáciles de usar que brindan ayuda con el análisis de estructuras geométricas e imágenes, para lograr esto realizan cálculos de píxel a píxel con cierto grado de paralelismo, de esta manera los algoritmos de procesamiento de imágenes necesitan de la ayuda de las operaciones morfológicas. (Zhou et al., 2022).

En esta etapa del algoritmo se redujo el ruido presente en la imagen binarizada que se obtuvo en el paso anterior, para dicha reducción de ruido se ocupó la operación morfológica “OPEN”. Dicha operación OPEN es una unión de las operaciones EROSION y DILATION, aplicando una después de otra, de esta forma se elimina el ruido existente en la imagen y después se suavizan los bordes dentro de la misma. (Zhou et al., 2022).

La librería OpenCV nos brinda la función cv2.morphologyEx() y conjunto a la operación cv2.MORPH_OPEN esta nos ayuda con la reducción de ruido de una imagen. La función pide los siguientes parámetros:

- Primer parámetro, la imagen binarizada.
- Segundo parámetro, la operación morfológica, para la reducción de ruido se utilizó la operación cv2.MORPH_OPEN.

- Tercer parámetro, es el elemento estructural o “kernel”, este se lo puede crear manualmente con la librería Numpy y su función np.ones(), en el presente algoritmo ocupamos un kernel de 2x2.

Para recudir el ruido de la imagen binarizada se utilizaron las siguientes líneas de código:

```
kernel = np.ones((2,2),np.uint8)
opening = cv2.morphologyEx(threshold, cv2.MORPH_OPEN, kernel)
cv2.imshow("Reduccion de ruido", opening)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

En la Fig. 24 se visualiza la imagen resultante de ejecutar el código anteriormente descrito, se obtuvo una imagen de las líneas de cultivo más limpias, con menos ruido.

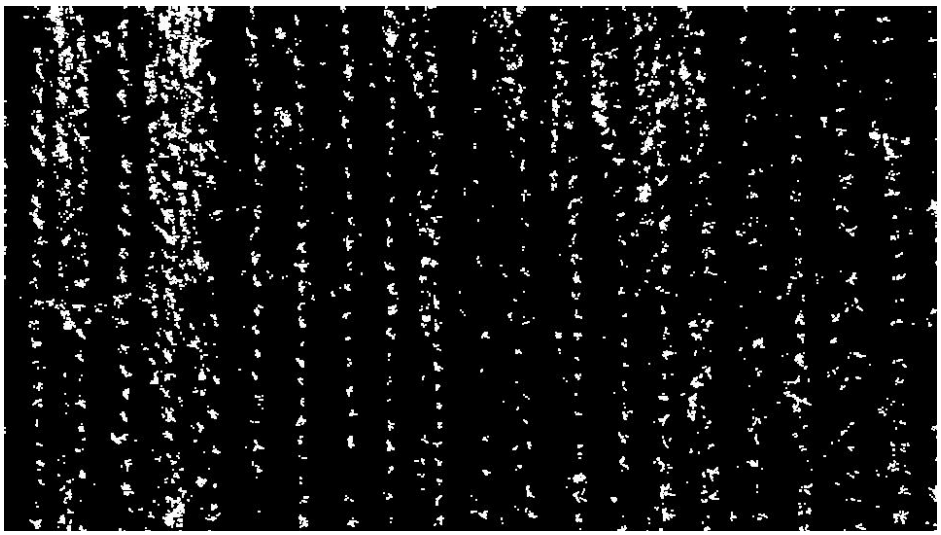


Fig. 24. Reducción de ruido.

Fuente: Propia

Transformada de Hough

La transformada de Hough es un método para detectar formas en una imagen, estas formas deben poder ser expresadas matemáticamente. Este método es bastante utilizado en el procesamiento de imágenes y en la visión por computador. (Han & Qu, 2020).

Para detectar una línea la trasformada de Hough mapea una línea del espacio de la imagen general en un punto en el espacio de los parámetros, de esta forma se busca el valor de un pico en el espacio de parámetros. (Tao Ma & Jie Ma, 2016).

Para aplicar la transformada de Hough se necesita representar una línea con los parámetros r y θ, en la Fig.25 se visualiza la representación de una línea en el plano cartesiano incluyendo los parámetros r y θ.

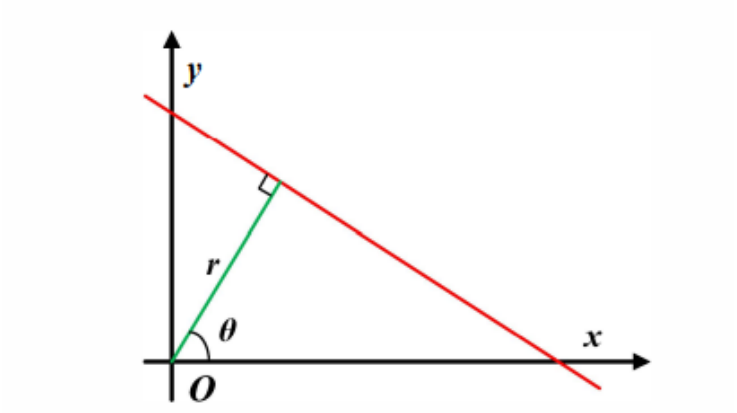


Fig. 25. Representación de una línea con parámetros r y θ.

Fuente: (Tao Ma & Jie Ma, 2016).

Basados en la imagen anterior, la línea se representa matemáticamente con la siguiente fórmula:

$$y = kx + b$$

Ahora se debe representar la ecuación anterior con los parámetros de r y θ, donde:

$$k = -\frac{cos\theta}{sin\theta}\ ;\ b = \frac{r}{sin\theta}$$

De esta manera tenemos la siguiente fórmula en función del parámetro r:

$$r = x\ cos\theta\ + \mathrm{y}\ \sin\theta$$

Cualquier punto de pares ordenados (x, y) que pase por una misma línea comparten los mismos parámetros r y θ, de esta forma un punto de la forma (r, θ) en el espacio puede representar una línea gracias a la transformada de Hough. (Tao Ma & Jie Ma, 2016).

La librería OpenCV nos brinda la función cv2.HoughLines() que aplica la transformada de Hough y esta nos devuelve un vector de pares (θ, rθ), este vector nos sirvió para dibujar las líneas detectadas en la imagen como se explica en el siguiente punto de identificación de líneas representativas. La función descrita anteriormente pide los siguientes parámetros:

- Primer parámetro, imagen en escala de grises y de preferencia una imagen detectados los bordes de cada objeto.
- Segundo parámetro, el valor de “rho” o el término “r” en pixeles.

- Tercer parámetro, el valor de “theta” o el término “θ” en radianes.
- Cuarto parámetro, el mínimo número de intersecciones que se van a detectar por línea.

Para la aplicación de la transformada de Hough se ejecutó la siguiente línea de código:

```
lines = cv2.HoughLines(opening, 7, np.pi, 200)
```

Antes de pasar al punto de identificar las líneas representativas, se procedió a sacar una copia de la imagen reducida el ruido (opening) que sirvió para el proceso de aislar las líneas de cultivo.

Para sacar la copia de una imagen se utilizó la función copy() de la librería copy, la cual nos devuelve una copia del objeto entregado para que no sea afectado en un futuro como en una asignación de variables o por referencia. La función se la utiliza como indica la siguiente línea de comando:

```
copyPlantas = copy.copy(opening)
```

Calcular promedio de distancias

En este punto del algoritmo se calcularon las distancias existentes entre cada punto detectado por la transformada de Hough, este promedio fue de ayuda para una mejor identificación de las líneas de cultivo en la imagen ingresada.

Primero se ordenó el arreglo de valores respecto al eje X (X1), la función de cv2.HoughLines() devuelve un arreglo de puntos al cual se lo recorrió para calcular los valores de las coordenadas X1 de cada línea y se los añadió a un nuevo arreglo que posteriormente se lo ordenó ascendentemente con la función np.sort() de la librería numpy. Lo descrito anteriormente se lo logró ejecutando la siguiente porción de código:

```
linesSorted = []
for i in range(len(lines)):
    for rho, theta in lines[i]:
        a = np.cos(theta) # 1
        b = np.sin(theta) # 0
        x0 = a * rho
        x1 = int(x0 + 1000 * (-b))
        linesSorted.append(x1)
```

