# SISTEMA DE NAVEGACIÓN AUTÓNOMA PARA ROBOTS MÓVILES TIPO ORUGA EN CULTIVOS DE TOMATE

**TESIS PRESENTADA PARA OPTAR AL TÍTULO DE MAGÍSTER EN INGENIERÍA MECÁNICA**


**FRANCISCO ARTURO ALVARADO ÁLVAREZ**

**UNIVERSIDAD INDUSTRIAL DE SANTANDER**

**ESCUELA DE INGENIERIA MECÁNICA**

**GRUPO DE INVESTIGACIÓN DICBoT**

**BUCARAMANGA**


**2025**

# SISTEMA DE NAVEGACIÓN AUTÓNOMA PARA ROBOTS MÓVILES TIPO ORUGA EN CULTIVOS DE TOMATE

**TESIS PRESENTADA PARA OPTAR AL TÍTULO DE MAGÍSTER EN INGENIERÍA MECÁNICA**


## AUTOR:

## FRANCISCO ARTURO ALVARADO ALVAREZ

## DIRECTOR:

## CARLOS BORRÁS PINILLA. PhD., MSc

## CODIRECTOR:

## HELIO SNEYDER ESTEBAN VILLEGAS. MSc

**UNIVERSIDAD INDUSTRIAL DE SANTANDER**

**FACULTAD DE INGENIERÍAS FÍSICO-MECÁNICAS**

**MAESTRÍA EN INGENIERÍA MECÁNICA**

**BUCARAMANGA**


**2025**

# RESUMEN

**Título:** Sistema de navegación autónoma para robots móviles tipo oruga en cultivos de tomate

**Autor:** Francisco Arturo Alvarado Álvarez

**Palabras Clave:** Navegación autónoma, Robótica móvil, LOAM, Algoritmo A* Híbrido, Planificación de trayectorias, Robot tipo oruga, Unreal Engine, Simulink.

**Descripción:**

La agricultura colombiana enfrenta pérdidas significativas por ineficiencia y baja tecnificación, especialmente en cultivos hortícolas como el tomate. Ante este contexto, se diseñó e implementó un sistema de navegación autónoma para un robot móvil tipo oruga orientado a operar en un invernadero de tomate simulado. El objetivo fue garantizar desplazamientos seguros y eficientes mediante técnicas de localización y mapeo simultáneos (SLAM) —con el algoritmo LOAM ejecutado en modo offline— y la planificación óptima de rutas basada en un A* híbrido posteriormente optimizado con TEB, cumpliendo restricciones cinemáticas y distancias de seguridad.

La metodología integró un entorno virtual de alta fidelidad construido en Unreal Engine 5 y acoplado a Simulink para la comunicación y el control; se modeló el robot, su sensórica (LiDAR 3D e INS/GNSS) y el invernadero. La validación se realizó en cuatro escenarios (normal/irregular, con/sin obstáculos) y cuatro sets de trayectorias por escenario, evaluando exactitud de seguimiento y eficiencia de ruta.

Los resultados muestran mapas precisos (error medio ~3%) y un seguimiento de trayectoria con errores promedio inferiores al 0,3% usando Pure Pursuit. El A* híbrido puro presentó colisiones en escenarios con obstáculos, mientras que la versión optimizada con TEB añadió márgenes de seguridad y completó exitosamente todas las pruebas, reduciendo hasta un 53,6% el error promedio en terrenos irregulares. Se concluye que la arquitectura propuesta es robusta para entornos estáticos previamente mapeados y sienta bases para futuras extensiones online y en campo real.

# ABSTRACT

**Title:** Autonomous Navigation System for Tracked Mobile Robots in Tomato Crops

**Author:** Francisco Arturo Alvarado Álvarez

**Keywords:** Autonomous navigation, Mobile robotics, LOAM, Hybrid A* Algorithm, Path planning, Tracked robot, Unreal Engine, Simulink.

**Description:**Colombian agriculture suffers substantial losses due to inefficiency and limited technological adoption, particularly in greenhouse tomato production. This work presents the design and implementation of an autonomous navigation system for a tracked mobile robot operating in a simulated tomato greenhouse. The system combines offline SLAM using the LOAM algorithm for mapping and localization with an optimal path planner based on a Hybrid A* algorithm refined through Timed Elastic Band (TEB) optimization to meet kinematic constraints and safety margins.

A high-fidelity virtual environment was built in Unreal Engine 5 and interfaced with Simulink to model robot dynamics, sensing (3D LiDAR and INS/GNSS), and the greenhouse. Validation covered four scenarios (normal/irregular terrain, with/without obstacles) and four trajectory sets per scenario, assessing tracking accuracy and route efficiency.

Results indicate accurate maps (≈3% average pose error) and trajectory tracking errors below 0.3% using a Pure Pursuit controller. The pure Hybrid A* planner produced collisions in obstacle-rich scenarios, whereas the TEB-optimized version successfully completed all tests, adding safety buffers and reducing average tracking error by up to 53.6% on irregular terrain. The proposed architecture proves robust for static, pre-mapped environments and provides a foundation for future online implementations and field deployment.

**TABLA DE CONTENIDO**

---

## LISTA DE FIGURAS

---

## LISTA DE TABLAS



---

# INTRODUCCIÓN

En la actualidad, la agricultura se enfrenta a diversos desafíos, como la creciente demanda de alimentos, la escasez de mano de obra y la necesidad de reducir el impacto ambiental de las prácticas agrícolas (FAO, 2017). La adopción de tecnologías de precisión y la automatización de tareas en el campo puede aumentar la eficiencia y la sostenibilidad de la producción agropecuaria (Zhang y otros, 2002). Entre las tecnologías emergentes en el ámbito de la agricultura de precisión, los robots móviles autónomos han demostrado su potencial para realizar diversas tareas como monitoreo de cultivos, aplicación de fertilizantes y pesticidas, y cosecha del cultivo, entre otros. (Blackmore y otros, 2005).

El cultivo de tomates es uno de los más importantes a nivel mundial, tanto por su demanda como por su valor económico (FAOSTAT., 2021). Sin embargo, enfrenta desafíos específicos, como la necesidad de manejo intensivo, la susceptibilidad a enfermedades y plagas, y la variabilidad en las condiciones del suelo y del clima (Mulla, 2013). En este contexto, los robots móviles autónomos tipo oruga podrían contribuir significativamente al manejo de los cultivos de tomate, proporcionando soluciones precisas y adaptadas a las necesidades específicas de este cultivo.

La navegación autónoma es una capacidad esencial para los robots móviles que operan en entornos agrícolas. Su diseño y desarrollo implican desafíos importantes, como la detección de obstáculos, la planificación de trayectorias y el control de la locomoción (Bac y otros, 2014). A lo largo de los últimos años, se han desarrollado diversos enfoques y algoritmos de navegación autónoma para robots móviles en entornos agrícolas, utilizando técnicas como la percepción basada en sensores (Li y otros, 2010), la navegación basada en mapas (Noguchi y otros, 2016) y el aprendizaje por refuerzo (Chen y otros, 2018).

Sin embargo, aún existen oportunidades para mejorar y adaptar los sistemas de navegación autónoma a las características específicas de los cultivos de tomate y a los robots móviles tipo oruga. En este contexto, la presente investigación tiene como objetivo principal diseñar, desarrollar e implementar un sistema de navegación autónoma para robots móviles tipo oruga en cultivos de tomate de invernadero simulado, que permita mejorar la eficiencia y la sostenibilidad de la producción de este cultivo en un futuro desarrollo. Este robot será puesto a prueba en un entorno de simulación realista creado en Unreal Engine, un motor de Videojuegos 3D de código abierto creado por la empresa Epic Games.

# 1. PLANTEAMIENTO DEL PROBLEMA.

Colombia es un país con un gran potencial agrícola debido a su diversidad climática y geográfica. Sin embargo, a pesar de este potencial, cerca de 10 millones de toneladas de alimentos se desperdician anualmente en un país donde cerca de 2.7 millones de colombianos sufren de hambre crónica, lo que equivale al 4.8% de la población (Asociación Nacional de Industriales, 2019).

En Colombia, la ineficiencia en el sector agrícola se ha convertido en un problema que agrava la situación de hambruna en el país. Según el Departamento Nacional de Planeación (DNP), la falta de infraestructuras, tecnología, manejo fitosanitario, habilidades y conocimientos en las cadenas de producción de alimentos ha llevado a una pérdida del 34% de los productos alimentarios disponibles para el consumo humano, lo que equivale a una cantidad de 9.76 millones de toneladas por año. (Departamento Nacional de Planeación, 2020).

En particular, la etapa de producción agropecuaria es la más afectada, donde se pierden 3.95 millones de toneladas de alimentos, lo que representa el 40.5% del total de los residuos alimentarios generados. Esto sugiere que la falta de tecnificación del campo es uno de los principales factores que contribuyen a este problema. Además, el hecho de que el 62% de estos desperdicios correspondan a frutas y verduras sugiere una falta de conocimiento y habilidades en la gestión de estos cultivos. En consecuencia, el mismo Departamento Nacional de Planeación concluye que la falta de inversión en ciencia, tecnología e innovación en la cadena productiva, junto a la falta de implementación de nuevas tecnologías en los procesos productivos son una de las principales causas de la generación de desperdicios en la producción agrícola. (Departamento Nacional de Planeación, 2020).

En la actualidad, se ha iniciado la implementación de nuevas técnicas en la producción agrícola con el objetivo de abordar la problemática de ineficiencia en el sector. Entre ellas, destaca la producción de tomate bajo invernadero o condiciones protegidas, la cual se presenta como una alternativa para la reconversión de cultivos en áreas con limitaciones y condiciones adversas. (Departamento Administrativo Nacional de Estadística, 2014)

A pesar de los avances en la tecnificación de la agricultura, gran parte de los cultivos de tomate en Colombia aún se manejan de forma manual y semi-mecanizada, lo que implica costos elevados de mano de obra, menor eficiencia y dificultades para mantener la calidad y homogeneidad del producto.

