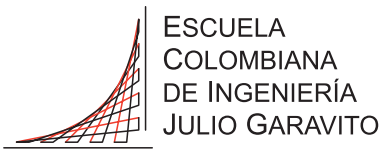


Supervisión autónoma de cultivos agrícolas por medio de vehículos aéreos no tripulados (UAV) usando técnicas de procesamiento de imágenes y fotogrametría

Daniel Santiago Palomino Suarez

Escuela Colombiana de Ingeniería Julio Garavito
Maestría en Ingeniería Electrónica
Bogotá D.C., Colombia
2020

Supervisión autónoma de cultivos agrícolas por medio de vehículos aéreos no tripulados (UAV) usando técnicas de procesamiento de imágenes y fotogrametría

Daniel Santiago Palomino Suarez

Trabajo de grado para optar al título de:
Magíster en Ingeniería Electrónica
con énfasis en Control y Automatización

Director:
Ing. Alexander Pérez Ruiz MSc. PhD.

Escuela Colombiana de Ingeniería Julio Garavito
Maestría en Ingeniería Electrónica
Bogotá D.C., Colombia
2020

Nota de aceptación:
La tesis de maestría titulada "Supervisión autónoma de cultivos agrícolas por medio de vehículos aéreos no tripulados (UAV) usando técnicas de procesamiento de imágenes y fotogrametría", presentada por Daniel Santiago Palomino Suarez, cumple con los requisitos establecidos para optar al título de Magíster en Ingeniería Electrónica con énfasis en Control y Automatización.

Director de Trabajo de Grado

Jurado

Jurado

Bogotá, D.C., 7 de febrero de 2020

Para mis padres, hermanos y sobrinos con todo el cariño y amor.

Agradecimientos

Doy gracias a la Escuela Colombiana de Ingeniería Julio Garavito y su programa de maestría en Ingeniería Electrónica por darme la oportunidad de fortalecer mi formación profesional.

Agradezco especialmente a mi director el Ingeniero Alexander Pérez Ruiz Msc. Phd., cuyo entusiasmo por desarrollar soluciones que mejoren los procesos y calidad de vida del país me motivaron a formar parte de este proyecto. Gracias por el tiempo dedicado en cada reunión, donde cada uno de los comentarios, recomendaciones y criticas constructivas, me ayudaron no solo a adquirir nuevos conocimientos técnicos sino nuevas habilidades para resolver problemas de forma creativa y eficiente a través de la curiosidad y el entusiasmo por nuevo conocimiento. Gracias por la dedicación, por las charlas que siempre ayudaron a mantener la motivación arriba y en general por todas las enseñanzas.

Doy gracias a todos los profesores que hicieron parte de mi formación en la maestría, con sus conocimientos y exigencia me ayudaron a resolver dificultades en diferentes etapas del desarrollo del proyecto, especialmente al Ingeniero Javier Soto Vargas Msc. Phd. Doy gracias al personal de laboratorio de ingeniería electrónica por su labor y buena disposición para proporcionar no solo las herramientas sino el apoyo en aquellos momentos que lo requerí, especialmente a Johnatan. Asimismo, agradezco a mis compañeros de maestría, con quienes trabaje y también aquellos que siempre tuvieron la disposición para brindarme su apoyo cuando lo necesité.

Finalmente, agradezco a toda mi familia y amigos, quienes siempre han sido mi mayor motivación y me han apoyado en todo momento.

Resumen

Las proyecciones del crecimiento en la población mundial para el año 2050, que de acuerdo a la ONU alcanzara los 9.7 millones de personas, requiere que desde ya se empiecen a generar herramientas y soluciones que permitan aprovechar los recursos disponibles de una manera eficiente y responsable, especialmente en el sector agrícola. Es allí donde elementos como los drones pueden ser utilizados para llevar a cabo tareas que integren metodologías de áreas como la agricultura de precisión, visión por computadora, entre muchas mas, para apoyar las diferentes tareas del agro.

Este trabajo de tesis se enfoca en la generación de un algoritmo para automatizar parte del flujo de trabajo al utilizar drones en tareas de supervisión. Mediante este trabajo se otorga al dron la capacidad de identificar los cultivos y características asociadas a ellos por medio de técnica de visión por computador, para posteriormente utilizar dichas características en la generación de un plan de vuelo que permita recorrer el cultivo de una manera eficiente, todo esto en tiempo real y siguiendo los lineamientos del uso del espacio aéreo en Colombia. Esto con el objetivo de obtener un registro fotográfico del entorno de trabajo para que pueda ser utilizado posteriormente para extraer información relevante y tomar decisiones.

A lo largo del documento se exponen las metodologías y herramientas utilizadas durante el desarrollo del proyecto, así como la justificación de su escogencia. Se desarrollo un entorno de simulación de ambientes agrícolas, el cual puede comunicarse directamente con un sistema autopiloto para emular en un alto grado de similitud las condiciones de operación reales de un dron. Se presentan los resultados obtenidos mediante el algoritmo, junto con su respectivo análisis. También se entregan recomendaciones y sugerencias para mejorar los resultados obtenidos, así como posibles soluciones que pueden hacer uso del algoritmo y dar continuidad al proyecto.

Palabras clave: Vehículo aéreo no tripulado (UAV), planificación de trayectorias, fotogrametría, agricultura de precisión, visión por computador, ROS, Distancia de muestra en tierra (GSD), simulación, automatización.

Abstract

Proyections regarding worldwide growth populition could reach a 9.7 million people amount by 2050, according to ONU. This requires an immidiate response to generate solutions that allow leverage available resources in an efficient and responsable man-

ner, specially in agricutural field. Here is where drones can be used to perform tasks that integrate metodologies of fields such as precision agriculture, computer vision, among others, in order to support agricultural activities.

This thesis work is focused in the generation of an algorithm to automate part of the workflow when working with drones in supervision activitites. By means of this work, the dron is given the capacity to identify crops and its features by using computer vision technichs, for latter use in the generation of flight plan that will allow go through the workspace in an efficient way. Everything is achieved in real-time and following the Colombia's uso of air space lineaments. The aim of this is to obtain a photographic register of the workspace in order to use it latter to extract information that allows make decisions.

Through this document the used metodologies and tools are exposed, as well as the corresponding justification for their choice. A simulation environment of agricultural scenarios was created, which can stablsih a direct communication with an autopilot system to emulate with a great similarity degree real operation conditions of drons. The obtained results through the algorithm are presented, along with its corresponding analysis. Recomendations and suggestions to improve obtained results are also given, as well as probable solutions that can use the algorithm and provide continuation to the project.

Keywords: UAV, path planning, photogrammetry, precision agriculture, Computer vision, ROS, Ground Sample Distance (GSD), simulation, automation

Contenido

Lista de Figuras

Lista de Tablas

Lista de símbolos

Símbolos con letras latinas

Símbolo	Término	Unidad SI
EL	Solape frontal *Endlap*	
FH	Altura de vuelo *Flight height*	m
FL	Distancia focal *Focal Length*	pixel;m
GSD	Resolución espacial en tierra *Ground Sample Distance*	m/pixel^2
H	Altura *Height*	m
IH	Altura de imagen *Image width*	pixel
IW	Ancho de imagen*Image height*	m
SL	Solape lateral*Sidelap*	

Símbolos con letras griegas

Símbolo	Término	Unidad SI
λ	Latitud	
φ	Longitud	
θ	Pitch	
ϕ	Roll	
ψ	Yaw	

Subíndices

Subíndice	**Término**
Body	*Body frame*
Cam	*Camera frame*
ECEF	*Earth center earth fixed*
IF	*Image Frame*
NAVF	Navigation Frame

Acrónimos

CPU Unidad de procesamiento central –del inglés *Central Process Unit*–.
CV visión por computador –del inglés *Computer vision*–.
ECEF centrado en la tierra y fijo en la tierra –del inglés *Earth center earth fixed*–.
ENU Este, Norte, arriba –del inglés *East, North, Up*–.
GIS Sistemas de información geografica –del inglés *Geographic information Systems*–.
GPU Unidad de procesamiento gráfico –del inglés *grapphical Process Unit*–.
HITL *Hardware in the loop*.
IMU Unidad de medición inercial –del inglés *Innertial measurement unit*–.
L4T *Linux for Tegra*.
MAVLink *Micro Air Vehicle Communication protocol*.
NED Norte, Este, abajo –del inglés *North, East, Down*–.
OpenCV *Open source computer vision library*.
RAC Reglamentos Aeronauticos de Colombia.
ROS *Robot Operating System*.
SITL *Software in the loop*.
UAEAC Unidad Administrativa Especial de Aeronáutica Cívil.
GPS Sistema de posicionamiento global–del inglés *Global Positioning System*–.
GSD *Ground Sample Distance*.
IOT Internet de las cosas –del inglés *Internet of things*–.
NDVI Indice de vegetación normalizado –del inglés *Normalized Vegetation Difference Index*–.
PA Agricultura de precisión –del inglés *Precision agriculture*–.
UAV Vehiculo aereo no tripulado –del inglés *Unmanned Aerial Vechicle*–.

1. Introducción

Colombia y el resto del mundo se enfrentan a un crecimiento poblacional que exige una gran demanda de recursos, especialmente en el sector agrícola. Para lograr satisfacer esta demanda se deben adoptar nuevas herramientas y metodologías para aumentar la productividad y eficiencia de los procesos de producción agrícola y es allí donde la adopción de vehículos no tripulados, particularmente los aéreos (Unmanned Aerial Vehicle o *UAV*), junto con técnicas de procesamiento de imágenes y aprendizaje de maquina desempeñarán un papel muy importante. Si bien son varios los avances que se han hecho en la generación de soluciones de esta naturaleza, el proceso de planificación de recorridos, vital para la adquisición de datos que permiten obtener información relevante en procesos de toma de decisiones, aun cuenta con varios retos a nivel de desarrollo. En ese sentido, este proyecto se orientó en la generación de un algoritmo que permite automatizar el proceso de planificación del recorrido o trayectoria de un *UAV* utilizado para la adquisición de imágenes de alta resolución en entornos agrícolas. Las trayectorias generadas por el algoritmo permitirán que se satisfagan los parámetros fotogramétricos seleccionados, los cuales son útiles para la ejecución de tareas de supervisión agrícola, como la identificación de frutos, la identificación de características en las hojas, la detección de estrés en los cultivos, el monitoreo del crecimiento del cultivo o cualquier otra que se haga rutinariamente sobre el cultivo en estudio.

1.1. Justificación

Debido al crecimiento de la población mundial, que de acuerdo a [1] se estima en 7200 y 8100 millones de habitantes para los años 2025 y 2050 respectivamente, se generará una mayor demanda de recursos, como los derivados de la actividad agrícola. Estudios como el de [2], valorado hasta en €5000, estiman que el crecimiento en el sector AgTech puede llegar a ser del 22.5 % para el año 2025, en donde se implementaran nuevas técnicas y metodologías que combinen inteligencia artificial, robótica, *big-data*, Internet de las cosas *IOT*[3], entre otras tendencias tecnológicas que ayuden a suplir la creciente demanda de recursos de una manera más eficiente. Esto ha contribuido a que se fortalezca la investigación y el desarrollo en el sector

agrícola, donde cada vez es más frecuente el uso de robots o vehículos no tripulados que ejecutan tareas repetitivas, en algunas ocasiones peligrosas para los humanos, con el fin de aumentar la eficiencia y la productividad, entre otros factores relevantes.

Particularmente el uso de *UAV*ofrece un gran potencial para obtener soluciones efectivas y de bajo costo, como los descritos por [4, 5], donde se presenta un estudio del estado del arte de este tipo de herramientas y sus usos potenciales en diversas áreas como la agricultura de precisión, el sensado remoto, la inspección de estructuras y la atención de desastres.

En general, todas estas aplicaciones se enfrentan a retos de adquisición y procesamiento de datos, los cuales pueden ser extraídos a partir de imágenes tomadas por los *UAVs*, ya que éstas pueden brindar información relevante y detallada de los entornos donde los *UAV* se desenvuelvan. Específicamente en el sector agrícola las imágenes de tipo RGB, Multi-espectrales e inclusive hiper-espectrales permiten obtener información especial, como por ejemplo los índices de vegetación, que dan cuenta del estado de salud de los cultivos; uno de los índices de vegetación ampliamente utilizado es el Indice de vegetación normalizado –del inglés *Normalized Vegetation Difference Index*– (NDVI). Estos índices se han utilizado en trabajos como los de [6] para mejorar el proceso de riego de los cultivos de uva en China, mientras que en [7] el trabajo se enfocó en monitorear el estado de los cultivos de arroz en Colombia.

Para lograr obtener imágenes de calidad mediante este tipo de vehículos una de las etapas más críticas es la planificación de la trayectoria, donde se debe tener en cuenta que dependiendo del uso se requerirá un nivel de detalle o resolución espacial diferente. Adicionalmente, por lo general estas imágenes deben satisfacer condiciones de solapamiento porque son utilizadas en un proceso denominado *stitching* en el que las imágenes individuales se unen para formar una imagen de mayor extensión, denominada ortofotomosaico, que ofrece una visión global del área de trabajo, pero con la resolución de la imagen tomada; adicionalmente a los procesos de *stistching*, existen procedimientos para elaborar modelos 3D y crear mapas como lo evidencian [8, 9, 10, 11].

De este modo la planificación de las trayectorias ofrece un reto interesante sobre el cual trabajar. Se han realizado diferentes avances respecto al tema, como los aportados por [12, 13, 14], donde se han desarrollado algoritmos que realizan la planificación de la trayectoria del *UAV*, los cuales permiten reunir la mayor cantidad de información del terreno a identificar y al mismo tiempo cumplen con un indicador de calidad, como el tiempo de procesamiento o la cantidad de memoria utilizada. Algunos de los algoritmos pueden ser ejecutados previamente al vuelo, luego de haber hecho un primer recorrido, y otros pueden hacerlo *on-line*, en tiempo real. Además de los aportes encontrados en la literatura existen herramientas comerciales que permiten

realizar la planificación de trayectorias, particularmente los software QGroundControl y Pix4D son muy populares para llevar a cabo este tipo de tareas. Mediante estas herramientas se pueden generar planes para realizar tareas de mapeo, modelamiento 3D y ejecución de rutas personalizadas.

Sin embargo, una de la desventajas de trabajar con este tipo de herramientas es que se limita la operación del vehículo en términos de autonomía. Por lo general, para la generación de un plan de vuelo usando estas herramientas requiere que se tenga un conocimiento previo sobre el espacio de trabajo, los planes deben generarse de manera manual antes de cada operación que se quiera realizar y además factores como una mala conexión a internet pueden entorpecer su operación.

Hasta donde conoce el autor en el momento del desarrollo de este documento, no hay una herramienta o algoritmo que permita automatizar parte del flujo de trabajo de la operación de *UAVs* en tareas de supervisión agrícola donde se prime un parámetro fotogramétrico muy importante como es el *Ground Sample Distance* (GSD); que representa la distancia real que existe entre el centro de dos pixeles adyacentes de una imagen capturada por un *UAV* y depende de las características del sensor y la altura de operación al momento de del registro fotográfico; y teniendo en cuenta la distribución y geometría del cultivo a supervisar.

Teniendo en cuenta lo expuesto a lo largo de esta sección, a través de este proyecto se trabajó en la generación de un algoritmo que permite cumplir la tarea de planificación de una manera eficiente y que al mismo tiempo cumple con el parámetro fotogramétrico **GSD** deseado, tomando como referencia una primera captura fotográfica a la máxima altura permitida por la regulación local. Particularmente este trabajo se orientó en brindarle autonomía al *UAV* mediante la automatización de los procesos de reconocimiento del espacio de trabajo y la planificación de trayectorias en ambientes agrícolas. Las herramientas y soluciones implementadas permitieron que estos procesos se lograran ejecutar en tiempo real.

El desarrollo de este trabajo más que un producto final, servirá como base para aumentar la autonomía de las vehículos utilizados, para que a través de ellos se puedan realizar análisis más avanzados de las condiciones de los cultivos, como los realizados por [3, 6, 12, 15, 16, 17], adaptados al contexto colombiano, con el fin de aumentar la calidad de los productos y optimizar los procesos agrícolas en el entorno nacional. La motivación para realizar este proyecto yace en el potencial que tiene Colombia como productor agrícola, derivado de factores como la abundancia de recursos hídricos y la variedad de pisos térmicos propios de la zona ecuatorial tropical, que ofrecen grandes ventajas respecto a otros países productores que no las tienen, como la posibilidad de producción a lo largo de todo el año, de manera que se busca generar herramientas que aporten en los procesos de transferencia tecnológica

por parte de entidades gubernamentales, gremios o *sponsors* hacía los agricultores.

1.2. Organización del documento

El resto del documento se encuentra organizado de la siguiente manera:

En el **capítulo 2** se expone una breve revisión bibliográfica de las áreas de conocimiento, metodologías y técnicas que contribuyen al desarrollo del algoritmo y el proyecto en general. Además se resaltan los aspectos más importantes de la regulación de *UAV*en Colombia. Luego en el **capítulo 3** se presenta la metodología y herramientas a nivel de *software* y *hardware* que permitieron el desarrollo del proyecto, para luego explicar los detalles de implementación de los diferentes procesos involucrados para automatizar la tarea de planificación de trayectorias en el **capítulo 4**.

En el **capítulo 5** se exponen y discuten los resultados generados por los procesos que conforman el algoritmo de planificación de trayectorias y finalmente en el **capítulo 6** se presentan las conclusiones del proyecto y los aportes generados, así como recomendaciones que permitan mejorarlo y continuar al desarrollo de herramientas que apoyen la labor del agro.

2. Marco teórico y regulación

2.1. Vehículos aéreos no tripulados UAV

Los vehículos aéreos no tripulados o UAV (Unmanned Aerial Vehicles) han venido experimentando gran acogida en un sinfín de actividades durante los últimos años, y se espera que este crecimiento se mantenga en el futuro, generando valor para este mercado. Actividades como la inspección de infraestructura o ambientes hostiles [18, 19, 20], agricultura de precisión [21], soporte en atención de desastres o misiones de rescate [22], sensado remoto y actividades de supervisión en entornos como el tráfico y los cultivos [23, 24, 25], son las que han atraído más la atención y a donde se están orientando grandes esfuerzos para fortalecer su desarrollo.

Lo anterior radica en que este tipo de vehículos ofrecen grandes facilidades y ventajas respecto a los sistemas aéreos tradicionales. En primer lugar, el costo para adquirir los UAVes bajo en comparación con los sistemas clásicos, debido a factores como el tamaño y los componentes usados para su fabricación. Además, aunque la complejidad y los riesgos asociados al operar estos vehículos variará dependiendo del tipo o categoría de vehículo a maniobrar (arreglo de rotores, helicóptero o ala fija), estos suelen ser muy bajos gracias a los sistemas de autopiloto que soportan los procesos de control de vuelo y posicionamiento[26]. Así mismo, el tamaño y el peso asociados a estos vehículos pueden variar según el tipo de aplicación, lo que puede ayudar a que sean fáciles de transportar y permitir el ingreso a ambientes o entornos de trabajo donde la operación mediante sistemas tradicionales sería imposible. Este tipo de características permiten que los UAVmejoren la eficiencia y la seguridad en las actividades donde puedan ser usados.

Un ejemplo puntual es el monitoreo de cultivos[27], el cual se hace tradicionalmente por medio de satélites o vehículos tripulados que toman fotografías para determinar el estado de los cultivos y establecer planes de acción. Las dos alternativas por lo general demandan cantidades considerables de esfuerzo y de recursos, además, se ven limitadas por condiciones climáticas a la vez que la información que ofrecen puede no tener la resolución necesaria. En cambio, un UAVes una solución de bajo costo a la que se le pueden integrar diferentes tipos de sensores y herramientas para tomar fotografías de mayor resolución, que conducen a información más amplia, relevante y

confiable. Cabe aclarar que este tipo de vehículos también tienen limitaciones como la condición del clima y el tiempo de operación, pero siempre y cuando no se enfrenten a condiciones extremas, se podría recolectar información que un satélite no sería capaz de adquirir si, por ejemplo, el campo de visión estuviera obstruido por nubes.

2.2. Regulación en Colombia

En Colombia, la Aeronáutica Civil es la entidad reguladora de los usos no militares del espacio aéreo del País, bien sean de transporte o de otra índole, para lo cual dispone de los Reglamentos Aeronauticos de Colombia (RAC). Por lo tanto, es esta entidad quien establece las condiciones para la operatividad de UAVs, sin importar si la actividad tiene fines recreativos o de otra naturaleza, como lo puede ser la utilización de estos sistemas en la agricultura. En la resolución Número O42O1 [28] se incorporaron una serie de disposiciones asociadas a la operación de sistemas de vehículos o aeronaves no tripuladas, allí se describen todas las condiciones y restricciones a las que están sujetos los UAVen Colombia. Sin embargo, valdría la pena mencionar algunos de los elementos más relevantes.

En primer lugar, se establece un sistema de clasificación con base en el riesgo asociado al tipo de operación que puedan desempeñar estos vehículos. La clase A (abierta) se caracteriza por no requerir la autorización de la Unidad Administrativa Especial de Aeronáutica Cívil ($UAEAC$), pues este tipo de operación representa un mínimo riesgo. Entre las condiciones más relevantes se encuentra el límite de carga al momento del despegue, el cual no debe ser superior a 25kg, la altura máxima de operación medida desde la superficie terrestre o acuática no debe ser superior a los 400 pies (alrededor de 123 metros), la velocidad máxima es de 50 MPH (44 kt u 80 km/h o 22 m/seg) y la distancia entre el piloto y la aeronave no debe ser superior a 500 metros lineales y debe ser línea de vista.

La clase B o regulada siempre requerirá previa autorización de la $UAEAC$ para su operación, aún si esta es de bajo riesgo. Esta clase permite una carga de hasta 150 kg al momento del despegue, una velocidad máxima de 100 MPH (87 kt o 160 km/h o 44 m/seg) y límite de línea de vista entre piloto y aeronave de 750 metros, adicionalmente, se permite el uso de UAV para tareas de aspersión agrícolas debidamente autorizadas, siendo esta la única aplicación donde un objeto puede ser lanzado o expulsado desde un vehículo de este tipo.

Finalmente, la clase C se caracteriza por que su operación está restringida dentro de la jurisdicción del estado colombiano, para la operación de vehículos con una carga mayor a 150 kg, aquellos usados como medio de transporte o donde se realicen sobrevuelos internacionales. Sin embargo, la aeronáutica civil podrá emitir autoriza-

ciones especiales para realizar actividades experimentales, orientadas a investigación y desarrollo. En el documento también se encuentran especificaciones de materiales que no deben tener los vehículos, o parte de ellos, aptitudes del piloto u operador, así como otras directivas que promueven la seguridad del espacio aéreo. Esto se hace con el fin de reducir los riesgos asociados a la operación, dentro de los cuales puede darse la obstrucción a otras aeronaves o el daño a infraestructura, personas, seres vivos, etc.

2.3. Agricultura de precisión

La Agricultura de precisión –del inglés *Precision agriculture*– (PA) hace referencia al conjunto de prácticas y técnicas que permiten ejecutar las actividades agrícolas de una manera eficiente, optimizando los recursos y tiempos de operación, con el objetivo de reducir perdidas y aumentar la productividad. Actualmente la PA propende por integrar elementos de visión de máquina, procesamiento de imágenes y *machine learning* junto con todos los desarrollos previos del sensado remoto, para buscar automatizar procesos agrícolas como la supervisión de los cultivos [15, 29].

Con la ayuda de vehículos especiales que recorren el terreno (*UAV*, *UGV*, etc), así como de sensores instalados a lo largo de ellos, se pueden extraer grandes cantidades de datos que al ser procesados brindarán información valiosa de la condición del cultivo[27, 30, 31]. Por ejemplo, los índices de vegetación (*Vegetation Index*) permiten detectar estrés y enfermedades en las plantas, también se pueden hacer procesos de reconocimiento de elementos de los cultivos (frutos, hojas, maleza, etc) por medio de procesamiento de imágenes y de *Machine learning*. Con esta información se pueden controlar los cultivos de una mejor manera, por medio de sistemas de irrigación, fertilización y fumigación que son capaces de administrar la cantidad necesaria de recursos de manera eficiente, disminuyendo costos de operación. Incluso se pueden generar soluciones que hagan más eficientes los procesos de recolección y siembra, por medio de la integración de vehículos aéreos y terrestres, tanto tripulados como no tripulados [7, 12, 32]. Por supuesto este tipo de actividades tienen un alcance mucho mayor, por la cantidad de tiempo y recursos que pueden llegar a demandar.

Es importante resaltar que los sistemas de *UAV*, independientemente del sector donde sean utilizados, se enfrentan a limitaciones como el tiempo de operación (asociado a los sistemas de alimentación de los vehículos), condiciones climáticas (que pueden generar daños considerables en los vehículos o alterar las condiciones de las mediciones), regulación y legislación (la cual puede variar según el contexto o región donde se lleve a cabo el estudio) e incluso en algunos casos limitaciones en los sistemas de adquisición y procesamiento de información [33], convirtiéndose en los retos a

superar y donde sigue existiendo mucho potencial de desarrollo e investigación[34].

2.4. Visión por computador y fotogrametría

El concepto de visión por computador –del inglés *Computer vision*– (CV)¸ surgió a la par con los primeros desarrollos de inteligencia artificial y su objetivo es que los sistemas de computación tengan la capacidad de describir el mundo real (percibido en 3D por animales y humanos) con base en imágenes o fotografías 2D, de manera que se tenga un mayor entendimiento del ambiente y se facilite la interacción con él. Es ampliamente usado en procesos de control de calidad en el sector industrial, modelado 3D de objetos y entornos, en el reconocimiento de objetos particulares en un determinado ambiente (rostros, vehículos, etc), en el apoyo al diagnóstico médico mediante imágenes, entre muchos otros [35, 36, 36, 37, 38].

Transformaciones entre espacios de color, filtros, análisis de histogramas, operaciones sobre pixeles, operaciones sobre regiones y operaciones morfológicas, son algunas de las herramientas y metodologías mas básicas utilizadas por los investigadores para resolver problemas en esta área [39, 40, 41]. Estas herramientas, junto con un mínimo conocimiento del problema a resolver (entorno de trabajo, objetos a identificar, características de la cámara) permiten resaltar, segregar o identificar características en imágenes o vídeos, con el fin de tomar decisiones. Sin embargo, muchas veces factores como las condiciones del ambiente (temperatura, iluminación), capacidades de los sistemas de computó y adquisición de datos pueden llegar a generar problemas en el desempeño de las soluciones generadas.

Otro campo o área del conocimiento importante a tener en cuenta es el de la fotogrametría (photogrammetry). Al igual que la visión por computador, la fotogrametría se enfoca en la extracción y análisis de información del ambiente por medio de imágenes. Sin embargo, la fotogrametría se enfoca en la medición de características de objetos y superficies reales a través de las imágenes lo que le permite modelar la superficie terrestre y con ello automatizar, completamente o parcialmente, procesos de mapeo e inspección de terrenos [42]. Las labores de fotogrametría de campo generalmente son apoyadas en sistemas de vehículos aéreos, tripulados o no tripulados, los cuales realizan un vuelo sobre el terreno o elemento sobre el cual se quiera obtener información. Durante el vuelo se realiza un registro fotográfico con características especiales de inclinación y cubrimiento, que junto con las características de la cámara y la información de posicionamiento y orientación del vehículo, hace posible realizar posteriormente procesos de modelamiento 3D y mapeo.

Particularmente en los procesos fotogramétricos de mapeo se usa el registro fotográfico de los vuelos para la obtención de un foto-mosaico. Este es una imagen de

alta resolución que se elabora a partir de imágenes individuales, donde cada una debe cumplir con características específicas de solapamiento longitudinal y lateral con respecto a las imágenes adyacentes. Por esta razón es importante poder asociar la ubicación de los elementos de las imágenes mediante sistemas de coordenadas, pues muchas aplicaciones que hacen uso de vehículos auto-guiados requieren este tipo de información para su operación. En el trabajo de [43] se hace un análisis de cómo ha evolucionado el uso de sistemas aéreos no tripulados en tareas de fotogrametría.

2.5. Sistemas de referencia y transformaciones

2.5.1. Sistemas de referencia

En las labores de planificación, navegación y control de trayectorias de vehículos intervienen múltiples sistemas de coordenadas, comúnmente denominados *frames*, especialmente si se trata de vehículos aéreos. El objetivo de estos sistemas de coordenadas es describir la posición de un objeto dentro de un entorno o espacio, como por ejemplo la tierra[44, 45]. A continuación se describen brevemente los sistemas coordenados involucrados al trabajar con vehículos aéreos, cuya relación se puede apreciar en las figuras 2-1 y 2-2.

Geodésico: Es ampliamente usado en los sistemas de navegación GPS , en donde un punto está definido por la combinación de latitud(λ), longitud(φ) y altitud (H) (parecido a un esférico), en este *frame* se introduce un elipsoide de referencia, el cual busca describir la superficie de la tierra de una manera más precisa. La latitud se mide como el angulo de rotación entre el meridiano principal y el punto de medición, la longitud representa el angulo que se forma entre el plano ecuatorial y la normal del elipsoide de referencia que pasa por el punto de interés, por último la altura se mide a partir de la superficie del elipsoide de referencia, siendo el WGS64 uno de los más populares.