```
linesSorted = np.sort(linesSorted)
```

Una vez ordenado el arreglo de los valores de las coordenadas X1 se procedió a encontrar las distancias existentes entre cada línea de cultivo de la imagen. Para lograr esto se recorrió el arreglo ordenado y se restó el valor de un primer X1 del valor del siguiente X1 en el arreglo, las distancias se las agregó a un nuevo arreglo para después sacar el promedio de estas con la función np.mean() de la librería numpy. Las siguientes líneas de código logran lo anteriormente descrito:

```
distances = []
for i in range(len(linesSorted)):
    if(i!=len(linesSorted)-1):
        distances.append(linesSorted[i+1]-linesSorted[i])
mean = np.mean(distances)
```

Para finalizar este punto se almacenaron en un nuevo arreglo los puntos de las líneas de cultivo cuya distancia entre ellas sea mayor o igual a la distancia promedio, esto se lo realizó para una mejor precisión del algoritmo al momento de identificar líneas de cultivo, así se evitó que las malezas que se encuentran muy unidas sean tomadas como una de las nombradas líneas.

Se recorrió el arreglo ordenado de los puntos, dentro del ciclo se guardó en un nuevo arreglo el primer punto otorgado debido a que la primera línea siempre se dibuja para a partir de esta verificar si las distancias cumplen con el promedio. También se verificó el promedio de la distancia entre líneas para así poder almacenar el punto de la segunda línea, en caso de que esto no se cumpla se volvió a recorrer el arreglo para ver la diferencia existente de la línea actual con las siguientes líneas de cultivo y, de la misma forma se almacenó el punto de la línea con la que sí cumplió el promedio de distancias. Antes de finalizar el ciclo se verificó si se encontraba en el último punto del arreglo para almacenarlo y evitar errores. La previamente mencionado se lo cumplió ejecutando la siguiente porción de código:

```
newLines=[]
i=0
while(i<len(linesSorted)-1):
    if(i==0):
        newLines.append(linesSorted[i])
```

```
        if(linesSorted[i+1]-linesSorted[i]>=(mean)):
            newLines.append(linesSorted[i+1])
            i+=1
        else:
            aux=i+1
            while(aux<len(linesSorted)):
                if(linesSorted[aux]-linesSorted[i]>=(mean)):
                    newLines.append(linesSorted[aux])
                    i=aux
                else:
                    if(aux==len(linesSorted)-1):
                        newLines.append(linesSorted[aux])
                        i=aux
                aux+=1
```

Identificar líneas representativas

Para detectar las líneas de cultivo de la imagen se recorrieron los datos del arreglo con los nuevos puntos que cumplieron el promedio de distancias, para esto recorremos dicho arreglo con un ciclo “for”. Del arreglo mencionado se obtuvieron los valores de las coordenadas en el eje X y con estos se calcularon los puntos de inicio y de fin de cada línea detectada, dicha línea se la marca de color rojo en la imagen. Es importante saber que para este algoritmo los valores de theta que devolvió la función cv2.HoughLines() para todos los puntos fueron de cero.

En la Fig. 26 se visualizan las líneas de cultivo detectadas, y todo lo mencionado anteriormente se le realizó a través de la siguiente porción de código:

```
for i in range(len(newLines)):
        the
        a = np.cos(theta)
        b = np.sin(theta)
```

```
x0 = a * newLines[i]

y0 = b * newLines[i]

x1 = int(x0 + 1000 * (-b))

y1 = int(y0 + 1000 * (a))

x2 = int(x0 - 1000 * (-b))

y2 = int(y0 - 1000 * (a))


cv2.line(img, (x1, y1), (x2, y2), (0, 0, 255), 2)
```

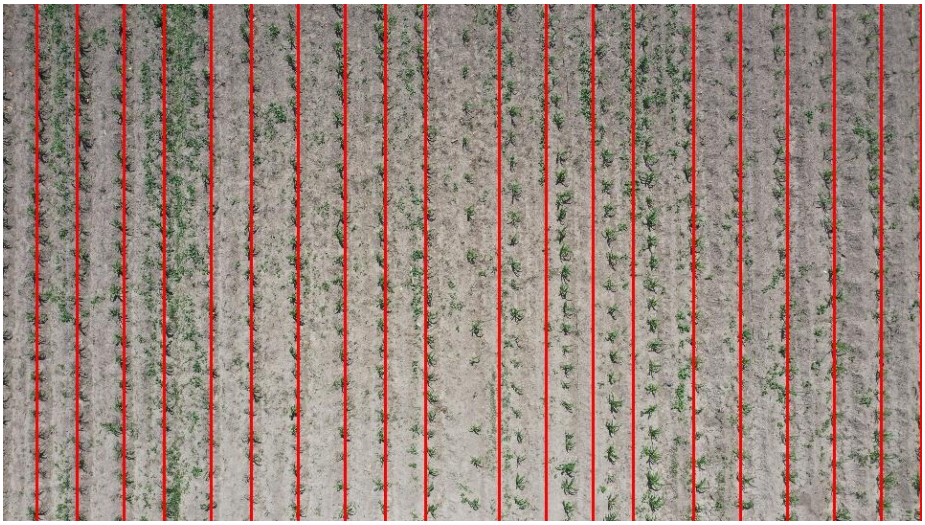


Fig. 26. Líneas de cultivo detectadas.

Fuente: Propia

Antes de pasar al punto del aislamiento de las malezas se dibujaron dos líneas paralelas a cada línea de cultivo marcada de rojo, una a su izquierda de color verde y otra a su derecha de color azul respectivamente.

La distancia que existe desde una línea verde a una línea azul crea un espacio que encierra a las plantas de las líneas de cultivo para así evitar que sean detectadas como una maleza.

Para dibujar las líneas paralelas se disminuyó y se aumentó el valor de las coordenadas en el eje x en 8 píxeles, tanto para x1 como para x2, y esto se lo realizó dentro de los dos ciclos “for” previamente mostrados y seguido de la función cv2.line() que dibuja la

línea de cultivo en la Fig.26. En la Fig. 27 se muestran las líneas paralelas dibujadas en la imagen y las cuales se obtienen al ejecutar las siguientes líneas de código:

```
cv2.line(img, (x1-8, y1), (x2-8, y2), (0, 255, 0), 2)
cv2.line(img, (x1+8, y1), (x2+8, y2), (255, 0, 0), 2)
```

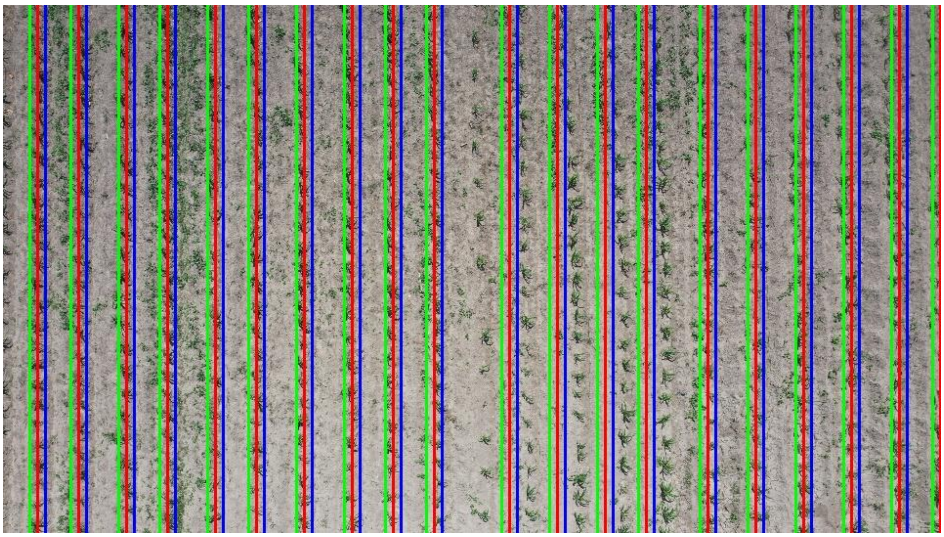


Fig. 27. Líneas paralelas.

Fuente: Propia

Aislamiento de líneas de cultivo

En este punto, después de haber identificado las líneas representativas de la imagen original, a cada espacio creado entre líneas verdes y líneas azules se lo llena de valores “cero” para así borrar las plantas que se encuentran en las líneas de cultivo y así poder aislar cada línea de cultivo.

Dentro de los dos ciclos “for” justo después de las funciones cv2.lines() que dibujan las líneas paralelas, se recorren los valores de las coordenadas en el eje X de la línea verde a la línea azul y cada píxel que se encuentra en este espacio se reemplaza su valor por “cero”.

La Fig. 28 presenta las líneas de cultivos borradas de la copia de la imagen reducida el ruido (copyPlantas) obtenida mediante la “Transformada de Hough”, la imagen resultante sirvió para los siguientes puntos en los cuales se detectaron las malezas. Todo lo anteriormente descrito se realizó con la ejecución de la siguiente porción de código:

```
if(i==len(lines)-1):
            for k in range(x1-8, x1-1):
```

```
                for l in range(0,539):
                    copyPlantas[l][k]=0
        else:
                for k in range(x1-8, x1+8):
                for l in range(0,539):
                    copyPlantas[l][k]=0
```

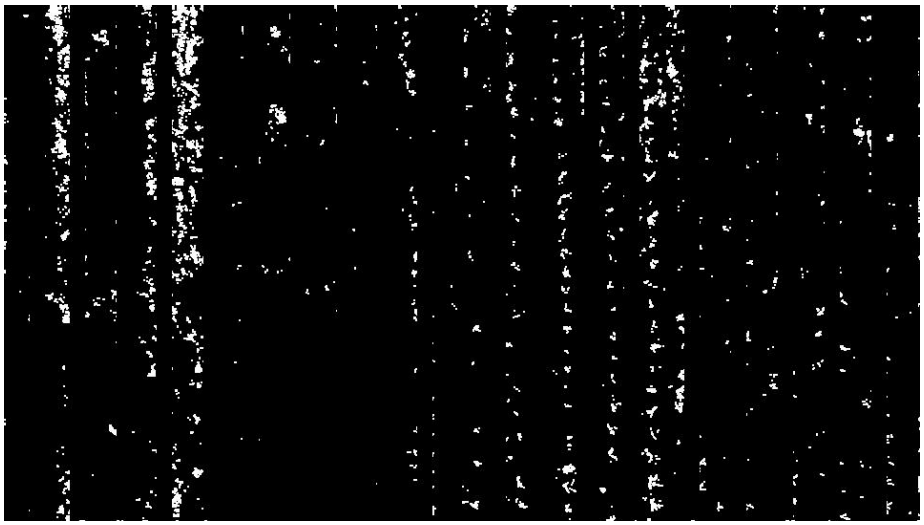


Fig. 28. Líneas de cultivo aisladas.

Fuente: Propia

Para un mejor entendimiento la Fig. 29 nos muestra las líneas de cultivo aisladas de la imagen original junto a las líneas paralelas verdes y azules que marcan los espacios borrados.

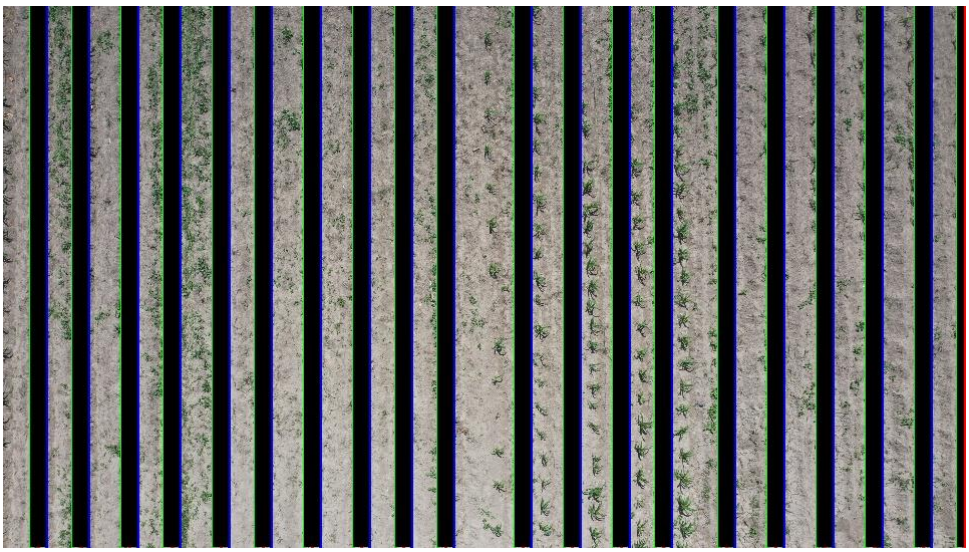


Fig. 29. Líneas de cultivo aisladas de la imagen original.

Fuente: Propia

Dilatar imagen de vegetación restante

En este punto se dilató la imagen aislada de las líneas de cultivo que se obtuvo en el paso anterior, para dicha dilatación se utilizó la operación formológica “dilation”.

La operación “dilation” o dilatación morfológica hace que cada píxel obtenga el valor de 1 si este o cualquiera de sus vecinos más cercanos valía 1 en la imagen binarizada original, de esta forma se amplían los espacios representativos brillantes (pixeles blancos) y se reducen las regiones oscuras (pixeles negros). (Mardiris & Chatzis, 2016).

En la Fig. 30 se visualiza la dilatación de un objeto aplicando la teoría anteriormente descrita.

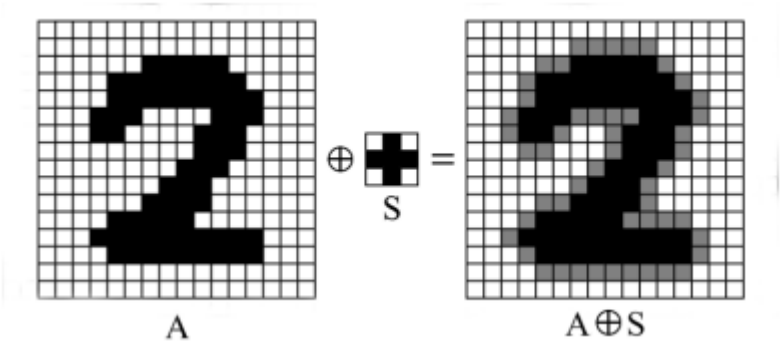


Fig. 30. Dilatación de un objeto.

Fuente: (Mardiris & Chatzis, 2016).

La función ejecutada para la operación de dilatación de la imagen fue dilation() junto a la función disk() con un valor de 1 (uno), estas dos funciones se importan de la librería skimage.morphology. La función dilation() pide los siguientes parámetros:

- Primer parámetro, la imagen binarizada.
- Segundo parámetro, es el elemento estructural o “kernel”, este se lo importa de la librería skimage.morphology y para este algoritmo el elemento es “disk”, también se puede crear manualmente con la librería Numpy.

En la Fig. 31 se presenta la imagen dilatada con los pixeles de los objetos más grandes. Para dilatar la imagen binarizada, sin las líneas de cultivo, se utilizaron las siguientes líneas de código:

```
dilated_image = dilation(copyPlantas, disk(1))
cv2.imshow("Resultado Dilatación",dilated_image)
cv2.waitKey(0)
```