La implementación de sistemas de navegación autónoma en robots móviles tipo oruga en cultivos de tomate puede representar una solución potencial para abordar esta problemática. Los robots autónomos pueden mejorar la eficiencia en la producción agrícola, reducir el desperdicio de alimentos y optimizar el uso de los recursos, efectuando tareas como el monitoreo del cultivo. Sin embargo, el desarrollo de sistemas de navegación autónoma para robots móviles en cultivos de invernadero es un desafío complejo y multifacético al encontrarse en un terreno irregular, donde no solo debe enfrentarse a las condiciones adversas del terreno sino también debe sortear con los diversos obstáculos cambiantes que conlleva el manejo del invernadero.

Las soluciones de navegación existentes han demostrado limitaciones. Por ejemplo, la navegación por carril guía, en la que se colocan rieles en el suelo para guiar a los robots, proporciona una solución sencilla, pero limita drásticamente la trayectoria y el alcance del robot

(Chiu et al., 2013; Hayashi et al., 2014; Lee et al., 2015). Estos sistemas encuentran limitados por el uso de rieles que restringen su trayectoria a una ruta predefinida. Esta configuración puede resultar problemática en situaciones donde el robot se enfrenta a obstáculos inesperados, ya que no puede desviarse de manera flexible y se ve obligado a retroceder para buscar una ruta alternativa. Además, los sistemas basados en rieles dificultan la colaboración entre el robot y el agricultor, ya que pueden interferir con el tránsito humano en el campo de cultivo, reduciendo la eficiencia del trabajo conjunto.

Aunque los sistemas de locomoción mediante ruedas son ampliamente utilizados en la mayoría de los casos de navegación autónoma, su aplicabilidad en los cultivos de tomate colombianos presenta ciertas limitaciones y desafíos específicos. Estos cultivos suelen crecer en terrenos irregulares, donde la tracción de las ruedas no es adecuada, lo que puede ocasionar atascamientos o deslizamientos en condiciones de suelo fangoso, resultando en una maniobrabilidad significativamente restringida. Además, el uso de ruedas en el campo agrícola puede dar lugar a una mayor compactación del suelo debido al peso del robot, lo que a su vez puede afectar negativamente la salud del suelo y el crecimiento óptimo de las plantas.

La navegación con visión artificial, que utiliza sensores de visión para recopilar información ambiental, también presenta desafíos. Aunque los métodos como la transformada de Hough, los métodos de mínimos cuadrados y los algoritmos de visión estéreo binocular pueden ser efectivos, estos sistemas requieren ajustes frecuentes basados en cambios en el entorno, como la iluminación, y pueden limitar la movilidad del robot (Hough, 1962; Cui et al., 2015; Mao et al., 2019; Zou et al., 2012).

Las tecnologías basadas en LiDAR 2D, como el mapeo simultáneo y la localización (SLAM), han obtenido resultados prometedores en la navegación de robots móviles en invernaderos (Juan et al., 2016; Obregón et al., 2019; Hou et al., 2020; Tiwari et al., 2020). Sin embargo, estos sistemas solo pueden detectar información en el mismo plano horizontal que la posición de instalación del LiDAR en el robot. Esto impone requisitos estrictos para la instalación de LiDAR y el entorno del invernadero, y no puede detectar información ambiental por encima o por debajo de sí mismo, dejando un gran peligro potencial para la seguridad en la navegación de robots. Por lo tanto, existe una clara necesidad de investigación para el desarrollo de un sistema de navegación más robusto y versátil para robots móviles.

Uno de los principales desafíos en la implementación de la navegación autónoma en cultivos es el diseño de un sistema que pueda garantizar una cobertura eficiente y uniforme del campo, una generación de trayectoria óptima que evite colisiones y obstáculos. Estos requisitos implican la necesidad de utilizar tecnologías y algoritmos avanzados que permitan al vehículo autónomo reconocer y adaptarse a la naturaleza del entorno agrícola.

Sin embargo, la investigación y desarrollo de sistemas de navegación autónoma adecuados para robots móviles en la agricultura aún es un desafío, ya que el entorno de conducción es más complejo, diverso y sus modos de navegación son significativamente diferentes a los que se implementan actualmente en los vehículos sin conductor (Binbin y otros, 2021).

- **Hipótesis de investigación:** En un entorno semiestructurado de un invernadero de tomate, la ejecución de un sistema de navegación autónoma para un robot móvil tipo oruga puede alcanzar un desplazamiento eficaz y autónomo, mediante la implementación integrada del SLAM y el Algoritmo A* Híbrido. Asegurando un mapeo, un seguimiento y

una planeación de trayectoria optima con base a un escaneo manual del entorno previamente ejecutado. Dicho enfoque potencia un sistema de navegación offline permitiéndole al robot moverse por un entorno estático previamente conocido.

## 2. JUSTIFICACIÓN.

El tomate (Solanum lycopersicum) se ha consolidado como la hortaliza más importante a nivel nacional e internacional, debido a su alta demanda, su extensa área de cultivo y su valor económico en la producción agrícola. Además de su relevancia económica y social, el tomate es cada vez más valorado por su importancia nutricional. En la actualidad, se ha reconocido que esta hortaliza es una fuente excepcional de sustancias antioxidantes, como el licopeno y el betacaroteno, así como de vitaminas, incluyendo la vitamina C y la vitamina A (Escobar & Lee, 2009).

Según la Organización de las Naciones Unidas para la Alimentación y la Agricultura (también conocida como FAO, por siglas en ingles), el tomate es una de las hortalizas de mayor crecimiento en su consumo y producción, aumentando un 29% en comparación a los ultimo 10 años. En el ámbito nacional, la producción anual de tomate en Colombia asciende a 632,268 toneladas, siendo Antioquia el departamento de mayor producción con una participación del 24.7%. Le siguen en importancia Norte de Santander con un 13.6%, Boyacá con un 11.5%, Cundinamarca con un 11.2% y Santander con un 10.4% (Octavio, 2018), demostrando que la producción de tomate es un motor de desarrollo para la economía rural en el país.

Ahora, la técnica de siembra de tomate en invernadero ha demostrado excelentes resultados en términos de productividad, rentabilidad y calidad del fruto, con la ventaja adicional de minimizar la aplicación de insecticidas. La producción bajo invernadero permite un mayor control sobre las condiciones ambientales, lo que se traduce en una mejor adaptación de la planta y una reducción del impacto de las enfermedades y plagas (Departamento Administrativo Nacional de Estadística, 2014). Álvaro Palacios, gerente general de ASOHOFRUCOL, ha señalado que el creciente uso de tecnologías innovadoras en el cultivo de tomate bajo invernadero, complementado con un manejo integral de las condiciones del cultivo, ha sido clave en el aumento de la demanda de esta hortaliza (Octavio, 2018).

La investigación llevada a cabo por Pedersen et al. (2008) evaluó la posibilidad de utilizar aplicaciones robóticas en la agricultura y reveló una importante disminución en los costos de producción. En consecuencia, la incorporación de sistemas de navegación autónoma para robots móviles en el ámbito agrícola constituye una solución altamente eficaz para potenciar tanto la eficiencia como la calidad de los procesos de producción y cosecha, la cual no solo es pertinente en el contexto de cultivos de tomate bajo invernadero, sino que abarca un alcance más amplio. Es importante destacar que la elección del tomate como cultivo de enfoque se fundamenta en su excepcional valor nutricional y en su significativa contribución a la economía agrícola a nivel local.

La incorporación de robots en la agricultura contribuye a mejorar la sostenibilidad y la consistencia en las tareas agrícolas, al mismo tiempo que disminuye los desechos y los costos al permitir, por ejemplo, la aplicación dirigida de fertilizantes y pesticidas, reduciendo la cantidad de químicos utilizados y minimizando el impacto ambiental (Hajjaj & Sahari, 2014), siendo esta una de las principales buenas prácticas agrícolas recomendadas para el tomate de invernadero (Escobar & Lee, 2009). Además, la implementación de robots en la agricultura también puede contribuir a la realización de tareas más laboriosas como el monitoreo de la salubridad y la madurez del cultivo, así como en su posterior recolección. Estos son ejemplos concretos de cómo se pueden apreciar beneficios significativos en sectores tradicionalmente poco automatizados.

La implementación de tecnologías avanzadas, como los robots móviles tipo oruga, presenta una prometedora oportunidad para mejorar la eficiencia y sostenibilidad de la producción de tomate en Colombia. Esta iniciativa adquiere especial relevancia al contribuir a la reducción significativa de los desperdicios alimentarios generados en la etapa de producción agropecuaria, al tiempo que fortalece la competitividad en el sector agrícola. Los robots móviles tipo oruga, debido a su versatilidad y capacidad para operar en terrenos difíciles, se erigen como una solución idónea para optimizar el manejo y monitoreo de los cultivos.

Un atributo distintivo de los robots móviles tipo oruga es su capacidad para distribuir el peso de manera uniforme, lo que resulta en una tracción y estabilidad mejoradas en terrenos irregulares y resbaladizos. Esta característica cobra especial importancia, ya que minimiza el riesgo de daños a las raíces y la estructura del suelo, preservando la salud y productividad del sustrato agrícola. En comparación con otras alternativas de locomoción, como los robots con ruedas, los robots tipo oruga se destacan por su habilidad para adaptarse a obstáculos, tales como pequeñas rocas o irregularidades del terreno. Igualmente, son una opción más económica y menos complejas como lo pueden ser los sistemas de locomoción por piernas. A su vez se ha visto un aumento exponencial en la bibliografía referente a “Agriculture Robot” desde el 2018, como se puede evidenciar en la Figura 1.

En la Figura 2 se puede evidenciar en el clúster bibliográfico de “Agriculture Robotics” que la investigación reciente en este tema se ha diversificado en diferentes ramas, entre las que se destacan la navegación autónoma, el radar óptico, el mapeo, la planeación de movimiento y los robots móviles. Cada una de estas ramificaciones desempeña un papel crucial en la mejora y eficiencia de las operaciones agrícolas, lo que ha generado un gran interés en la comunidad científica y tecnológica, y ha llevado a una amplia investigación en estos campos.