***ECEF*:** El sistema referencia centrado en la tierra y fijo en la tierra –del inglés *Earth center earth fixed*– (*ECEF*) es un sistema de coordenadas cartesiano que rota junto con la tierra y cuyo origen coincide con el centro de la misma, donde un punto es representado con las coordenadas (x_{ecef}, y_{ecef} y z_{ecef}). El eje z de este sistema coincide con el eje de rotación de la tierra apuntando al polo norte, el eje x coincide con la normal que pasa por el meridiano principal ubicado sobre el plano ecuatorial, mientras el eje y completa el sistema para que cumpla con la regla de la mano derecha.

Local: Este es un sistema de coordenadas cartesiano, cuyo plano ***xy*** es tangente a la superficie de la tierra. En labores asociadas a robótica este *frame* coincide con el *frame* de mapeo y con el *frame* de navegación, por lo cual un objeto en este sistema

sera representado por las coordenadas (x_{navf}, y_{navf} y z_{navf}). Se tienen dos notaciones para definir los ejes del sistema coordenado, con la notación Este, Norte, arriba –del inglés *East, North, Up*– (*ENU*), el eje $\boldsymbol{y}$ coincide con el norte, el eje $\boldsymbol{x}$ con el este y el eje $\boldsymbol{z}$ es perpendicular a la superficie de la tierra. Con la notación Norte, Este, abajo –del inglés *North, East, Down*– (*NED*) el norte coincide con el eje x, el este coincide con el eje y, mientras el eje z va hacia el centro de la tierra desde la superficie, siendo esta la notación preferida al trabajar con vehículos aéreos. El origen de este sistema coordenado se fija de manera arbitraria, siendo el lugar de despegue o aquel donde se activen los sistemas de sensores el que usualmente se elige para tal propósito.

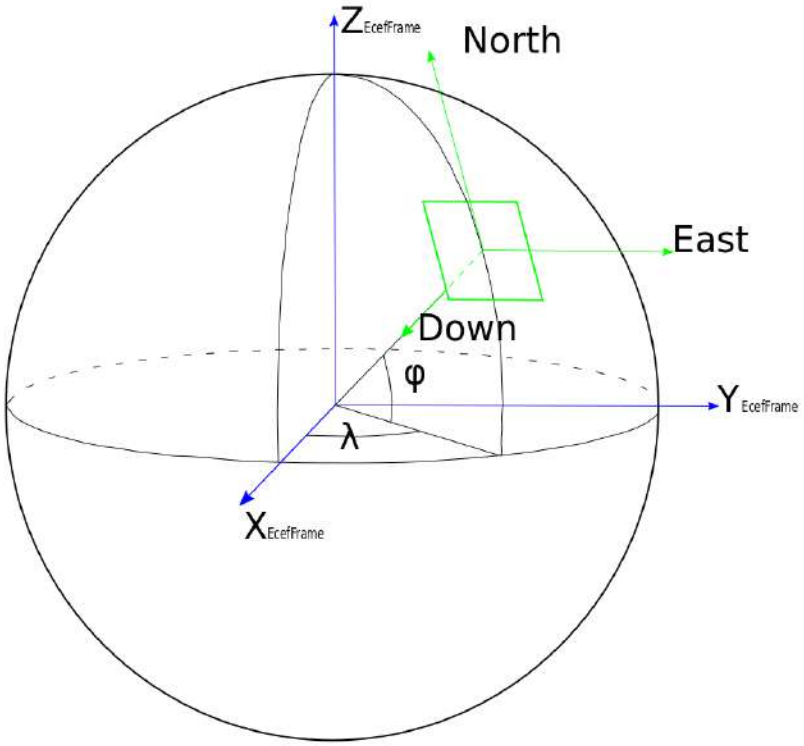


Figura 2-1: *Frames*: *Geodetic*, ECEF, Local (NED).

Body: Este es un sistema de coordenadas cartesiano atado al centro de gravedad del vehículo, siendo este el origen del mismo. Independientemente del tipo de vehículo o aplicación el eje $\boldsymbol{x}$ generalmente se alinea en la dirección de avance frontal, sin embargo, los demás ejes pueden ser fijados de forma arbitraría; particularmente para vehículos aéreos el eje $\boldsymbol{z}$ se define saliendo del centro de masa del vehículo y

apuntando a la superficie, similar a la notación *NED* del *frame* local, con lo cual el eje $\boldsymbol{y}$ apunta al lado derecho del vehículo para completar el sistema coordenado, cumpliendo la regla de la mano derecha.

Camera: Este es un sistema de coordenadas cartesiano con origen en el punto principal de la cámara. El eje $\boldsymbol{z}$ es perpendicular a la superficie del sensor, el eje $\boldsymbol{x}$ apunta a la derecha de la cámara y el eje $\boldsymbol{y}$ completa el sistema coordenado para cumplir la regla de la mano derecha. Adicionalmente se encuentra el *frame* de la imagen, que es un sistema cartesiano de dos dimensiones cuyo origen suele ser la esquina superior izquierda de la imagen; por medio de modelos como el de *pinhole* es posible establecer una relación con el *frame* de la cámara.

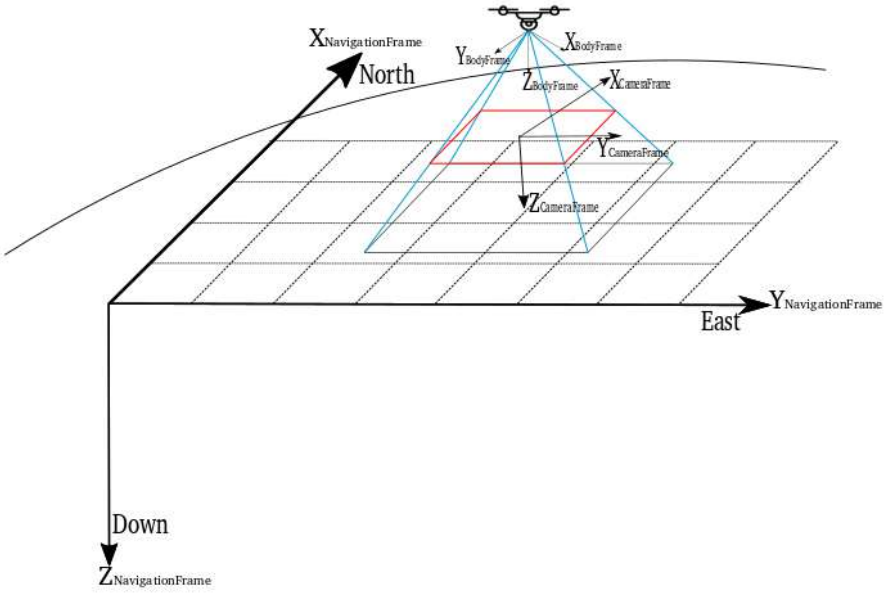


Figura 2-2: Frames: Local, *Body*, *Camera*.

2.5.2. Transformaciones

Cuando se involucran múltiples sistemas coordenados es necesario contar con herramientas o métodos que permitan relacionarlos, y es allí donde se definen las transformaciones. Una transformación es la combinación de una traslación y una rotación, con las cuales se pueden expresar las coordenadas de un punto definido en un sistema coordenado en las coordenadas de otro u otros sistemas coordenados. Para definir las

matrices de rotación se suelen usar conceptos como los ángulos de Euler, cosenos directores o cuaterniones, cada uno con sus respectivas ventajas y restricciones[45, 46].

En actividades como las de mapeo y planificación que involucran vehículos aéreos y cámaras, el primer paso consiste en trasladar la información de los múltiples *frames* hacia un *frame* local, como el *NED*, donde se ejecutan todas los procesos necesarios para completar la tarea objetivo. En el caso de la planificación de trayectorias serían los puntos de control[46, 47, 48, 49]. Posteriormente, si es necesario se pueden aplicar las transformaciones inversas para expresar las coordenadas de los puntos de control dentro del *frame* geodésico, es decir en coordenadas GPS[45, 50, 51], que el *UAV* pueda seguir.

Para trasladar las coordenadas de un punto p_{gps} con coordenadas (λ, φ, h) en el *frame* geodésico hacia el *frame* local, primero se deben expresar las coordenadas geodésicas como coordenadas cartesianas, específicamente como coordenadas del *frame* ECEF, cuya relación se expresa en la ecuación (2-1), donde N y e se definen con base en las características del elipsoide de referencia, específicamente para WGS84 $e = 0.08181919$, $N = a/\sqrt{1 - e^2 \sin^2 \varphi}$, donde $a = 6,356,752.3142m$ es la longitud del semieje mayor del elipsoide.

$$\begin{bmatrix} x_{ECEF} \\ y_{ECEF} \\ z_{ECEF} \end{bmatrix} = \begin{bmatrix} (N+h)\cos\varphi\cos\lambda \\ (N+h)\cos\varphi\cos\lambda \\ (N(1-e^2)+h)\sin\varphi \end{bmatrix} \tag{2-1}$$

Luego, la relación entre un punto del *frame* ECEF y el local (navf) está dado por la expresión (2-2), donde R_{ecef}^{navf} es la matriz de rotación entre el *frame* ECEF y el *frame* de navegación (2-3), p_{ecef} es el punto de interés, y $p_{ecef_{ref}}$ es el punto de referencia donde se definirá el origen del *frame* local y que para el dron corresponderá al lugar donde este sea ubicado para el despegue (***HOME***).

$$p_{navf} = R_{ecef}^{navf}(p_{ecef} - p_{ecef_{ref}}) \tag{2-2}$$

$$R_{ecef}^{navf} = \begin{bmatrix} -\sin\lambda\cos\varphi & -\sin\lambda\sin\varphi & -\sin\lambda\cos\varphi \\ -\sin\varphi & \cos\lambda & 0 \\ -\cos\lambda\cos\varphi & -\cos\lambda\sin\varphi & -\sin\lambda \end{bmatrix} \tag{2-3}$$

Lo anterior permite obtener las coordenadas en el *frame* local bajo la convención *NED* , pero si se requiere trabajar con la convención *ENU* se puede usar la ecuación (2-4).

$$R_{ENU}^{NED} = R_{NED}^{ENU} = \begin{bmatrix} 0 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & -1 \end{bmatrix} \tag{2-4}$$

El siguiente paso es determinar la relación entre las coordinadas del *frame* del vehículo (*body*) y el *frame* local, lo cual se logra mediante la expresión (2-5). Donde $p_{uav_{navf}}$ es la posición del vehículo en el *frame* local (obtenida a partir de la posición GPS), R_{body}^{navf} es la matriz de rotación entre el *frame body* y el local (2-6). Esta matriz de rotación se elabora con base en la orientación del vehículo, que en el sector aeronáutico son definidos a partir de tres ángulos de Euler conocidos como $yaw(\psi)$, $pitch(\theta)$ y $roll(\phi)$, los cuales guardan una relación con los ejes del *frame* local y pueden ser obtenidos con una Unidad de medición inercial –del inglés *Innertial measurement unit*– (IMU).

$$p_{navf} = R_{body}^{navf} p_{body} + p_{uav_{navf}} \tag{2-5}$$

$$R_{body}^{navf} = \begin{bmatrix} \cos\theta\cos\psi & \sin\phi\sin\theta\cos\psi - \cos\phi\sin\psi & \cos\phi\sin\theta\cos\psi + \sin\phi\sin\psi \\ \cos\theta\sin\psi & \cos\phi\sin\theta\sin\psi + \cos\phi\cos\psi & \cos\phi\sin\theta\sin\psi - \sin\phi\cos\psi \\ -\sin\theta & \sin\phi\cos\theta & \cos\phi\cos\theta \end{bmatrix} \tag{2-6}$$

Ahora es momento de determinar la relación entre el *frame* de la cámara y el *frame* del vehículo. Si la cámara se encuentra fija al vehículo la relación esta dada por la matriz de rotación unitaria sobre el eje z ((2-7)). Si en cambio la cámara cuenta con un dispositivo estabilizador se deberá tener en cuenta la rotación sobre los tres ejes, sin embargo, tal caso no es cubierto en este trabajo.

$$R_{cam}^{body} = \begin{bmatrix} \cos\alpha & -\sin\alpha & 0 \\ \sin\alpha & \cos\alpha & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{2-7}$$

Finalmente, para obtener la relación entre las coordenadas del *frame* de la imagen y el de navegación, se deben relacionar dos transformaciones, además del modelo que define las coordenadas de la cámara en términos de la imagen, donde k es la matriz

de parámetros intrínsecos y s es un factor de escala.

$$T_{camera}^{navf} = T_{camera}^{navf} * T_{camera}^{navf} \tag{2-8}$$

$$s \begin{bmatrix} x_{if} \\ y_{if} \\ 1 \end{bmatrix} = k \begin{bmatrix} x_{cam} \\ y_{cam} \\ z_{cam} \end{bmatrix} \tag{2-9}$$

Para mayor información sobre las transformaciones se puede consultar los trabajos de [44, 45, 47, 49], sobre los cuales se obtuvieron las transformaciones y ecuaciones aquí presentadas.

2.6. Algoritmos de planificación de trayectorias

En el sector de la robótica, particularmente los vehículos autónomos (AGV) dependen de algoritmos que planifican la trayectoria que estos deberán seguir en un entorno determinado para llegar de un punto a otro, donde pueden existir obstáculos que deberán ser evadidos. En este proceso se presentan dificultades asociadas a elementos como la incertidumbre o desconocimiento del entorno (si se tiene total conocimiento del entorno se denomina un problema estático, en caso contrario se denomina un problema dinámico), limitaciones en los sistemas de sensores y actuadores que permitan identificar y reaccionar ante elementos del ambiente, así como el nivel de complejidad del sistema de control del vehículo (lo cual está relacionado en gran parte con el número de grados de libertad del vehículo) [14, 52]. Sin embargo, independientemente de esas dificultades los algoritmos deberán tener la capacidad de orientar el vehículo a lo largo del entorno.

En ocasiones estos algoritmos son clasificados por la capacidad del vehículo (en particular su sistema de procesamiento) de ejecutarlos en tiempo real, si cuentan con ella se denominan *Online* u *On-Board*, en cambio si el algoritmo se ejecuta previamente a la operación se denomina *Offline* u *Off-Board*. Esto dependerá directamente de la complejidad de la tarea a ejecutar y de las limitaciones mencionadas anteriormente, pues los problemas dinámicos por su naturaleza demandan que los sistemas tengan la capacidad de responder en tiempo real.

En el caso de los *UAV* de acuerdo a [52, 53] las condiciones climáticas, junto con la limitada capacidad de los sistemas de sensores y autonomía hacen que la planeación de trayectorias sea más desafiante respecto a vehículos terrestres. Sin embargo, hay muchos conceptos que pueden ser aplicados sin importar el tipo de vehículo que esté

involucrado. Generalmente los vehículos se definen como objetos con la capacidad para desplazarse y pueden representarte mediante vectores o como un punto dentro de un sistema de coordenadas (tridimensional o bidimensional) que representa el espacio o mundo donde el objeto puede desplazarse.

Los algoritmos de planificación tendrán que determinar el conjunto de puntos, así como la secuencia que el vehículo ejecutará para desplazarse a través del espacio. En los trabajos de [52, 54, 55] se hace una revisión bastante detallada de los algoritmos que se han desarrollado para la planificación de trayectorias de vehículos, donde se identifican técnicas como la descomposición celular (exactas y aproximadas), descomposición por retículas y redes neuronales, los autores indican las bondades y limitaciones que tienen los algoritmos desarrollados hasta el momento. Autores como [7, 12, 13, 32, 56, 57] han hecho aportes donde desarrollan algoritmos para la planificación de trayectorias de UAV en tareas de supervisión agrícola, como la estimación de la cobertura del campo, clasificación activa de elementos de ambientes agrícolas (arboles, plantas, etc) y monitoreo de los cultivos.

3. Metodología y herramientas

3.1. Metodología

Para desarrollar y probar el algoritmo se optó por crear un entorno de simulación que permitiera replicar el comportamiento de un *UAV* real, así como generar diversos escenarios agrícolas en lo que respecta a la geometría de los cultivos y la orientación de los surcos. Posteriormente se definió la secuencia de pasos que el dron debería ejecutar una vez se encuentre ubicado en el cultivo para considerar que la planificación se realizó de manera automática. Para empezar, se debe seleccionar el GSD y solape entre fotos deseados, una vez que el vehículo esté en condiciones de volar y se emita la orden, el deberá despegar hasta la máxima altura permitida por regulación, una vez alcanzada dicha altura se toma una fotografía. Con base en esta última se debe realizar la identificación de las características del cultivo, más específicamente el contorno o geometría del mismo, así como la orientación de los surcos. Por último, con base en los parámetros fotográficos deseados y las características identificadas, se debe realizar la planificación de la trayectoria. Una vez se cuente con un plan de vuelo el vehículo deberá ejecutarlo, de manera que se obtenga un registro fotográfico en cada uno de los puntos de control (*waypoint*). Esta secuencia se ilustra en la figura 3-1.

Con base en lo anterior, se identificaron tres macro-procesos cuyos detalles de implementación son presentados en el capítulo 4. En primer lugar, se encuentra el posicionamiento y control del vehículo, donde a partir de las lecturas de los sensores se deben proporcionar las indicaciones necesarias para que el vehículo se ubique en las posiciones deseadas dentro del espacio de trabajo, como en el momento del despegue, o durante la ejecución del plan. En segundo lugar, se tiene la etapa de procesamiento de imágenes, donde a partir de la captura fotográfica del cultivo se deben ejecutar técnicas que permitan identificar las zonas del cultivo con vegetación y cuáles son las características geométricas en dichas zonas. Finalmente, se tiene la etapa de planificación y geo-referenciación en donde se usan conceptos geodésicos, geométricos y algebraicos que permitan establecer una relación entre los elementos del espacio de trabajo y definir una trayectoria adecuada.

Es importante resaltar que este proyecto se desarrolló para cultivos ubicados en

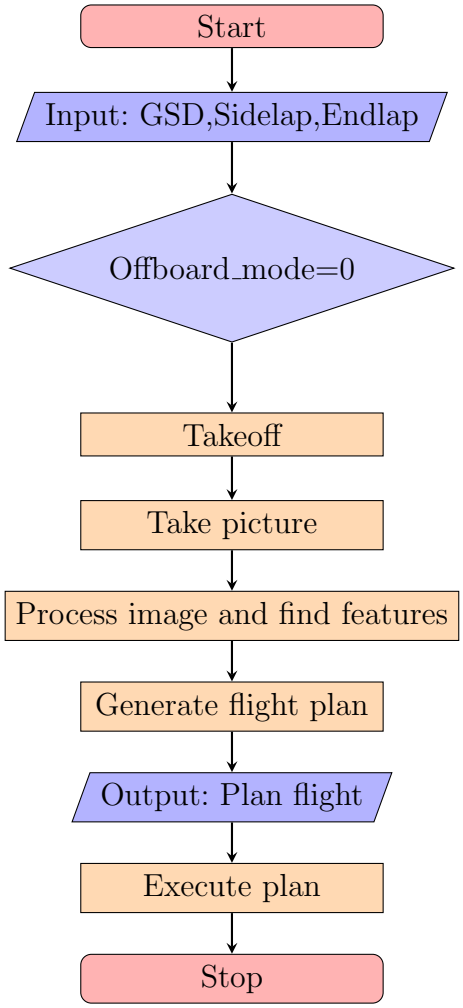


Figura 3-1: Secuencia para obtener el esqueleto.

terrenos llanos o cuya altimetría no varíe significativamente, pues si este no es el caso el parámetro GSD no permanece constante.

3.2. Herramientas

Durante la ejecución del proyecto se usaron preferentemente herramientas de desarrollo de código abierto, pues en ellas se generan múltiples contribuciones que permiten a los investigadores enfocarse en sus áreas de investigación al disminuir los tiempos de implementación de soluciones tecnológicas. A continuación se presenta una breve descripción de las herramientas más relevantes del proyecto.

3.2.1. Robot Operating System (ROS)

ROS más que un sistema operativo, es un *framework* de desarrollo para soluciones robóticas desarrollado por la universidad de Stanford y Willow Garage [1], pues este no ejecuta las tareas tradicionales de manejo y planificación de procesos, sino que se enfoca en proveer una estructura de comunicaciones robusta, la cual corre sobre un sistema operativo huésped (tradicionalmente Linux), que permite ejecutar las tareas críticas de un sistema robótico [58]. Las principales características que definen a *ROS* es que implementa una tipología *peer-to-peer* entre los procesos y *hosts* desplegados; permite programar en múltiples lenguajes, entre los más destacados C++ y Python, reduce la complejidad del *framework* al hacer uso de herramientas del sistema operativo huésped para ejecutar algunos de sus procesos de control principales y es de acceso abierto.

A lo largo del desarrollo de *ROS* se han generado múltiples soluciones para sistemas robóticos en lo que respecta a controladores de *hardware*, navegación, manipulación de objetos, visión por computadora (*OpenCV*), simuladores (Gazebo), entre muchos otros componenetes que al ser públicos, contribuyen a que los desarrolladores se enfoquen en sus trabajos sin tener que empezar cada proyecto desde cero y evitar procesos de configuración tediosos que pueden consumir tiempo valioso.

Estructura de comunicación

ROS cuenta con una estructura en la que cada proceso, comúnmente definido como nodo, ejecuta una o vareas tareas dentro de una porción de software (*main*) y se comunica con otros nodos. La comunicación entre los nodos se puede realizar en diferentes modalidades de acuerdo a la necesidad, siendo la de publicador-suscriptor la más utilizada, seguida por cliente-servidor y action-lib.

Publicador-suscriptor: En esta modalidad el nodo publicador está transmitiendo constantemente algún tipo de información en un tema (*topic*), identificado mediante una palabra clave como velocidad u odometría, mientras que el suscriptor accede a esa información a través de esa palabra clave. La información contenida en cada tema puede estar conformada por uno o varios **mensajes**.

Un mensaje es una estructura que define algún tipo de dato (*string, bool, etc*). Un mensaje puede estar compuesto por otros mensajes o arreglos de mensajes y se define mediante un formato simple que no depende de ningún lenguaje de programación. Una vez se han definido los mensajes, *ROS* se encarga de crear objetos o estructuras compatibles con cada uno de los lenguajes de programación con los que se esté trabajando y a través de procesos de serialización y deserialización permite que por

[1] Pagina oficial Willow Garage: http://www.willowgarage.com/

ejemplo, un nodo publicador ejecutado con Python pueda transmitir información a un suscriptor desarrollado en C++.

Cliente-servidor: En esta modalidad el nodo servidor se encarga de ejecutar un proceso y emitir una respuesta a un nodo cliente, pero a diferencia de la modalidad anterior esto se realiza por demanda, o sea específicamente en el momento que el cliente lo solicite.

Para ello se define la estructura de **servicio** (*service*), que está conformada por una solicitud (*request*) y una respuesta (*response*). La solicitud está conformada por mensajes que permitan al servidor ejecutar el proceso deseado, mientras que la respuesta contiene mensajes sobre el resultado del proceso. Al igual que el publicador-suscriptor cada servicio es definido mediante una palabra clave.

Por lo general los procesos ejecutados mediante servicios se caracterizan por ser de bajo costo computacional.

Action-lib: Por último, las *action-lib* se crearon para las ocasiones donde se tienen varios procesos ejecutándose simultáneamente y alguno de ellos consume un tiempo considerable para su ejecución. Las *action-lib* se definen con una estructura de tres elementos que son la meta (*goal*), la retroalimentación (*feedback*) y el resultado (result), cada una definida por medio de mensajes. La ventaja sobre un servicio es que estos no inhabilitan el nodo que solicita la ejecución del proceso, pues son hilos independientes, en cambio, va proporcionando una retroalimentación del estado de la ejecución hasta el momento en que el proceso se ejecute completamente o se solicite la cancelación del mismo.

Vale la pena resaltar que un solo nodo puede implementar múltiples suscriptores, publicadores, servicios y action-libs, de acuerdo a los requerimientos de cada aplicación y debido a que múltiples nodos pueden estar relacionados con un tipo de problemática común a resolver, como en la planificación y todos estos nodos suelen agruparse en un paquete (*package*).

Un paquete está conformado por un conjunto de archivos y carpetas donde se definen los mensajes, servicios, nodos, dependencias y demás elementos asociados a una aplicación. Por lo general un proyecto está conformado por varios paquetes, pero al ser elementos individuales los desarrolladores pueden acceder o generar contenidos que atiendan temas muy específicos sin necesidad de copiar los proyectos completamente, de esta manera se promueve la reutilización de código.

Gazebo

ROS trabaja junto con el simulador Gazebo, también de acceso abierto, pero que es una herramienta muy útil pues cuenta con un motor de física potente que permite simular el comportamiento físico-mecánico de varios tipos de robots, sensores

y actuadores con una precisión elevada. Gazebo cuenta con modelos predefinidos que permiten construir múltiples entornos de simulación, pero también permite importar modelos elaborados en otros softwares, lo cual amplía aún más las posibilidades. Lo anterior contribuye a realizar pruebas con robots de una manera más segura evitando el uso de componentes de *hardware* hasta el momento en que las aplicaciones, como los algoritmos de control, hayan sido probadas y pulidas hasta el punto que sea seguro acceder al *hardware*. *ROS* ofrece un paquete para interactuar con el simulador, mediante el cual se pueden programar *plugins* que controlan el comportamiento de elementos como motores o sensores.

3.2.2. Pixhawk 2

Pixhawk 2 es un sistema de autopiloto cuyas principales especificaciones técnicas se muestran en la tabla 3-1 y es compatible con sensores GPS RTK, que ofrecen una mayor precisión a nivel de posicionamiento y tiene la capacidad de brindar redundancia en elementos como la *IMU*, GPS y alimentación, el cual fue diseñado originalmente para integrar la visión por computadora en los procesos de navegación y control de *UAVs*[59, 60]. Puede operar bien sea con el *firmware ArduPilot* o PX4, ambos de acceso abierto y desarrollados especialmente para el control de vehículos aéreos y terrestres. Entre los dos *firmware* el que se uso en el proyecto fue el PX4, pues este ofrece mayores prestaciones en comparación a *ArduPilot*.

PX4 es definido como un *framework* robótico multiproceso basado en nodos para sistemas profundamente embebidos [61], el cual opera sobre un sistema operativo Nuttx, y al igual que *ROS*, a nivel de comunicación interna cuenta con un sistema de distribución de datos basados en la modalidad publicador-suscriptor. Cuenta con una interfaz que permite la interoperabilidad entre el microcontrolador sobre el cual corre el framework y computadores de compañia basados en Linux. Lo anterior le permite la interacción con procesos de comunicación y control de *ROS* que corran en un computador de compañía, y a diferencia de ArduPilot es capaz de ejecutar nodos nativos de *ROS*. Adicionalmente, cuenta con un sistema de comunicación *offboard* que usa el protocolo *MAVLink*[2], diseñado especialmente para la comunicación de vehículos aéreos, aunque su uso haya sido extendido a vehículos terrestres [62].

En el proyecto, la comunicación entre *ROS* y el autopiloto se realizó mediante el sistema de comunicación *offboard*, por medio del paquete *MAVROS*, el cual traduce los mensajes en formato *MAVLink* a formato *ROS* y viceversa. Ademas cuenta con una serie de nodos definidos para cada *framework*, PX4 o *ArduPilot*, que permiten acceder a la información de los sensores, así como enviar ordenes o comandos al

[2]Sitio web *MAVLink*: https://mavlink.io/en/

autoplito a través de una serie de servicios y temas (tópicos).

Figura 3-2: Pixhawk2 Autopilot.

Pixhawk2 technical especifications	
Processor	Interfaces
32bit STM32F427 Cortex M4 core with FPU	5x UART (serial ports), one high-power capable, 2x with HW flow control
168 MHz / 252 MIPS	2x CAN (one with internal 3.3V transceiver, one on expansion connector)
256 KB RAM	Spektrum DSM / DSM2 / DSM-X® Satellite compatible input
2 MB Flash (fully accessible)	Futaba S.BUS® compatible input and output
32 bit STM32F103 failsafe co-processor	PPM sum signal input
	RSSI (PWM or voltage) input
	I2C
	SPI
	3.3v ADC input
	Internal microUSB port and external microUSB port extension

Tabla 3-1: Especificaciones técnicas Pixhawk 2.

3.2.3. Nvidia Jetson TX2

El computador de placa reducida de altas prestaciones Nvidia Jetson TX2 es un sistema de cómputo embebido con una gran capacidad de procesamiento, especialmente diseñado para imagen y vídeo, gracias a su GPU de 256 núcleos. La tabla 3-2 muestra un resumen de sus características técnicas más relevantes, ademas, el módulo tiene unas dimensiones de apenas 50x87mm, lo que lo vuelve ideal para la integración en robots. Ademas del módulo, Nvidia también ofrece un kit de desarrollo que cuenta con múltiples puertos y periféricos listos para usar, permitiendo agilizar

la etapa de desarrollo, razón por la cual se uso en el proyecto. La figura 3-3 muestra tanto el módulo (izquierda) como el kit de desarrollo (derecha).