```
cv2.destroyAllWindows()
```

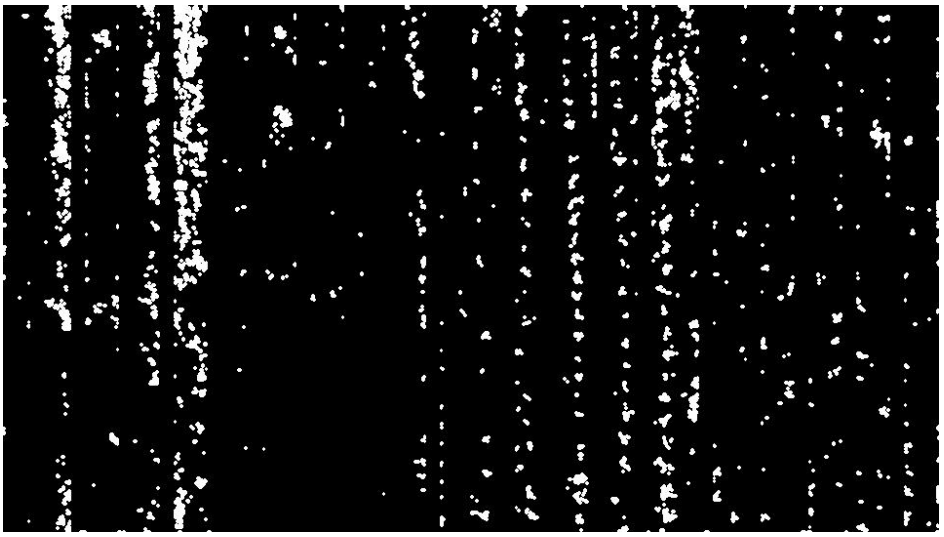


Fig. 31. Imagen de la vegetación restante dilatada.

Fuente: Propia

Identificar malezas en la vegetación restante

Una vez dilatada la vegetación restante se procedió a encontrar los contornos de cada objeto presente en la imagen, para esto se utilizó la función cv2.findContours() de la librería OpenCV, dicha función nos devuelve una lista de todos los contornos que se encuentran en una imagen binarizada y pide los siguientes parámetros:

- Primer parámetro, la imagen binarizada.
- Segundo Parámetro, el modo de recuperación de contornos, para el presente algoritmo se utilizó el modo cv2.RETR_EXTERNAL.
- Tercer parámetro, el método de aproximación de contornos, para el presente algoritmo se utilizó el método cv2.CHAIN_APPROX_SIMPLE que nos devuelve 4 puntos representativos del contorno.

Se ejecutaron las siguientes líneas de código para encontrar los contornos de la imagen binarizada con la vegetación restante:

```
cnts = cv2.findContours(dilated_image, cv2.RETR_EXTERNAL,
cv2.CHAIN_APPROX_SIMPLE)
cnts = cnts[0] if len(cnts) == 2 else cnts[1]
```

Una vez obtenida la lista de contornos de la imagen, se la recorre en un ciclo “for” para obtener los valores de los 4 puntos del rectángulo con la función cv2.boundignRect(), esta nos devuelve los valores numéricos de la coordenada en el eje X, la coordenada en el eje Y, el ancho (w) y la altura (h).

Con los 4 valores numéricos que nos devuelve la función cv2.boundignRect() se procedió a dibujar un marco de color rojo en forma de rectángulo en cada objeto de la imagen binarizada y cada rectángulo representa una maleza. Lo mencionado anteriormente se lo realizó con la función cv2.rectangle() de la librería OpenCV, esta función pide los siguientes parámetros:

- Primer parámetro, la imagen binarizada.
- Segundo parámetro, coordenadas de comienzo del rectángulo.
- Tercer parámetro, coordenadas de finalización del rectángulo.
- Cuarto parámetro, color de la línea de borde del rectángulo.
- Quinto parámetro, ancho de la línea de borde del rectángulo.

Para dibujar los rectángulos que enmarcan a cada maleza identificada se ejecutaron las siguientes líneas de comando:

```
for c in cnts:
    x,y,w,h = cv2.boundingRect(c)
    cv2.rectangle(copyImage1, (x, y), (x + w, y + h),
(0,0,255), 1)
```

En la Fig. 32 se visualizan las malezas identificadas en la vegetación restante del proceso de aislamiento de las líneas de cultivo.

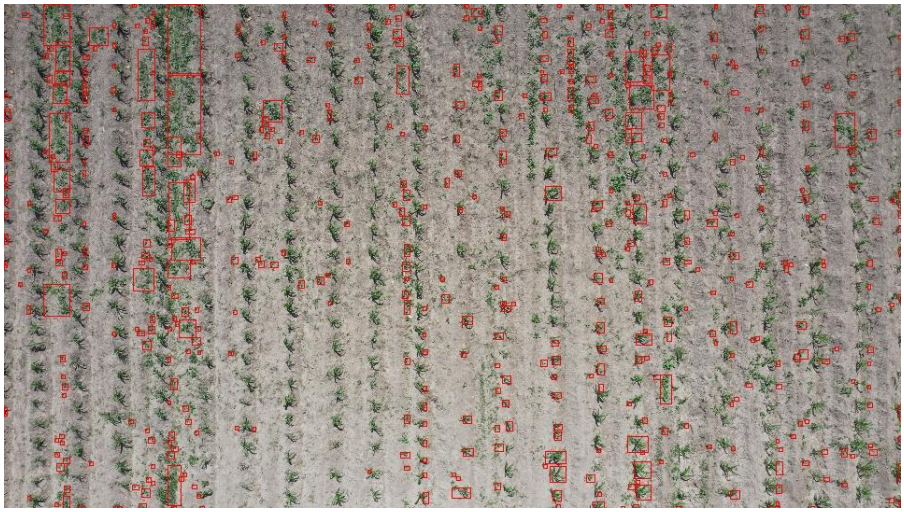


Fig. 32. Malezas Identificadas en la vegetación restante.

Fuente: Propia

En este punto solo se tienen las malezas que se encuentran fuera de las líneas de cultivo, lo siguiente es identificar malezas dentro de las líneas de cultivo, para lograr esto primero se obtuvo una imagen con las líneas de cultivo que fueron aisladas y reducidas el ruido en un punto anterior del algoritmo, después de eso se volvió a realizar el proceso de identificar malezas pero esta vez se tomaron en cuenta las áreas de las plantas presentes en las líneas de cultivo, y toda área menor a estas se identificó como maleza.

Obtener imagen de líneas de cultivo aisladas

En este punto, se restó la imagen que tiene aisladas las líneas de cultivo de la imagen que tiene reducido el ruido, la imagen resultante se visualiza en la Fig.33 y esta imagen nos muestra las líneas de cultivo que fueron aisladas, dicha imagen fue de gran ayuda en el siguiente punto para una mejor detección de malezas. Lo anteriormente mencionado se obtiene de la ejecución de la siguiente línea de código:

```
copyMalezas = opening - copyPlantas
```

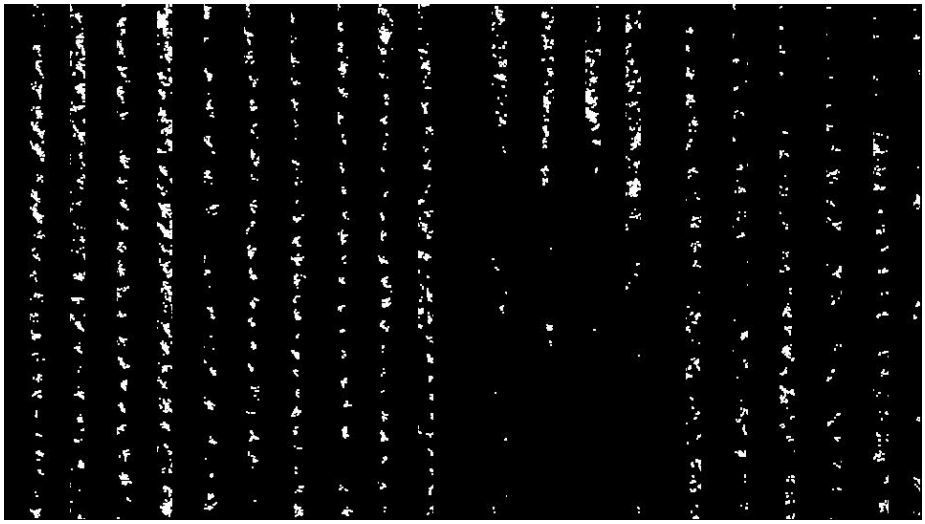


Fig. 33. Líneas de cultivo que fueron aisladas.

Fuente: Propia

Identificar malezas con condición de áreas

Ya obtenida la imagen de las líneas de cultivo que fueron aisladas, se procedió a encontrar los contornos de todos los objetos en dicha imagen, se volvió a ocupar la función cv2.findContours() y con esto obtuvimos la lista de contornos encontrados.

Se recorrió la lista mencionada con un ciclo “for”, primero se obtuvieron los valores numéricos de los 4 puntos de cada rectángulo, segundo se obtuvieron las áreas de estos, de estas áreas se identificaron cuáles son las áreas de las plantas y se estableció un área mínima para las mismas. Antes de pasar a dibujar dichos rectángulos se preguntó con el condicional “if” si el área del rectángulo que se quiere dibujar es menor al área mínima establecida y el área que cumplió esta condición procedió su rectángulo a ser dibujado.

Los rectángulos que fueron dibujados enmarcan las nuevas malezas identificadas en las líneas de cultivo, para lograr lo explicado previamente se ejecutó la siguiente porción de código:

```
areas = []
cnts = cv2.findContours(copyMalezas, cv2.RETR_EXTERNAL,
cv2.CHAIN_APPROX_SIMPLE)
cnts = cnts[0] if len(cnts) == 2 else cnts[1]
for c in cnts:
    x,y,w,h = cv2.boundingRect(c)
    areaBB = w*h
```

```
    areaMin = 20
    if(areaBB < areaMin):
    # if(areaBB < 20 or areaBB > 200):
        areas.append(areaBB)
        cv2.rectangle(copyImage1, (x, y), (x + w, y + h),
(0,0,255), 1)
```

En la Fig. 34 se presenta la nueva maleza identificada al ejecutar el código previo y, la maleza total identificada como resultado final del algoritmo se la expone en el capítulo 3 de resultados del presente trabajo en la Fig. 40, en esta imagen final se encuentran tanto la maleza que se identificó de la imagen de vegetación restante que nos muestra la Fig. 32 y también se encuentra la nueva maleza identificada en este punto.

Fig. 34. Maleza identificada solo en líneas de cultivo.

Fuente: Propia

Capítulo 3

3. Resultados

3.1. Validación de resultados

En esta sección se exponen los resultados tanto del estudio de mapeo sistemático (SMS) que se encuentra realizado en el capítulo 1 y del desarrollo de prototipo que se encuentra en el capítulo 2.

3.1.1. Resultados de la revisión literaria

Como se mencionó en el capítulo 1, en esta sección se interpretan los resultados cuantitativos expuestos en la Tabla 6 y Tabla 7 para dar una mejor explicación y respuesta a las preguntas de investigación.

Interpretación de resultados

Con respecto a la PI1 “¿Cuáles son los lenguajes de programación orientados a GPU más utilizados en el procesamiento de imágenes?” los resultados que se exponen en la Tabla 6 nos muestran una clara tendencia de los investigadores hacia los lenguajes de programación Python y Matlab, siendo Python el lenguaje más utilizado para el procesamiento de imágenes. También nos muestra el poco uso de los lenguajes C y Java que se da para dicho tema y, de la misma forma, se visualiza una mínima inclinación para los siguientes lenguajes de programación: OpenGL Shading Language, PHP, CUDA, la unión de Matlab - Python y la unión de Java - C++. Los resultados de la PI1 y sus porcentajes de frecuencia se los puede visualizar en la Fig. 35.

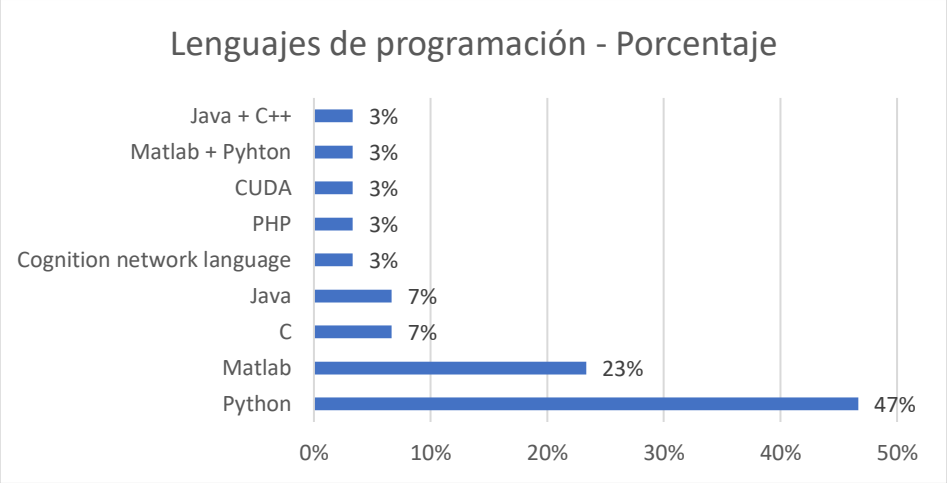


Fig. 35. Gráfica de resultados PI1

Fuente: Propia

Teniendo en consideración la PI2 “¿Qué tipo de aplicaciones se han desarrollado en actividades agrícolas mediante procesamiento de imágenes con GPU?” en la Tabla 7 se puede visualizar que casi la mitad de las aplicaciones desarrolladas por los investigadores son respecto al tema de “Detección y clasificación de plantas por sus características”. La Recolección de datos de plantas para toma de decisiones ocupa el segundo lugar, seguido de la detección de plagas, enfermedades y malezas en los cultivos en tercer lugar, y por último se está desarrollando aplicaciones de detección de líneas de cultivo. La Fig. 36 muestra los resultados de la PI2 con sus porcentajes de frecuencia.

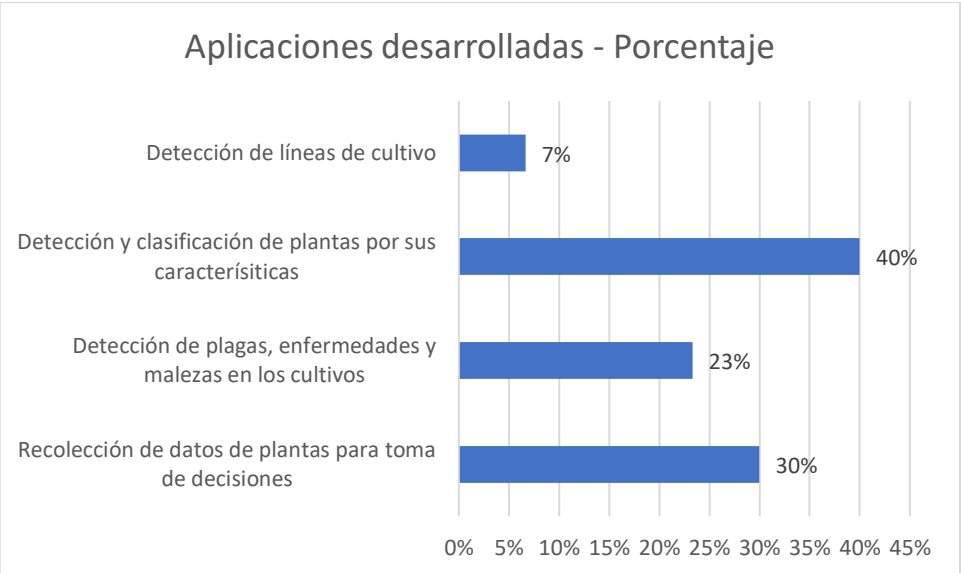


Fig. 36. Gráfica de resultados PI2

Fuente: Propia

Resultados adicionales

Como información adicional a los resultados relevantes a este SMS realizado se obtuvo la cantidad de artículos que se publicaron en cada año la cual se muestra en la Figura 6, esto nos sirvió para visualizar la evolución de los estudios primarios sobre el tema por medio de los años, siendo el año 2021 (47%) el que más tendencia tiene respecto a la investigación de procesamiento de imágenes de actividades agrícolas, seguido del año 2020 (23%) y del año 2017 (13%) en segundo y tercer lugar respectivamente. Esto nos dio a conocer que la investigación sobre el tema ha venido creciendo y siendo más fuerte en los últimos años, dichos resultados se muestran en la Fig. 37.

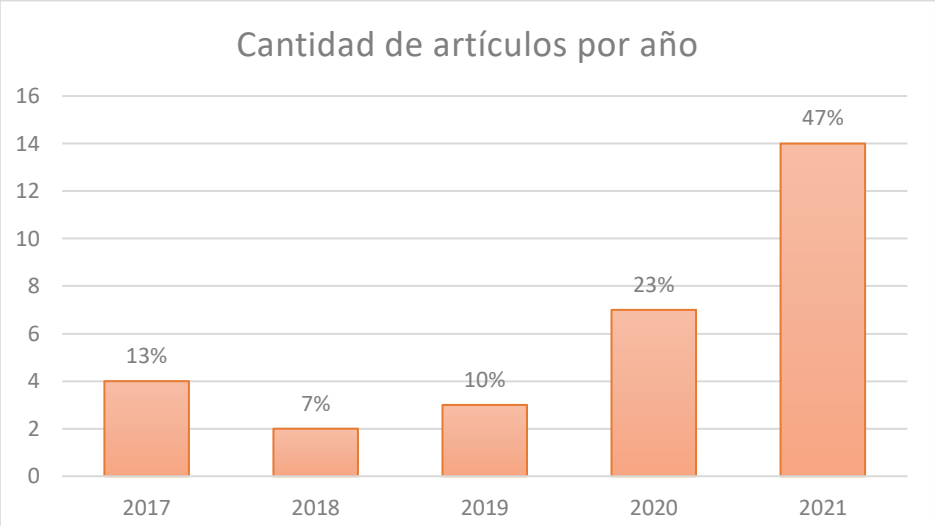


Fig. 37. Cantidad de artículos publicados por año.

Fuente: Propia

También se pudo analizar la cantidad de artículos que se publicaron por cada revista indexada, lo cual se puede visualizar en la Tabla 19. La revista con mayor número de estudios primarios publicados es “Computers and Electronics in Agriculture” (50%) con la mitad de los artículos de los 30 seleccionados en este SMS, seguido de “Biosystems Engineering” (13%) y de “International Journal of Remote Sensing” (7%). Las demás revistas publican una minoría de estudios siendo estas un total de nueve revistas que conforman cada una un 3% de la frecuencia de publicación de los estudios que se seleccionaron. La Fig. 38 muestra en modo de gráfico de barras todo el análisis de las revistas con mayor número de publicaciones para un mejor entendimiento.

TABLA 19. Cantidad de artículos publicados por revista indexada.

Revista	Cantidad	Porcentaje
ISPRS Journal of Photogrammetry and Remote Sensing.	1	3%
Chemometrics and Intelligent Laboratory Systems.	1	3%
Computers and Electronics in Agriculture.	15	50%
Alexandria Engineering Journal.	1	3%
Biosystems Engineering.	4	13%
Acta Agriculturae Scandinavica, Section B — Soil & Plant Science.	1	3%
European Journal of Remote Sensing.	1	3%
Ecological Informatics.	1	3%
Agricultural and Forest Meteorology.	1	3%
International Journal of Remote Sensing.	2	7%
Mathematics and Computers in Simulation.	1	3%
IEEE ROBOTICS AND AUTOMATION LETTERS.	1	3%
Total	30	100%

Fuente: Propia

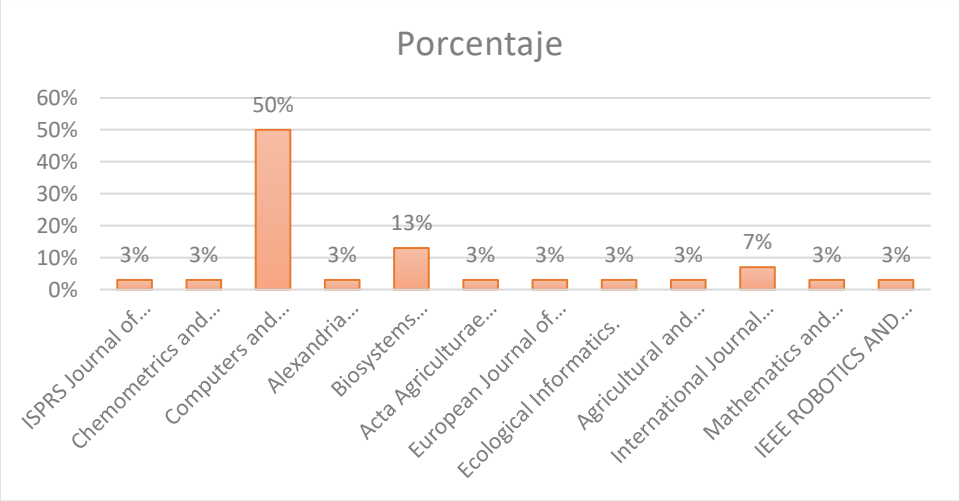