La navegación autónoma es una característica clave de estos robots, ya que permite realizar tareas agrícolas de manera independiente y eficiente. El uso de un sistema SLAM es fundamental para la navegación autónoma ya que puede construir y actualizar constantemente un mapa preciso del entorno mientras simultáneamente estima su propia posición en dicho mapa. Esto permite al sistema navegar de manera segura y eficiente a través de terrenos desconocidos.

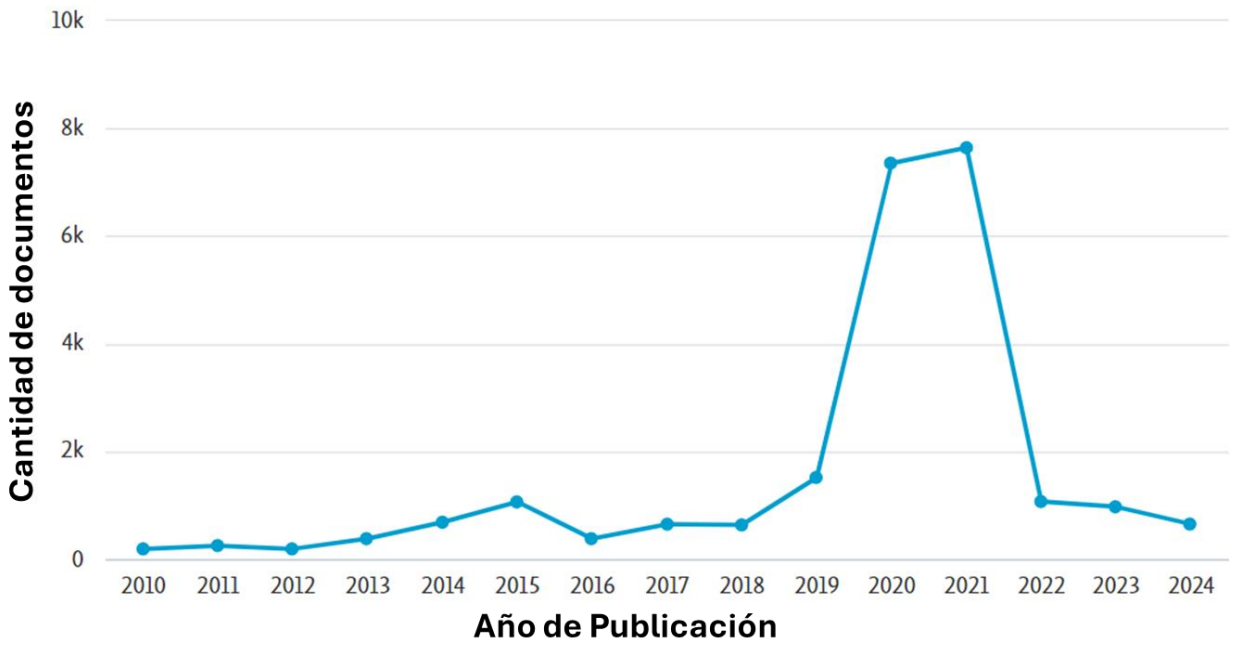



**Figura 1.** Gráfico de Documentos por año para las palabras clave “Agriculture” y “Robot”. **Fuente:** Scopus®

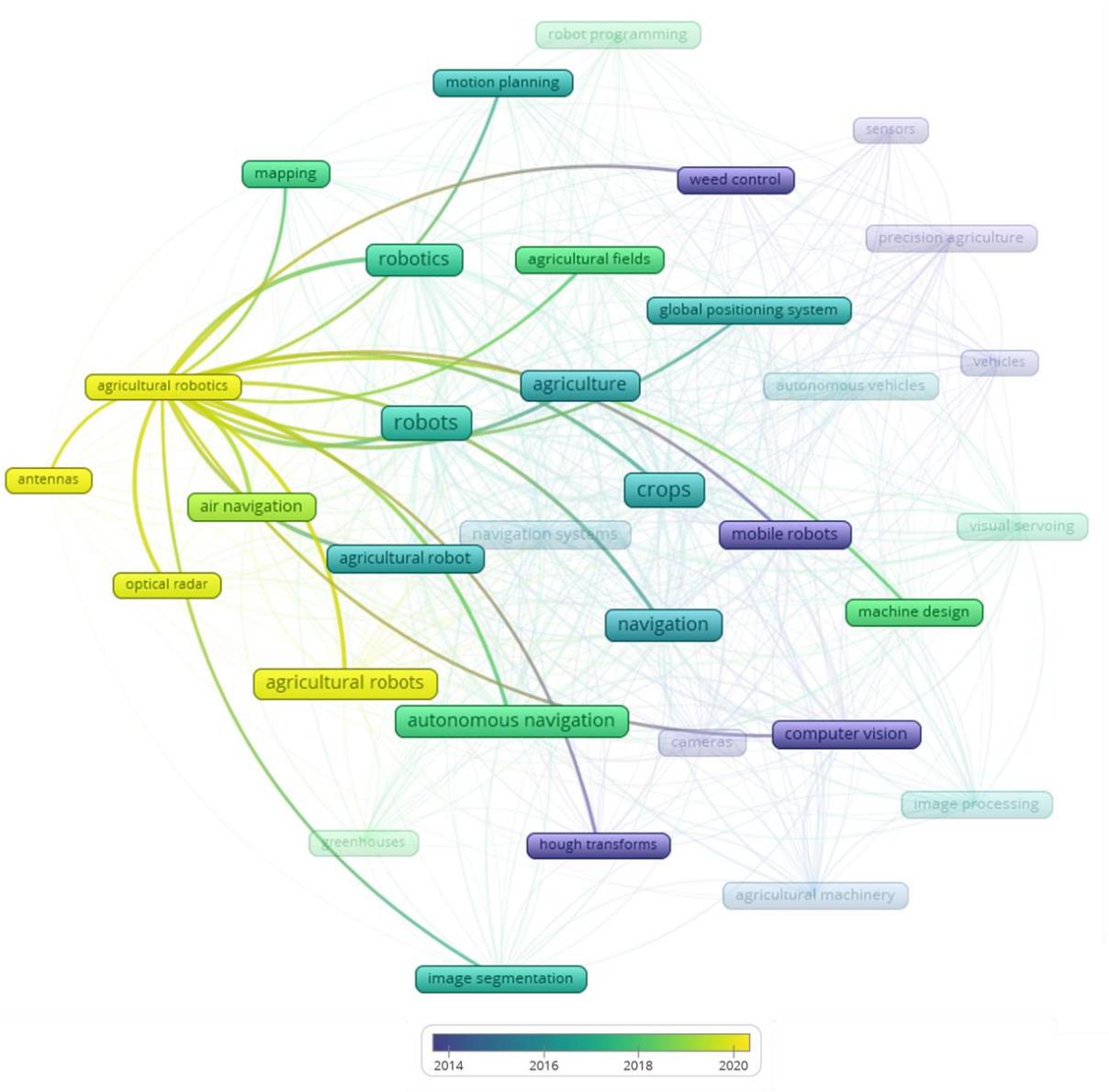



**Figura 2.** Clúster bibliográfico para “Agricultural Robotics”. **Fuente:** Scopus®

# 3. OBJETIVOS.

## 3.1. Objetivo General.

Diseñar un sistema de navegación en un Robot Móvil tipo oruga, que permita el desplazamiento autónomo en un cultivo de tomate de invernadero, por medio de técnicas de reconocimiento de ubicación y mapeo del entorno con el fin de plantear una posible trayectoria en un ambiente simulado.

## 3.2. Objetivos Específicos.

- Simular el comportamiento de un robot móvil tipo oruga en un entorno virtual, utilizando un prototipo simplificado desarrollado en Unreal Engine 5, como método de validación del desempeño de la navegación autónoma bajo condiciones controladas.
- Implementar un sistema de localización a través del mapeo (SLAM) fuera de línea (offline) de un cultivo de tomate a transitar con el propósito de conocer la ubicación del robot móvil en un ambiente simulado, utilizando únicamente los datos proporcionados por un sensor LiDAR.
- Implementar un sistema de generación de trayectorias utilizando el algoritmo A* Híbrido, que se base en el mapa adquirido por el sistema SLAM para generar una ruta óptima, considerando las restricciones cinemáticas del vehículo, garantizando una curvatura suave y manteniendo una distancia segura respecto a los obstáculos.
- Validar el modelo de navegación autónoma propuesto mediante la simulación de un cultivo en un entorno de desarrollo virtual, con el objetivo de evaluar el desempeño del robot móvil tipo oruga en un invernadero de tomate, para esto se evaluará el error de seguimiento en la ruta planteada y su oportuna respuesta ante obstáculos ya conocidos.

# 4. ANTECEDENTES

En la última década, se ha dedicado un esfuerzo significativo a la investigación en el campo de la automatización de los procesos agrícolas, con el objetivo de identificar áreas específicas en las que se puedan aplicar la capacidad que tiene la robótica móvil para aumentar la eficiencia en las tareas agrícolas, reducir la dependencia de mano de obra y minimizar el impacto ambiental. Entre las áreas de aplicación que se han identificado se incluyen la preparación del terreno, la siembra, el monitoreo de la salud del cultivo, la cosecha de los frutos, la estimación del rendimiento, entre otras (Oliveira y otros, 2021).

En el ámbito de la fertilización, control de plagas y manejo del suelo, el robot Cäsar se destaca como un ejemplo comercial que ayuda a los trabajadores rurales en estas tareas, así como en la cosecha y transporte (The Raussendorf Company, s.f.). Agbot es otro robot en fase de investigación que se centra en la aplicación de fertilizantes y herbicidas en granjas de maíz (Khan y otros, 2018). La reducción del tamaño, peso y complejidad mecánica de robots agrícolas también se ha abordado mediante el diseño de Di-Wheel, un robot que se apoya y se desplaza únicamente bajo dos ruedas, utilizado en la siembra de precisión, fumigación y deshierbe (Sukkarieh, 2017).