El módulo es capaz de operar bajo una distribución de Linux denominada *Linux for Tegra* (*L4T*), sobre la cual se puede instalar *ROS*. De esta forma puede funcionar como computadora de compañía para ejecutar las tareas de procesamiento a través de paquetes de *ROS*. Ahora bien, existen otros sistemas embebidos como la Raspberry Pi, que también son compatibles con *ROS* y pueden trabajar como computadores de compañía, sin embargo la mayor ventaja de la TX2 yace en su capacidad de computación en paralelo, por medio de su Unidad de procesamiento gráfico –del inglés *grapphical Process Unit*– (*GPU*), pues existe un conjunto bastante amplio de herramientas y librerías, como TensorFlow, cuDNN y OpenCV elaboradas con soporte para CUDA, que es la plataforma de computación paralela creada por Nvidia, con la cual la ejecución de los procesos se realiza con mayor velocidad en comparación a la ejecución en CPU, lo cual permite generar soluciones de gran calidad como es el caso de Skydio[3].

Dentro del conjunto de herramientas disponibles en la Jetson TX2 vale la pena resaltar el conjunto de librerías *Open source computer vision library* (*OpenCV*) [63], de acceso abierto y desarrolladas especialmente para procesos de visión por computadora, en donde se encuentran implementados múltiples algoritmos y técnicas que permiten por ejemplo, realizar cambios entre espacios de color, detectar bordes, aplicar operaciones morfológicas, entre muchas aplicaciones más, las cuales resultan muy convenientes para la etapa de identificación de características y aunque la generación del foto-mosaico está por fuera del alcance del proyecto, *OpenCV* cuenta con librerías especiales para realizar dicho proceso.

[3]sitio web ficial Skydio: https://www.skydio.com/

Figura 3-3: Nvidia Jetson TX2 Module and Developer kit.

Nvidia TX2 technical Specifications	
Processor	Ports and peripherals
dual-core NVIDIA Denver2 + quad-core ARM Cortex-A57	HDMI 2.0
256-core Pascal GPU	802.11a/b/g/n/ac 2×2 867Mbps WiFi
8GB LPDDR4, 128-bit interface	Bluetooth 4.1
32GB eMMC	USB3, USB2
4kp60 H.264/H.265 encoder	10/100/1000 BASE-T Ethernet
Dual ISPs (Image Signal Processors)	12 lanes MIPI CSI 2.0, 2.5 Gb/sec per lane
1.4 gigapixel/sec MIPI CSI camera ingest	PCIe gen 2.0, 1×4 + 1×1 or 2×1 + 1×2
	SATA, SDcard
	dual CAN bus
	UART, SPI, I2C, I2S, GPIOs

Tabla 3-2: Especificaciones técnicas Jetson TX2.

4. Implementación

4.1. Configuración experimental

El proyecto se desarrolló netamente a nivel de simulación siguiendo una arquitectura como la mostrada en la figura 4-1, donde se cuenta con un computador que corre la simulación, el autopiloto Pixhawk 2 operando en modo *Hardware in the loop* (*HITL*) y la NVidia funcionando como computadora de compañía (*companion computer*).

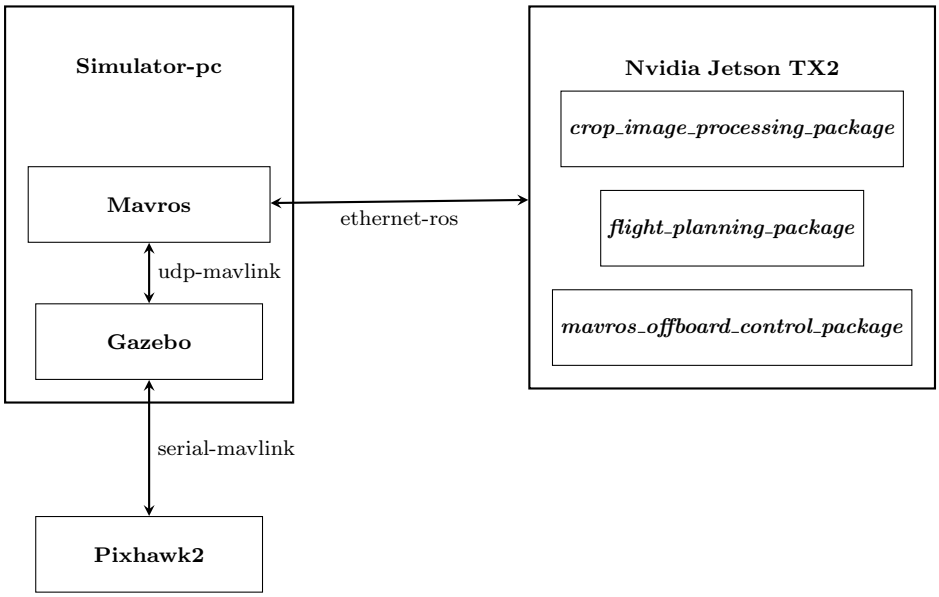


Figura 4-1: Configuración experimental.

El computador usado para correr la simulación fue un HP-Pavilion gaming 15, al cual se le instalo el sistema operativo Linux y la distribución Kinetic de *ROS* que incluye todas las herramientas de simulación y apoyo gráfico (En el anexo B se proporcionan unos *scripts* y recomendaciones sobre la instalación de *ROS* según las características del equipo), con lo cual fue posible construir el entorno de simulación. En primera instancia se usó el software libre para diseño de modelado 3D **Blender** para construir el modelo de los cultivos que harían parte del entorno de simulación,

allí se generaron cultivos donde se variaba la orientación de los cultivos y las formas geométricas, aunque vale la pena aclarar que cualquier otro software que permita exportar sus modelos a formato *Collada*, que es compatible con Gazebo, puede ser usado para elaborar los modelos de interés. Adicionalmente, se instaló CUDA 8.0 y se compiló OpenCV 3.3 con soporte para dicha versión con el fin de comparar el rendimiento con la Nvidia Jetson TX2.

Vale la pena resaltar que

Figura 4-2: Cultivos del entorno de simulación.

El repositorio de PX4 contiene los archivos fuente para compilar y cargar el *firmware* a los dispositivos de autopiloto en sus diferentes versiones, pero además incluye unas herramientas para realizar simulación en varias aplicaciones, incluyendo Gazebo. Entre tales herramientas se encuentran modelos de *UAVs* y sensores, con sus respectivos *plugins*, basados en el simulador *RotorS* [64] que sirven para interactuar con *ROS*; particularmente hay un *plugin* que genera una interfaz *Mavlink* razón por la cual se usó para generar el modelo del dron usado en el proyecto. Entre los modelos disponibles se tomó el del Iris Quadcopter y le fueron integrados sensores GPS, IMU, magnetómetro y una cámara ubicada sobre la parte inferior apuntando perpendicularmente al suelo. Teniendo el modelo de los cultivos y del vehículo se generó un archivo *launch* en el cual se inicializa Gazebo y se instancian los dos modelos, completando de esta manera el entorno de simulación. Es importante mencionar que durante la etapa inicial del proyecto se usó el modelo *hector quadrotor* [65] en lugar

del Iris, sin embargo, al no contar con el *pluggin* para interactuar en el modo *HITL*, se descartó su uso. El anexo D incluye instrucciones de como instalar y configurar este entorno.

El *firmware* PX4 cuenta con dos modos de simulación en los que permite reemplazar los sensores físicos por sensores virtuales, como los implementados por Gazebo. Estos dos modos de simulación se llaman *Software in the loop* (*SITL*) y *Hardware in the loop* (*HITL*). En ambos casos el simulador se encarga de enviar las lecturas de los sensores al *firmware* por medio de mensajes en formato *mavlink* y este último luego de procesar los mensajes emitirá uno o varios comandos como respuesta, también en formato *mavlink*, la cual puede ser por ejemplo, las velocidades normalizadas que deben aplicarse a cada motor para efectuar un cambio de posicionamiento desde un punto a otro. Por otro lado, lo que diferencia los dos modos es que en el caso de *SITL* el *firmware* es una copia que se compila en un computador de escritorio o laptop (generalmente el mismo computador donde se ejecuta la simulación), por lo tanto nunca se tendrá acceso a los componentes de *hardware*, mientras que en *HITL* el *firmware* es ejecutado directamente por el autopiloto, el cual si tiene acceso a dichos componentes. Durante el proyecto se hicieron pruebas con ambas modalidades, pero se enfatizó en el uso de simulación *HITL*, cuyas instrucciones de configuración para el Pixhawk 2 se encuentran en el anexo E.

En cuanto al tercer componente de la configuración experimental, la NVidia Jetson TX2 fue configurada inicialmente a través de Jetpack 3.3, la cual instala: L4T 28.2 y CUDA 9.0, entre otras herramientas para inteligencia artificial; también se instalaron las librerías de OpenCV con soporte para CUDA (instrucciones disponibles en el anexo A). Posteriormente se instaló la distribución Kinetic de *ROS*, así como los paquetes de planificación desarrollados en el proyecto (instrucciones disponibles en el anexo C).

En cuanto al alcance y las bondades del sistema y entorno de simulación, vale la pena aclarar que durante el diseño de los surcos de los cultivos se intento emular factores como la textura, tanto de la tierra como de las plantas y que las medidas sensores tuvieran un buen grado de similitud respecto a los de vida real. En la figura 4-3, se alcanza a observar la textura de las hojas en los surcos, la cual se puede modificar en los modelos de Blender. Como se mencionó en la metodología, se generaron cultivos en donde se asumen condiciones de altimetría y topográficas planas para el terreno, con el fin de mantener las condiciones para que el GSD no varíe a lo largo de él. Los modelos de los sensores simulados permiten agregar condiciones como el ruido, lo que conlleva los respectivos desafíos que se tendrían al trabajar con sensores reales. Por último, se desprecian factores asociados a las condiciones climáticas como el viento o la lluvia y se hace énfasis que en la modalidad HIL el

autopiloto es quien controla las velocidades de cada uno de los motores para hacer los desplazamientos basandose en la información de los sensores.

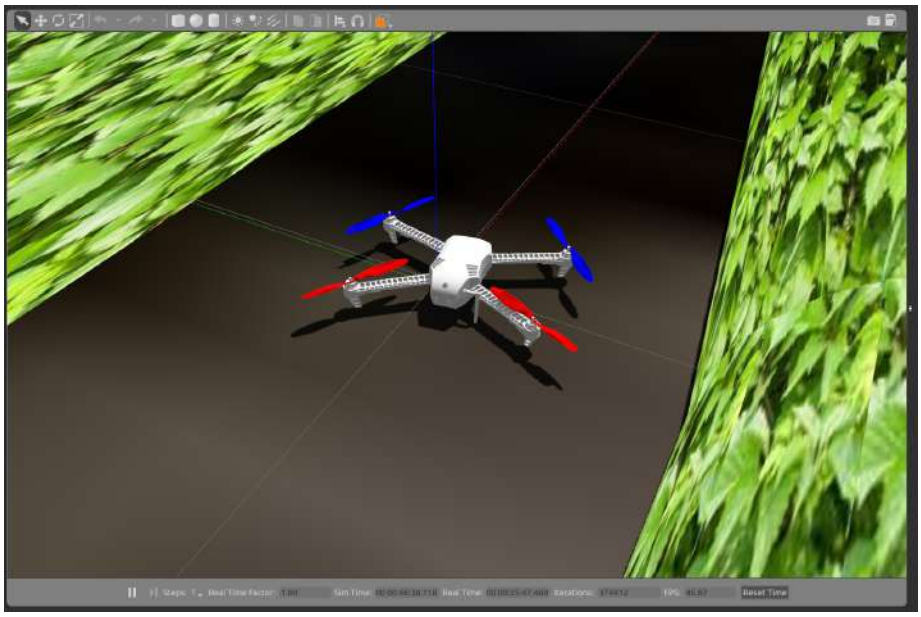


Figura 4-3: Iris quadcopter.

4.2. Identificación características del cultivo

Una vez que el dron ha completado el despegue hasta la altura máxima y tomado una fotografía, la identificación de las características se divide en tres tareas principales, en primer lugar está la identificación o segmentación de las zonas donde existe vegetación, de este proceso se obtiene una máscara que sirve como insumo para las otras dos tareas, que son la identificación de los surcos y con su orientación y la identificación de la geometría de los límites del cultivo [66, 67, 68, 69].

4.2.1. Segmentación de vegetación

Para identificar las zonas donde existe vegetación se debe definir una variable relevante que pueda ser asociada a la vegetación, lo cual va ligado al tipo de sensor que se esté utilizando. En el caso del entorno de simulación, donde se tiene un sensor de tipo RGB, se pueden extraer principalmente características de color, en cambio con una cámara multiespectral como la RedEdge MX, (la cual no se puede simular en Gazebo), se puede calcular el Indice de vegetación normalizado –del inglés *Normalized*

Vegetation Difference Index– (NDVI). Una vez elegida la característica que permite determinar la presencia de vegetación, se debe establecer el valor umbral con el que se pretende separar de todo aquello que no es considerado vegetación.

En el caso de la identificación de vegetación con imágenes RGB se optó por realizar una segmentación basado en color, más específicamente en el color verde, pues este es un color característico de muchos cultivos. En primer lugar se realizó un cambio del espacio de color RGB al HSV, pues el canal de tono (*hue*) es ideal para la identificación de color, luego se establecieron los umbrales que se registran en la tabla 4-1, los cuales al ser aplicados a la imagen HSV, generan la máscara de segmentación. Aunque OpenCV cuenta con la función para realizar el cambio de espacio de color y la umbralización simultánea en la *CPU* de los tres canales, esta función no se encuentra implementada para la *GPU* por lo que se hace necesario hacer la umbralización canal por canal de forma independiente.

El umbral de tono se definió entre los rangos indicados en la tabla 4-1 dado que el espectro de color verde se encuentra centrado a 120° y eligiendo un rango de ±40° se cubre la mayoría de dicho espectro. Por otro lado los umbrales inferiores de los canales de saturación y valor se eligieron para excluir el color blanco y negro respectivamente.

Canal	Mínimo	Máximo
Tono (*Hue*)	80°	160°
Saturación (*Saturation*)	100	255
Valor (*Value*)	50	255

Tabla 4-1: Umbrales de segmentación por color.

En cuanto a las cámaras multiespectrales, se realizó el ejercicio de obtener las imágenes NDVI a través de la ecuación 4-1, y hacer la segmetación de vegetación basados en esta característica. Para ello se utilizó un set de imágenes tomado del repositorio de Micasense[1], empresa que fabrica las cámaras RedEdge y Atum, quienes indican que por lo general las zonas cuyo valor de NDVI se encuentra entre un 50 % y un 99.5 %, pueden ser definida como zonas de vegetación. Los resultados se muestran en las figuras 5-3 y 5-3 con el fin de resaltar las ventajas de utilizar este tipo de cámara ya que simplifica la carga computacional al procesarse únicamente un canal y aumenta la eficiencia en la detección de vegetación, sin embargo los demás procesos

[1]Repositorio Micasense: https://github.com/micasense

se realizaron con base en la máscara obtenida con la segmentación de color, pues es la única compatible con el sensor del simulador.

$$NDVI = \frac{NIR - RED}{NIR + RED} \tag{4-1}$$

4.2.2. Identificación de la orientación de los surcos

OpenCV tiene implementado el algoritmo de la transformada de Hough, que sirve para identificar las líneas presentes en una imagen, así como su orientación con respecto al origen de la misma, lo cual resulta bastante útil al momento de identificar los surcos a partir de registros fotográficos [67, 68, 69]. Si bien en la literatura se encuentran trabajos donde se implementan técnicas como máquinas de soporte vectorial o redes neuronales para realizar la identificación de los cultivos o surcos, se decidió no implementarlas partiendo de la premisa que el interés no es identificar cada surco de manera individual, sino en cambio identificar de manera general las características de orientación y límites del cultivo, para lo cual la transformada de Hough hasta donde conoce el autor es la alternativa más sencilla y que entrega resultados eficientes.

Ahora, en la máscara obtenida en el proceso de segmentación dependiendo de qué tan anchos sean los surcos del cultivo se pueden identificar líneas falsas que no reflejan la orientación real de los surcos del cultivo. Para evitar esta problemática es necesario obtener el esqueleto (*skeleton*) de los surcos. Esto es básicamente un proceso en el que se busca eliminar parte del primer plano de la máscara, sin deshacerse de los detalles de la forma de los elementos que lo conforman [70, 70, 71, 72], en el caso de los surcos reducirá el ancho de los mismos hasta el punto en que cada uno se asemeje a una línea. Esto se logra a través de un proceso iterativo donde se ejecutan operaciones morfológicas y aritméticas sobre la máscara o imagen binaria siguiendo la secuencia mostrada en la figura 4-4.

Tan pronto como se obtiene el esqueleto de los surcos se procede a aplicar el algoritmo de la transformada de Hough[73], el cual entrega un arreglo con las líneas identificadas, las cuales son definidas en coordenadas polares, es decir mediante la combinación de radio y ángulo[74, 75]. A continuación, se toma el valor correspondiente al ángulo de cada uno de los elementos del arreglo y se calcula la moda, este valor es el que se asociará como la orientación de los surcos del cultivo.

La razón para elegir la moda yace en que representa la tendencia de los elementos, en este caso la orientación de las líneas y dentro del contexto de un cultivo, si bien en la práctica no todos los surcos irán en la misma dirección si se tendría una tendencia. En cambio, al utilizar otro criterio como por ejemplo la media, puede suceder que elementos diferentes al del cultivo de interés, (surcos de cultivos contiguos, vegetación,

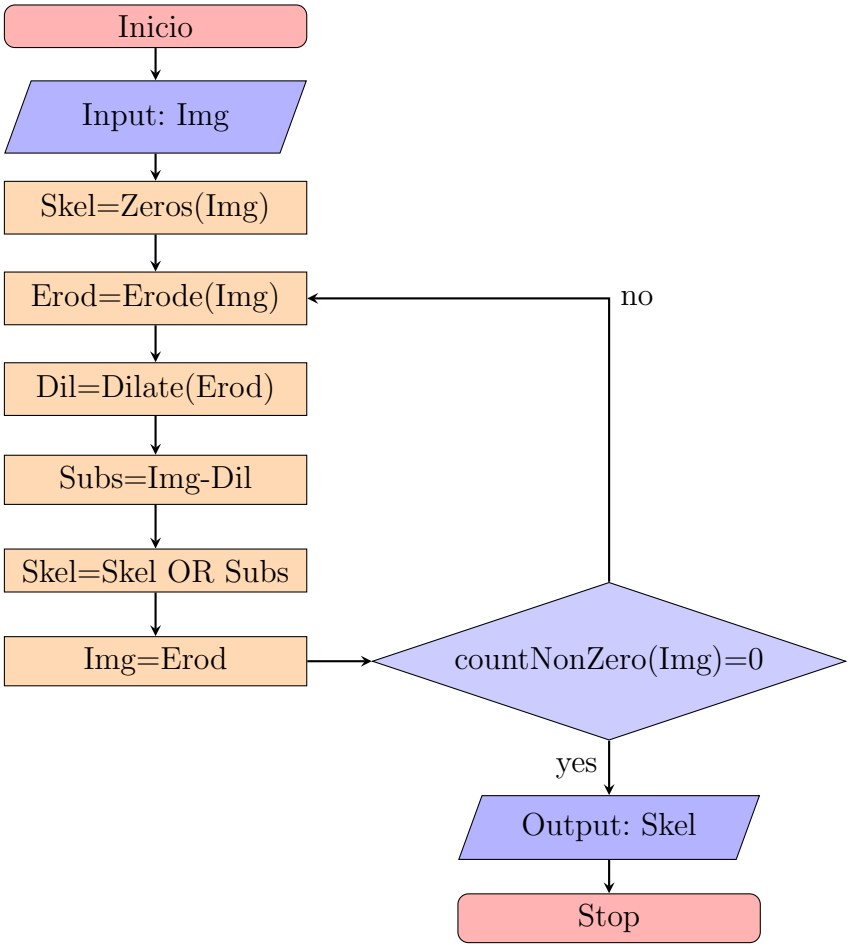


Figura 4-4: Secuencia para obtener el esqueleto.

bordes) alteren la aproximación asociada de la orientación de los cultivos.

4.2.3. Identificación del contorno

Al igual que en la identificación de la orientación de los surcos, a partir de la máscara de segmentación se aplican operaciones morfológicas, pero en este caso se busca expandir los surcos de manera que se solapen entre sí. Para ello se aplica en primera instancia una operación de dilatación, luego a este resultado se le aplica una operación de apertura, de esta manera se obtiene una máscara en la que se puede distinguir el contorno del cultivo, pero aun hace falta establecer las coordenadas de los puntos que definen tal contorno.

Afortunadamente OpenCV cuenta con una función que se encarga de realizar tal operación, cuyo resultado es un arreglo con los contornos identificados en la máscara,

cada uno de ellos se define como una secuencia de puntos. El siguiente paso consiste en establecer cuál es el contorno que más información proporciona del cultivo, para lo cual se calcula el área de cada uno de los elementos del arreglo y se define como contorno del cultivo aquel cuya área sea la mayor.

Por último, debido a las operaciones morfológicas los contornos identificados pueden tener formas irregulares, lo cual incrementa considerablemente el número de puntos que definen cada contorno. Teniendo en cuenta lo anterior, el último paso consiste en reducir tal cantidad, para lo cual existe una función que se encarga de aproximar un contorno a un polígono con un número mucho menor de puntos, de esta manera se concluye el proceso de identificación de características.

4.2.4. Implementación en ROS

Debido a que la identificación de características es un proceso que no requiere ser ejecutado de forma continua y se puede llegar a realizar por demanda en un momento específico, es acertado afirmar que su implementación en ROS coincide con la de un servicio. Con esto en mente, se creó el paquete *crop_image_processing*, en el que se define un nodo dentro del cual se cuenta con un suscriptor y dos servicios. La suscripción se realiza al tema en el que se publica la imagen captada por la cámara; uno de los servicios permite almacenar la imagen disponible en el momento que sea solicitado el servicio y especificar la ruta donde se desee almacenar la imagen (lo cual resulta útil en el momento de la ejecución del plan).

El segundo servicio se encarga de capturar una imagen y a partir de ella ejecutar los procesos de identificación de características descritas en esta sección apoyandose en las librerías de OpenCV. Al igual que en el primer servicio se debe especificar la ruta donde se almacenaran los resultados, mientras que la respuesta de este servicio consta del valor que define la orientación de los surcos, un vector en el que se almacenan los puntos del contorno y una imagen del cultivo donde se resaltan los elementos identificados. En la figura 4-5 se muestra el grafo que representa el nodo asociado a este paquete.

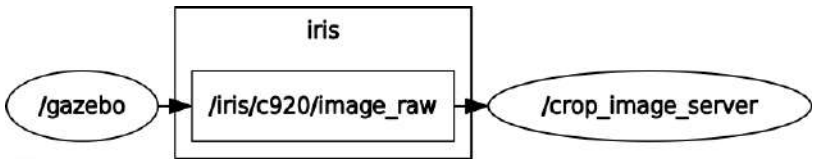


Figura 4-5: Nodo *crop_image_server*.

4.3. Control y posicionamiento del vehículo

Al igual que en la mayoría de vehículos aéreos no tripulados el autopiloto Pixhawk 2 cuenta con múltiples modos de vuelo para realizar su posicionamiento. Los más usados son el modo manual en el que el vehículo se opera a través del sistema de radio control; y el modo misión donde el dron opera de forma automática siguiendo una serie de *Waypoints* cargados previamente, lo cual por lo general implica conocimiento del área de trabajo. Sin embargo, Pixhawk 2 cuenta con el modo *offboard*, en el que una aplicación o programa ejecutado por un sistema externo, como un computador de compañía, se encarga de realizar el posicionamiento del vehículo a través de comandos específicos.

El sistema autopiloto en modo *offbaord* puede ser maniobrado bien sea a través de comandos de velocidad, comúnmente usados en proyectos de robótica asociados con el control y posicionamiento de un robot en un espacio de trabajo donde puede haber obstáculos. Asimismo el autopiloto también puede ser operado por medio de *setpoints* de posición y velocidad, los cuales pueden ser indicados tanto en el *frame* local, como en el *frame* global; vale la pena indicar que al trabajar en el *frame* local se debe tener en cuenta que el origen se define con base en las coordenadas GPS del lugar donde el vehículo es ubicado al momento de ser activado, mientras que para el *frame* global es necesario definir si la altura con la que se operará es relativa respecto el nivel del mar o relativa respecto al plano xy del *frame* local.

Debido a que el objetivo principal del proyecto es la planificación de las trayectorias y no el control de posicionamiento del vehículo, además del hecho que el entorno de trabajo donde este operaría es un espacio libre de obstáculos, se decidió realizar el posicionamiento por medio del sistema de *setpoints* en el que los algoritmos de control son implementados y ejecutados directamente en el *firmware* del autopiloto.

Ahora, si bien el autopilto es capaz de posicionar el dron en un *setpoint* indicado es necesario que la aplicación *offboard* tenga la capacidad de identificar cuándo se da esta condición para que se decidan y ejecuten las siguientes acciones. Para ello se cuenta con las lecturas de los sensores que para el caso del proyecto están disponibles a través de tópicos proveídos por los nodos de MAVROS. Estos tópicos son usados para monitorear dos tipos particulares de posición, la correspondiente al despegue y la de un punto de control del plan de vuelo.

En el momento del despegue únicamente es necesario hacer el seguimiento de la altura, que puede ser determinada a través de la lectura del GPS. En cambio, para un punto de control es necesario hacer seguimiento de la posición, a través del GPS, así como la orientación, donde se tiene en cuenta la *IMU* y el magnetómetro, pues uno de los objetivos es que el dron recorra los cultivos siguiendo la orientación de los surcos.

4.3.1. Implementación ROS

Teniendo en cuenta lo anterior se diseñó el paquete *mavros_offboard_control*, el cual está conformado por tres nodos. El primero se encarga de validar el despegue, en él se establece un umbral de aceptación alrededor de la altura deseada y se considera que el despegue ha sido exitoso si el vehículo se mantiene dentro de dicho umbral luego de tres segundos, pues de esta manera se puede tener mayor seguridad de que el vehículo se estabilizo adecuadamente en la altura indicada. El nodo se encuentra suscrito a la posición global, así como a la posición local, de donde se extrae la información que permite realizar la validación de altura. Este proceso se ofrece a través de un servicio, pues es una tarea que se puede ejecutar por demanda tan pronto como se indique al vehículo iniciar la operación del sistema.

El segundo nodo se encarga de validar si el dron ha alcanzado satisfactoriamente un punto de control. Al igual que en el nodo de despegue se requieren umbrales de aceptación, en este caso uno para la posición y otro para la orientación. La diferencia radica en que la posición será considerada valida si se encuentra dentro de una esfera con un radio menor al umbral de aceptación, allí la librería *Geographiclib* resulta de bastante utilidad pues cuenta con una función para determinar la distancia sobre el plano xy a partir de los valores de latitud y longitud del punto de inicio y del punto final, mientras que la componente z de la distancia se puede determinar a través de la altura registrada en el GPS. En cuanto a la orientación, se compara que el ángulo de *heading* del dron se encuentre alrededor de la orientación deseada, dentro del umbral establecido. Este nodo también tiene en cuenta información del GPS, el magnetómetro y la IMU para realizar los cálculos que permiten establecer si el punto de control fue alcanzado. Este proceso de validación es implementado por medio de un servicio en cuya solicitud se indica la posición deseada y la posición actual del vehículo, así como los umbrales de aceptación de distancia y orientación que el usuario considere adecuados; la respuesta indica si la posición deseada fue o no alcanzada, así como la distancia y ángulo de giro restantes para completar el posicionamiento.

Finalmente, el nodo más importante es donde se implementa la secuencia de control del *UAV*. Este nodo se suscribe a múltiples tópicos como el GPS, la IMU, el magnetómetro y el estado del vehículo, con el fin de utilizar dicha información para llevar a cabo las labores de posicionamiento y planificación, donde básicamente se implementa la secuencia presentada en la figura 3-1. Para lograr tal propósito este nodo se apoya en los servicios ofrecidos por los paquetes *mavros_offboard_control*, *crop_image_processing* y *flight_planning*; con base en los resultados obtenidos en cada servicio se va avanzando en la secuencia y se publica en los tópicos relacionados a los *setpoints* de posición locales y globales. A continuación, se presenta una explicación

más detallada de la secuencia.

El primer paso consiste en monitorear constantemente el tópico del estado del vehículo (*mavros/state*), que incluye la información del modo de vuelo del dron, de manera que tan pronto como se identifique que se ha configurado en el modo *offboard* (bien sea por una orden de consola o mediante el radio control), el vehículo empieza la secuencia de despegue. Periódicamente se llama el servicio de despegue hasta que se obtenga un resultado exitoso, momento en el que es solicitado el servicio de identificación de características.

Si el servicio de identificación de características no es exitoso, se aplica una política de reintentos (un número fijo de tres veces) y si se supera ese número se procede a terminar la misión, enviando un comando para que el dron vuelva a la posición donde despegó. Si en cambio el proceso de identificación de características es exitoso se almacenan los resultados asociados a las características identificadas en la imagen (orientación de surcos, contorno en el frame de la imagen), así como los datos de posicionamiento en el momento de la captura (GPS, orientación, etc). Estos son usados para solicitar el servicio de planificación de trayectorias.