Fig. 38. Análisis de cantidad de artículos publicados por revista.

Fuente: Propia

3.1.2. Resultados del prototipo desarrollado

En esta sección se exponen los resultados del prototipo de detección de malezas en líneas de cultivo. Primero se obtuvo el prototipo en sí, y a su interfaz gráfica que se visualiza en la Fig. 39, en este prototipo es donde se debe ingresar la imagen a ser procesada. Segundo, se muestra en la Fig. 40 la maleza total que se identificó en la imagen de prueba ingresada en dicho prototipo.

Fig. 39. Interfaz gráfica del prototipo desarrollado.

Fuente: Propia

Fig. 40. Maleza total identificada.

Fuente: Propia

Se realizó una comparación del número de líneas detectadas por el humano contra las líneas detectadas por el prototipo para identificar el porcentaje de precisión de este, la mencionada comparación se encuentra en el Anexo C del presente trabajo de titulación. Los resultados fueron los siguientes: Para el etiquetado manual se obtuvo un promedio de 20.9 (21 aproximado) líneas de cultivo de 10 imágenes, mientras que para el prototipo se obtuvo un promedio de 19.7 (20 aproximado) líneas de cultivo identificadas de las mismas 10 imágenes, de esta forma se calculó el porcentaje de precisión y el resultado fue del 94.25%.

3.2. Fundamentos de ingeniería de SWEBOK

Para demostrar lo obtenido en el estudio de mapeo sistemático de mejor manera se realizó un prototipo de procesamiento de imágenes en actividades agrícolas, para validar este prototipo desarrollado se necesitó de la ayuda del cuerpo del conocimiento de la Ingeniería de Software (SWEBOK), se utilizó el capítulo 15 llamado Fundamentos de la Ingeniería para dicha validación, específicamente del subcapítulo 5 que habla sobre el modelamiento, simulación y Prototipado.

El prototipado es un proceso de abstracción, en este se crea una representación parcial del sistema o producto software como una versión inicial. El prototipo carece de todas las funcionalidades del sistema final, pero este es de mucha ayuda para demostrar a las partes interesadas de ciertos requerimientos o diseños de partes del sistema. (Bourque et al., 2014).

Según lo descrito por el cuerpo del conocimiento SWEBOK, el prototipo desarrollado demuestra el procesamiento de imágenes en actividades agrícolas con el lenguaje de programación Python. El lenguaje se obtuvo respecto a la primera pregunta de investigación formulada en el SMS realizado en el primer capítulo y, basados en la información que se expone en la Tabla 6, se visualiza al lenguaje Python como el resultado de la PI1 que demuestra la tendencia de los investigadores hacia los lenguajes de programación orientados a la GP. El prototipo desarrollado serviría de complemento para un sistema o software más grande y completo. La Fig. 39 muestra la ventana inicial del prototipo y la Fig. 41 muestra la ventana con los resultados del procesamiento justo después de haber escogido la imagen.

Fig. 41. Ventana de Resultados.

Fuente: Propia

3.3. Análisis de impacto

En el presente trabajo de titulación se obtuvieron algunos impactos que se describen a continuación:

Impacto Científico

El Estudio de Mapeo Sistemático realizado en el presente documento aporta con la comunidad científica no solo interna, sino, también externa a la Universidad. Los resultados provistos del SMS ayudan a visualizar la tendencia de muchos investigadores al utilizar lenguajes de programación orientados al uso de la GPU en el procesamiento de imágenes agrícolas, de esta forma otros investigadores se basarían en dichos resultados para asegurarse al escoger la herramienta adecuada para este tipo de aplicaciones.

Impacto Tecnológico

El prototipo desarrollado brinda a la comunidad una herramienta que ayuda con la detección de malezas en líneas de cultivo sin necesitar de mucha mano de obra que revise línea a línea la maleza que pueda existir en ellas y entre ellas, de esta manera se automatiza el proceso de detección de malezas el cual se puede implementar o servir de base para todo un sistema o aplicación informática.

Impacto Económico

Todo el prototipo se encuentra desarrollado con herramientas y librerías open source (software libre), de esta forma los costos se reducirían al equipo donde se quiera implementarlo. Al automatizar el proceso de detección de malezas se está reduciendo costos en mano de obra debido a que ya no tendrán que revisar cada línea de cultivo en busca de malezas, ahora se lo podrá hacer por medio de un vehículo aéreo no tripulado que proporciones imágenes para ser analizadas en el programa, de esta manera solo se invertiría en personal para el manejo del aplicativo.

Conclusiones y recomendaciones

Conclusiones

El estudio de mapeo sistemático (SMS) es una herramienta que sirvió de mucho para dar respuesta a las preguntas de investigación, y sobre todo saber la tendencia de los lenguajes de programación orientados a la GPU escogidos por los investigadores a la hora de desarrollar programas o algoritmos de procesamiento de imágenes en actividades agrícolas. Al realizar el SMS se identificó que el lenguaje más utilizado es Python, que se encontraba por arriba del lenguaje Matlab, esto debido a que Python mostró optimización de tiempos y mejor manejo del lenguaje en los artículos que fueron revisados.