Además, la investigación en el campo de la agricultura continúa explorando nuevas áreas de aplicación para la robótica móvil, incluyendo la identificación de plagas en cultivos de algodón y maní utilizando tecnología como el AGROBOT (Pilli y otros, 2015), la implementación de cosecha autónoma de frutos basada en diferentes algoritmos de inteligencia artificial, como los utilizados en los robots móviles Harvey (Lehnert y otros, 2020) y SWEEPER (Arad y otros, 2020), la adquisición de datos y estimación del rendimiento en cultivos de naranja y caña de azúcar mediante robots como el AgriBot (Ceres y otros, 1998), entre otros ejemplos de aplicaciones que se están desarrollando actualmente en este campo.

La navegación autónoma es una tecnología crucial para el desarrollo de robots móviles en la agricultura, ya que difiere significativamente de otros sistemas de navegación autónoma debido a la complejidad y diversidad del entorno agrícola (Oliveira y otros, 2021). Mohd Basri y Adnan (2022) implementaron un algoritmo de seguimiento de línea con características adicionales para la creación de un prototipo de robot móvil autónomo implementado al monitoreo y captura de imagen de un cultivo. En cambio, Jasinski et al. (2016) ideo la concepción de un robot agrícola autónomo para la siembra y la plantación en hileras de cultivos orgánicos utilizando un sistema de navegación inercial. Santhi et al. (2017) presento un prototipo de robot móvil capaz de navegar autónomamente en cualquier terreno agrícola con la finalidad de sembrar semillas a su paso, gracias a su sistema de navegación autónomo que combina la autoconciencia de la posición del robot por medio de GPS y el uso de un sistema de visión a bordo del robot.

En cuanto al uso de sistemas de navegación autónoma en invernaderos, Tiwari et al. (2020) presento un sistema de navegación autónoma para la detección y el tratamiento de plagas en invernaderos comerciales usando 3 funciones básicas de navegación: localización absoluta, localización relativa y planificación de ruta. Sin embargo, el sistema más avanzado de navegación en invernadero lo presentó Jiang (2022) planteando un sistema de control de navegación del robot que filtra inicialmente los datos del entorno recopilado por un LiDAR 3D y lo fusiona con la información recopilada por un LiDAR 2D, luego basándose en los odómetros del robot y la información de la unidad de medición inercial, el sistema logra posicionar y construir

el entorno del invernadero utilizando un algoritmo de localización y navegación simultanea (SLAM, por siglas en inglés) 2D.

Localmente, en la Universidad Industrial de Santander ha desarrollado varios proyectos de grado relacionados con la navegación autónoma. La escuela de Ingeniería Eléctrica, Electrónica y Telecomunicaciones ha desarrollado una tesis de pregrado sobre algoritmos genéticos aplicados al planeamiento de trayectorias de un robot móvil (Navas Gomez & Ortiz Ortega, 2006) y otra tesis, esta vez a nivel de posgrado, sobre el planeamiento de trayectorias de un robot móvil (Ibaduiza Burgos & Barrero Perez, 2006). Por su parte, la escuela de ingeniería mecánica ha desarrollado una tesis de pregrado orientada al diseño y construcción de un prototipo de robot móvil para exploración (Pineda Quijano & Prada Largo, 2009) y otra orientada al diseño y construcción de un sistema de navegación autónoma para un vehículo aplicado a un campo de maíz (Niño Rodriguez & Salazar Triana, 2018).

# 5. MARCO TEÓRICO.

## 5.1. Introducción a la robótica móvil y la navegación autónoma.

Los robots móviles son plataformas electromecánicas dotadas de un sistema de locomoción que le brinda la capacidad de navegar a lo largo de un determinado espacio de trabajo, de esta forma se le dota al robot de un nivel considerable de autonomía a la hora realizar diversos trabajos. La robótica móvil parte de la necesidad de extender el campo de aplicación de la robótica incrementando la autonomía al aplicar la capacidad de trasladarse a lo largo de su entorno, de esta forma aumentar el rango del espacio de trabajo y aumentando el número de tareas que el robot puede realizar (OLLERO, 2001).

### 5.1.1. Clasificación de los robots móviles según sus restricciones de movimiento.

Los robots móviles pueden clasificarse según sus grados de libertad y sus controles independientes, específicamente en relación con las restricciones de movimiento que tienen. Según estos criterios, los robots móviles pueden clasificarse en dos categorías principales: Robots móviles holonómicos y no holonómicos.

#### 5.1.1.1. Robots móviles holonómicos.

Un robot móvil holonómico es aquel que tiene la misma cantidad de grados de libertad que de controles independientes. Esto significa que puede moverse instantáneamente en cualquier dirección en el espacio en el que opera sin necesidad de girar previamente. Los robots holonómicos son capaces de realizar movimientos en todas las direcciones del plano sin restricciones, lo que les permite una mayor maniobrabilidad en entornos complejos. Estos robots no tienen restricciones en su velocidad o dirección de movimiento que dependan de su orientación o posición actual (Siciliano y otros, 2009).

#### 5.1.1.2. Robots móviles no holonómicos.

Un robot móvil no holonómico es un sistema mecánico cuyos grados de libertad son mayores que el número de controles independientes. Esto significa que existen restricciones en su movimiento que no pueden ser expresadas como una relación fija entre las posiciones de las partes móviles del sistema (restricciones de posición), sino que dependen también de la dirección y/o velocidad de movimiento (restricciones de velocidad). Estas restricciones limitan la capacidad del robot para moverse en ciertas direcciones en un momento dado.

Las restricciones no holonómicas suelen surgir de la interacción entre las ruedas y el suelo. Por ejemplo, una rueda que rueda sin deslizarse tiene una restricción no holonómica: en cualquier momento, su velocidad de traslación debe ser perpendicular a su eje de rotación (Siciliano y otros, 2009).

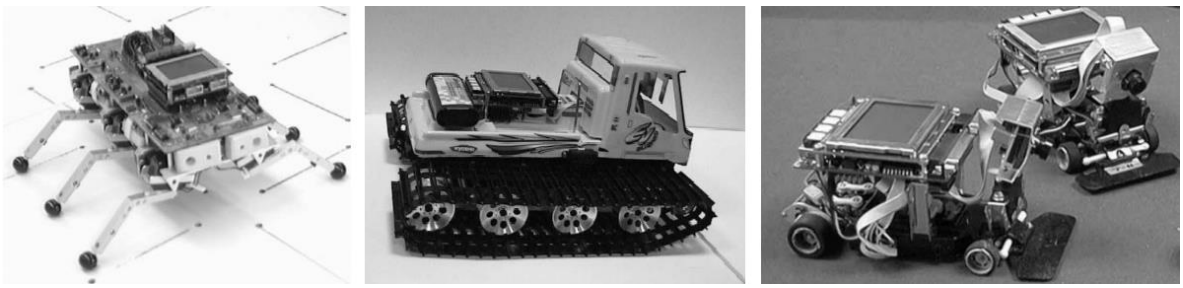
### 5.1.2. Clasificación de los robots móviles según su entorno de trabajo.

Los robots móviles también se pueden clasificar según su entorno de trabajo en tres categorías principales:

- **Terrestres:** Capaz de navegar en diversos terrenos terrestres.
- **Acuáticos:** Capaz de navegar en ambientes acuáticos, ya sea flotando o sumergiéndose.
- **Aéreos:** Capaz de navegar en el aire o el espacio.

#### 5.1.2.1. Clasificación de los robots móviles terrestres.

Los robots móviles terrestres pueden ser clasificados en tres grupos de acuerdo a su sistema de locomoción: con piernas, orugas y ruedas, como se puede observar en la siguiente figura:



(a) (b) (c)

**Figura 3.** Clasificación de los robots móviles terrestres según su sistema de locomoción. (a) Sistema de locomoción de piernas, en este caso un hexápodo. (b) Sistema de locomoción por orugas. (c) Sistema de locomoción por ruedas, en este caso un sistema simple de 4 llantas. **Fuente:** (Siegwart & Nourbakhsh, 2004)

**Robots móviles con piernas.**

Los robots con sistema de ***locomoción de piernas*** se caracterizan por tener una serie de puntos de contacto entre el robot y el suelo, permitiéndole una mayor maniobrabilidad ante un terreno irregular debido a que cuentan con características antropomorfas que le dan la capacidad de adaptarse a su entorno de trabajo, pero son sistemas que cuentan con muchos grados de libertad, volviéndolos difícil de controlar y requieren una gran potencia para su funcionamiento (ver Figura 3a). En este grupo se encuentran los bípedos, cuadrúpedos u hexápodos (Siegwart & Nourbakhsh, 2004). Este sistema de locomoción puede ser considerado ***holonómico*** porque pueden moverse en cualquier dirección sin girar.

**Robots móviles con orugas.**

De igual forma, los ***robots móviles con orugas*** cuentan con parches de contacto con el suelo mucho más grandes, lo cual puede mejorar su maniobrabilidad en terrenos irregulares (ver Figura 3b). Sin embargo, no dispone de un modelo preciso de giro, debido a que el parche de contacto genera una mayor fricción con la superficie, requiriendo gran potencia por parte del actuador (Siegwart & Nourbakhsh, 2004). Aunque pueden girar en su lugar y moverse hacia adelante y hacia atrás, los robots con orugas se consideran ***no holonómicos*** porque no pueden moverse lateralmente.

**Robots móviles con ruedas.**

Por último, está la ***locomoción por ruedas***, la cual presenta una mayor eficiencia energética, especialmente en superficies planas y compactas, dado a que cuenta con una superficie de contacto menor y un menor grado de complejidad, en comparación (ver Figura 3c). Su principal desventaja es la perdida de estabilidad en terrenos irregulares, llegando incluso a volcarse. A este grupo se puede subclasificar según la disposición de sus ruedas, presentando configuraciones tales como diferencial, síncrono, triciclo, monociclo, Ackerman, omnidireccional, entre otras (Braunl, 2006). Dependiendo de su configuración los robots móviles con ruedas

pueden ser holonómicos o no holonómicos. ***Holonómicos***, si cuentan con *ruedas omnidireccionales* que le permitan moverse a cualquier dirección sin girar. ***No holonómicos*** si su configuración de ruedas es diferencial, Ackerman o cualquier otra que *no sea omnidireccional*.