Una vez que se termina de ejecutar el servicio de planificación de trayectorias y se cuenta con un plan de vuelo, se publica en el tópico del *setpoint* de posición cada uno de los puntos de control generados en el orden establecido por el plan; cada uno de estos puntos de control son validados periódicamente por el servicio de verificación de *waypoints*. Si el punto de control es alcanzado el dron se detiene para tomar una foto a través del servicio del paquete *crop_image_processing* definido para tal tarea, para luego cambiar el *setpoint* de posición y así continuar con el siguiente punto de control.

Tan pronto como el plan de vuelo ha sido ejecutado en su totalidad se cambia el modo de vuelo para que se retorne al punto de lanzamiento donde el control del vehículo puede ser tomado nuevamente por el operador. Adicionalmente, en caso de que alguno de los servicios llamados o procesos ejecutados en este nodo terminara en error, el modo de vuelo se cambia para que el vehículo retorne al punto de lanzamiento. En la figura 4-6 se muestra la representación por grafos de los nodos asociados a este paquete.

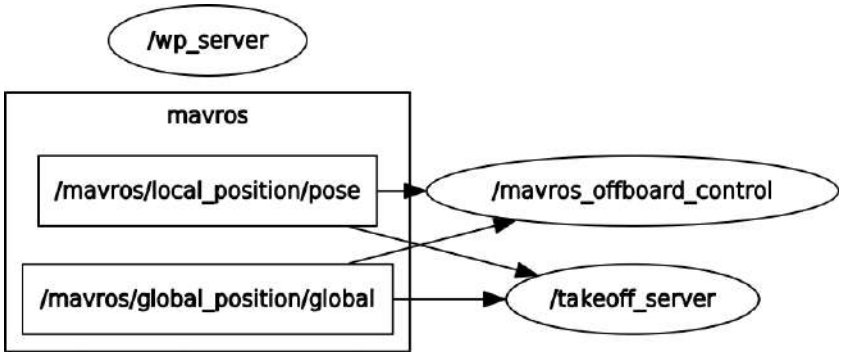


Figura 4-6: Nodos paquete *mavros_offboard_control*.

4.4. Algoritmo de planificación de trayectorias

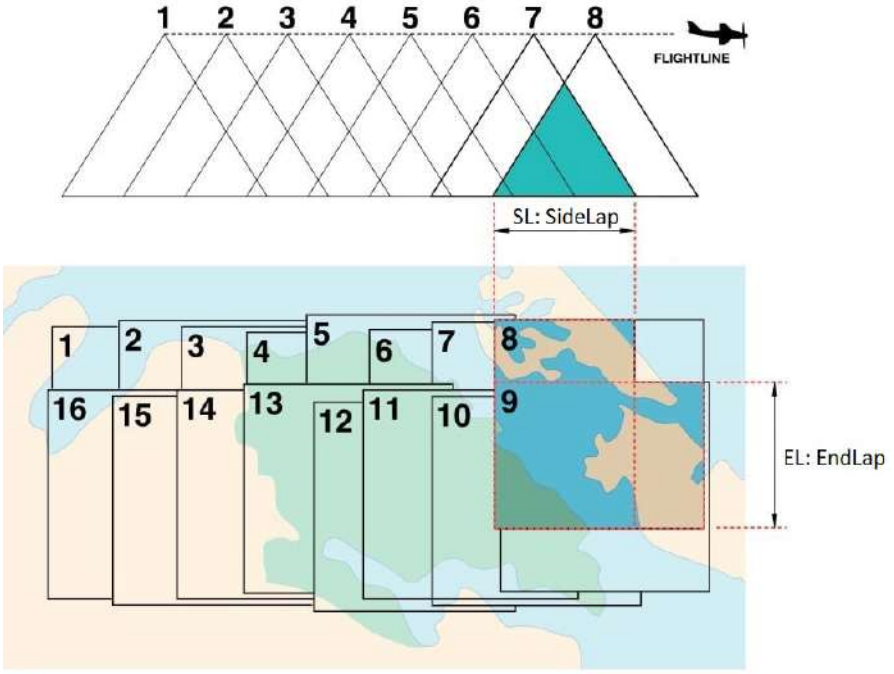
Partiendo del hecho de que la premisa del algoritmo de planificación de trayectorias para *UAVs* en entornos agrícolas es garantizar las condiciones fotogramétricas que permitan generar un fotomosaico de calidad, los principales insumos del algoritmo son el GSD (representa la distancia física entre los centros de dos pixeles adyacentes y es directamente proporcional a la altura de vuelo), y los porcentajes de solape lateral y frontal entre fotos, comúnmente denominados *sidelap* y *endlap* respectivamente; la figura 4-7 muestra una representación gráfica de las variables fotográmetricas y su relación. Adicionalmente, el algoritmo debe conocer las características del cultivo (orientación de los surcos y contorno del límite del cultivo), así como la información geoespacial y de orientación del vehículo en el momento que realizo la captura fotográfica que permitió identificar dichas características. Con base en estos datos el algoritmo se divide en los siguientes momentos principales: Determinación de variables fotogramétricas, obtención de contornos y vector de orientación en el *frame* de navegación, determinación punto de arranque y número de líneas de vuelo y generación de puntos de control.

$$GSD = \frac{FH \times SW}{FL \times IW} \tag{4-2}$$

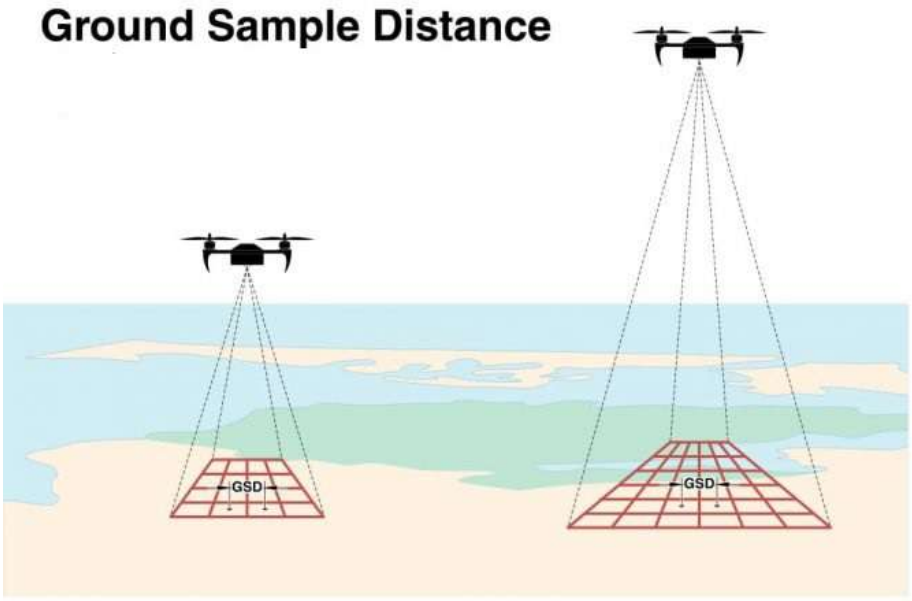
En la ecuación 4-2 se define el GSD en donde FH es la altura de vuelo (*flight height*), SW es el ancho del sensor (*sensor width*), FL es la distancia focal (*focal length*) y IW indica el ancho de la imagen en pixeles (*image width*).

4.4.1. Determinación variables fotogramétricas

El primer paso para poder generar un plan de vuelo fotogramétrico es determinar la distancia que debe haber entre fotos consecutivas para que garanticen el *endlap*



(a) Solapamiento entre fotos.



(b) Ground Sample Distance.

Figura 4-7: Representación gráfica variables fotográficamente.
Tomado de: https://www.aviassist.com.au/improving-accuracy-aerial-survey-drone/

deseado; esta distancia es normalmente conocida como base y está definida mediante la fórmula 4-3, en donde *IH* es la altura de la imagen en pixeles, GSD representa el

Ground Sample Distance y *EL* define el *endlap*.

$$Base = GSD \times IH \times \left[\frac{100 - EL}{100}\right] \tag{4-3}$$

De manera similar se debe calcular la distancia entre líneas de vuelo o distancia de línea (*line distance*) que garantice el *sidelap* (*SL*) deseado, la cual está definida mediante la ecuación 4-4, donde IW indica el ancho de la imagen en pixeles.

$$LineDistance = GSD \times IW \times \left[\frac{100 - SL}{100}\right] \tag{4-4}$$

Finalmente, es necesario hallar la altura a la cual el dron tendrá que sobrevolar el cultivo para que la resolución espacial de las fotos corresponda a la del *GSD* ingresado. Para ello se puede partir de la definición de *GSD*, dada por la ecuación 4-2, donde la altura de vuelo (*flight height*) puede ser fácilmente despejada.

Como se pudo evidenciar, estas variables dependerán directamente de las características de la cámara, las cuales son indicadas en los manuales o especificaciones, sin embargo la distancia focal en ocasiones debe ser identificada a través de un proceso de calibración de cámara, que servirá también para eliminar distorsiones asociadas a los procesos de fabricación, lo cual resulta determinante al trabajar con cámaras multiespectrales pues se trabaja con fotografías generadas por lentes diferentes y que en ocasiones tienen que ser registradas en una sola imagen; para el caso del simulador estas ser modificadas al momento de definir el sensor de la cámara. El algoritmo 1 muestra un resumen de la secuencia de pasos para completar los procesos descritos.

Algoritmo 1: CalculatePhotogrammetricVariables.

Input: GSD, EL,SL, IW,IH,FL,SW
Result: Base, LineDistance,FH
Base=Compute_GSD(Ecuación: 4-3)
LineDistance=Compute_LineDistance(Ecuación: 4-4)
FH=Compute_FH(Ecuación: 4-2)

4.4.2. Obtención de contornos y vector de orientación en el frame de navegación

Debido a que las características del cultivo fueron identificadas dentro del *frame* de la imagen y teniendo en cuenta que la planificación de trayectorias se facilita dentro del *frame* local, es necesario realizar la transformación de estas características al *frame* local. En el caso de las coordenadas de los puntos que conforman el contorno,

resulta fácil porque se puede hacer directamente por medio de las ecuaciones 2-1–2-9, donde se tiene en cuenta los datos de posicionamiento (GPS, IMU, magnetḿetro, etc) almacenados en el momento de la captura.

Por otro lado, la orientación de los surcos esta expresada como un ángulo en el sistema coordenado de la imagen, y para trasladarlo al sistema coordenado del terreno, se definió un vector unitario, que a partir de las transformaciones expresadas en las ecuaciones 2-1–2-9 se convierte en un vector referenciado al sistema coordenado del terreno.

Adicionalmente se hallan los puntos mínimos y máximos en los ejes x y y del contorno del cultivo en el *frame* de mapeo, y a partir de ellos se genera un segundo contorno, similar a un *bounding-box*, el cual es usado en algunos casos especiales de los procesos posteriores. El algoritmo 2 muestra un resumen de la secuencia de pasos para completar los procesos descritos.

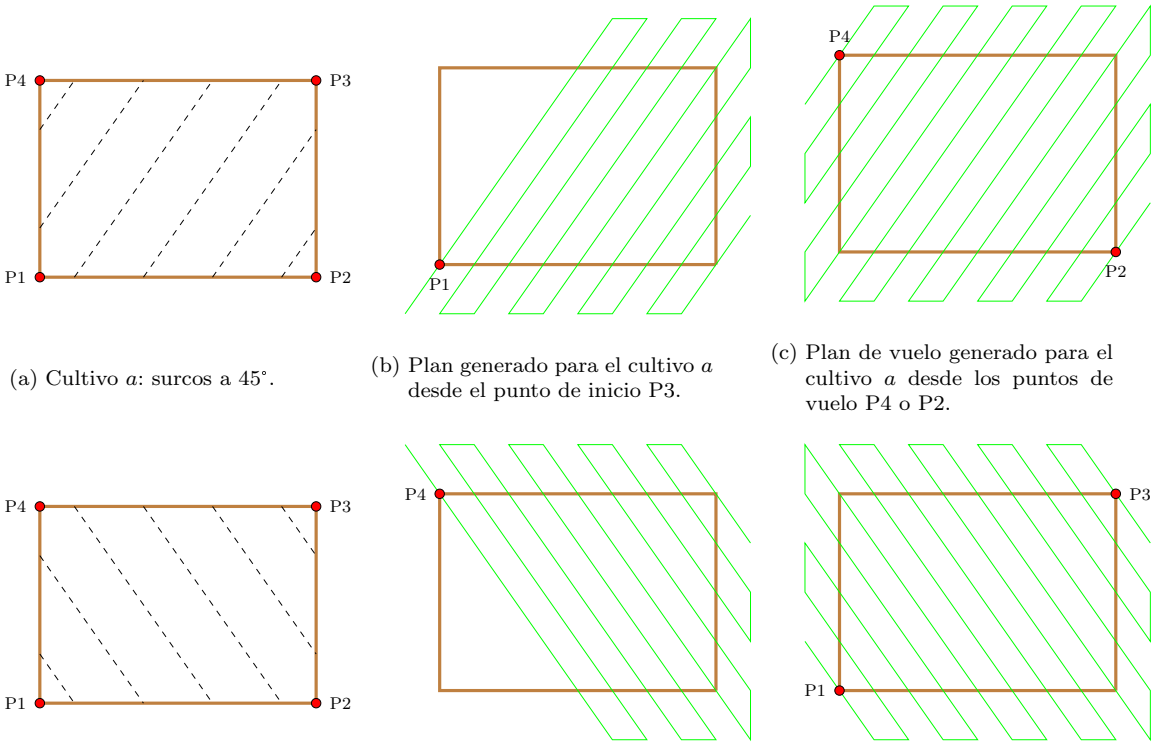
Algoritmo 2: GetContours_and_orientationVector.

Input: CropContour$_{if}$,Base,
CropOrientation$_{if}$,UAV_GPS$_{picture}$,UAV_Heading$_{picture}$,RefGPS
Result: CropContour$_{navf}$,OV$_{navf}$,BoundingBox$_{navf}$
CropContour$_{navf}$=ApplyIF2NAVFTransformation(CropContour$_{if}$,PositioningData)
%(Ecuaciones: 2-1–2-9)
OV$_{navf}$=Compute_OV$_{navf}$(CropOrientation, positioning_data) %(Ecuaciones: 2-1–2-9)
$x_{min}, x_{max}, y_{min}, y_{max}$=FindMinMax(CropContour$_{navf}$)
BoundingBox$_{navf}$=Define_BoundingBox[$(x_{min}, y_{min}), (x_{min}, y_{max}), (x_{max}, y_{max}), (x_{max}, y_{max})$]

4.4.3. Determinación punto de inicio y número de líneas de vuelo

El siguiente paso consiste en la determinación del punto de inicio más optimo del plan de vuelo, que generalmente se ubicará en un vértice del contorno y dependerá considerablemente de la combinación entre la geometría del cultivo y la orientación de los surcos. En la figura 4-8 se muestra un ejemplo simple donde se comprueba que para un cultivo con la misma geometría, pero con orientaciones de surco diferentes el punto de inicio optimo difiere entre sí.

Para resolver tal problemática, se usaron conceptos geométricos sencillos. El primer paso consiste en determinar el centroide del contorno del cultivo, posteriormente se genera una recta cuyo vector director siga la misma dirección que la orientación de



(a) Cultivo a: surcos a 45°.

(b) Plan generado para el cultivo a desde el punto de inicio P3.

(c) Plan de vuelo generado para el cultivo a desde los puntos de vuelo P4 o P2.

(d) Cultivo b: surcos a -45°.

(e) Plan generado para el cultivo b desde el punto P4.

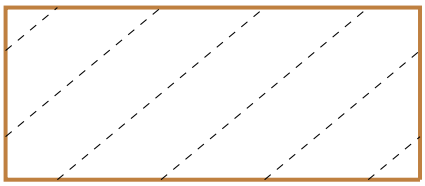
(f) Plan generado para el cultivo b desde el punto P1 o P3.

Figura 4-8: Puntos de inicio y su relación con las características del cultivo.

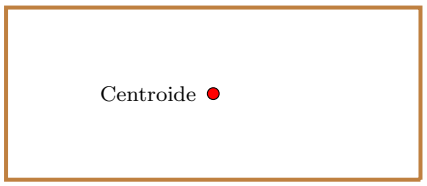
los cultivos y que pase por el centroide. Teniendo en cuenta que el camino más corto de un punto en el espacio hasta una recta es a través de un vector perpendicular al vector director de la recta y que aquel punto del contorno que este más lejos de la recta definida recientemente permitirá abarcar una mayor cantidad de terreno respecto a los otros puntos del contorno, se debe calcular la distancia a cada uno de los puntos del contorno a la recta y encontrar aquel punto donde esta sea la máxima para elegirlo como el punto de inicio óptimo.

En la figura 4-9 se muestra la secuencia gráfica para hallar este punto y se evidencia que siguiendo esta secuencia los puntos P2 y P4 serían los más adecuados como puntos de inicio, lo cual se puede confirmar viendo la figura 4-8, donde el plan de vuelo generado para el cultivo con las mismas características cubre todo el terreno si se inicia desde alguno de ellos.

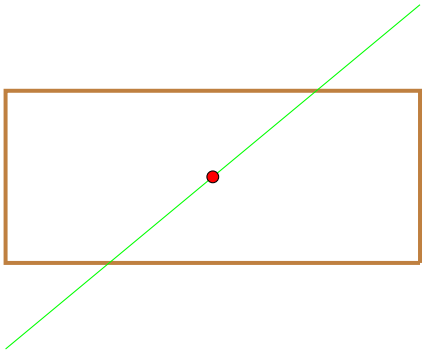
Con el punto de inicio (SP) ahora es posible determinar la cantidad de líneas de vuelo necesarias para recorrer el cultivo. Lo primero es determinar el vector de avance lateral (LAV) a partir del punto inicial, la dirección de este vector debe ser perpendicular a la dirección de avance frontal, es decir que debe ser perpendicular a



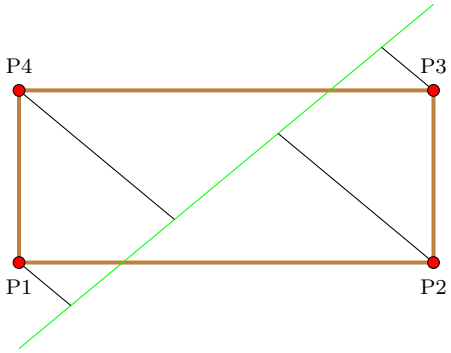
(a) Cultivo original con orientación de surcos a 45°.



(b) Centroide del contorno.



(c) Recta que contiene el centroide con vector director que sigue la orientación delos surcos.



(d) Distancia desde los vértices hasta la recta.

Figura 4-9: Plan generado desde un punto de inicio no optimo.

la dirección de orientación de los surcos, sin embargo, hay dos vectores que cumplen estas condiciones. La manera para decidir cuál es el adecuado es de hecho simple, pues basta con avanzar una distancia arbitraria, como la distancia de línea, desde el punto inicial en la dirección de cada uno de los vectores candidatos y aquel punto que se encuentre dentro del contorno *boundingbox* definirá el vector de avance lateral.

Continuando con la determinación del número de líneas de vuelo, se genera una recta cuyo vector director siga la dirección de los surcos, pero que esta vez pase por el punto hasta donde se avanzó desde el punto inicial, enseguida se establece si esta recta tiene puntos de intersección con el contorno del cultivo, en cuyo caso el número de líneas de vuelo incrementa. Este proceso se repite desde cada nuevo punto hasta el momento en que la recta no tenga ninguna intersección con el contorno del cultivo, momento en el que se sabe el número mínimo de líneas de vuelo que se necesitan, sin

embargo, en la práctica se recomienda incluir una línea adicional para garantizar el cubrimiento del terreno. El algoritmo 3 muestra un resumen de la secuencia de pasos para completar los procesos descritos.

Algoritmo 3: FindStartPoint_and_NumberOfLines.

Input: CropContour$_{navf}$,BoundingBox$_{navf}$,LineDistance, OV$_{navf}$
Result: LAV$_{navf}$,NOL,SP
Centroid=FindCentroid(CropContour)
AuxRect=GenerateRect(OV$_{navf}$,Centroid)
MaxDist=CalculateDistance(AuxRect,CropContour$_{navf}$[0])
SP=CropContour$_{navf}$[0]
for $i \leftarrow 1$ **to** *Size(CropContour$_{navf}$)* **do**
    if *CalculateDistance(AuxRect,CropContour$_{navf}$[i])¿MaxDistance* **then**
        MaxDistance=CalculateDistance(AuxRect,CropContour$_{navf}$[i])
        SP=CropContour$_{navf}$[i]
    end
end
LAV$_{navf}$=FindLAV(OV$_{navf}$,LineDistance)
AuxRect=MoveRect(AuxRect,LAV$_{navf}$)
NOL=1
while *FindInterception(AuxRect,CropContour$_{navf}$* **do**
    NOL=NOL+1
    AuxRect=MoveRect(AuxRect,LAV$_{navf}$)
end

4.4.4. Generación de puntos de control

Luego de saber cuántas líneas de vuelo deben ser generadas es momento de calcular los puntos de control que serán utilizados para tomar las fotografía en cada una de las líneas encontradas en el paso anterior. Para llevar a cabo esta tarea se deben identificar tres elementos fundamentales que son el punto inicial de línea (ILP), la longitud de la línea (LL) y el vector de avance frontal (FAV). La longitud de la línea sirve para estimar el número de fotografías (number of photographies NOP) por línea de vuelo, lo cual se logra mediante la ecuación 4-5, donde LL es la longitud de la línea y AP es un número arbitrario de puntos adicionales, que por lo general son recomendados para alinear el dron con la línea de vuelo, estos son ubicados al inicio y final de cada línea.

$$NOP = \frac{LL}{base} + AP \tag{4-5}$$

Inicialización primera línea

Aunque en el paso anterior se encontró el punto de inicio que permite recorrer el cultivo de forma óptima, la inclusión de los puntos adicionales implican que el punto inicial de la línea (ILP) se vea desplazado, si n_{ap} es el número de puntos de control adicionales, ILP se define mediante la expresión 4-6.

$$\begin{aligned} ILP_x &= SP_x - \frac{n_{ap}}{2} FAV_x \\ ILP_y &= SP_y - \frac{n_{ap}}{2} FAV_y \end{aligned} \tag{4-6}$$

$$(ILP_x, ILP_y) = \left(SP_x - \frac{n_{ap}}{2} FAV_x, SP_y - \frac{n_{ap}}{2} FAV_y\right) \tag{4-7}$$

De manera que hace falta hallar el vector de avance frontal y la longitud. Para ello se genera una recta que contenga el punto SP y cuyo vector director siga la misma dirección que el vector de orientación de los surcos, luego se hallan las intersecciones de esta recta con el contorno *bounding-box*. La razón para elegir este contorno y no el del cultivo es que para la primera línea existen casos donde solo habrá una intersección con este contorno, de manera que la primera línea quedaría conformada por el punto inicial y los puntos adicionales, lo que podría implicar perdida de información del cultivo. A continuación, se genera un vector que vaya desde el punto inicial hasta la intersección más lejana y la magnitud de dicho vector es definida como la longitud de la línea o LL. Enseguida, la versión normalizada de dicho vector se multiplica por un valor igual a la base, obteniendo así el vector de avance frontal. Con estos datos se puede calcular el número de puntos de control de la línea y las coordenadas de cada punto de control wp_i estarán definidas mediante la expresión 4-8, teniendo en cuenta que $wp(0)$ es igual a ILP.

$$(wp_x(i), wp_y(i)) = (wp_x(i-1) + FAV_x, wp_y(i-1) + FAV_y) \quad 1 \leq i \leq NOP \tag{4-8}$$

Ahora, aunque el uso del contorno *bounding-box* en la primera línea de vuelo contribuye a que no se excluya información relevante del cultivo, también podría generar puntos cuya información no resulte útil, por lo cual a medida que se va encontrando un nuevo punto de control, se calcula la distancia de este punto al contorno del cultivo y si esta supera un umbral arbitrario, que para el caso del proyecto se eligió como la distancia de línea, el punto sera descartado. De esta manera

se busca reducir la inclusión de puntos innecesarios que implicarían mayor tiempo de vuelo y almacenamiento.

Tan pronto como se terminen de calcular las coordenadas de los puntos para la primera línea, la recta que se usó para hallar las intersecciones con el *boumding-box* se desplaza una distancia igual a la magnitud del vector de avance lateral, en la dirección de dicho vector, de tal manera que queda ubicada en el lugar de la siguiente línea de vuelo.

Generación puntos de control líneas restantes

Le generación de los puntos de control para las demás líneas se realiza por medio de un proceso iterativo. En cada una de las iteraciones el primer paso consistirá en determinar el nuevo punto de inicio, para ello se halla el número de intersecciones que tiene la recta definida anteriormente con el contorno del cultivo y con base en el resultado se definen los siguientes casos.

- **Número de intersecciones >1:** En este caso se calcula la distancia desde el último punto generado en la línea de vuelo anterior (LGP) a cada uno de los puntos de intersección y aquel que le corresponda la menor distancia definirá las nuevas coordenadas de SP. LL sera definido como la distancia entre SP y el punto de intersección más lejano mientras que la dirección de av tendrá que coincidir con la del vector que se forma SP y el punto de intersección más distante.

- **Número de intersecciones =1:** El que exista una única intersección entre la recta y el contorno del cultivo significa que la línea de vuelo pasará por un vértice del contorno, lo cual es similar al caso que se tiene para la primera línea. El punto de intersección es almacenado temporalmente como IP_{crop} mientras se hallan las intersecciones de la recta con el contorno *bounding-box*.

  Si hay más de un punto de intersección, se calcula la distancia de cada uno hasta IP_{crop} y se define $IP_{bounding-box}$ como aquel punto en el que esta distancia es la mayor, así mismo esta distancia definirá LL. Luego, si la distancia entre IP_{crop} y el último elemento de la línea anterior es menor a la distancia entre $IP_{bounding-box}$ y el último elemento, IP_{crop} se convierte en SP, mientras que la dirección de avance debe coincidir con la del vector que se forma desde IP_{crop} hasta $IP_{bounding-box}$. En caso contrario SP se definirá con base en $IP_{bounding-box}$ y la dirección de avance ira en el otro sentido.

  Si en cambio hay menos de dos intersecciones, se optó por repetir el patrón de la línea anterior. Para ello se mantiene el valor de LL, SP se define a partir del

último punto generado en la línea anterior desplazado por el vector de avance lateral y se cambia el sentido de av, es decir se rota 180°, respecto a la dirección de avance en la línea anterior.

- **Número de intersecciones = 0:** Si no existen intersecciones con el contorno del cultivo, se usa el contorno *boounding-box* para hallar SP, av y LL. Nuevamente se calculan las intersecciones de la recta con el contorno *boounding-box*. Si existe más de una intersección se ejecuta el mismo procedimiento explicado en el primer item, en caso contrario se repite el patrón de la línea anterior como se explicó en el item 2.

A partir de los resultados obtenidos en el caso correspondiente es posible usar las expresiones 4-6 y 4-8 para generar los puntos de control de la línea actual, de la misma manera como se describió para la primera línea, es decir que a partir de ILP se va desplazando hasta cada punto usando el vector av y se determina si el punto hallado está a una distancia adecuada del cultivo para que sea agregado al plan de vuelo. Una vez que se ha finalizado dicho proceso, la recta se traslada nuevamente con el vector de avance lateral y se empieza una nueva iteración, proceso que se repite hasta cuando se haya generado los puntos de cada línea.

El último paso del algoritmo de planificación consiste en realizar la georreferenciación de los puntos de control. Debido a que la labor de planificación de trayectorias se hace sobre el *frame* de navegación, es necesario ejecutar una serie de transformaciones para obtener las coordenadas GPS asociadas a cada punto de control, como se mencionó en el la sección 2.5, pues es la forma estándar de definir el posicionamiento de un vehículo aéreo. Afortunadamente mediante el paquete *GeographicLib* [76], esta tarea se simplifica porque cuenta con un conjunto de funciones para realizar las operaciones de transformación de coordenadas entre el sistema geodésico y el *frame* local y viceversa. El algoritmo 4 muestra un resumen de la secuencia de pasos para completar los procesos descritos.

El algoritmo 5 muestra la secuencia de pasos más general asociada a la planificación de trayectorias, en donde se integran cada uno de las funciones y algoritmos descritos a lo largo de esta sección.

4.4.5. Implementación en ROS

El proceso de planificación de trayectorias fue implementado con el paquete de ROS *flight_planning*. En este paquete se creó un nodo que cuenta con dos servicios, uno para la georreferenciación de puntos de una imagen y el otro donde se ejecuta el proceso completo de planificación.

Algoritmo 4: GeneratePoints.