El prototipo desarrollado mostró un muy buen porcentaje de precisión al momento de identificar las líneas de cultivo de las imágenes ingresadas al mismo y, de igual forma, el prototipo identificó una gran cantidad de maleza entre dichas líneas. El buen porcentaje obtenido fue gracias al buen uso de las librerías de Python, en especial de la librería que se utiliza específicamente para el procesamiento de imágenes “OpenCV”.

Los mejores resultados se lograron con imágenes a una altura de 15 metros del suelo y que la edad del cultivo sea de 4 semanas, con estas condiciones el prototipo detecta de mejor manera la maleza existente en las líneas de cultivo. Si la vegetación es de menor edad hay la posibilidad de que a ciertas plantas se las detecte como maleza y eso no es lo adecuado.

Gracias al cuerpo del conocimiento de la ingeniería de Software (SWEBOK) se logró definir el prototipo procesamiento de imágenes y de esta forma se representó solo una pequeña parte de un sistema entero de detección de malezas en líneas de cultivo o ya a un mayor nivel, un sistema de agricultura de precisión.

Se obtuvo una muy buena experiencia con el uso del lenguaje de programación Python, tiene una gran cantidad de librerías que son de gran ayuda al momento de desarrollar software, de esta forma no puedes enfocarte a seguir una sola estrategia de desarrollo porque con dos librerías diferentes puedes llegar al mismo resultado, además el tiempo de respuesta de algoritmo en Python es mucho más rápido que el desarrollado en Matlab.

Recomendaciones

Para dar con mejores y con más cantidad de resultados en artículos de investigación al momento de realizar el SMS, recomiendo realizar la búsqueda en las bases bibliográficas: Science Direct e IEEE Xplorer, porque son bases más consolidadas y utilizadas por científicos e investigadores.

Para tener una muy buena precisión al momento de identificar las líneas de cultivo recomiendo medir las distancias entre cada línea de cultivo, después calcular el promedio de dichas distancias, e identificar como línea de cultivo únicamente las que cumplan con dicho promedio de distancia entre ellas, de esta forma se evitará que algún tipo de maleza grande y densa sea identificada como línea de cultivo.

Para evitar que una planta sea marcada como maleza, ya sea porque se encontraba fuera de la línea identificada, o no se identificó toda una línea de cultivo, recomiendo calcular una área mínima y máxima de las plantas marcadas por la función cv2.boundingRect() y de esta forma solo marcar como maleza aquella que no se encuentre dentro del rango de las áreas calculadas.

Recomiendo utilizar imágenes que tengan una altura de 15 metros del suelo y que la vegetación presente 4 semanas de edad, de esta forma se detectará de manera adecuada la maleza existente y se podrá evitar que, a plantas más pequeñas por tener menor edad sean detectadas como maleza.

Al momento de desarrollar un algoritmo con el lenguaje de programación Python, recomiendo escoger librerías con varios años de desarrollo y soporte y también comprobar la compatibilidad entre estas, debido a que ciertas librerías solo funcionan con una versión igual o mayor de otra librería.

Referencias y bibliografía

Referencias

Alashqar, A. M. (2022). Studying the commonalities, mappings and relationships between non-functional requirements using machine learning. *Science of Computer Programming*, *218*, 102806. https://doi.org/10.1016/j.scico.2022.102806

Bourque, P., Fairley, R. E., & IEEE Computer Society. (2014). *SWEBOK: Guide to the software engineering body of knowledge*.

Boursianis, A. D., Papadopoulou, M. S., Diamantoulakis, P., Liopa-Tsakalidi, A., Barouchas, P., Salahas, G., Karagiannidis, G., Wan, S., & Goudos, S. K. (2022). Internet of Things (IoT) and Agricultural Unmanned Aerial Vehicles (UAVs) in smart farming: A comprehensive review. *Internet of Things*, *18*, 100187. https://doi.org/10.1016/j.iot.2020.100187

Coulibaly, S., Kamsu-Foguem, B., Kamissoko, D., & Traore, D. (2022). Deep learning for precision agriculture: A bibliometric analysis. *Intelligent Systems with Applications*, *16*, 200102. https://doi.org/10.1016/j.iswa.2022.200102

Cui, M., Qian, J., & Cui, L. (2022). Developing precision agriculture through creating information processing capability in rural China. *Journal of Rural Studies*, *92*, 237–252. https://doi.org/10.1016/j.jrurstud.2022.04.002

Gonzalez, R. C., & Woods, R. E. (2018). *Digital image processing*. Pearson.

Gruppetta, S. (23 de Marzo de 2022). *Real Python*. Obtenido de https://realpython.com/image-processing-with-the-python-pillow-library/

Han, C., & Qu, L. (2020). Appearance Monitoring of the Transmission Lines based on Hough Transform. *2020 13th International Congress on Image and Signal Processing, BioMedical Engineering and Informatics (CISP-BMEI)*, 71–75. https://doi.org/10.1109/CISP-BMEI51763.2020.9263568

Kim, M., Yeo, Y., & Shin, H. (2021). Binarization for eliminating calibration in fiberscope image processing. *Optics Communications*, *497*, 127198. https://doi.org/10.1016/j.optcom.2021.127198

Lainjo, B. (2019). Enhancing Program Management with Predictive Analytics Algorithms (PAAs). *International Journal of Machine Learning and Computing*, *9*(5), 539–553. https://doi.org/10.18178/ijmlc.2019.9.5.838