### 5.1.3. Navegación autónoma de robots móviles.

La navegación autónoma se define como la metodología que permite guiar el curso de un robot móvil a través de un entorno con obstáculos, con el objetivo de llevar autónomamente el vehículo a su destino de forma segura y eficiente (Muñoz Martinez, 1997). Las tareas involucradas en la navegación de un robot móvil son:

- **La percepción del entorno:** La capacidad de percepción del robot es la síntesis de toda la información provisionada por los sensores, con el objetivo de determinar una relación con su entorno (Massino, 2008).
- **La planificación:** en el caso más general, puede descomponerse en planificación global de una misión, una ruta, una trayectoria y la evasión de ósculos inesperados. (OLLERO, 2001).
- **El Razonamiento y el control:** capacidad de decidir qué acciones son requeridas en cada movimiento, basándose en el estado del robot y el de su entorno, con el fin de alcanzar su objetivo. (Massino, 2008)

El diagrama de bloques de un sistema de navegación autónoma para robots móviles puede observarse en la siguiente figura:

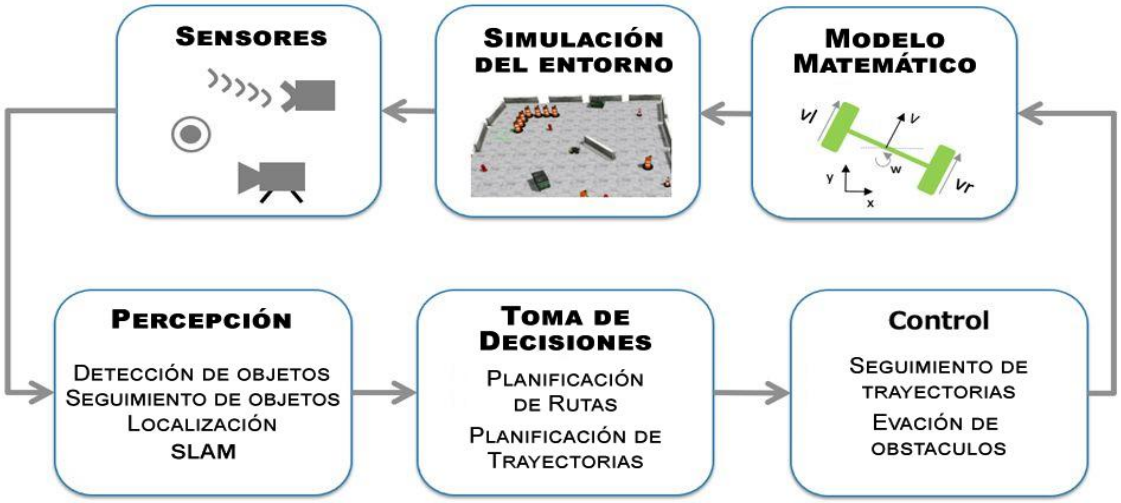



**Figura 4.** Diagrama de bloques de un sistema de navegación autónoma. **Fuente:** adaptado de **(MathWorks, 2022)**

Los sistemas de robots móviles autónomos están diseñados para llevar a cabo tareas que implican un alto nivel de complejidad cognitiva, lo que requiere la implementación de múltiples algoritmos. Por ejemplo, para realizar tareas como la detección y evasión de obstáculos, se necesita una combinación de algoritmos multidisciplinarios, el procesamiento de nubes de puntos obtenidas por un sensor LiDAR, la fusión de datos obtenidos por diferentes sensores para llevar a cabo el seguimiento de obstáculos, la generación de mapas mediante el uso de un sistema de localización y mapeo simultáneos (SLAM, por siglas en ingles), así como la planificación y el control de la trayectoria del robot (MathWorks, 2022).

### 5.2. Sistema de Localización y Mapeo Simultáneos (SLAM):

La técnica de localización y mapeo simultáneos (SLAM) es ampliamente utilizada en la industria de los vehículos y robots móviles autónomos, permitiendo la construcción de mapas precisos del entorno y la localización del vehículo en tiempo real. Los algoritmos de SLAM permiten el mapeo de entornos desconocidos, proporcionando información valiosa para la planificación de rutas y la evasión de obstáculos.

En el ámbito de la tecnología SLAM, se identifican dos tipos de componentes fundamentales, tal y como se puede observar en la Figura 5. El primero se refiere al procesamiento de señales de los sensores (Front end), donde el procesamiento frontal varía según los tipos de sensores empleados. El segundo tipo se relaciona con la optimización de los gráficos de posición (Back end), donde el procesamiento final es independiente de los sensores (Cadena y otros, 2016). Ambos tipos de componentes son esenciales en la construcción de un mapa y la localización simultánea de un vehículo autónomo en entornos desconocidos, como se puede observar en el siguiente gráfico:

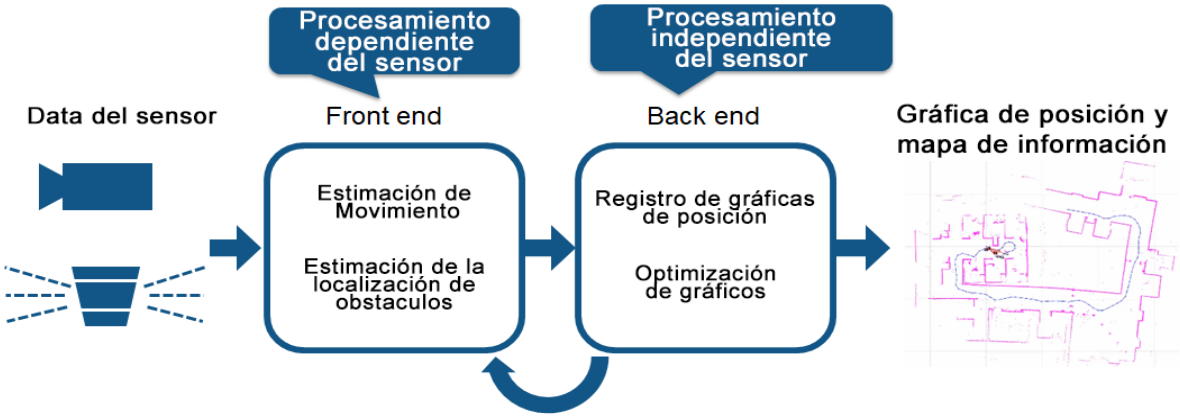



**Figura 5.** Diagrama de bloques de un sistema SLAM. **Fuente:** Adaptado de **(MathWorks, s.f.)**

#### 5.2.1. Métodos principales de SLAM.

##### 5.2.1.1. SLAM Visual.

La técnica de navegación y mapeo simultáneo mediante visión artificial, conocida como SLAM visual o vSLAM, se basa en el uso de cámaras y otros sensores de imagen para capturar información del entorno y así construir un mapa del mismo mientras se localiza el vehículo en él. Esta técnica puede implementarse de manera económica utilizando cámaras simples de gran angular, ojo de pez o esféricas, cámaras estereoscópicas o multicámaras, y cámaras RGB-D de profundidad. Debido a que las cámaras proporcionan una gran cantidad de información, se pueden utilizar para detectar puntos de referencia, que a su vez pueden ser combinados con la optimización basada en gráficos, lo que proporciona mayor flexibilidad en la implementación de SLAM. Un ejemplo vSLAM puede observarse en la Figura 6.

En términos generales, los algoritmos de SLAM visual se pueden dividir en dos categorías principales:

- **Métodos dispersos:** son aquellos que identifican correspondencias entre los puntos de características de las imágenes, utilizando algoritmos como PTAM y ORB-SLAM.

- **Métodos densos:** utilizan el brillo total de las imágenes para la creación de mapas, haciendo uso de algoritmos como DTAM, LSD-SLAM, DSO y SVO (Dailey & Parnichkun, 2006).

Ambos enfoques tienen sus ventajas y desventajas, y su elección dependerá del objetivo específico y de las condiciones del entorno en el que se aplicarán.

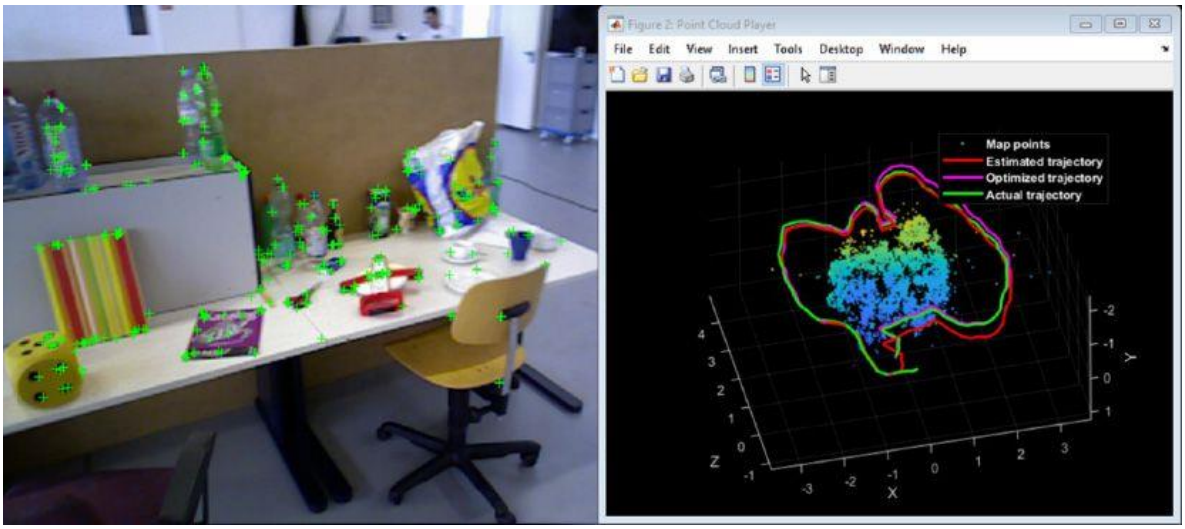



**Figura 6.** Visual SLAM (ORB-SLAM) con cámara monocular. **Fuente: (MathWorks, 2022)**

#### 5.2.1.2. SLAM de LiDAR.

El LiDAR (Light Detection and Ranging) es un método que utiliza principalmente sensores láser para medir la distancia y detectar objetos en un entorno. A diferencia de otros sensores, como cámaras o sensores TOF, los láseres son mucho más precisos y se utilizan comúnmente en aplicaciones de alta velocidad como la conducción autónoma o en drones. Los valores de salida de los sensores LiDAR suelen ser datos de nubes de puntos en 2D (x, y) o 3D (x, y, z), que proporcionan mediciones de distancia altamente precisas y son efectivas para generar mapas con SLAM.