Input: LAV$_{navf}$,NOL,SP,CropContour$_{navf}$,OV$_{navf}$,BoundingBox$_{navf}$,RefGPS,FH
Result: WaypointS[],UAV_Heading[]
$ILP = SP - \frac{n_{ap}}{2} * FAV$ AuxRect=GenerateRect(SP,OV$_{navf}$)
Interceptions=FindInterc(BoundingBox$_{navf}$,AuxRect)
EndPoint=FindFurthestPoint(SP,BoundingBox$_{navf}$)
LL=Distance(EndPoint,SP)
$NOP = \frac{LL}{Base} + AP$
LineHeading=ComputeOrientation(EndPoint,SP,OV)
wp[0]=SP
UAV_Heading[0]=LineHeading;
for $i \leftarrow 1$ **to** NOP **do**
    auxPoint=wp[i-1]+FAV
    if *Distance(auxPoint,CropContour$_{navf}$)* **then**
        append(wp,auxPoint)
        append(UAV_heading,orientation)
    end
end
AuxRect=moveRect(AuxRect,LAV)
for $k \leftarrow 1$ **to** NOL **do**
    Interceptions=FindInterc(CropContour$_{navf}$,AuxRect)
    FindSP_EndPoint(Interceptions,CropContour$_{navf}$,BoundingBox$_{navf}$,wp.last)
    LL=Distance(EndPoint,SP)
    $NOP = \frac{LL}{Base} + AP$
    LineHeading=ComputeOrientation(EndPoint,SP,OV)
    wp$_{navf}$[size(wp$_{navf}$)]=SP
    UAV_Heading[size(wp$_{navf}$)]=LineHeading;
    for $i \leftarrow 1$ **to** NOP **do**
        auxPoint=wp$_{navf}$[i-1]+FAV
        if *Distance(auxPoint,CropContour$_{navf}$)* **then**
            append(wp$_{navf}$,auxPoint)
            append(UAV_heading,orientation)
        end
    end
end
wp$_{if}$=applyNAVF2ECEF(RefGPS)

Se incluyó la georreferenciación para los puntos de una imagen como un servicio en caso que por ejemplo, se quiera obtener las coordenadas GPS de los puntos que definen el contorno o cualquier otro punto de interés identificado en una imagen. La solicitud del servicio debe incluir las características de la cámara utilizada, los datos de orientación y posición en el momento de realizar la captura (GPS, IMU,

Algoritmo 5: Algoritmo de planificación.

Input: Desired Photogrammetric variables (GSD, EL and SL), Image and camera features (IW,IH,SW,FL), GeoSpatial data (GPS, Attitude, heading)
Result: Waypoints$_{GPS}$[]
LineDistance, Base=CalculatePhotogrammetricVariables(()
BoundingBox$_{navf}$,crop_contour$_{navf}$,oriantation_vetor$_{navf}$=GetContours_and_OrientationVector()
SP, NOL=FindStartPoint_and_NumberOfLines()
Waypoints$_{GPS}$[]=GeneratePoints()

magnetómetro) y un arreglo con las coordenadas de la imagen que se van a georreferenciar. La respuesta del servicio contiene el arreglo con las coordenadas GPSde los puntos de interés.

En el caso del servicio de planificación de trayectorias, la solicitud esta conformada por los mismos elementos que el servicio de georreferenciación, así como el GSD, solape lateral y solape frontal de interés. La respuesta del servicio consta de un arreglo con los puntos de control expresados en coordenadas GPS, un arreglo de igual tamaño con la orientación que debe asumir el dron en cada punto, las coordenadas GPS de los puntos del contorno y el área estimada del contorno del cultivo.

Figura 4-10: Nodo Paquete *flight_planning*.

5. Resultados y discusión

Una vez implementados los paquetes de ROS descritos en el capítulo 4, se procedió a realizar las pruebas necesarias para validar el funcionamiento individual de cada uno de los paquetes, así como el funcionamiento en conjunto de todos ellos, es decir cuando todos los elementos interactuaban al tiempo entre sí. Las figuras 5-1 y 5-1 muestran la representación por medio de grafos de todos los paquetes cuando se encuentran activos y de los tópicos con los cuales se comunican los nodos.

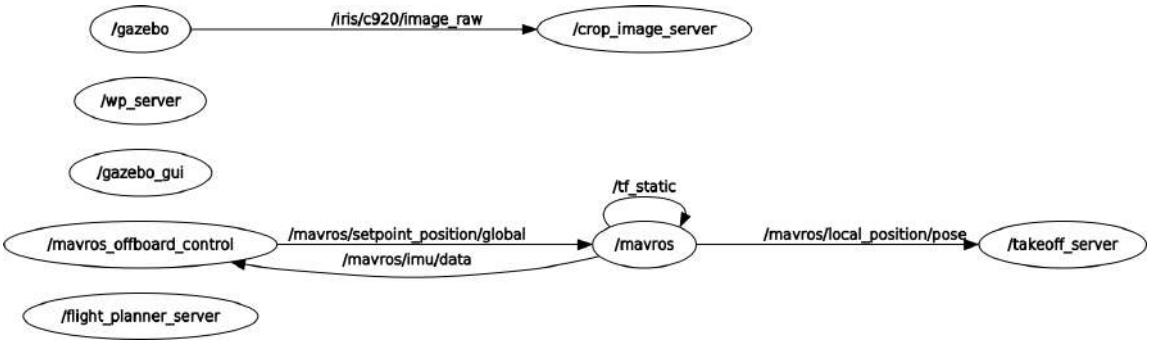


Figura 5-1: Nodos activos durante la ejecución de la simulación.

El primer paquete que se pusó a prueba fue el de posicionamiento y control, en donde a través de los servicios del paquete MAVROS se comprobó que se pudiera cambiar el modo de vuelo y armar el vehículo, luego, para el posicionamiento del dron se proporcionaban las coordenadas de los *setpoints* tanto en el *frame* local como el el *frame* global. En todos los casos se obtuvieron resultados positivos, es decir que fue posible ejecutar las secuencias básicas de operación (cambio de modo de vuelo, armado, y posicionamiento) a través de comandos emitidos por medio de ROS para ubicar el dron a la altura de despegue para capturar la fotografía del cultivo o en algún *waypoints* arbitrario. Comprobando de esta manera el comportamiento de los nodos descritos en la sección 4 para el posicionamiento y control.

En cuanto al paquete asociado al procesamiento de imágenes, la primera prueba consistió en comprobar la capacidad de identificar las zonas con vegetación. En las

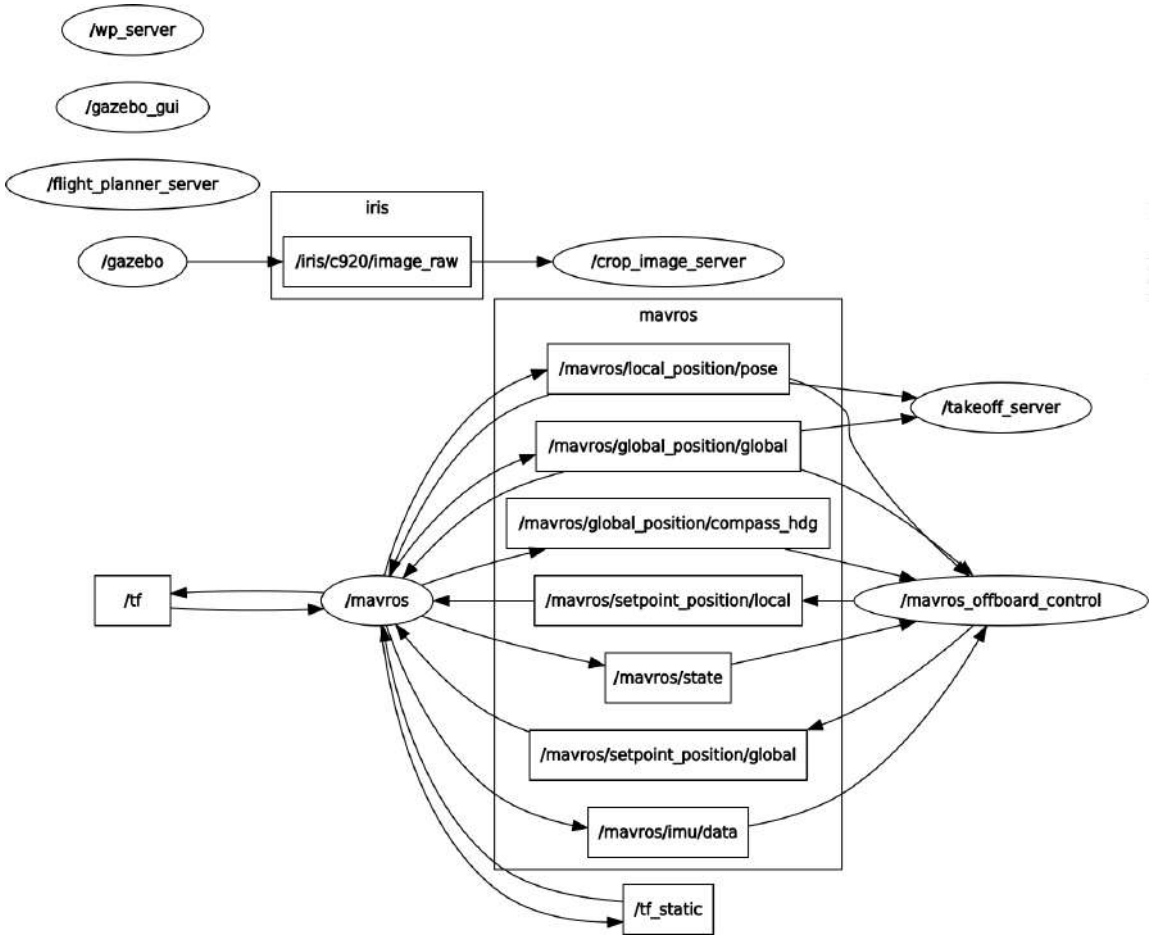


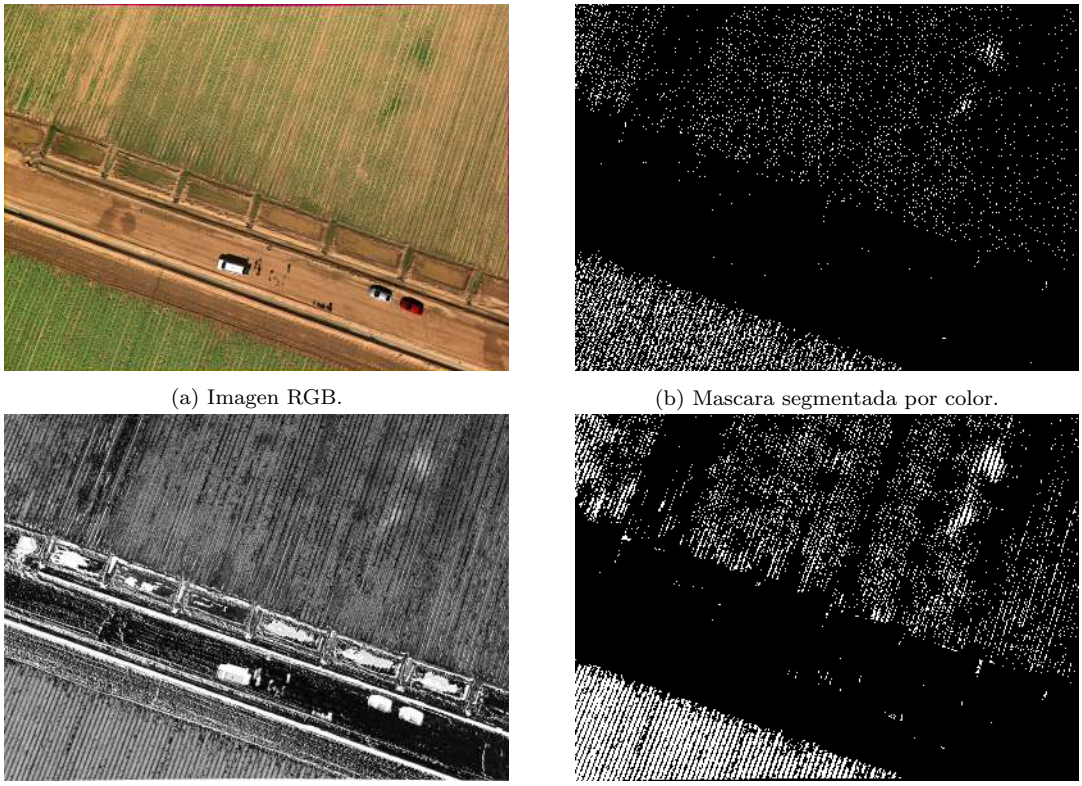
Figura 5-2: Nodos y tópicos activos durante la ejecución de la simulación.

figuras 5-3 y 5-4, la imagen (a) es el resultado de hacer la composición de las imágenes de las bandas Red, Green y Blue de una toma aérea sobre un cultivo con una cámara multiespectral Atum, a la cual luego de aplicar la segmentación de color se obtiene como resultado la imagen (b). Por otro lado, la imagen (c) es la imagen NDVI, obtenida a partir de la imágenes individuales de las bandas NIR y Red, de la cual se obtiene la imagen (d) tras la aplicación de la segmentación basado en NDVI.

Para la figura 5-3 se observa que los dos tipos de segmentación funcionan adecuadamente al momento de identificar las zonas de vegetación. Se observan diferencias leves, pero se puede observar que con la segmentación basada en NDVI el nivel de detalle es mayor, es decir que se identifican más zonas que corresponden a vegetación, lo cual se hace más evidente al comparar los resultados obtenidos en la 5-4.

Sin embargo, la segmentación por color presenta resultados bastante buenos al momento de identificar la vegetación, pues al comparar las zonas de vegetación de la máscara basada en color y compararla con la imagen RGB, se observa que se

lográ identificar casi la totalidad de la vegetación. Teniendo en cuenta lo anterior y como se mencionó anteriormente, por factores de compatibilidad con el simulador, la segmentación basada en color fue la que se usó como base para los demás procesos y resultados que se expondrán en el resto de esta sección.



(a) Imagen RGB.

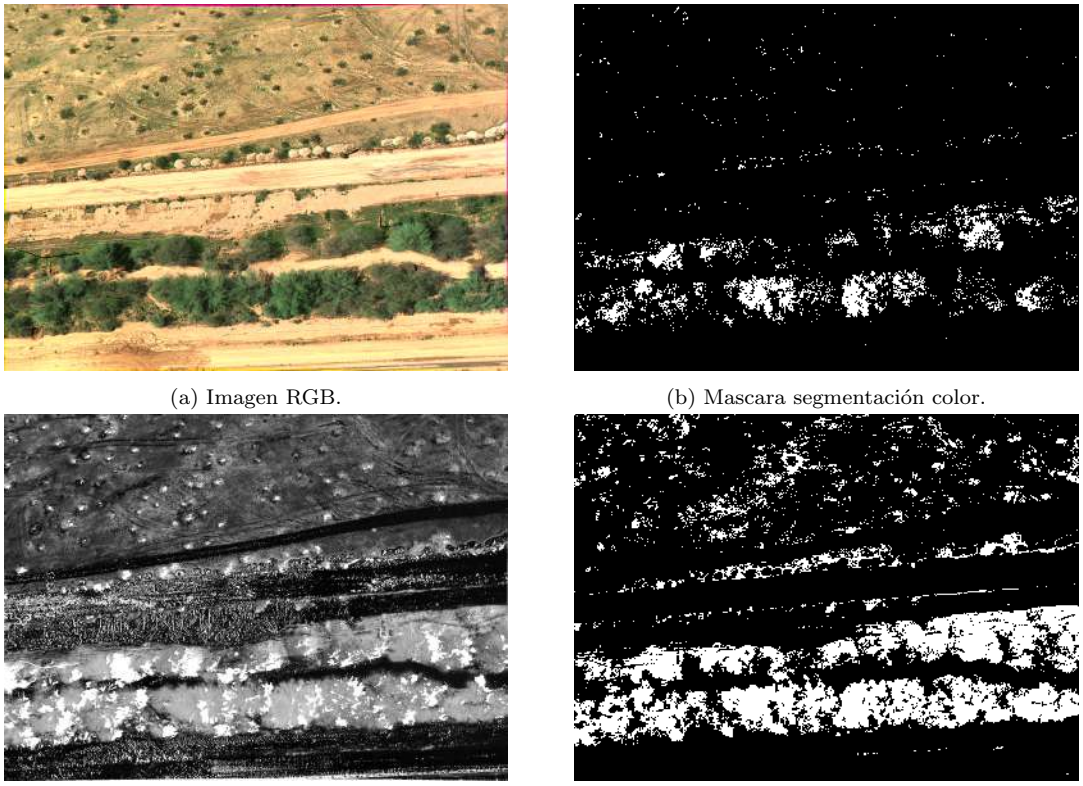
(b) Mascara segmentada por color.

(c) Imagen NDVI.

(d) Mascara segmentación NDVI.

Figura 5-3: Identificación de vegetación sobre un cultivo.

Con la identificación de vegetación se obtiene la máscara que se usa para obtener las características del cultivo. La figura 5-5 muestra un ejemplo de la secuencia mediante la cual se logra tal propósito usando una imagen obtenida por la cámara del entorno de simulación (a). La imagen (b) es el resultado de aplicar la segmentación por color en la imagen original, la imagen (c) es el resultado del proceso de esqueletización a partir de la máscara, la imagen (d) se obtiene a través de las operaciones morfológicas de expansión sobre la máscara, la imagen (e) es el resultado de aplicar detección de bordes sobre la imagen (d) y finalmente, en la imagen (f) sobre la fotografía original se superponen las líneas identificadas por el algoritmo de la transformada de Hough aplicado sobre la imagen (c), así como el contorno identificado mediante la imagen (e).



(a) Imagen RGB.

(b) Mascara segmentación color.

(c) Imagen NDVI.

(d) Mascara segmentación NDVI.

Figura 5-4: Identificación vegetación.

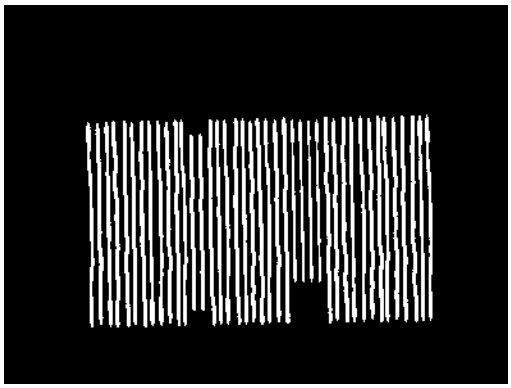
Las imágenes de la secuencia permiten evidenciar que los procesamientos de imágenes definidos permiten aislar satisfactoriamente los elementos de interés del cultivo. Por un lado, el contorno identificado representa muy bien la geometría del contorno si se compara con la figura RGB. Así mismo, se observa que a partir de la transformada de Hough aplicada al *skeleton* se logra identificar en cada surco del cultivo de interés por lo menos una línea que representa su orientación, es decir que el 100 % de los surcos tiene una representación de la orientación para ser tenida en cuenta al momento de estimar la orientación general.

Vale la pena resaltar que el hecho de que no se logre representar cada surco del cultivo mediante una única línea no representa ningún problema para el propósito de este proyecto que es la planificación, pues lo que se busca es establecer la tendencia de la orientación, mas no factores como la extensión de cada surco.

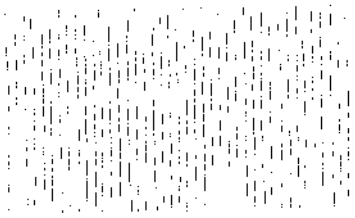
Así mismo es importante destacar que los resultados asociados al procesamiento de imágenes presentados son asociados a las condiciones de luminosidad del ambiente de simulación, sin embargo, para el momento en el que se vayan a realizar las pruebas de campo bajo condiciones reales, debe tenerse en cuenta este factor. Otra razón para

haber seleccionado a cámara Red-Edge como sensor de adquisición es que cuenta con un sensor de luz y un panel de reflectancia que sirven como apoyo para hacer un análisis más exacto al tener en cuenta las variaciones de luminosidad.

(a) Imagen RGB.



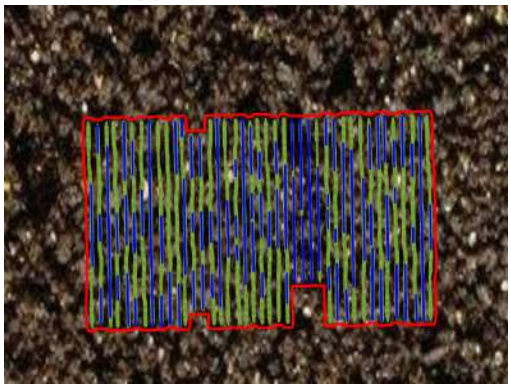
(b) Mascara segmentación color.



(c) *Skeleton.*

(d) *Expansión de surcos.*

(e) Contorno identificado.



(f) Contorno y líneas identificadas superpuestas en la imagen original.

Figura 5-5: Secuencia de identificación de características de cultivo.

La siguiente prueba consistió en validar el desempeño de los algoritmos asociados

al procesamiento de imágenes cuando son ejecutados tanto en GPU, como en CPU. En tal prueba se midió el tiempo de ejecución desde el momento cuando la fotografía se encuentra disponible para el procesamiento, hasta el momento en que las características del cultivo han sido identificadas completamente, es decir que se mide el tiempo de ejecución de la secuencia mostrada en la figura 5-5. La prueba se realizó no solo en la NVIDIA TX2, sino también en el portátil que corría la simulación, en ambos casos el vehículo permaneció a una altura de 120 metros hasta que la identificación de características del cultivo fue ejecutada como mínimo 120 veces. De las 120 iteraciones ejecutadas se calculó el promedio del tiempo de ejecución y los resultados fueron consignados en la tabla 5-1.

Promedio de tiempo de ejecución identificación de características (ms)			
HP-Gaming		NVIDIA TX2	
CPU	GPU	CPU	GPU
2660,90	556,81	11004,52	549,00

Tabla 5-1: Desempeño de procesamiento de imágenes.

Se puede observar que tanto para el portátil, como para la NVIDIA TX2, el rendimiento de la GPU es mucho mayor frente al que ofrece la CPU, pero más aún, la NVIDIA sobrepasa ligeramente el rendimiento usando la GPU en comparación con la GPU del portátil. Especificamente el procesamiento mediante GPU en el computador de simulación es 478 % más rápido en comparación con su CPU. Para el caso de la Nvidia TX2 el margen alcanza a ser del 2004 %. Al comparar el rendimento en CPU de ambos sistemas el del computador de simulación es superior por un factor de 413 %, en cambio, para el caso de las GPU la Nvidia supera al computador de simulación por un margen de 1.14 %.

Sin embargo, hay que tener en cuenta que el portátil estaba corriendo la simulación de forma simultánea con lo que se genera un mayor gasto computacional frente a la ejecución en la NVIDIA TX2, aunque esto no desvirtúa la validez de la comparación. Lo que se resalta con esta prueba es la versatilidad y desempeño de los sistemas embebidos como la NVIDIA TX2 para realizar este tipo de procesamientos y su posibilidad de ser integrada en dispositivos de bajo peso.

Finalmente, se realizaron pruebas para validar el funcionamiento del algoritmo de planificación. El primer paso consistió en ubicar el dron en el centro de un cultivo

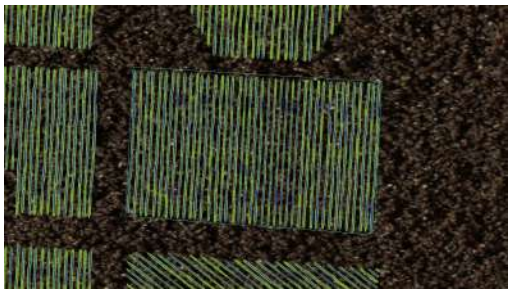
y cambiar al modo *offboard*, ingresando como parámetros de entrada para el plan de vuelo un GSD de 4cm, *sidelap* de 50 % y *endlap* de 70 %, con lo cual se ejecutó la secuencia completa, desde el despegue hasta la ejecución del plan de vuelo. De esta ejecución se obtuvo una imagen con las características identificadas y además se guardaron los datos de posicionamiento con los cuales se generó el plan. Esta información permitió ejecutar el servicio de planificación nuevamente, pero variando unicamente el valor de GSD y sin necesidad de ejecutar toda la secuencia desde el despegue; si no se hiciese de este modo, en cada ejecución se podría capturar una imagen distinta (debido a variaciones en el posicionamiento del vehículo asociadas a la precisión del GPS), lo cual no permitiría comparar los planes generados con distintos valores de GSD . Este proceso se repitió en diez (10) de los cultivos generados para el entorno de simulación y se generaron tres (3) planes de vuelo diferentes para cada cultivo.

Las figuras 5-7 a 5-15 muestran los resultados obtenidos mediante esta prueba. En cada una de ellas la imagen (a) muestra el cultivo con las características identificadas, la imagen (b) muestra un plan de vuelo con un GSD de 1cm, la imagen (c) muestra un plan de vuelo con un GSD de 2cm y la imagen (d) muestra un plan de vuelo con un GSD de 4cm.

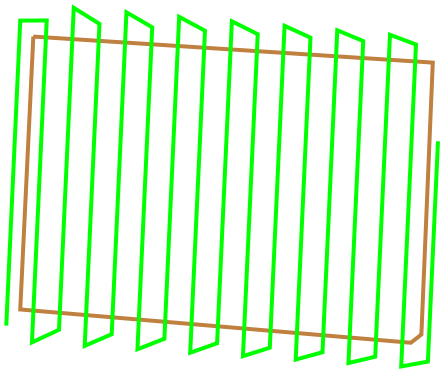
De los resultados obtenidos se puede observar que el algoritmo desarrollado, se adapta adecuadamente a variaciones en la geometría de los cultivos. Mas aun, se puede observar que entre menor sea el valor del GSD los planes generados describen mucho mejor la geometría de los cultivos, sin embargo en figuras como la 5-10, 5-11 y 5-6 se alcanza a omitir una leve zona del cultivo.

De igual forma en las figuras 5-13, 5-14 y 5-15 en donde la orientación de los surcos no esta alineada con la geometría del cultivo, los resultados se consideran satisfactorios. Esto debido a que en cada una de las líneas de vuelo los puntos generados siguen la orientación de los surcos, las líneas de vuelo generadas abarcan la totalidad del terreno y siguen adaptándose a la geometría de los cultivos descrita por el contorno de cada cultivo.

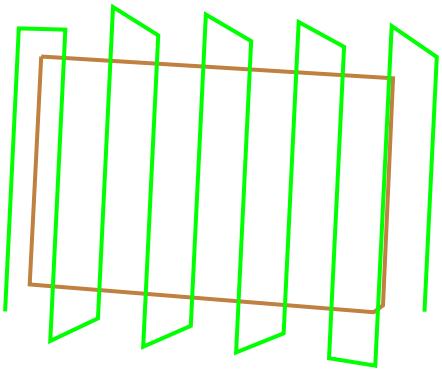
La capacidad de adaptarse a la geometría de los cultivos en la mayoría de los casos ayuda a evitar incluir puntos de control innecesarios, como se evidencia en figuras como la 5-7,5-8, 5-9 y 5-12, de manera que se contribuye a reducir el número de fotos a tomar, en comparación a si se considerara un *bounding box* rectangular del cultivo. Esto contribuye a disminuir la memoria de almacenamiento requerida, así como el tiempo de ejecución por plan de vuelo, siendo este último uno de los factores críticos a tener en cuenta cuando se operan *UAVs*.



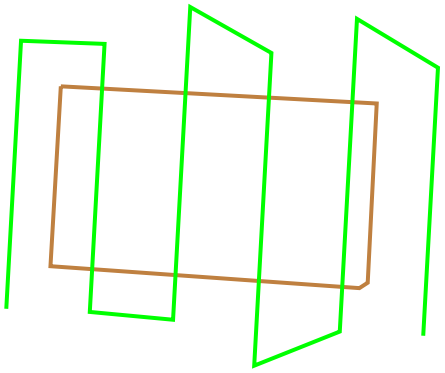
(a) Cultivo con orientación de surcos a 87°.



(b) Plan de vuelo GSD 1 cm.

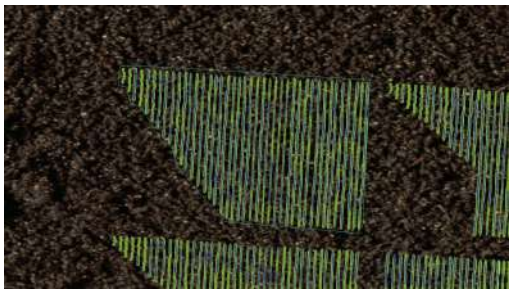


(c) Plan de vuelo GSD 2 cm.

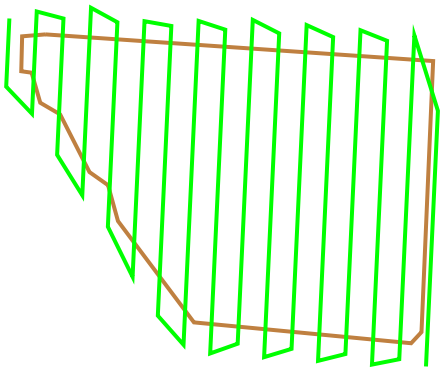


(d) Plan de vuelo GSD 4 cm.

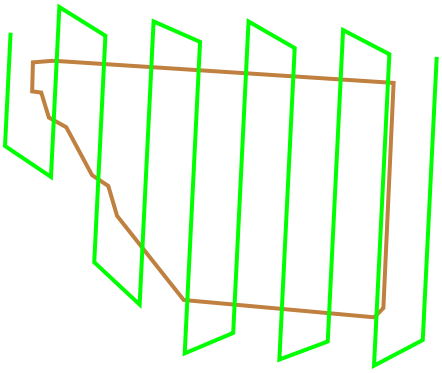
Figura 5-6: Planes de vuelo cultivo 1.