Ma, G., & Yue, X. (2022). An improved whale optimization algorithm based on multilevel threshold image segmentation using the Otsu method. *Engineering Applications of Artificial Intelligence*, *113*, 104960. https://doi.org/10.1016/j.engappai.2022.104960

Mahnken, M. (9 de Enero de 2021). *Course Report*. Obtenido de https://www.coursereport.com/blog/what-is-python-programming

Mardiris, V., & Chatzis, V. (2016). A Configurable Design for Morphological Erosion and Dilation Operations in Image Processing using Quantum-dot Cellular Automata. *Journal of Engineering Science and Technology Review*, 6.

MathWorks. (2022). *Cálculo paralelo con MATLAB y Simulink*. Obtenido de MathWorks: https://la.mathworks.com/solutions/parallel-computing.html

Njoroge, B. M., Fei, T. K., & Thiruchelvam, V. (2018). *A Research Review of Precision Farming Techniques and Technology*. *2*(1), 9.

NumPy. (2022). *Numpy documentation*. Obtenido de NumPy: https://numpy.org/doc/stable/#

OpenCV. (2022). *Introduction*. Obtenido de Open Source Computer Vision: https://docs.opencv.org/4.x/d1/dfb/intro.html

Pillow. (2022). *Pillow (PIL Fork) 9.2.0 documentation*. Obtenido de Pillow: https://pillow.readthedocs.io/en/stable/

Python. (2022). *copy — Shallow and deep copy operations*. Obtenido de Python: https://docs.python.org/3/library/copy.html

Python. (2022). *os — Miscellaneous operating system interfaces*. Obtenido de Python: https://docs.python.org/3/library/os.html

Python. (2022). *tkinter — Python interface to Tcl/Tk*. Obtenido de Python: https://docs.python.org/es/3/library/tkinter.html

Radoglou-Grammatikis, P., Sarigiannidis, P., Lagkas, T., & Moscholios, I. (2020). A compilation of UAV applications for precision agriculture. *Computer Networks*, *172*, 107148. https://doi.org/10.1016/j.comnet.2020.107148

Rao, A. (15 de Julio de 2021). *edureka!* Obtenido de https://www.edureka.co/blog/python-opencv-tutorial/

Scikit-image. (2022). *scikit-image 0.20.0.dev0 docs*. Obtenido de scikit-image: https://scikit-image.org/docs/dev/index.html

Tao Ma & Jie Ma. (2016). A sea-sky line detection method based on line segment detector and Hough transform. *2016 2nd IEEE International Conference on Computer and Communications (ICCC)*, 700–703. https://doi.org/10.1109/CompComm.2016.7924792

Tsouros, D. C., Bibi, S., & Sarigiannidis, P. G. (2019). A Review on UAV-Based Applications for Precision Agriculture. *Information*, *10*(11), 349. https://doi.org/10.3390/info10110349

van der Walt, S., Schönberger, J. L., Nunez-Iglesias, J., Boulogne, F., Warner, J. D., Yager, N., Gouillart, E., & Yu, T. (2014). scikit-image: Image processing in Python. *PeerJ*, *2*, e453. https://doi.org/10.7717/peerj.453

Wooldridge, M. (2020). Artificial Intelligence requires more than deep learning—But what, exactly? *Artificial Intelligence*, *289*, 103386. https://doi.org/10.1016/j.artint.2020.103386

Xiang, Y., Yu, B., Yuan, Q., & Sun, D. (2017). GPU Acceleration of CFD Algorithm: HSMAC and SIMPLE. *Procedia Computer Science*, *108*, 1982–1989. https://doi.org/10.1016/j.procs.2017.05.124

Zhang, J., Pang, H., Cai, W., & Yan, Z. (2022). Using image processing technology to create a novel fry counting algorithm. *Aquaculture and Fisheries*, *7*(4), 441–449. https://doi.org/10.1016/j.aaf.2020.11.004

Zhou, Y., Gao, B., Zhang, Q., Yao, P., Geng, Y., Li, X., Sun, W., Zhao, M., Xi, Y., Tang, J., Qian, H., & Wu, H. (2022). Application of mathematical morphology operation with memristor-based computation-in-memory architecture for detecting manufacturing defects. *Fundamental Research*, *2*(1), 123–130. https://doi.org/10.1016/j.fmre.2021.06.020

Anexos

Anexo A: Matriz de metadatos

Todos los metadatos extraídos de cada artículo se encuentran almacenados en una carpeta de OneDrive como un archivo Excel en la siguiente url: Matriz_Metadatos.

Anexo B: Encuesta para levantar requisitos

Entrevistado: MsC. Marco Pusdá (Parte Interesada)

Entrevista:

1. ¿Cuál es el formato de la imagen que ingresa al algoritmo?

Formato JPG

2. ¿Las imágenes estarán a una sola altura o más?

1 sola altura

3. ¿Cuál o cuáles son las alturas a las que van a estar las imágenes?

15 metros

4. ¿Cuál es la resolución de las imágenes que ingresan al algoritmo?

Dimensiones: 5472px * 3648px (ancho*alto) y resolución: 28.346 píxeles/centímetro

5. ¿Qué tipo de plantas son las que se van a analizar en los cultivos?

Cultivo de maíz

6. ¿Cuál es la finalidad u objetivo del algoritmo? (mejorar la pregunta)

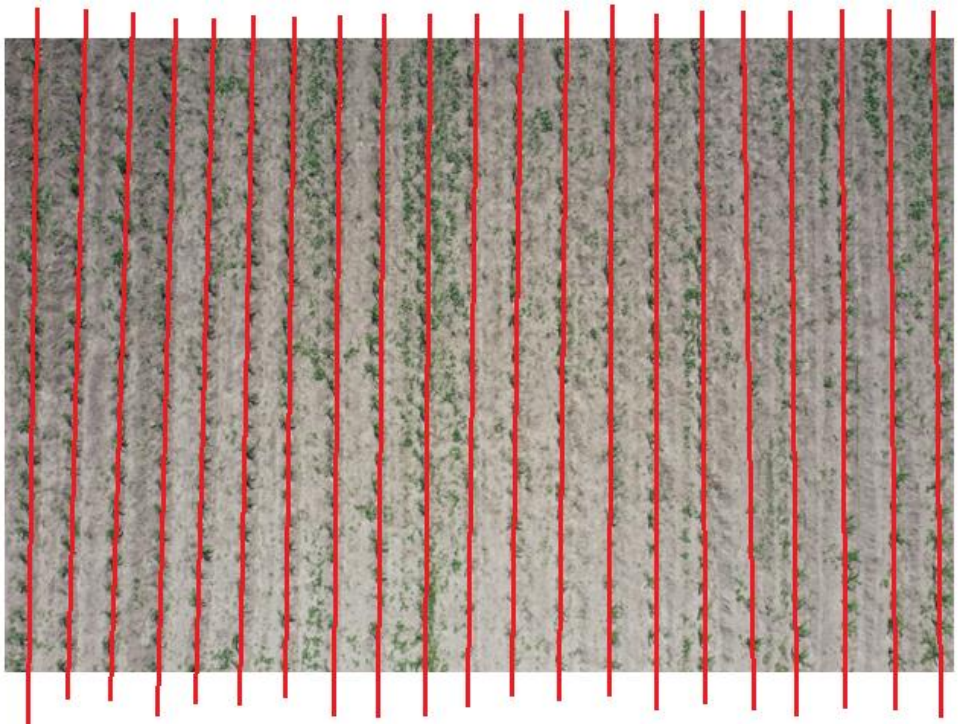
Procesamiento de imágenes mediante visión por computador.

7. ¿Se requiere de una vista para ingresar la imagen y realizar el procesamiento?

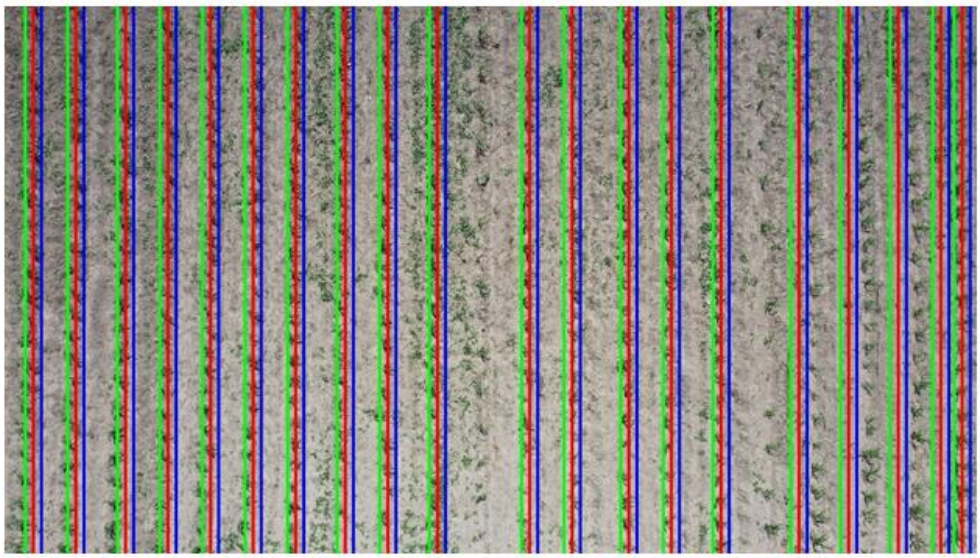
Debe tener una vista en la cual mediante un botón se seleccione una imagen y después se muestren los resultados del procesamiento de la misma.

Anexo C: Etiquetado manual y comparación

- Primera comparación



21 Líneas etiquetadas manualmente



20 Líneas identificadas por el algoritmo

Porcentaje:

Etiquetado manual: 21 líneas = 100%

Prototipo: 20 líneas = 95.23%

- Comparaciones

Nro.	Líneas etiquetado manual	Líneas prototipo	Precisión (%)
1	21	20	95.23%
2	21	19	90.47%
3	21	20	95.23%
4	21	20	95.23%
5	20	19	95.00%
6	21	20	95.23%
7	21	20	95.23%
8	21	19	90.47%
9	21	20	95.23%
10	21	20	95.23%

Promedios:

- Líneas etiquetado manual: 20.9
- Líneas prototipo: 19.7
- Porcentaje de precisión: 94.25%