El movimiento se estima de manera secuencial identificando correspondencias entre las nubes de puntos, en donde el movimiento calculado (distancia recorrida) se utiliza para localizar el vehículo. Para la identificación de correspondencias de nubes de puntos de LiDAR, se utilizan algoritmos de registro tales como los algoritmos del punto iterativo más cercano (ICP) y de la transformada de distribuciones normales (NDT). Estos mapas se pueden representar como un mapa de cuadrícula o un mapa de vóxeles. Las representaciones de mapas gestionadas por LiDAR SLAM se pueden observar en la Figura 7, donde la primera fila muestra mapas de cuadrícula de ocupación en 2D y la segunda fila muestra nubes de puntos en 3D. Cabe resaltar que los mapas de nubes de puntos son resultados sin procesar del sensor LiDAR.

Sin embargo, las nubes de puntos no son tan detalladas como las imágenes en cuanto a densidad, lo que puede dificultar la identificación de correspondencias. Además, la identificación de coincidencias de nubes de puntos suele requerir una alta carga de procesamiento, lo que puede afectar la velocidad. Debido a estos desafíos, la localización de vehículos autónomos puede requerir la combinación de otros datos de medición, como la odometría de las ruedas, los

datos del sistema global de navegación por satélite (GNSS) y los datos de IMU (Zhang, Lai, Xu, Li, & Fu, 2020; Droeschel & Behnke, 2018).

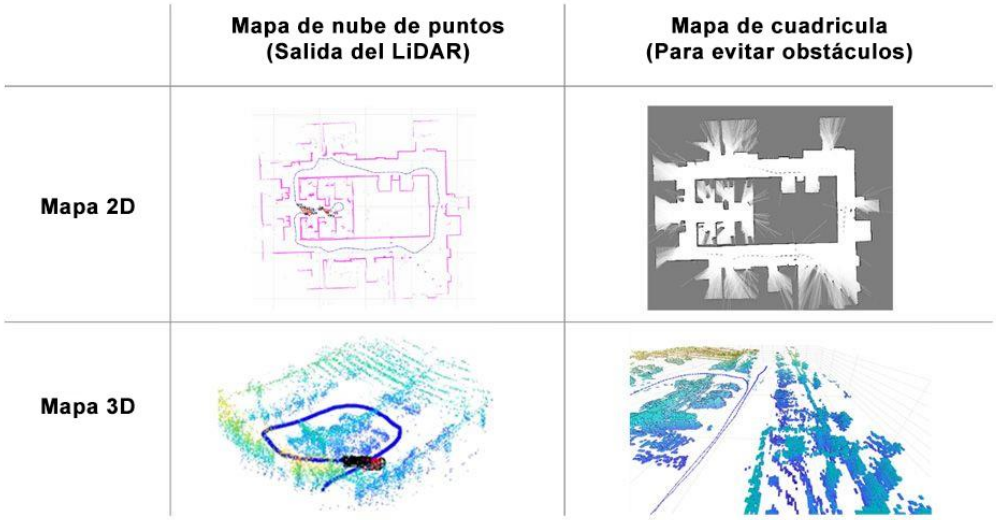



**Figura 7.** Comparativa entre la generación de SLAM con Mapas 2D y 3D utilizando LiDAR. **Fuente:** Adaptado de **(MathWorks, 2022)**.

### 5.2.2. Desafíos comunes del SLAM.

#### 5.2.2.1. Errores de localización acumulada.

La estimación del movimiento secuencial en SLAM es susceptible a errores que se acumulan con el tiempo, lo que puede provocar desviaciones significativas respecto de los valores reales. Además, estos errores pueden afectar la validez y la precisión de los datos del mapa, lo que dificulta su uso en futuras búsquedas y aplicaciones. A medida que la tasa de error aumenta, la posición final del robot ya no coincide con su posición inicial, lo que se conoce como **problema de cierre de bucle** y aunque este error acumulado resulta inevitable, es importante detectarlos y corregirlos (o en el peor de los casos, neutralizarlos).

Una **contramedida** para este problema consiste en registrar características de lugares previamente visitados como puntos de referencia para reducir el error de localización (MathWorks, s.f.). Para esto se utilizan los **gráficos de posición**, los cuales son estructuras que contienen nodos conectados por aristas y se generan para corregir los errores y mejorar la precisión del mapa de datos. Cada estimación de nodo se conecta al gráfico mediante restricciones de borde que establecen la posición relativa entre ellos y la incertidumbre en esa medición. En la construcción de los gráficos de pose, es posible incluir objetos de referencia conocidos como marcadores de realidad aumentada o tableros de ajedrez. Asimismo, la salida de sensores como las unidades de medición inercial (IMU) pueden fusionarse en los gráficos para medir la posición de forma directa (Cadena y otros, 2016). Ejemplo de generación de un gráfico de posición y minimización de errores puede observase en la siguiente figura:


**Figura 8.** Ejemplo de generación de un gráfico de posición y minimización de errores. **Fuente: (MathWorks, s.f.)**

#### 5.2.2.2. Localización fallida y perdida de la posición en el mapa

Las técnicas de mapeo de imágenes y nubes de puntos no tienen en cuenta las características del movimiento de un robot, lo que puede generar estimaciones de posición discontinuas o errores de localización. Para evitar estos problemas, se pueden emplear algoritmos de recuperación o fusionar el modelo de movimiento con múltiples sensores para realizar cálculos precisos con base a los datos obtenidos por los sensores.

Entre los métodos más utilizados para la **fusión de sensores**, se encuentra el filtrado de Kalman extendido y los filtros de partículas, que se emplea en la localización de robots con modelos de movimiento no lineales, como los vehículos de tracción diferencial y los de cuatro ruedas. Los sensores más utilizados para este fin son los dispositivos de medición inercial, tales como IMU, Sistemas de Referencia de Actitud y Rumbo (AHRS, por sus siglas en inglés) o sistemas de navegación inercial (INS, por sus siglas en inglés), así como los sensores de aceleración, giroscopios y magnéticos. Además, se pueden añadir codificadores de ruedas en el vehículo para realizar odometría.

Cuando falla la localización, se puede utilizar una **contramedida** para recuperarla, como *recordar un punto de referencia* de un lugar visitado previamente como marco clave y aplicar un proceso de extracción de características para realizar barridos a gran velocidad. Algunos de los métodos más utilizados para la extracción de características de imagen son la bolsa de características (BoF, por sus siglas en inglés) y la bolsa de palabras visuales (BoVW, por sus siglas en inglés). También se utiliza Deep Learning para comparar las distancias respecto de las características (Cadena y otros, 2016).

#### 5.2.2.3. Alto costo computacional.

La implementación de SLAM en hardware de robots móviles presenta desafíos en términos de capacidad de procesamiento limitada. El procesamiento de imágenes y la identificación de correspondencias de nubes de puntos deben ejecutarse a alta frecuencia para lograr una localización precisa. Los cálculos de optimización, como el cierre de bucle, son especialmente costosos. Para **abordar este problema**, se puede recurrir a la paralelización de procesos como la extracción de características, y el uso de tecnologías como SIMD (“instrucción única, datos múltiples”, por sus siglas en ingles), CPU multinúcleo y GPU integradas. Además, la reducción de la prioridad y la realización periódica de la optimización de gráficos de posición también pueden mejorar el rendimiento.

### 5.3. LiDAR Odometry and Mapping (LOAM).

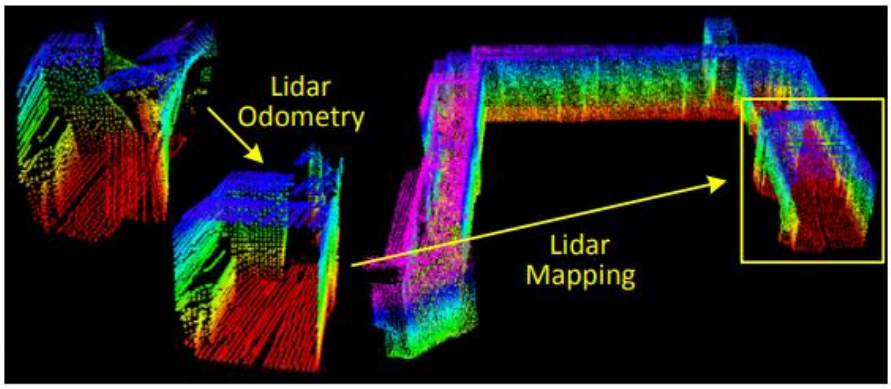



**Figura 9.** LiDAR Odometry and Mapping (Loam). **Fuente: (Zhang & Singh, 2014)**

LOAM, siglas en inglés para "Odometría LiDAR y Mapeo Simultáneo", es un enfoque para SLAM (Simultaneous Localization and Mapping) que ha ganado reconocimiento por su efectividad en entornos complejos. Este método utiliza únicamente un sensor LiDAR 3D, un dispositivo que emite pulsos de luz láser y mide el tiempo que tardan en reflejarse, para generar un mapa 3D detallado de un entorno y ubicar un robot dentro de ese mapa simultáneamente, tal y como puede observarse en la Figura 9. LOAM fue propuesto por Zhang y Singh en 2014 y ha sido un componente crucial para la navegación de vehículos autónomos, robots móviles y drones.

Este algoritmo se divide principalmente en cuatro procesos se ejecutan en paralelo: *Point Cloud Registration, LiDAR Odometry, LiDAR Mapping y Transform Integration,* tal y como puede observarse en la Figura 10.