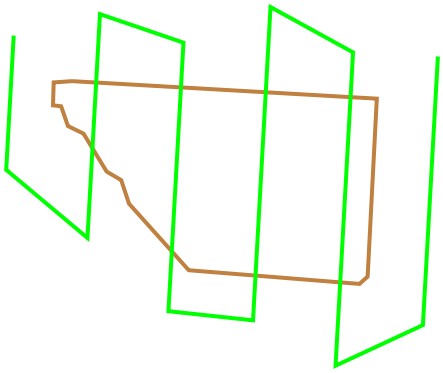
(a) Cultivo con orientación de surcos a 87°.



(b) Plan de vuelo GSD 1 cm.

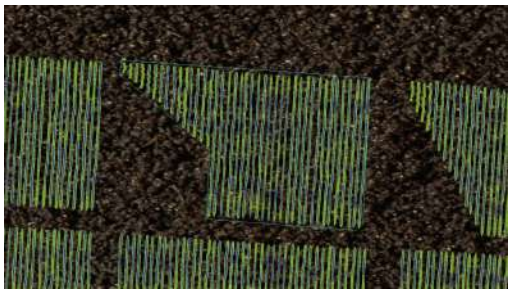


(c) Plan de vuelo GSD 2 cm.

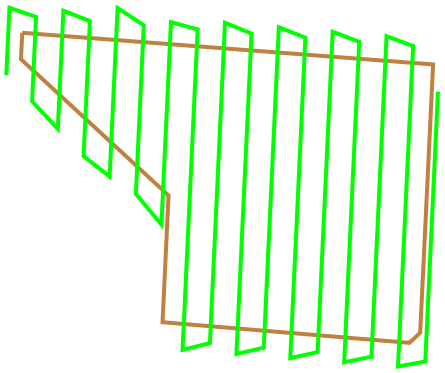


(d) Plan de vuelo GSD 4 cm.

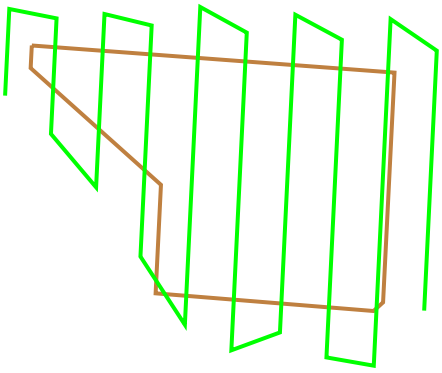
Figura 5-7: Planes de vuelo cultivo 2.



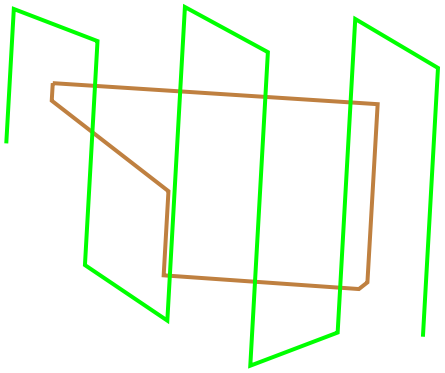
(a) Cultivo con orientación de surcos a 87°.



(b) Plan de vuelo GSD 1 cm.

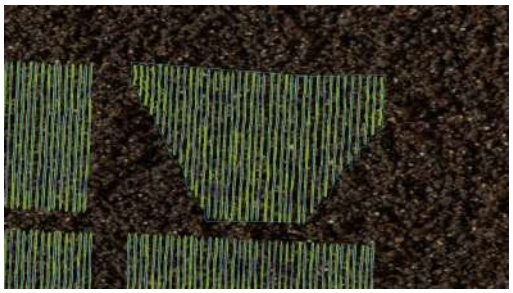


(c) Plan de vuelo GSD 2 cm.

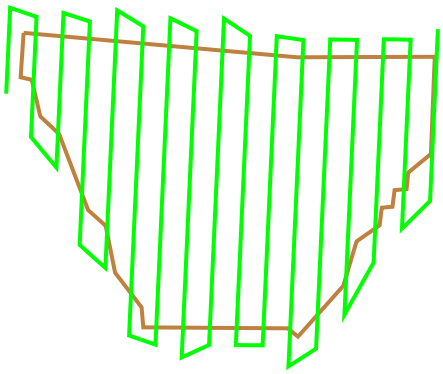


(d) Plan de vuelo GSD 4 cm.

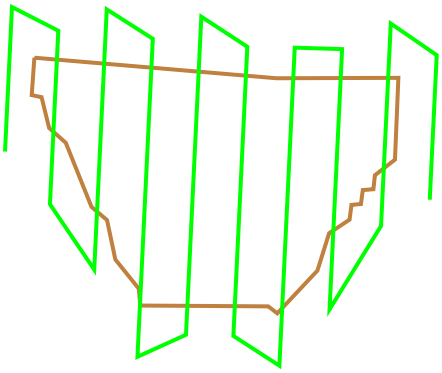
Figura 5-8: Planes de vuelo cultivo 3.



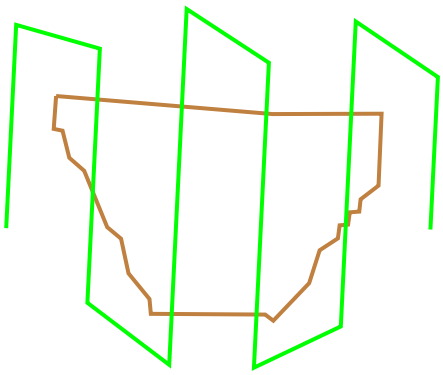
(a) Cultivo con orientación de surcos a 87°.



(b) Plan de vuelo GSD 1 cm.

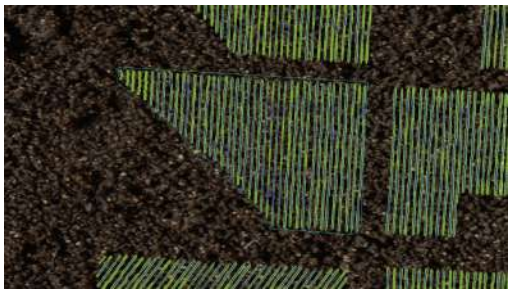


(c) Plan de vuelo GSD 2 cm.

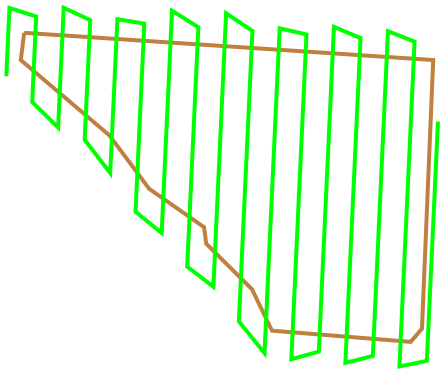


(d) Plan de vuelo GSD 4 cm.

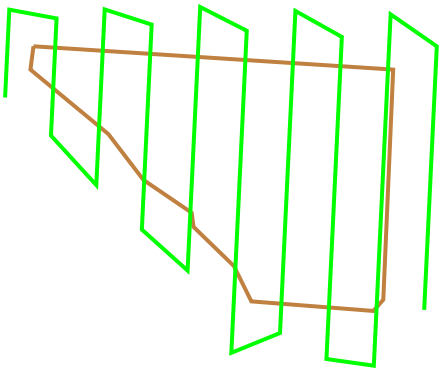
Figura 5-9: Planes de vuelo cultivo 4.



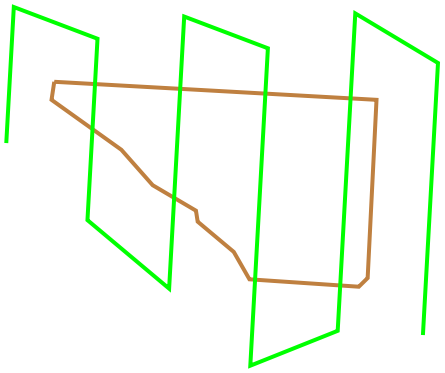
(a) Cultivo con orientación de surcos a 87°.



(b) Plan de vuelo GSD 1 cm.

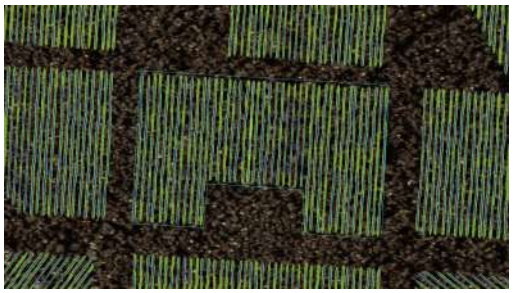


(c) Plan de vuelo GSD 2 cm.

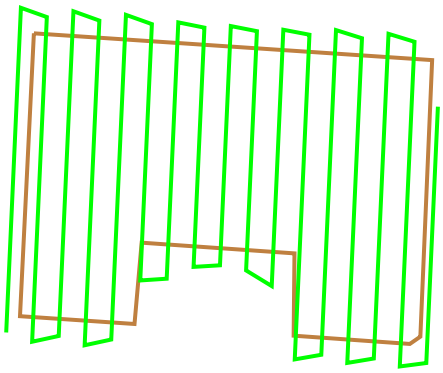


(d) Plan de vuelo GSD 4 cm.

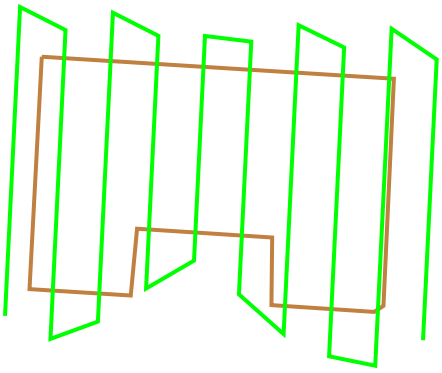
Figura 5-10: Planes de vuelo cultivo 5.



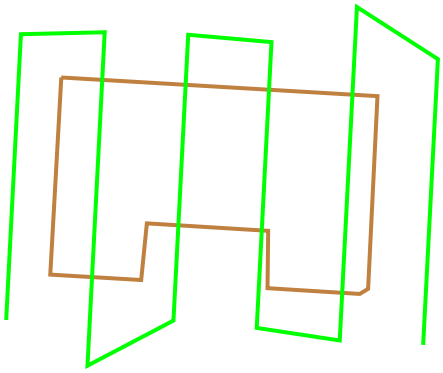
(a) Cultivo con orientación de surcos a 87°.



(b) Plan de vuelo GSD 1 cm.

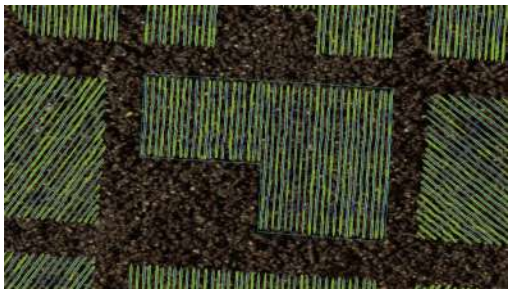


(c) Plan de vuelo GSD 2 cm.

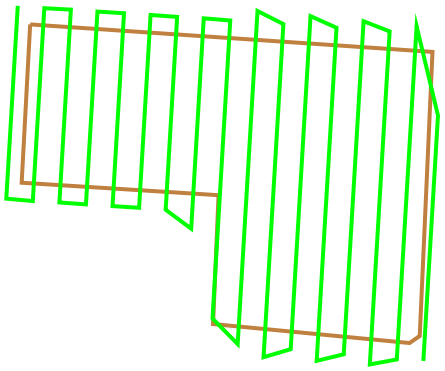


(d) Plan de vuelo GSD 4 cm.

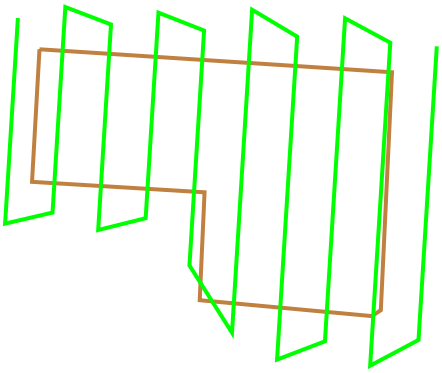
Figura 5-11: Planes de vuelo cultivo 6.



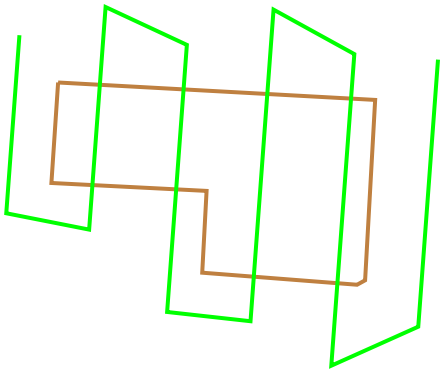
(a) Cultivo con orientación de surcos a 86°.



(b) Plan de vuelo GSD 1 cm.



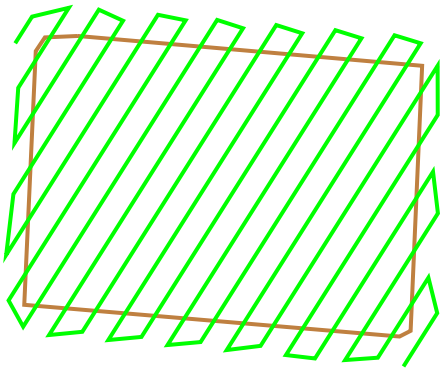
(c) Plan de vuelo GSD 2 cm.



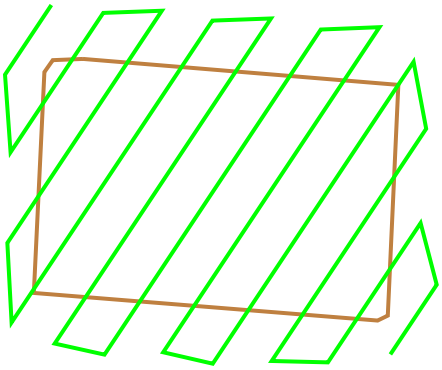
(d) Plan de vuelo GSD 4 cm.

Figura 5-12: Planes de vuelo cultivo 7.

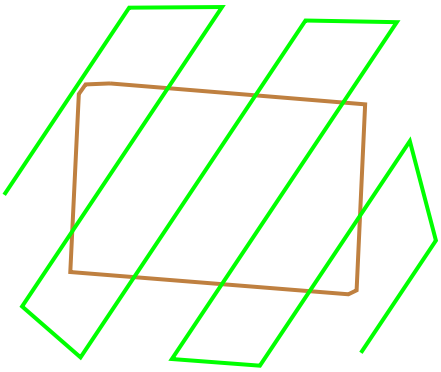
(a) Cultivo con orientación de surcos a 53.000004°.



(b) Plan de vuelo GSD 1 cm.

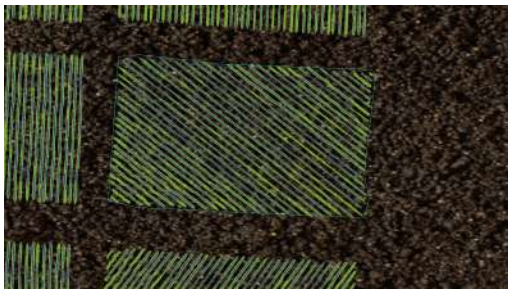


(c) Plan de vuelo GSD 2 cm.

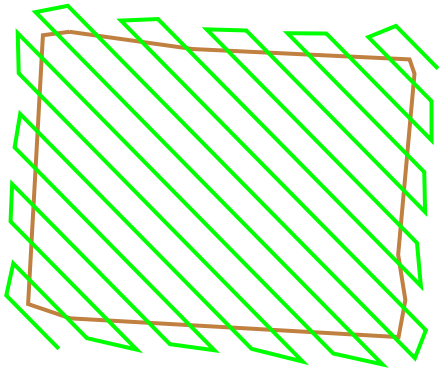


(d) Plan de vuelo GSD 4 cm.

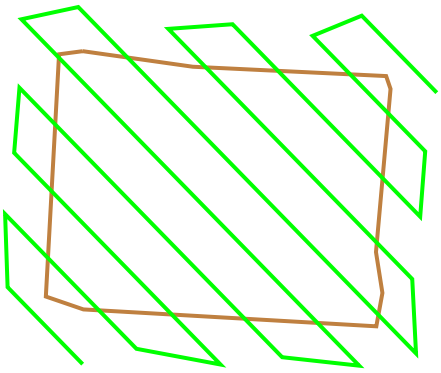
Figura 5-13: Planes de vuelo cultivo 8.



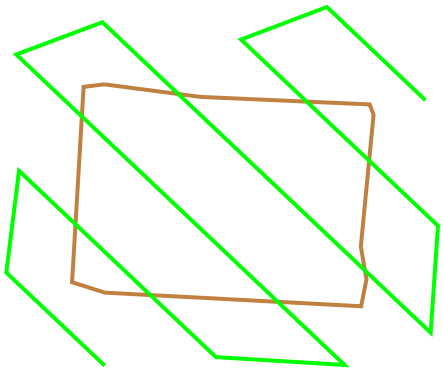
(a) Cultivo con orientación de surcos a -37.999992°.



(b) Plan de vuelo GSD 1 cm.



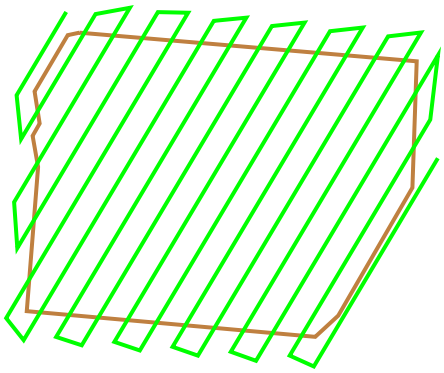
(c) Plan de vuelo GSD 2 cm.



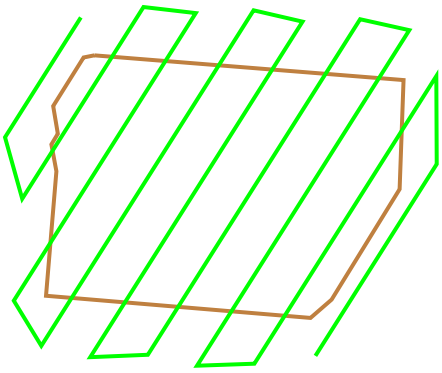
(d) Plan de vuelo GSD 4 cm.

Figura 5-14: Planes de vuelo cultivo 9.

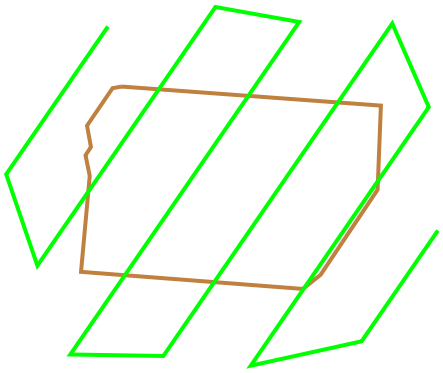
(a) Cultivo con orientación de surcos a 53.000004°.



(b) Plan de vuelo GSD 1 cm.



(c) Plan de vuelo GSD 2 cm.



(d) Plan de vuelo GSD 4 cm.

Figura 5-15: Planes de vuelo cultivo 10.

6. Conclusiones y trabajo futuro

6.1. Conclusiones

Los resultados obtenidos tras la realización del proyecto permiten afirmar que tanto los objetivos específicos, como el objetivo general fueron logrados, sin embargo, existen aspectos por mejorar que garanticen el cubrimiento de todo el terreno.

En primer lugar, fue posible hacer la identificación de las características geométricas del cultivo con base en una toma fotográfica aérea. Para identificar las zonas de vegetación se presentaron dos métodos, uno en el que se hace una segmentación basado en el color verde, usando información de los canales *Red*,*Green* y *Blue*, presentes tanto en cámaras multiespectrales como la Red-Edge, así como la gran mayoría de cámaras comerciales. Por otro lado la segmentación basada en NDVI, utiliza información de las bandas, *NIR* y *Red*, lo cual implica el uso de una cámara de tipo multiespectral. Aunque se evidenció que ambos métodos son efectivos, también se pudo observar que la identificación mediante NDVI es mejor. Posteriormente, mediante el uso de operaciones morfológicas, así como de la transformada de Hough, es posible identificar los límites del cultivo y la orientación de los surcos dentro del sistema coordenado de la cámara.

En segundo lugar, fue posible extraer la ubicación geoespacial de los puntos que definen el contorno del cultivo identificado. Para ello se hizo uso de una serie de transformaciones que permiten trasladar las coordenadas entre los diferentes *frames* o sistemas coordenados involucrados, empezando por pasar las coordenadas de la imagen al *frame* de navegación y posteriormente al sistema GPS , siendo las coordenadas del *frame* de navegación las que se usan para la generación de las trayectorias.

Finalmente, se generó un algoritmo que permite realizar la planificación de trayectorias para *UAVs* garantizando el cumplimiento de requerimientos fotogramétricos, específicamente el GSD y los porcentajes de solape. El algoritmo se elaboró integrando conceptos sencillos de geometría, álgebra y de fotogrametría. El algoritmo desarrollado tiene la capacidad de generar planes de vuelo que se adapten a la geometría de los cultivos y que siguen la orientación de los surcos, lo cual contribuye a que se recorra el cultivo de una manera eficiente, evitando la inclusión de puntos de control innecesarios que no lleguen a brindar información relevante y a la vez ayuda

a optimizar la duración de batería y el espacio de almacenamiento. Es posible que para algunos valores bajos de GSD y con geometrías y alineaciones de los surcos fuera de lo común el cubrimiento del plan de vuelo no sea del 100 % del área del cultivo identificado, sin embargo, se deberá evaluar si es necesario contemplar estas situaciones como una oportunidad de mejora del algoritmo.

Ademas del algoritmo, se creó un entorno de simulación en el cual se pueden realizar pruebas con múltiples modelos de vehículos no tripulados, usando un sistema de autopiloto real operando en la modalidad *HIL*, el cual no es solo compatible con vehículos aéreos, sino con algunos terrestres. Todo este sistema es posible gracias al *framework* de *ROS* que se ha vuelto un estándar de *facto* en el desarrollo de soluciones robóticas. Esto favorece a que se puedan realizar pruebas de una forma segura cuando se estén desarrollando nuevas soluciones tecnológicas que empleen este tipo de vehículos sin tener que poner en riesgo la integridad de los componentes reales de *hardware* hasta el momento en que los resultados de simulación muestren que existe un nivel de confiabilidad en la funcionalidad de las soluciones.

Los anexos del documento contienen información que permiten facilitar los procesos de instalación y configuración, tanto del simulador, como de los componentes de *hardware* utilizados a lo largo del proyecto. Con lo cual se busca simplificar y reducir la curva de aprendizaje para aquel que esté interesado en explorar este tipo de metodologías y herramientas, bien sea para dar continuidad a la línea de trabajo del proyecto o explorar una nueva donde estas le sean útiles. En la documentación se buscó incluir la mayor cantidad de detalles que permitan evitar las dificultades que se encontraron durante el desarrollo del proyecto.

Aunque inicialmente en la propuesta del trabajo de grado se contempló la posibilidad de realizar pruebas de campo, no fue posible su realización debido a dificultades técnicas. En primer lugar, la cámara multiespectral RedEdge que se pretendía integrar al dron, presentó fallas que la dejaron fuera de funcionamiento y no ha sido posible reparar al menos hasta el momento en que se elaboró este documento. Adicionalmente, el dron DIY de 3DR disponible para el desarrollo del proyecto, como el que se muestra en la figura 6-1, no contaba con la estructura adecuada para integrar los componentes de *hardware* adicionales (kit de desarrollo TX2 y cámara RedEdge), lo cual implica una mejora en la capacidad de carga del dron, es decir aumentar la potencia de los motores y ajustar la electrónica de potencia que los alimenta. Por esta razón se decidió incrementar la importancia de la simulación *HIL* para el proyecto y asegurar de esta forma la confiabilidad de la ejecución de toda aplicación.

Este tipo de proyectos de integración de software especializado tienen curvas de aprendizaje que inicialmente pueden ser subestimadas dada la complejidad inherente

Figura 6-1: Cultivos del entorno de simulación.

a la integración tanto de elementos de hardware dispares como a los diferentes niveles del software que se ejecuta en ellos. Se pasa de compilar *firmwares* particulares para el piloto automático, de ejecutar nodos de ROS en el computador de compañía y de ejecutar la simulación que permite probar el funcionamiento del sistema en un entorno que emula una situación real. De esta forma se demostró que es posible generar un algoritmo que automatice la generación de trayectorias para vehículos aéreos no tripulados en entornos agrícolas, primando variables fotogramétricas como el GSD, con lo cual también se cumple el objetivo de implementar la autonomía del dron como herramienta dentro de un sistema de supervisión agrícola.

De los resultados obtenidos durante el desarrollo del proyecto, se elaboró el artículo titulado *Towards automatic UAV path planning in agriculture oversight activities*, que fue presentado en el congreso LACAR (*Latin American Congress on Automation and Robotics*), llevado a cabo en la ciudad de Cali entre el 31 de Octubre y el 1 de Noviembre de 2019 y con los resultados finales se elaboró otro artículo titulado

"Entornos de simulación para prueba de sistemas robóticos autónomos: *Hardware in the loop*"que está en proceso de presentación a una revista indexada nacional.

Los resultados obtenidos durante el desarrollo de este trabajo se convierten en la primera etapa del flujo de trabajo de un vehículo de supervisión agrícola autónomo. Se necesitan realizar ajustes y mejoras para que las soluciones generadas sean aplicables dentro del contexto productivo colombiano.

6.2. Trabajo futuro

Como trabajo futuro que dé continuidad al proyecto se deben realizar las pruebas de campo sobre un cultivo real, como por ejemplo de papa, en donde se pueda ejecutar el algoritmo de planificación y se ejecute el plan de vuelo para la captura de las imágenes con el GSD y demás parámetros deseados. Posteriormente con las fotografías obtenidas se genere el foto-mosaico, para validar la efectividad de las soluciones desarrolladas al usar los componentes de *hardware* reales.

En cuanto a la NVIDIA TX2, se pueden explorar las herramientas de aprendizaje de máquina y procesamiento de imágenes avanzadas, para por ejemplo, realizar la identificación de frutos donde se tendrá que proporcionar un valor de GSD pequeño al momento de genera el plan de vuelo.

Una vez que se ha identificado el entorno de trabajo se podría explorar la implementación de un sistema de vehículos no tripulados, donde vehículos aéreos y terrestres operen en conjunto para completar labores de supervisión. Los vehículos terrestres podrían hacer uso de la información asociada a la orientación de los surcos para recorrer los cultivos sin pasar por encima de las plantas y mediante la implementación de sensores inerciales y/o laser, se podría considerar la elaboración de modelos detallados del cultivo.

Los resultados del proyecto también permitieron demostrar que el uso de sistemas embebidos que cuentan con una tarjeta gráfica de procesamiento ofrecen grandes ventajas respecto a sistemas embebidos que solo cuentan con CPU, porque permiten reducir el tiempo computacional asociado al procesamiento de imágenes, siendo comparables con el rendimiento de un computador portátil. Esto puede ser útil para realizar actividades en tiempo real, como por ejemplo la detección de obstáculos mediante cámaras de profundidad o bifocales, que no fue tenido en cuenta dentro del alcance del presente trabajo ya que se tomó como premisa que el entorno de trabajo no tendría obstáculos.

Otra característica del algoritmo que se puede mejorar, es que los cultivos sobre los cuales es efectivo tienen que estar ubicados sobre un terreno de geografía plana, de lo contrario no se puede obtener una fotografía de tipo NADIR, con lo cual el valor

del GSD cambiara a lo largo del terreno. Una posible solución sería la integración de un sistema de *Gimbal*, el cual se maneja de tal manera que las imágenes tomadas cumplan con tal característica. Además, durante el proceso de ejecución del plan de vuelo también favorecería a que la cámara permanezca estable, permitiendo que las imágenes obtenidas no sufran efectos de distorsión u otro tipo de inconvenientes que puedan afectar el proceso de *Stitching*.

Los sistemas de autopiloto Pixhawk y el *firmware* PX4 son herramientas de las cuales se exploraron solo algunas de sus funcionalidades, por lo cual aún existe un potencial grande para generar soluciones más robustas al mejorar los conocimientos técnicos sobre estas, ya que por ejemplo se podría desarrollar un vehículo que satisfaga las necesidades de una problemática especial y con el uso de estas herramientas se podrían implementar los controladores de posicionamiento del vehículo utilizando comandos de velocidad, en lugar de *setpoints* lo que permitiría operar el vehículo de forma más controlada.

En cuanto a las cámaras multiespectrales, para fortalecer las tareas de supervisión, se podría hacer un estudio de índices de vegetación diferentes al NDVI, que permitan a los agricultores establecer el estado de salud del cultivo mediante un solo vuelo, pues debido a las capacidades de procesamiento de la NVIDIA TX2 se podrían calcular estos índices a gran velocidad, cuya información permitiría generar informes de calidad.

La exploración y capacitación sobre Sistemas de información geografica –del inglés *Geographic information Systems*– (*GIS*) puede contribuir a generar soluciones más compatibles con las labores realizadas por profesionales del sector agrícola o catastral, quienes constantemente se ven involucrados con el manejo de coordenadas y sistemas de referencia para desempeñar las labores de mapeo o gestión de los terrenos.

Referencias Bibliográficas

[1] UN, "Population," 2017.

[2] Research and Markets, "Ai in agriculture market by technology (machine learning, computer vision, predictive analytics), offering, application (precision farming, drone analytics, agriculture robots, livestock monitoring), offering, and geography - global forecast to 2025," 2017.

[3] D. Vasisht, Z. Kapetanovic, J. Won, X. Jin, R. Chandra, S. Sinha, A. Kapoor, M. Sudarshan, and S. Stratman, "Farmbeats: An iot platform for data-driven agriculture," in *14th USENIX Symposium on Networked Systems Design and Implementation (NSDI 17)*, (Boston, MA), pp. 515–529, USENIX Association, 2017.

[4] J. J. Roldán, J. del Cerro, D. Garzón-Ramos, P. Garcia-Aunon, M. Garzón, J. de León, and A. Barrientos, "Robots in agriculture: State of art and practical experiences," in *Service Robots*, IntechOpen, 2017.

[5] H. Shakhatreh, A. Sawalmeh, A. I. Al-Fuqaha, Z. Dou, E. Almaita, I. Khalil, N. S. Othman, A. Khreishah, and M. Guizani, "Unmanned aerial vehicles: A survey on civil applications and key research challenges," *CoRR*, vol. abs/1805.00881, 2018.

[6] M. Romero, Y. Luo, B. Su, and S. Fuentes, "Vineyard water status estimation using multispectral imagery from an uav platform and machine learning algorithms for irrigation scheduling management," *Computers and Electronics in Agriculture*, vol. 147, pp. 109 – 117, 2018.

[7] J. Rojas, C. Martinez, I. Mondragon, and J. Colorado, "Towards image mosaicking with aerial images for monitoring rice crops," in *Advances in Automation and Robotics Research in Latin America* (I. Chang, J. Baca, H. A. Moreno, I. G. Carrera, and M. N. Cardona, eds.), (Cham), pp. 279–296, Springer International Publishing, 2017.

[8] T. Bergen and T. Wittenberg, "Stitching and surface reconstruction from endoscopic image sequences: A review of applications and methods," *IEEE Journal of Biomedical and Health Informatics*, vol. 20, pp. 304–321, Jan 2016.

[9] F. Nex and F. Remondino, "Uav for 3d mapping applications: a review," *Applied Geomatics*, vol. 6, pp. 1–15, Mar 2014.

[10] M. E. Tjahjadi, F. Handoko, and S. S. Sai, "Novel image mosaicking of uav's imagery using collinearity condition," *International Journal of Electrical and Computer Engineering (IJECE)*, vol. 7, no. 3, pp. 1188–1196, 2017.

[11] J. Zhao, X. Zhang, C. Gao, X. Qiu, Y. Tian, Y. Zhu, and W. Cao, "Rapid mosaicking of unmanned aerial vehicle (uav) images for crop growth monitoring using the sift algorithm," *Remote Sensing*, vol. 11, no. 10, p. 1226, 2019.

[12] G. Ristorto, P. D'Incalci, R. Gallo, F. Mazzetto, and G. Guglieri, "Mission planning for the estimation of the field coverage of unmanned aerial systems in monitoring mission in precision farming," *Chemical Engineering Transactions*, vol. 58, pp. 649–654, 2017.

[13] M. Popović, G. Hitz, J. Nieto, I. Sa, R. Siegwart, and E. Galceran, "Online informative path planning for active classification using uavs," in *2017 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 5753–5758, May 2017.

[14] C. S. Paul Sponagle, "Automatic mission planning algorithms for aerial collection of imaging-specific tasks," vol. 10218, 2017.

[15] D. I. Patrício and R. Rieder, "Computer vision and artificial intelligence in precision agriculture for grain crops: A systematic review," *Computers and Electronics in Agriculture*, vol. 153, pp. 69–81, 2018.

[16] O. Brovkina, E. Cienciala, P. Surový, and P. Janata, "Unmanned aerial vehicles (uav) for assessment of qualitative classification of norway spruce in temperate forest stands," *Geo-spatial Information Science*, vol. 21, no. 1, pp. 12–20, 2018.

[17] O. Nevalainen, E. Honkavaara, S. Tuominen, N. Viljanen, T. Hakala, X. Yu, J. Hyyppä, H. Saari, I. Pölönen, N. N. Imai, *et al.*, "Individual tree detection and classification with uav-based photogrammetric point clouds and hyperspectral imaging," *Remote Sensing*, vol. 9, no. 3, p. 185, 2017.

[18] D. Falanga, K. Kleber, S. Mintchev, D. Floreano, and D. Scaramuzza, "The foldable drone: A morphing quadrotor that can squeeze and fly," *IEEE Robotics and Automation Letters*, vol. 4, no. 2, pp. 209–216, 2018.

[19] C. Gomez and H. Purdie, "Uav-based photogrammetry and geocomputing for hazards and disaster risk monitoring–a review," *Geoenvironmental Disasters*, vol. 3, no. 1, p. 23, 2016.

[20] C. A. F. Ezequiel, M. Cua, N. C. Libatique, G. L. Tangonan, R. Alampay, R. T. Labuguen, C. M. Favila, J. L. E. Honrado, V. Caños, C. Devaney, A. B. Loreto, J. Bacusmo, and B. Palma, "Uav aerial imaging applications for post-disaster assessment, environmental management and infrastructure development," in *2014 International Conference on Unmanned Aircraft Systems (ICUAS)*, pp. 274–283, May 2014.

[21] S. Norouzi Ghazbi, Y. Aghli, M. Alimohammadi, and A. Akbari, "Quadrotors unmanned aerial vehicles: A review.," *International Journal on Smart Sensing & Intelligent Systems*, vol. 9, no. 1, 2016.

[22] S. M. Adams and C. J. Friedland, "A survey of unmanned aerial vehicle (uav) usage for imagery collection in disaster research and management," in *9th International Workshop on Remote Sensing for Disaster Response*, vol. 8, 2011.

[23] A. Puri, "A survey of unmanned aerial vehicles (uav) for traffic surveillance," 2005.

[24] Y. Zeng, R. Zhang, and T. J. Lim, "Wireless communications with unmanned aerial vehicles: opportunities and challenges," *IEEE Communications Magazine*, vol. 54, pp. 36–42, May 2016.

[25] A. Carrio, C. Sampedro, A. Rodriguez-Ramos, and P. Campoy, "A review of deep learning methods and applications for unmanned aerial vehicles," *Journal of Sensors*, vol. 2017, 2017.

[26] M. Hassanalian and A. Abdelkefi, "Classifications, applications, and design challenges of drones: A review," *Progress in Aerospace Sciences*, vol. 91, pp. 99–131, 2017.

[27] P. Lottes, R. Khanna, J. Pfeifer, R. Siegwart, and C. Stachniss, "Uav-based crop and weed classification for smart farming," in *2017 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 3024–3031, May 2017.

[28] A. C. de Colombia, "Rac 91 - reglas generales de vuelo y operación," Oct.

[29] A. Chlingaryan, S. Sukkarieh, and B. Whelan, "Machine learning approaches for crop yield prediction and nitrogen status estimation in precision agriculture: A review," *Computers and Electronics in Agriculture*, vol. 151, pp. 61 – 69, 2018.

[30] C. A. Rokhmana, "The potential of uav-based remote sensing for supporting precision agriculture in indonesia," *Procedia Environmental Sciences*, vol. 24, pp. 245 – 253, 2015. The 1st International Symposium on LAPAN-IPB Satellite (LISAT) for Food Security and Environmental Monitoring.

[31] J. A. Arroyo, C. Gomez-Castaneda, E. Ruiz, E. M. de Cote, F. Gavi, and L. E. Sucar, "Uav technology and machine learning techniques applied to the yield improvement in precision agriculture," in *2017 IEEE Mexican Humanitarian Technology Conference (MHTC)*, pp. 137–143, IEEE, 2017.

[32] K. C. T. Vivaldini, V. Guizilini, M. D. C. Oliveira, T. H. Martinelli, D. F. Wolf, and F. Ramos, "Route planning for active classification with uavs," in *2016 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 2563–2568, May 2016.

[33] H. Shakhatreh, A. H. Sawalmeh, A. Al-Fuqaha, Z. Dou, E. Almaita, I. Khalil, N. S. Othman, A. Khreishah, and M. Guizani, "Unmanned aerial vehicles (uavs): A survey on civil applications and key research challenges," *IEEE Access*, vol. 7, pp. 48572–48634, 2019.

[34] T. Ojha, S. Misra, and N. S. Raghuwanshi, "Wireless sensor networks for agriculture: The state-of-the-art in practice and future challenges," *Computers and Electronics in Agriculture*, vol. 118, pp. 66–84, 2015.

[35] R. Szeliski, *Computer Vision - Algorithms and Applications*. Texts in Computer Science, Springer, London, 2011.

[36] K. H. Lim, K. P. Seng, and L. M. Ang, "Intra color-shape classification for traffic sign recognition," in *2010 International Computer Symposium (ICS2010)*, pp. 642–647, IEEE, 2010.

[37] S. Song and J. Xiao, "Deep sliding shapes for amodal 3d object detection in rgb-d images," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 808–816, 2016.

[38] M. Mariappan, T. C. T. Ming, and M. Nadarajan, "Automated visual inspection: Position identification of object for industrial robot application based on color and shape," *International Journal of Intelligent Systems and Applications*, vol. 8, no. 1, p. 9, 2016.

[39] G. Bradski and A. Kaehler, *Learning OpenCV: Computer vision with the OpenCV library*. .ºReilly Media, Inc.", 2008.

[40] L. Saxena and L. Armstrong, "A survey of image processing techniques for agriculture," in *Proceedings of Asian Federation for Information Technology in Agriculture*, pp. 401,413, Australian Society of Information and Communication Technologies in Agriculture, 2014.

[41] H. C. Oliveira, V. C. Guizilini, I. P. Nunes, and J. R. Souza, "Failure detection in row crops from uav images using morphological operators," *IEEE Geoscience and Remote Sensing Letters*, vol. 15, pp. 991–995, July 2018.

[42] W. Förstner and B. P. Wrobel, *Photogrammetric Computer Vision: Statistics, Geometry, Orientation and Reconstruction*. Springer Publishing Company, Incorporated, 1st ed., 2016.

[43] I. Colomina and P. Molina, "Unmanned aerial systems for photogrammetry and remote sensing: A review," *ISPRS Journal of Photogrammetry and Remote Sensing*, vol. 92, pp. 79 – 97, 2014.

[44] M. S. Grewal, L. R. Weill, and A. P. Andrews, *Global positioning systems, inertial navigation, and integration*, pp. 324–346. John Wiley & Sons, 2007.

[45] G. Cai, B. M. Chen, and T. H. Lee, *Coordinate Systems and Transformations*, pp. 23–34. London: Springer London, 2011.

[46] G. B. Ladd, A. Nagchaudhuri, T. J. Earl, M. Mitra, and G. L. Bland, "Rectification, georeferencing, and mosaicking of images acquired with remotely operated aerial platforms," in *Proceedings of the American Society for Photogrammetry and Remote Sensing Annual Conference ASPRS*, vol. 10, 2006.

[47] E. M. Hemerly, "Automatic georeferencing of images acquired by uav's," *International Journal of Automation and Computing*, vol. 11, pp. 347–352, Aug 2014.

[48] H. Xiang and L. Tian, "Method for automatic georeferencing aerial remote sensing (rs) images from an unmanned aerial vehicle (uav) platform," *Biosystems Engineering*, vol. 108, no. 2, pp. 104 – 113, 2011.

[49] S. Wang, L. Ding, Z. Chen, and A. Dou, "A rapid uav image georeference algorithm developed for emergency response," *Journal of Sensors*, vol. 2018, 2018.

[50] M. Ligas and P. Banasik, "Conversion between cartesian and geodetic coordinates on a rotational ellipsoid by solving a system of nonlinear equations," *Geodesy and cartography*, vol. 60, no. 2, pp. 145–159, 2011.

[51] H. Vermeille, "An analytical method to transform geocentric into geodetic coordinates," *Journal of Geodesy*, vol. 85, no. 2, pp. 105–117, 2011.

[52] C. Goerzen, Z. Kong, and B. Mettler, "A survey of motion planning algorithms from the perspective of autonomous uav guidance," *Journal of Intelligent and Robotic Systems*, vol. 57, p. 65, Nov 2009.

[53] M. Monwar, O. Semiari, and W. Saad, "Optimized path planning for inspection by unmanned aerial vehicles swarm with energy constraints," in *2018 IEEE Global Communications Conference (GLOBECOM)*, pp. 1–6, IEEE, 2018.

[54] E. Galceran and M. Carreras, "A survey on coverage path planning for robotics," *Robotics and Autonomous Systems*, vol. 61, no. 12, pp. 1258 – 1276, 2013.

[55] T. Cabreira, L. Brisolara, and P. R Ferreira, "Survey on coverage path planning with unmanned aerial vehicles," *Drones*, vol. 3, no. 1, p. 4, 2019.

[56] M. Popović, T. Vidal-Calleja, G. Hitz, I. Sa, R. Siegwart, and J. Nieto, "Multiresolution mapping and informative path planning for uav-based terrain monitoring," in *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 1382–1388, Sep. 2017.

[57] S. Javed, A. Tariq, and Y. N. Khan, "Optimized path planning for unmanned aircraft vehicle systems in crop surveillance," tech. rep., Department of Computer Science FAST-NU Lahore Campus, Pakistan., 2014.

[58] M. Quigley, K. Conley, B. P. Gerkey, J. Faust, T. Foote, J. Leibs, R. Wheeler, and A. Y. Ng, "Ros: an open-source robot operating system," in *ICRA Workshop on Open Source Software*, 2009.

[59] L. Meier, P. Tanskanen, L. Heng, G. H. Lee, F. Fraundorfer, and M. Pollefeys, "Pixhawk: A micro aerial vehicle design for autonomous flight using onboard computer vision," *Autonomous Robots*, vol. 33, no. 1-2, pp. 21–39, 2012.

[60] L. Meier, P. Tanskanen, F. Fraundorfer, and M. Pollefeys, "Pixhawk: A system for autonomous flight using onboard computer vision," in *2011 IEEE International Conference on Robotics and Automation*, pp. 2992–2997, IEEE, 2011.

[61] L. Meier, D. Honegger, and M. Pollefeys, "Px4: A node-based multithreaded open source robotics framework for deeply embedded platforms," in *2015 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 6235–6240, May 2015.

[62] A. Koubaa, A. Allouch, M. Alajlan, Y. Javed, A. Belghith, and M. Khalgui, "Micro air vehicle link (mavlink) in a nutshell: A survey," *arXiv preprint arXiv:1906.10641*, 2019.

[63] G. Bradski, "The OpenCV Library," *Dr. Dobb's Journal of Software Tools*, 2000.

[64] F. Furrer, M. Burri, M. Achtelik, and R. Siegwart, *Robot Operating System (ROS): The Complete Reference (Volume 1)*, ch. RotorS—A Modular Gazebo MAV Simulator Framework, pp. 595–625. Cham: Springer International Publishing, 2016.

[65] J. Meyer, A. Sendobry, S. Kohlbrecher, U. Klingauf, and O. von Stryk, "Comprehensive simulation of quadrotor uavs using ros and gazebo," in *3rd Int. Conf. on Simulation, Modeling and Programming for Autonomous Robots (SIMPAR)*, p. to appear, 2012.

[66] L. Comba, P. Gay, J. Primicerio, and D. R. Aimonino, "Vineyard detection from unmanned aerial systems images," *Computers and Electronics in Agriculture*, vol. 114, pp. 78 – 87, 2015.

[67] R. Ji and L. Qi, "Crop-row detection algorithm based on random hough transformation," *Mathematical and Computer Modelling*, vol. 54, no. 3, pp. 1016 – 1020, 2011. Mathematical and Computer Modeling in agriculture (CCTA 2010).

[68] K. Ramesh, N. Chandrika, S. Omkar, M. Meenavathi, and V. Rekha, "Detection of rows in agricultural crop images acquired by remote sensing from a uav," *International Journal of Image, Graphics and Signal Processing*, vol. 8, no. 11, p. 25, 2016.

[69] G. A. Soares, D. D. Abdala, and M. Escarpinati, "Plantation rows identification by means of image tiling and hough transform.," in *VISIGRAPP (4: VISAPP)*, pp. 453–459, 2018.

[70] R. Kresch and D. Malah, "Skeleton-based morphological coding of binary images," *IEEE Transactions on Image Processing*, vol. 7, pp. 1387–1399, Oct 1998.

[71] A. W. Robert Fisher, Simon Perkins and E. Wolfart, "Skeletonization/medial axis transform," 2003.

[72] J. Serra and P. Soille, *Mathematical morphology and its applications to image processing*, vol. 2. Springer Science & Business Media, 2012.

[73] P. V. Hough, "Machine analysis of bubble chamber pictures," in *Conf. Proc.*, vol. 590914, pp. 554–558, 1959.

[74] D. H. Ballard, "Generalizing the hough transform to detect arbitrary shapes," *Pattern recognition*, vol. 13, no. 2, pp. 111–122, 1981.

[75] R. O. Duda and P. E. Hart, "Use of the hough transformation to detect lines and curves in pictures," tech. rep., Sri International Menlo Park Ca Artificial Intelligence Center, 1971.

[76] C. F. F. Karney, "Geographiclib."

Anexos

A. Instalación JetPack y OpenCV con soporte para CUDA en NVIDIA TX2

A.1. Instalación JetPack

Para instalar los componentes de *software* en la NVIDIA Jetson TX2 se puede hacer uso de los *SDK Manager* proveídos por NVIDIA. Para usar esa herramienta se debe contar con un computador con sistema operativo Linux, en el cual se instalará el *Manager*, este computador actuara como *host* o huésped para descargar los paquetes o componentes de software que posteriormente serán instalados en el *target* (Jetson TX2). Para el proyecto se uso la versión 3.3 del *SDK Manager*[1].

Una vez se otorguen permisos para ejecutar al archivo se puede proceder con el proceso de instalación, cuyas instrucciones se encuentran resumidas en tutoriales[2]Para computadores que actúen como huésped y que cuenten con una tarjeta gráfica NVIDIA, el *manager* ofrece una forma sencilla de instalar CUDA y OpenCV, por lo cual se recomienda dejar las opciones por defecto, pero si este no es el caso, se recomienda desmarcar todas las opciones del dispositivo huésped. En cuanto al dispositivo objetivo es importante mencionar que si se habilitan las opciones *Jetson OS Image* y *Flash Jetson OS*, todos los datos de la tarjeta TX2 serán borrados para instalar el sistema operativo *L4T*, por lo que se debe tener un*back-up* de la información si se ha venido trabajando previamente en otros proyectos.

Durante la instalación del JetPack, existe un paso en el que se requiere el uso de una red de Internet para comunicar los dos dispositivos y se observo que usando una red Wi-Fi se presentaron dificultades, por lo cual se recomienda usar una conexión *Ethernet* tanto en el dispositivo huésped, como en el objetivo. Adicionalmente, se debe conocer la interfaz de red por la cual el dispositivo huésped se conecta a la red que lo comunica con el dispositivo objetivo, lo cual puede hacerse mediante comandos

[1]SDK Manager v3.3: https://developer.NVIDIA.com/embedded/jetpack-3_3

[2]Tutorial 1: https://docs.NVIDIA.com/sdk-manager/install-with-sdkm-jetson/index.html
Video Tutorial: https://www.youtube.com/watch?v=D7lkth34rgM

como *ip link show* y *ifconfig -a*

Para verificar que el proceso de instalación fue satisfactorio se pueden ejecutar algunos programas de muestra que se generan automáticamente en el proceso de instalación. En el vídeo tutorial se explica como ejecutar uno de ellos, aunque también se pueden seguir las instrucciones de la pagina oficial[3].

A.2. Instalación OpenCV con soporte para GPU

Aunque entre los componentes instalados del JetPack se encuentran las librerías de OpenCV, estas no cuentan con soporte para GPU, por lo cual es necesario descargar o clonar los archivos fuentes de las librerías que si brindan tal soporte y compilarlos en la TX2. Sin embargo, es importante mencionar que para poder llevar a cabo la instalación de las librerías de OpenCV optimizadas para GPU, se debe contar con alguna versión de CUDA en el dispositivo y conocer la ruta donde se llevo a cabo la instalación. Durante la instalación de CUDA mediante el *SDK manager* se escogió la ruta /usr/local, como se recomienda en las guías encontradas.

A continuación se proporcionan las instrucciones necesarias para realizar la instalación. Sin embargo, también se sugiere seguir las recomendaciones de los siguientes enlaces

El siguiente *script* contiene las comandos ejecutados para la instalación sobre la NVIDIA TX2.

```
# /bin/bash.sh
## Instalacion de librerias sobre las que open-cv depnede
sudo apt-get install -y libglew-dev libtiff5-dev zlib1g-dev \
libjpeg-dev libpng12-dev libjasper-dev libavcodec-dev\
libavformat-dev libavutil-dev libpostproc-dev libswscale-dev \
libeigen3-dev libtbb-dev libgtk2.0-dev pkg-config
sudo apt-get install -y python-dev python-numpy python-py python-pytest
sudo apt-get install -y python3-dev python3-numpy python3-py python3-pytest

cd
mkdir opencv3
cd opencv3
##  Clonar repositorio
git clone https://github.com/opencv/opencv.git opencv
cd opencv
git checkout 3.3.1
cd ..
##  OpenCV Extras
git clone https://github.com/opencv/opencv_extra.git
cd opencv_extra/
git checkout 3.3.1
cd ..
cd opencv
```

[3]Ejemplos pagina oficial: https://docs.NVIDIA.com/jetson/l4t-multimedia/l4t_mm_test_group.html

```
##  Crear directorio para construir los paquetes
mkdir build
cd build/
##  Configuracion de la comiplacion e instalacion
export PATH=$PATH:/usr/local/cuda/bin
export LD_LIBRARY_PATH=/usr/local/cuda-9.0/lib64
cmake     -D CMAKE_BUILD_TYPE=Release     -D CMAKE_INSTALL_PREFIX=/usr/local     -
  D BUILD_PNG=OFF     -D BUILD_TIFF=OFF     -D BUILD_TBB=OFF     -D BUILD_JPEG=OFF
       -D BUILD_JASPER=OFF     -D BUILD_ZLIB=OFF     -D BUILD_EXAMPLES=ON     -D
  BUILD_opencv_java=OFF     -D BUILD_opencv_python2=ON     -D BUILD_opencv_python3
  =ON     -D ENABLE_PRECOMPILED_HEADERS=OFF     -D WITH_OPENCL=ON     -D
  WITH_OPENMP=OFF     -D WITH_FFMPEG=ON     -D WITH_GSTREAMER=OFF     -D
  WITH_GSTREAMER_0_10=OFF     -D WITH_CUDA=ON     -D WITH_GTK=ON     -D WITH_VTK=
  OFF     -D WITH_TBB=ON     -D WITH_1394=OFF     -D WITH_OPENEXR=OFF     -D
  CUDA_TOOLKIT_ROOT_DIR=/usr/local/cuda-9.0     -D CUDA_ARCH_BIN=6.2     -D
  CUDA_ARCH_PTX=""     -D INSTALL_C_EXAMPLES=ON     -D INSTALL_TESTS=OFF     -D
  OPENCV_TEST_DATA_PATH=../opencv_extra/testdata     ../opencv ..
##  Comiplar librerias con 4 de los procesadores de la CPU del dispositivo
make -j4
##  Instalar librerias
sudo make install
```

Listing A.1: Instalación OpenCV en TX2

El siguiente *script* contiene las comandos ejecutados para la instalación de las librerías de openCV en el computador de simulación. La principal diferencia radica en la versión de CUDA utilizada, ademas que para el computador de simulación no se sabe el valor especifico de la variable CUDA_ARCH_BIN, razón por la cual es necesario incluir múltiples valores, lo cual puede incrementar el tiempo de compilación, sin embargo, el computador cuenta con mas procesadores para realizar la compilación, compensando un poco esta situación.

```
# /bin/bash.sh
## Instalacion de librerias sobre las que open-cv depnede
sudo apt-get install -y libglew-dev libtiff5-dev zlib1g-dev \
libjpeg-dev libpng12-dev libjasper-dev libavcodec-dev\
libavformat-dev libavutil-dev libpostproc-dev libswscale-dev \
libeigen3-dev libtbb-dev libgtk2.0-dev pkg-config
sudo apt-get install -y python-dev python-numpy python-py python-pytest
sudo apt-get install -y python3-dev python3-numpy python3-py python3-pytest

cd
mkdir opencv3
cd opencv3
##  Clonar repositorio
git clone https://github.com/opencv/opencv.git opencv
cd opencv
git checkout 3.3.1
cd ..
##  OpenCV Extras
git clone https://github.com/opencv/opencv_extra.git
cd opencv_extra/
git checkout 3.3.1
cd ..
cd opencv
##  Crear directorio para construir los paquetes
```

```
mkdir build
cd build/
##  Configuracion de la comiplacion e instalacion
export PATH=$PATH:/usr/local/cuda/bin
export LD_LIBRARY_PATH=/usr/local/cuda-8.0/lib64
cmake -D CMAKE_BUILD_TYPE=Release -D CMAKE_INSTALL_PREFIX=/usr \
-D BUILD_PNG=OFF -D BUILD_TIFF=OFF -D BUILD_TBB=OFF -D BUILD_JPEG=OFF \
-D BUILD_JASPER=OFF -D BUILD_ZLIB=OFF -D BUILD_EXAMPLES=ON \
-D BUILD_JAVA=OFF -D BUILD_opencv_python2=ON -D BUILD_opencv_python3=OFF \
-D WITH_OPENCL=OFF -D WITH_OPENMP=OFF -D WITH_FFMPEG=ON \
-DWITH_GSTREAMER=OFF -DWITH_GSTREAMER_0_10=OFF -DWITH_CUDA=ON \
-DWITH_GTK=ON -DWITH_VTK=OFF -DWITH_TBB=ON -DWITH_1394=OFF \
-DWITH_OPENEXR=OFF -DCUDA_TOOLKIT_ROOT_DIR=/usr/local/cuda-8.0 \
-DCUDA_ARCH_BIN='3.0 3.5 5.0 6.0 6.2' -DCUDA_ARCH_PTX="" \
-DINSTALL_C_EXAMPLES=ON -DINSTALL_TESTS=OFF \
-DOPENCV_TEST_DATA_PATH=../opencv_extra/testdata \
../opencv ..
##  Comiplar librerias con siete de los procesadores de la CPU del dispositivo
make -j7
##  Instalar librerias
sudo make install
```

Listing A.2: Instalación OpenCV en Computador de simulación

B. Instalación ROS

El tipo de instalación de *ROS* dependerá de las tareas a desarrollar en el computador huésped. En el caso del computador portátil es conveniente instalar la distribución *desktop*, pues incluye las herramientas de simulación e interfaz gráfica que facilitan la interacción con el sistema. Por otro lado y teniendo en cuenta que el objetivo de la NVIDIA es funcionar como computador de compañía, no es necesario instalar ni el simulador ni herramientas de soporte gráfico, por lo cual basta con instalar la distribución base, donde se incluyen los paquetes esenciales de comunicación.

B.1. Instalación NVIDIA TX2

B.1.1. Sin soporte para GPU

A continuación se proporciona un *script* de bash que permite realizar la instalación base de la distribución *ROS*-kinetic, en la que se usan las librerías de OpenCv precompiladas por *ROS* sin soporte para GPU, a partir de las instrucciones proveidas en la pagina oficial[1].

```
# /bin/bash.sh
# Actualizar repositorios
sudo apt-get update
# Actualizar llaves
sudo sh -c 'echo "deb http://packages.ros.org/ros/ubuntu $(lsb_release -sc) main"
  > /etc/apt/sources.list.d/ros-latest.list'
sudo apt-key adv --keyserver 'hkp://keyserver.ubuntu.com:80' --recv-key
  C1CF6E31E6BADE8868B172B4F42ED6FBAB17C654
sudo apt-get update
# Instalar ros kinetik base
sudo apt-get install ros-kinetic-base
# inicializar rosdep
sudo rosdep init
rosdep update
# configurar variables de entorno
source /opt/ros/kinetic/setup.bash
# Intalar dependencias para construir paquetes de ros
sudo apt-get -y install python-rosinstall python-rosinstall-generator python-
  wstool build-essential
sudo apt-get -y install ros-kinetic-controller* ros-kinetic-octomap* ros-kinetic-
  hardware-interface ros-kinetic-mav*
# Crear workspace
```

[1]Instalación ROS pagina oficial: http://wiki.ros.org/kinetic/Installation

```
cd ~
mkdir -p ~/catkin_ws/src
cd ~/catkin_ws/
catkin_make

source devel/setup.bash
cd ~
# Redirecciona el texto al archivo indicado para que es habilite la opcion auto-
  completar de los comandos de ROS cada vez que se inicie una nueva consola.
echo "source /opt/ros/kinetic/setup.bash" >> .bashrc
echo "source ~/catkin_ws/devel/setup.bash" >> .bashrc
```

Listing B.1: Instalación ROS-Kinetic base sin oporte GPU

B.1.2. Con soporte para GPU

Para realizar la instalación de *ROS* con soporte para GPU es necesario contar con las librerías de OpenCv compiladas con compatibilidad para CUDA, bien sea siguiendo las recomendaciones del anexo A o de cualquier otra manera.