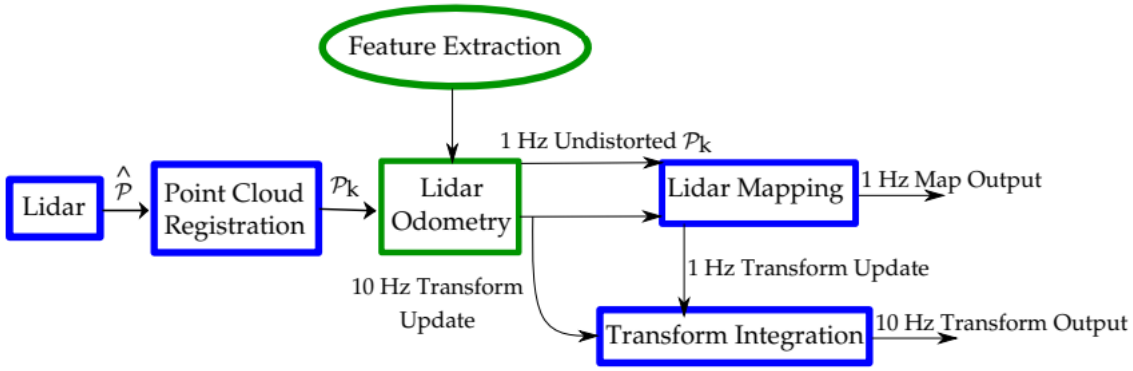



**Figura 10.** Arquitectura en Software del Algoritmo LOAM. **Fuente: (Gonzalez & Adams, 2019)**

La arquitectura del algoritmo LOAM funciona básicamente de la siguiente forma:

1. **Point Cloud Registration:** este bloque se encarga de registrar la nube de puntos $\hat{P}$ obtenida por el sensor LiDAR y procesarla, asociándole una propiedad de barrido llamada $k$. Esto da como resultado una nube de puntos $P_k$.
2. **Odometría LiDAR (LiDAR Odometry)**: Esta parte del algoritmo se encarga de estimar el movimiento entre escaneos consecutivos del LiDAR. Lo hace al extraer características (***Feature Extraction***) de los datos del LiDAR, incluyendo puntos de borde (características lineales) y puntos planos (características planas). Luego, utiliza estas características para

calcular la transformación que mejor alinea estos puntos entre los escaneos consecutivos. Este proceso se repite en un bucle rápido para proporcionar una estimación en tiempo real del movimiento del robot.

- El bloque de odemetría LiDAR alimenta constante el bloque de mapeo LiDAR con datos a 1 Hz y el bloque de Tramsform Integration con datos a 10 Hz (Gonzalez & Adams, 2019).

3. **Mapeo LiDAR (LiDAR Mapping)**: En paralelo con la odometría, el algoritmo realiza el mapeo. Utiliza la odometría estimada para transformar los puntos de las nubes de puntos del LiDAR al marco de referencia global y luego alinea estos puntos con el mapa existente. El algoritmo utiliza un optimizador para refinar la odometría y el mapa simultáneamente para garantizar la consistencia a largo plazo.

    - El Mapa se actualiza a una frecuencia de 1 Hz

4. **Transform Integration:** Se encarga de integrar las transformaciones calculadas por la odometría y el mapeo. Actualiza las transformaciones a una frecuencia de 10 Hz, proporcionando una salida final a esa velocidad para una navegación precisa.

Este algoritmo ha sido seleccionado para el estudio en este trabajo.

### 5.3.1. Descomposición del problema del algoritmo LOAM

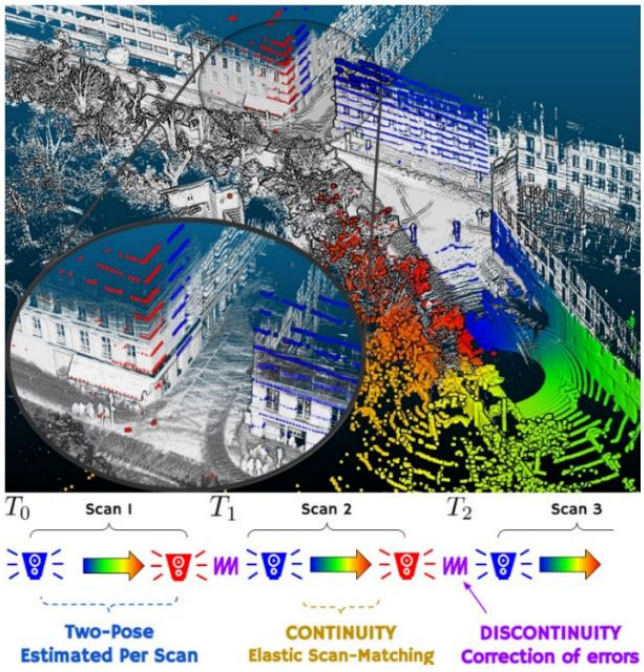



**Figura 11.** Proceso de alineamiento de nubes de puntos**. Fuente: (Biber & Strasser, 2003)**

El problema del registro de nubes de puntos en el algoritmo LOAM se aborda mediante un enfoque dual, utilizando la ejecución en paralelo de los algoritmos de ***odometría*** y ***mapeo***. El primero se encarga de estimar la velocidad del LiDAR en alta frecuencia, pero con baja precisión, mientras que el segundo realiza un ajuste fino de las nubes de puntos a una frecuencia menor. Este enfoque de doble frecuencia permite una baja deriva acumulada en la estimación del

movimiento, lo que resulta en mapas coherentes sin la necesidad de métodos como el cierre de bucles para corregir la deriva a lo largo del tiempo.

En cada barrido del sensor, LOAM extrae puntos clave, conocidos como puntos característicos, que pueden ser bordes afilados o superficies planas y luego, se realiza la búsqueda de correspondencias entre estos puntos en la nube anterior y la actual. Una vez que se identifican las correspondencias, la transformación entre las nubes se estima utilizando un método basado en mínimos cuadrados que minimiza las distancias entre los puntos clave correspondientes. Este enfoque garantiza que las nubes de puntos se alineen con precisión, lo que es crucial para obtener estimaciones exactas del movimiento y construir mapas precisos del entorno. (Zhang & Singh, 2014; Gonzalez & Adams, 2019)

En las siguientes subsecciones presentan mas a detalle el funcionamiento de los dos bloques principales: Odometría LiDAR y Mapeo LiDAR.

### 5.3.2. Odometría LiDAR

#### 5.3.2.1. Extracción de puntos característicos.

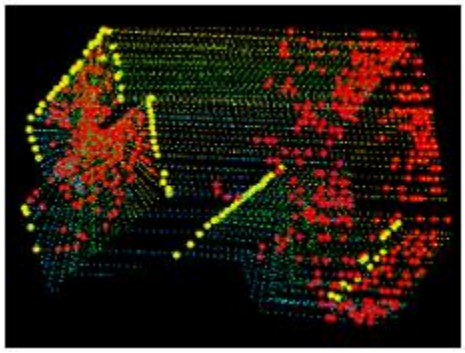


**Figura 12.** Ejemplo de puntos de bordes y puntos planos extraídos de una nube LiDAR. **Fuente: (Zhang & Singh, 2014)**

Para realizar una correcta estimación del movimiento, primero es necesario extraer ***puntos característicos*** del conjunto de datos percibidos por el LiDAR durante cada barrido. Dado que los escaneos generados por el sensor LiDAR tienen una resolución angular de 0.25° dentro del plano de escaneo, pero una resolución mucho menor en la dirección perpendicular (alrededor de 4.5°), los puntos característicos se extraen considerando solo relaciones geométricas coplanares.

Los puntos seleccionados pertenecen a dos categorías principales:

1. **Puntos de borde** (sharp edge points): son aquellos ubicados en las aristas de las superficies. Un ejemplo de ellos son los puntos amarillos en la Figura 12.
2. **Puntos de superficie plana** (planar surface points): son puntos ubicados en superficies lisas. Un ejemplo de ellos son los puntos rojos en la Figura 12.

El algoritmo utiliza un criterio de **suavidad** para seleccionar los puntos característicos. Para un punto $\boldsymbol{i}$ del conjunto $\boldsymbol{P_k}$, la suavidad se calcula como:

$$c = \frac{1}{|S| \cdot \|X^L_{(k,i)}\|} \left\| \sum_{j \in S, j \neq i} X^L_{(k,i)} - X^L_{(k,J)} \right\| \quad (1)$$

Donde:

- $S$ es el conjunto de puntos consecutivos en la misma exploración-
- $X^L_{(k,i)}$ representa las coordenadas del punto $i$ en el sistema de coordenadas del lidar.

Los puntos con los mayores valores de $c$ se seleccionan como **puntos en bordes** y aquellos con los valores más bajos se seleccionan como **puntos en superficies planas**. Este proceso asegura que los puntos seleccionados sean representativos de las características estructurales del entorno.

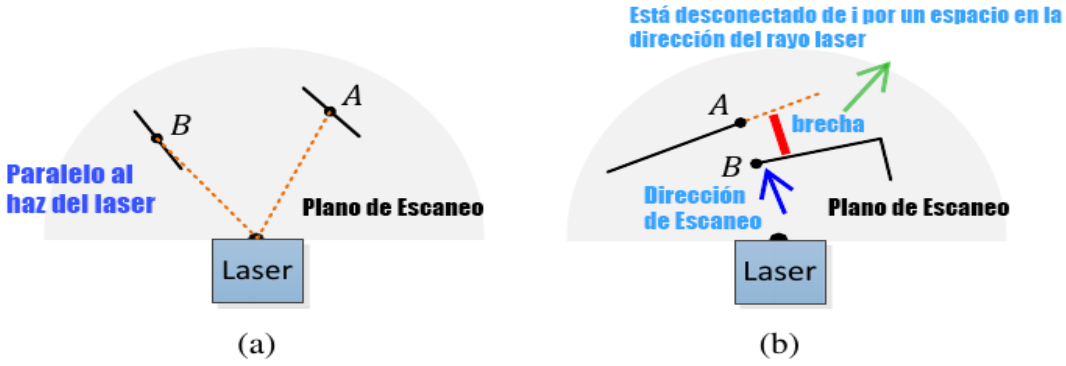



**Figura 13.** Selección de características en el algoritmo LOAM. (a) Caso 1. (b) Caso 2. **Fuente:** Adaptado de **(Zhang & Singh, 2014)**

Para mantener una distribución uniforme de los puntos, se divide el escaneo en cuatro subregiones, de las cuales se seleccionan un máximo de dos puntos de borde y cuatro puntos de superficie por subregión. Al seleccionar estos puntos se deben tener en cuenta que:

- El número de puntos de borde o planos seleccionados no puede superar el máximo de la subregión.
- Los puntos que están demasiado cerca de los ya seleccionados se omiten.
- Los puntos que se encuentran en superficies paralelas a los haces del láser o en regiones ocluidas se descartan debido a su baja fiabilidad.