El primer paso es ejecutar el *script* de instalación para ROS B.1. Luego, se debe eliminar el paquete *cv_bridge* que viene por defecto, para compilarlo en el *workspace* a partir de los archivos fuentes. A continuación se proporciona un *script bash* en el que se ejecutan los pasos que permiten habilitar el soporte para GPU. Se usa un repositorio donde se modifica el *cmake list* de manea que indique que la compilación de hacerse usando las librerías de OpenCv con soporte para CUDA, en lugar de las librerías pre-compiladas por ROS.

```
# /bin/bash.sh
# Actualizar repositorios
sudo apt-get update
########### Desinstalar cv_bridge
sudo apt-get remove ros-kinetic-cv-bridge
########### Instalar cv_bridge apartir de los archivos fuente (src)
cd ~/catkin_ws/src/
git clone https://github.com/daniel-dsouza/vision_opencv.git
cd ~/catkin_ws/
catkin_make
```

Listing B.2: bash example

B.2. Instalación Computador de simulación

B.2.1. Sin soporte para GPU

Aunque la instalación de la versión *desktop* de *ROS*que incluye herramientas de simulación se recomienda que el equipo cuente con tarjeta de procesamiento gráfico independiente, es posible realizar la instalación en computadores que no la tengan. A

continuación se proporciona un *script* que ejecuta tal instalación de la versión *desktop* de *ROS*-kinetic, que incluye los paquetes de simulación y algunos para soporte de interfaz gráfica.

```
# /bin/bash.sh
# Actualizar repositorios
sudo apt-get update
# Actualizar llaves
sudo sh -c 'echo "deb http://packages.ros.org/ros/ubuntu $(lsb_release -sc) main"
  > /etc/apt/sources.list.d/ros-latest.list'
sudo apt-key adv --keyserver 'hkp://keyserver.ubuntu.com:80' --recv-key
  C1CF6E31E6BADE8868B172B4F42ED6FBAB17C654
sudo apt-get update
# Instalar ros kinetik desktop
sudo apt-get install ros-kinetic-desktop-full
# inicializar rosdep
sudo rosdep init
rosdep update
# configurar variables de entorno
source /opt/ros/kinetic/setup.bash
# Intalar dependencias para construir paquetes de ros
sudo apt-get -y install python-rosinstall python-rosinstall-generator python-
  wstool build-essential
sudo apt-get -y install ros-kinetic-controller* ros-kinetic-octomap* ros-kinetic-
  hardware-interface ros-kinetic-mav*
# Crear workspace
cd ~
mkdir -p ~/catkin_ws/src
cd ~/catkin_ws/
catkin_make
source devel/setup.bash
cd ~
# Redirecciona el texto al archivo indicado para que es habilite la opcion auto-
  completar de los comandos de ROS cada vez que se inicie una nueva consola.
echo "source /opt/ros/kinetic/setup.bash" >> .bashrc
echo "source ~/catkin_ws/devel/setup.bash" >> .bashrc
```

Listing B.3: Instalación ROS-Kinetic base sin oporte GPU

B.2.2. Con soporte para GPU

Si el equipo donde se ejecutara la simulación cuenta con una tarjeta de video NVIDIA y se han compilado las librerías de OpenCv con soporte para CUDA, es posible habilitar *ROS* para que trabaje con soporte para GPU, al igual que se hizo con la NVIDIA.

El primer paso es ejecutar el *script* de instalación para ROS en su versión *desktop*. Luego, se debe eliminar el paquete *cv_bridge* que viene por defecto, el cual a su vez eliminara algunos paquetes adicionales. Algunos de estos paquetes deberán ser compilados directamente en el *workspace* de *ROS* , como se muestra en el siguiente script.

```
# /bin/bash.sh
# Actualizar repositorios
sudo apt-get update
sudo apt-get install -y libgazebo7-dev
cd ~/catkin_ws/src
sudo apt-get install -y ros-kinetic-control* ros-kinetic-transmission-interface
  ros-kinetic-joint-limits-interface
git clone https://github.com/ros-simulation/gazebo_ros_pkgs.git -b kinetic-devel
rosdep update
#Verificar dependencias faltantes:
rosdep check --from-paths . --ignore-src --rosdistro kinetic
#Instalar dependencias faltantes:
rosdep install --from-paths . --ignore-src --rosdistro kinetic -y
cd ~/catkin_ws/
catkin_make
```

Listing B.4: Instalación Gazebo con soporte GPU

```
# /bin/bash.sh

# Actualizar repositorios
sudo apt-get update
#clonar paquete image_common
cd ~/catkin_ws/src
git clone https://github.com/ros-perception/image_common.git
rosdep update
#Verificar dependencias faltantes:
rosdep check --from-paths . --ignore-src --rosdistro kinetic
#Instalar dependencias faltantes:
rosdep install --from-paths . --ignore-src --rosdistro kinetic -y
cd ~/catkin_ws/
catkin_make
# clonar el paquete image_geometry de vision_opencv. No se clona el repositorio
  directamente en el workspace, pues ya se tiene el paquete de cv_bridge que
  compila con soporte para GPU.
cd ~/Documents
git clone https://github.com/ros-perception/vision_opencv.git vision_opncv
cp ~/Documents/vision_opencv/image_geometry ~/catkin_ws/src/image_geometry
```

Listing B.5: Instalación componentes ROS-Percpection

```
# /bin/bash.sh

# Actualizar repositorios
sudo apt-get update
cd ~/catkin_ws/src
git clone https://github.com/ros-visualization/rqt_image_view.git
git clone https://github.com/ros-visualization/rqt_common_plugins.git
rosdep update
#Verificar dependencias faltantes:
rosdep check --from-paths . --ignore-src --rosdistro kinetic
#Instalar dependencias faltantes:
rosdep install --from-paths . --ignore-src --rosdistro kinetic -y
cd ~/catkin_ws/
catkin_make
```

Listing B.6: Instalación componentes ros-visualization

La razón para habilitar esta opción es que el proceso de escritura y desarrollo de código es más sencillo al poder realizar pruebas sobre el mismo computador de simulación, pues existe un *plugin* con soporte para ROS para el IDE Qt-Creator. Este es capaz de establecer enlaces con las rutas donde se encuentran las librerías indicadas dentro de un archivo *source* o *header* que pertenezcan a un paquete o proyecto de *ROS* desarrollado en lenguaje c_{++}, sin necesidad de hacer configuraciones tediosas y brindando opciones útiles como auto-completar. Lamentablemente, aunque Qt-creator puede instalarse en dispositivos con arquitectura ARM, como la NVIDIA TX2, para el momento de la elaboración de este documento, el *plugin* aun no era compatible con tal arquitectura, razón por la cual se opto por hacer el desarrollo y pruebas iniciales en el computador de simulación, para luego hacer las pruebas definitivas en el dispositivo embebido. Otra alternativa es utilizar Qt-Creator para desarrollar directamente en la TX2, sin embargo, se deberá buscar información sobre como realizar la configuración para asociar las librerías al proyecto y poder verse beneficiado de las utilidades del IDE.

Para instalar el *plugin* se puede acceder al repositorio[2] y descargar el archivo de instalación preferido, se recomiendan las versiones 4.4 y 4.8. También se pueden seguir los pasos de instalación avanzados disponibles en el la guia oficial [3]. Una vez instalado el *plugin* se debe ejecutar el programa Qt-Creator e importar el *workspace* de ROS.

[2]plugin ROS industrial: https://qtcreator-ros.datasys.swri.edu/downloads/installers/xenial/archived/
[3]Guia instalación avanzada plugin: https://ros-qtc-plugin.readthedocs.io/en/latest/_source/Improve-ROS-Qt-Creator-Plugin-Developers-ONLY.html

C. Instalación Paquetes de planeación

C.1. Instalación Geographic-Lib y mavros

Las librerías del paquete GeographicLib ofrecen herramientas para realizar transformaciones (directas e inversas) entre múltiples sistemas de referencia geográficos con un alto grado de precisión, como se describe en su [1] , este conjunto de herramientas es utilizado por el paquete mavros, pero también fueron utilizadas en los paquetes *flight_planning* y *mavros_offboard_control*, por lo cual se considera importante proveer asistencia para poder usarlas.

Para instalar las librerías del paquete geographiclib se pueden seguir los pasos indicados por los desarrolladores de las librerías[2], donde ofrecen varias alternativas para realizar la instalación dependiendo del sistema operativo, en caso de que se quiera usar la librería en otras aplicaciones. Sin embargo, a continuación se presenta un *script* en donde se ejecutan los pasos para realizar la instalación con las mismas características usadas para el proyecto.

```
# /bin/bash.sh
git clone git://git.code.sourceforge.net/p/geographiclib/code geographiclib
cd geographiclib
mkdir BUILD
cd BUILD
cmake ..
cmake -D CMAKE_INSTALL_PREFIX=/usr/local -D GEOGRAPHICLIB_DATA=/usr/local/share/
  GeographicLib -D GEOGRAPHICLIB_LIB_TYPE=SHARED -D CMAKE_BUILD_TYPE=Release .
make # compile the library and utilities
make test # run some tests
sudo make install # as root, if CMAKE_INSTALL_PREFIX is a system directory
```

Por otro lado el siguiente comando instalara todos los paquetes de mavros y *MAVLink* necesarios para el funcionamiento de los paquetes y el simulador.

```
# /bin/bash.sh
sudo apt-get install -y ros-kinetic-mav*
```

[1]Pagina oficial GeographicLib: https://geographiclib.sourceforge.io/html/intro.html
[2]Instrucciones instalación GeographicLib: https://geographiclib.sourceforge.io/html/install.html

C.2. Instalación paquete crop_image_server

A continuación se proveen una serie de *scripts* para instalar los paquetes generados durante el proyecto, se recomienda instalar los paquetes del proyecto en el orden que se presentan en el documento, pues por ejemplo los paquetes *flight_plannig* y *mavros_offboard_control* dependen de los mensajes y servicios generados en el paquete *crop_image_processing*.

```
# /bin/bash.sh
##Instalar dependencias aparte de open_cv y cv_bridge
sudo apt-get install -y ros-kinetic-sensor_msgs
##Clonar y compilar paquete
cd ~/catkin_ws/src
git clone git@github.com:DanielPalominoS/crop_image_procesing.git
  crop_image_procesing
cd ~/catkin_ws
catkin_make
catkin_make
catkin_make
```

En ocasiones es necesario ejecutar el comando **catkin_make** varias veces, pues se deben generar los mensajes y servicios asociados al paquete. Una primera compilación generar los mensajes del paquete, la segunda compilación generar los servicios que dependan de los mensajes recien generados, mientras que la ultima compilación generará los ejecutables que usan estos mensajes y servicios. Aveces es necesario comentar las lineas del archivo **CMakeLists** asociadas al ejecutable y una vez que los mensajes y servicios sean compilados satisfactoriamente se puede proceder a descomentar y compilar los ejecutables.

C.3. Instalación paquete flight_planning

```
# /bin/bash.sh
##Instalar dependencias
sudo apt-get install -y ros-kinetic-geometry-msgs ros-kinetic-tf2-geometry-msgs
##Clonar y compilar paquete
cd ~/catkin_ws/src
git clone git@github.com:DanielPalominoS/flight_planning.git flight_planning
cd ~/catkin_ws
catkin_make
catkin_make
catkin_make
```

C.4. Instalación paquete mavros_offboard_control

```
# /bin/bash.sh
##Instalar dependencias
```

```
sudo apt-get install -y ros-kinetic-mav-planning-msgs
##Clonar y compilar paquete
cd ~/catkin_ws/src
git clone git@github.com:DanielPalominoS/mavros_offboard_control.git
  mavros_offboard_control
cd ~/catkin_ws
catkin_make
catkin_make
catkin_make
```

D. Instalación Entorno de simulación

Para realizar la instalación del entorno de simulación es necesario clonar el repositorio donde se encuentra el *firmware* de PX4 y compilar algunos archivos. La guía para desarrollador [1] cuenta con una amplia documentación de muchas de las utilidades disponibles en el *firmware* , así como del proceso de instalación. Sin embargo, a continuación se muestra un resumen con los pasos mas importantes a tener en cuenta para poder trabajar con las herramientas de simulación compatibles con Gazebo.

La guía de desarrollo recomienda hacer una bifurcación(*fork*) del repositorio en caso de que se vayan a realizar cambios permanentes sobre los archivos del *firmware* , para lo cual es necesario tener una cuenta en GitHub, posteriormente el repositorio se podrá clonar a nivel local con un *script* como el que se muestra a continuación.

```
cd ~/<instalation_directory_path>
git clone https://github.com/<youraccountname>/Firmware.git
```

Si no se busca realizar cambios permanentes sobre los archivos fuente basta con clonar el repositorio original, para lo cual se puede ejecutar un *script* como el siguiente.

```
cd ~/<instalation_directory_path>
git clone https://github.com/PX4/Firmware.git
cd Firmware
```

Una vez el repositorio ha sido clonado es posible obtener una versión especifica del software mediante la instrucción *checkout* de git, lo cual es útil en casos donde se quiere cargar el *firmware* en una versión mas antigua o estable. Para la compilación de los componentes de simulación se uso la version 1.9.0, que era la ultima versión estable en el momento donde se clono el repositorio.

```
git checkout v1.9.0
```

El siguiente paso es compilar los archivos que crean una copia virtual del *firmware* , la cual puede ser ejecutada directamente en el computador de simulación en la modalidad SITL, así como los *plugins* para controlar los modelos de Gazebo. El siguiente *script* permite compilar los archivos y agregar el directorio del *firmware* a

[1] Pagina web guía para desarrollador: https://dev.px4.io/master/en/setup/building_px4.html

la variable de entorno que maneja los paquetes de *ROS* , así como añadir los archivos bash asociados, para que en cada inicio de la ventana de comandos se pueda hacer uso de los elementos compilados en el entorno de ROS. Es necesario indicar la ruta adecuada al directorio del *firmware* .

```
cd <Firmware_clone>
DONT_RUN=1 make px4_sitl_default gazebo
source ~/catkin_ws/devel/setup.bash    # (optional)
source ~/<Firmware_clone>/Tools/setup_gazebo.bash ~/<Firmware_clone>/ ~/<
  Firmware_clone>//build/px4_sitl_default
export ROS_PACKAGE_PATH=$ROS_PACKAGE_PATH:~/<Firmware_clone>/
export ROS_PACKAGE_PATH=$ROS_PACKAGE_PATH:~/<Firmware_clone>//Tools/sitl_gazebo
```

Posteriormente se puede verificar la instalación haciendo uso de los archivos *launch* disponibles en el repositorio clonado. Algunos de esos archivos ejemplo requieren que se haya instalado el paquete mavros, que permite la comunicación del *firmware* px4 con el entorno *ROS* por medio del protocolo MavLink. El primer ejemplo que se sugiere ejecutar es el de la siguiente linea de comando, donde unicamente se ejecutara una instancia de Gazebo con un mundo básico y un modelo de *UAV* conectado bien sea al *firmware* virtual (modo SITL) o al autopiloto real (modo HITL).

```
roslaunch px4 posix_sitl.launch
```

Con el fin de establecer el modo de operación se debe acceder al archivo SDF del modelo a utilizar, el cual estará ubicado en la ruta. Para el proyecto la variable *"model"* corresponde a iris, para usar el modelo del Iris Quadcopter.

```
"Firmware_clone_path"/Tools/sitl\_gazebo/models/"model"/"model".sdf
```

Como primer paso para operar en modo SITL se debe garantizar que las etiquetas **serialEnabled** y **hil_mode** tengan asignado el valor false. La condición anterior es importante, pues existe un *plugin* denominado *mavlink_interface*, que se encarga de establecer la comunicación entre el simulador y el *firmware*.

Este *plugin* es ejecutado cuando el modelo es lanzado en el entorno de simulación, donde toma la información publicada por el simulador, como la de los sensores y la organiza en mensajes de formato *MAVLink* [2], luego transmite la información al *firmware* usando comunicación serial, UDP o IP. Una vez el *firmware* recibe y procesa la información, transmite los comandos de los motores para operar el vehículo, cuya información es recibida por el *plugin*, para finalmente publicarla en los actuadores del entorno de simulación. Específicamente en el modo SITL la comunicación se realiza mediante el protocolo UDP a través de unos puertos especificados en los archivos SDF del modelo a utilizar, por lo cual es necesario que no se encuentre habilitada la comunicación serial.

[2]Mensajes *MAVLink*: https://mavlink.io/en/messages/common.html

En caso de que ya se haya instalado el paquete mavros, se puede hacer uso de la versión virtual del *firmware* para ejecutar el nodo ejemplo de control[3], con el fin de verificar que la comunicación en el entorno de *ROS* entre el dron, los sensores de simulación y el *firmware* funciona correctamente. Para el proyecto se implemento dicho ejemplo con ligeras modificaciones en el paquete *mavros_offboard_control* y para ejecutarlo, asumiendo que ya fue compilado exitosamente, se deben seguir los siguientes pasos.

1. Seleccionar el modelo a utilizar en el archivo launch.
2. Verificar que las variables **serialEnabled** y **hil_mode**, del archivo sdf asociado al modelo seleccionado sean igual a *false*.
3. Ejecutar el archivo *launch*:

```
roslaunch px4 mavros_posix.launch
```

4. Ejecutar nodo offboard:

```
rosrun mavros_offboard_control offb_node
```

Si todo funcionan correctamente se obtendrá un comportamiento similar al mostrado en la sección correspondiente de la pagina de PX4. En este punto aun no es necesario involucrarse con el sistema autopiloto y se pueden hacer múltiples pruebas.

Ahora, si en cambio se quiere que el modelo del entorno de simulación funcione en la modalidad HITL se deben cambiar los valores de los parámetros **serialEnabled** y **hil_mode** a *true*, ademas, antes de ejecutar la simulación se debe conocer el nombre del puerto serial donde el autopiloto esta conectado para configurar el parámetro **serialDevice** en el archivo SDF del modelo seleccionado, ya que el *plugin mavlink_interface* necesita tal información para establecer la comunicación entre el simulador y el autopiloto. También es importante que el *Baudrate* sea por lo menos de 921600, ya que se requiere una tasa de transmisión de alta velocidad pues por ejemplo, el modo *offboard* cuenta con un *timeout* dentro del cual se tiene que recibir un mensaje para que el autopiloto se mantenga operando en este modo.

Finalmente, para poder hacer uso de los modelos virtuales de cultivos desarrollados en Blender se deben seguir las siguientes indicaciones.

1. Descargar o clonar los archivos de los modelos de los cultivos.

[3]Código ejemplo control offboard:https://dev.px4.io/master/en/ros/mavros_offboard.html

```
cd <DownloadPath>
git clone git@github.com:DanielPalominoS/sim_crop.git
```

2. De los archivos recién descargados acceder a la carpeta *models* y copiar todas los archivos que esta contiene. Vale la pena destacar que en la carpeta terreno se encuentran archivos de extensión .blend y .dae, donde los primeros son los archivos que contienen los modelos de los surcos diseñados dentro del software Blender, mientras que los archivos .dae son generados desde el mismo *software* al ejecutar un proceso de exportación a formato collada, ya que este es el formato compatible para describir modelos en Gazebo

   Pegar los archivos en la ruta *"Firmware_clone_path"*/Tools/sitl_gazebo/models/. En esta carpeta es donde se encuentran los archivos que describen los diferentes modelos, descritos en archivos *SDF*, desarrollados para llevar a cabo la simulación en Gazebo.

3. Acceder a la carpeta *world* de *sim_crop* y copiar el archivo crop.world.

   Pegar el archivo en la ruta *"Firmware_clone_path"*/Tools/sitl_gazebo/worlds/. En esta carpeta se encuentran los archivos que describen los diferentes mundos o entornos de simulación, donde en cada uno de ellos se pueden integrar usar uno o varios de los modelos disponibles en la carpeta *models*. Particularmente el archivo crop.world hace uso de los planos de tierra y del modelo terreno, pero se puede generar cualquier tipo de escenario, bien sea usando los modelos predefinidos o modelos de creación propia.

4. Finalmente, se deben copiar los archivos *launch* del repositorio descargado o clonado, estos se deberán pegar en la ruta *"Firmware_clone"*/launch/, en donde se encuentran los archivos para lanzar uno o varios procesos de ROS.

   En cuanto a los archivos launch generados, el archivo *spawn_vehicle.launch* se encarga de ejecutar Gazebo con el mundo indicado mediante el argumento *"world"*, así como lanzar un vehículo, especificado mediante el argumento *vehicle*.

   Por otro lado el archivo *spawn_vehicle_with_mavros.launch* cumple la misma función que el archivo recien descrito, pero adicionalmente inicia una instancia de mavros que se puede configurar meidante los parametros *fcu_url* y *gcs_url*, siendo el primero el usado para establecer la comunicación entre ros y el autopiloto, mientras el segundo se usa para la comunicación entre ros y una estación terrestre (si esta opera en el mismo computador).

Para el proyecto el parámetro *fcu_url* se configuro para establecer una conexión UDP entre ROS y el *plugin mavlink_interface* (fcu_url=udp://:14540@localhost:14555), donde este ultimo al estar conectado serialmente con el autopiloto actúa como puente o enlace para transmitir información entre el simulador, ros y el autopiloto. Sin embargo, cabe aclarar que también es posible establecer una comunicación serial directa entre el auotipilto y ROS, como se indica en el repositorio[4], siendo esta la configuración a ser usada cuando se remueva el componente de simulación.

Para ejecutar los archivos *launch* se deberá ejecutar un comando como el siguiente:

```
roslaunch px4 spawn_vehicle.launch x:=0 y:=0 z:=0
```

Los argumentos del archivo launch como *x,y,z y los demás* pueden ser configurados al momento del lanzamiento en caso de querer iniciar la simulación con condiciones personalizadas. Sin embargo, no es obligatorio configurar realizar tal configuración, pues simplemente se pueden usar los valores por defecto definidos en el archivo.

[4]mavros: https://github.com/mavlink/mavros/tree/master/mavros

E. Configuración Pixhawk

E.1. Actualizar firmware

Para cargar el *firmware* en el sistema embebido Pixhawk 2 existen dos alternativas principales.

E.1.1. Linea de comandos

Usando la ventana de comandos y una vez se tiene el dispositivo conectado, se debe ir al directorio donde se tenga una copia del *firmware* , allí se pueden compilar los codigos fuentes en sus diferentes versiones de desarrollo de acuerdo al dispositivo objetivo.Para los dispositivos de la linea Pixhawk o aquellos otros basadas en NuttX se cuenta con una distribución específica de *firmware* según se indica en la pagina oficial, particularmente para Pixhawk 2 se recomienda usar la distribución v3. Si no se ha compilado el *firmware* previamente, se puede ejecutar el comando ***make px4_fmu-v3_default***, donde x se reemplaza por la distribución deseada para el dispositivo. Una vez culmina la compilación se generara un archivo de extensión .px4, el cual puede ser cargado mediante la opción ***upload***.

```
make px4_fmu-v3_default
make px4_fmu-v3_default upload
```

En el caso de que no se haya hecho uso del comando *checkout* de Git para registrarse en una versión o entrega anterior del repositorio se cargara la mas reciente para cada distribución, sin embargo, para el momento en que el documento fue realizado se encontró que tanto los ediciones 1.8.x y 1.9.x presentaban algunas fallas para operar en la modalidad HITL, similar a como se indica en los *issues* 146[1] y 159[2] del repositorio. Con base en lo anterior se clono el *firmware* en una ruta diferente de donde se guardo la copia utilizada para la simulación y allí se cambio a la versión 1.7.3, que fue la versión mas avanzada donde la modalidad HITL funciono adecuadamente para la *board* Pixhawk 2, siendo esta la versión del *firmware* que se cargo. El siguiente script resume tal procedimiento.

[1]Issue 146: https://github.com/PX4/sitl_gazebo/pull/146
[2]Issue 159: https://github.com/PX4/sitl_gazebo/pull/157

```
mkdir ~/<instalation_directory_path_2>
cd ~/<instalation_directory_path_2>
git clone https://github.com/<youraccountname>/Firmware.git
git checkout v1.7.3
make px4_fmu-v3_default
make px4_fmu-v3_default upload
```

E.1.2. Q-GroundControl

La segunda opción para cargar el *firmware* es usar el software libre QGroundControl. En la sección de ajustes existe una pestaña dedicada para tal propósito, allí se podrá elegir entre PX4 y ArduPilot según las preferencias del usuario. Al seleccionar la opción PX4 se puede cargar la ultima versión estable oprimiendo el botón **ok**, sin embargo, mediante la opción *Advanced settings* se puede buscar un archivo de extensión *.px4*, para ser subido, lo cual sería útil en casos donde se ha compilado el *firmware* a partir de los archivos fuentes, para por ejemplo generar una versión personalizada del *firmware* . En la sección *releases*[3] del repositorio existen disponibles algunas distribuciones pre-compiladas en las diferentes versiones, en caso de que no se quiera hacer la compilación de forma manual.

E.2. Hardware in the loop

Con el fin de habilitar el autopiloto Pixhawk 2 para operar en el modo HITL se deben realizar varios procesos. En primer lugar se deben seguir las instrucciones de la sección *PX4 Configuration* de la guía de desarrollo[4]. En cuanto al paso 5, la guía no provee información para configurar la UDP, sin embargo a continuación se resumen los pasos para realizarla.

- Abrir QGroundControl
- Acceder a la sección de configuración de aplicaciones (*application settings*) haciendo click en el icono con el logo del programa.
- Acceder a la sección *comm links*
- Oprimir el botón *Add* para añadir un nuevo enlace de comunicación.
- Configurar el enlace con las opciones de preferencia.

[3] *Releases Firmware PX4*: https://github.com/PX4/Firmware/releases
[4] Guía de desarrollo operación HITL: https://dev.px4.io/master/en/simulation/hitl.html

Para el proyecto se estableció un enlace UDP cuyo *Listening Port* era el 14550 y el *target host* era el 14560.

El siguiente paso es configurar las interfaces mavlink del autopiloto a través de sus puertos de telemetría (TELEM1 y TELEM2 para Pixhawk2), lo cual se hace en la sección *parameters* de la configuración del vehículo. En la versión mas reciente del *firmware* se ofrecen mayores opciones de configuración a traves de la interfaz grafica de QGroundControl, permitiendo configurar el BaudRate y demas parametros de una manera sencilla e intuitiva, sin embargo, como en el proyecto se uso la versión 1.7.3, esta ofrecia limitaciones en las opciones de configuración. Para resolver este problema fue necesario intervenir en la secuencia de inicilización del modulo, cada vez que es energizado.

Dentro del sistema de archivos del autopiloto existen unos archivos utilizados para configurar la funcionalidad del autopiloto, con base en el tipo de vehículo (*AirFrame*), modo de operación (Simulación o tiempo real). Particularmente el archivo RCs o RC, cuenta con una serie de instrucciones en las que se configuran y lanzan las instancias de comunicación *MAVLink*. El primer paso para configurar el autopiloto es inspeccionar dicho archivo e identificar los comandos usados para configurar los enlaces de comunicación *MAVLink*, para ello se puede descargar una copia del archivo usando el *widget Onboard Files* de QGroundControl, también se pueden usar comandos de Linux a través de la consola Mavlink disponible en QGroundControl, para explorar el sistema de archivos y su contenido. Para el *firmware* utilizado en el proyecto se encontro que la interfaz *MAVLink* principal (TELEM1) se configuraba con la variable MAVLINK_F, mientras que la interfaz para computador de compañía se configuraba con la variable SYS_COMPANION.

PX4 ofrece la posibilidad de intervenir en el proceso de inicialización del modulo por medio de archivos de configuración en la memoria SD del dispositivo. Lo primero que se debe hacer es crear un directorio en la SD y nombrarlo etc, luego se puede crear el archivo config.txt. si el sistema de inicio del dispositivo encuentra este archivo tendra en cuenta los parametros y configuraciones escritos en el, ignorando los valores por defecto que se tienen en la secuencia normal. Si bien se puede crear una secuencia de inicio personalizada en la que se tenga un ayor control de los perifericos y procesos a ejecutar para el desarrollo del proyecto basto con agregar las siguientes lineas en el archivo config.

```
set MAVLINK_F "-b 921600 -r 20000 -f"
set SYS_COMPANION 1921600
mavlink start -d /dev/ttyS2 -b 921600 -r 20000 -f
```

La primera linea configura los parámetros de la interfaz *MAVLink* principal, que luego es iniciada pro el sistema de arranque, la bandera -b configura el *BaudRate*,

-r configura la máxima tasa de transmisión , y -f indica que se habilita el envio de mensajes a otras instancias de *MAVLink*; las demás opciones para iniciar una instancia pueden verse en la consola mavlink de QGroundControl. La segunda linea indica que el modo de operación del enlace SYS_COMPANION sera como computador de compañia operando a un BAUDRATE de 921600 y finalmente, la ultima linea lanza la instancia de *MAVLink* del computador de compañía sobre el puerto serial ttys2 (TELEM2) con un BAUD_RATE de 921600 y las misma opciones que el puerto principal. Al configurar las instancias *MAVLink* de esta manera se garantiza que la comunicación en el modo HITL con el *plugin mavlink_interface* no presentara problemas de sincronia asociadas a la velocidad de transmisión.