El diagrama anterior representa visualmente este tercer punto. La Figura 13(a) muestra que el punto B se encuentra en una zona de superficie paralela al haz laser, por lo que es descartado. Sin embargo, la superficie del punto A forma un ángulo con el haz, por lo que el punto A se toma como un punto característico. Por su parte, en la Figura 13(b), el punto A no se toma como punto característico porque, si bien desde el punto de vista actual parece ocluido, desde una perspectiva diferente podría ser observable, lo que lo hace poco confiable.

#### 5.3.2.2. Correspondencia de puntos característicos.

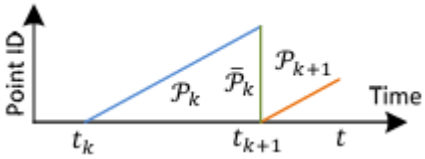



**Figura 14.** Re-proyectar una nube de puntos hasta el final de un barrido. **Fuente: (Zhang & Singh, 2014)**

Una vez seleccionados los puntos característicos, el siguiente paso en la odometría es **encontrar correspondencias** entre los puntos extraídos en un barrido actual y los puntos percibidos en barridos anteriores. Para este propósito, se re-proyectan los puntos del barrido anterior ($\boldsymbol{P_k}$) al tiempo del siguiente barrido ($\boldsymbol{P_{k+1}}$), lo que genera un nuevo conjunto de puntos proyectados $\bar{\boldsymbol{P}}_{\boldsymbol{k}}$, tal y como se puede visualizar el Figura 14.

Para cada barrido, el algoritmo identifica **líneas de borde** (*edge lines*) y **parches de superficie** (*surface patches*) que sirven como correspondencias de los puntos extraídos. Este proceso se realiza utilizando una búsqueda en un árbol KD tridimensional, lo que permite una búsqueda rápida de los puntos más cercanos. El proceso de correspondencia de puntos de borde y de puntos en superficies planas se describe resumidamente a continuación:

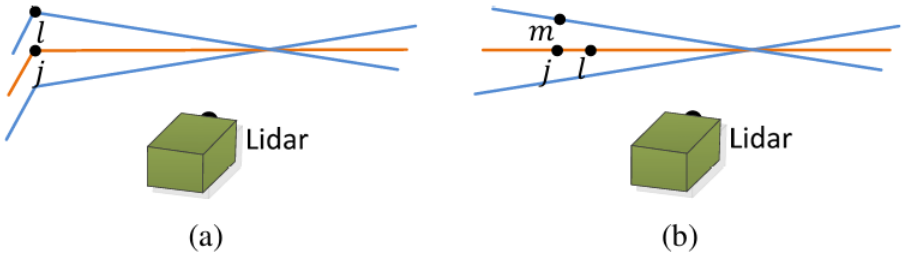



**Figura 15.** Correspondencias de características en LOAM. **Fuente: (Zhang & Singh, 2014)**

- **Puntos de borde**: Se buscan líneas de borde como correspondencias para los puntos de borde. Una **línea de borde** se define por dos puntos consecutivos en diferentes escaneos del barrido anterior, que forman una línea en el espacio tridimensional. El proceso comienza identificando el vecino más cercano $\boldsymbol{j}$ en $\boldsymbol{P_k}$ y un segundo vecino $\boldsymbol{l}$ en los barridos consecutivos al de $\boldsymbol{j}$. La correspondencia de un punto $i \in \boldsymbol{E_{k+1}}$ se representa mediante los puntos $\boldsymbol{j}$ y $\boldsymbol{l}$, cómo se puede observar en la Figura 15 (a).

  La distancia entre el punto en el borde ($\boldsymbol{i}$) y la línea formada por los puntos $\boldsymbol{j}$ y $\boldsymbol{l}$ se calcula usando la siguiente fórmula de **distancia punto-línea**:

$$d_E = \frac{\left|\left(\tilde{\boldsymbol{X}}^L_{(k+1,i)} - \bar{\boldsymbol{X}}^L_{(k,j)}\right) \times \left(\tilde{\boldsymbol{X}}^L_{(k+1,i)} - \bar{\boldsymbol{X}}^L_{(k,l)}\right)\right|}{\left|\bar{\boldsymbol{X}}^L_{(k,j)} - \bar{X}^L_{(k,l)}\right|} \qquad (2)$$

  Donde $\tilde{\boldsymbol{X}}^L_{(k+1,i)}$, $\bar{\boldsymbol{X}}^L_{(k,j)}$ y $\bar{\boldsymbol{X}}^L_{(k,l)}$ son las coordenadas de los puntos $i$, $j$ y $l$ en $\{L\}$, respectivamente. Esta fórmula mide la distancia perpendicular entre el punto $i$ y la línea formada por $j$ y $l$.

- **Puntos de superficie**: De manera similar, se identifican planos como correspondencias para los puntos de superficie plana. Un **parche de superficie** se define por tres puntos consecutivos en la nube de puntos del barrido anterior que, cuando se conectan, forman un pequeño plano en el espacio. Esto se logra localizando tres puntos cercanos $\boldsymbol{j}$, $\boldsymbol{l}$, y $\boldsymbol{m}$ en las cercanías del punto $i \in H_{k+1}$, asegurando que no sean colineales, como puede observar en la Figura 15 (b).

  La distancia entre el punto en la superficie plana ($\boldsymbol{i}$) y el parche correspondiente se calcula mediante la fórmula de **distancia punto-plano**:

$$d_E = \frac{\left|\left(\tilde{\boldsymbol{X}}^L_{(k+1,i)} - \bar{\boldsymbol{X}}^L_{(k,j)}\right)\left(\left(\bar{\boldsymbol{X}}^L_{(k,j)} - \bar{X}^L_{(k,l)}\right) \times \left(\bar{\boldsymbol{X}}^L_{(k,j)} - \bar{X}^L_{(k,m)}\right)\right)\right|}{\left|\left(\bar{\boldsymbol{X}}^L_{(k,j)} - \bar{X}^L_{(k,l)}\right) \times \left(\bar{\boldsymbol{X}}^L_{(k,j)} - \bar{X}^L_{(k,m)}\right)\right|} \tag{3}$$

  Donde $\bar{X}^L_{(k,m)}$ son las coordenadas de los puntos $m$ en $\{L\}$.

#### 5.3.2.3. Estimación del movimiento.

En la estimación del movimiento del LiDAR, se calcula la transformación de su posición y orientación entre dos barridos consecutivos utilizando un modelo que asume que el movimiento del LiDAR durante un barrido es continuo y que las velocidades lineales y angulares se mantienen constantes dentro de este período.

La transformación total del LiDAR entre dos barridos consecutivos se expresa como:

$$\boldsymbol{T}^{\boldsymbol{L}}_{\boldsymbol{(k+1,i)}} = \left[\boldsymbol{t_x}, \boldsymbol{t_y}, \boldsymbol{t_z}, \boldsymbol{\theta_x}, \boldsymbol{\theta_y}, \boldsymbol{\theta_z}\right]^T \tag{4}$$

donde $\boldsymbol{t_x}$, $\boldsymbol{t_y}$, $\boldsymbol{t_z}$ son las traslaciones y $\boldsymbol{\theta_x}$, $\boldsymbol{\theta_y}$, $\boldsymbol{\theta_z}$ son las rotaciones. Para calcular la transformación de un punto en particular durante un barrido, se interpola la pose a lo largo del tiempo con:

$$\boldsymbol{T}^{\boldsymbol{L}}_{\boldsymbol{(k+1,i)}} = \frac{t_i - t_{k+1}}{t - t_{k+1}} T^L_{k+1} \tag{5}$$

Esta fórmula ajusta la posición de cada punto en función de su momento de captura dentro del barrido.

El cálculo del movimiento del LiDAR se realiza estableciendo una relación geométrica entre los puntos re-proyectados del barrido anterior y los puntos actuales. La posición de un punto característico $i$ en el barrido actual está relacionada con su punto correspondiente re-proyectado por la transformación de rotación y traslación:

$$X^L_{(k+1,i)} = R\tilde{X}^L_{(k+1,i)} + \boldsymbol{T}^{\boldsymbol{L}}_{\boldsymbol{(k+1,i)}}(\boldsymbol{1}:\boldsymbol{3}) \tag{6}$$

Donde $\boldsymbol{R}$ es una matriz de rotación calculada usando la fórmula de Rodríguez, que ajusta la orientación del LiDAR.

Finalmente, el algoritmo ajusta la transformación $\boldsymbol{T^L_{(k+1,i)}}$ para minimizar la distancia entre los puntos característicos y sus correspondencias usando el método de **Levenberg-Marquardt**. La función objetivo a minimizar es la suma de las distancias de los puntos a las líneas y planos correspondientes:

$$T^L_{(k+1)} \leftarrow T^L_{(k+1)} - \left(J^T J + \lambda diag(J^T J)\right)^{-1} J^T d \tag{7}$$

Donde $\boldsymbol{J}$ es la matriz Jacobiana y $\boldsymbol{d}$ contiene las distancias calculadas. Este proceso iterativo asegura una estimación precisa de la pose del LiDAR al finalizar cada barrido.

#### 5.3.2.4. Algoritmo de Odometría LiDAR.

En resumen, el algoritmo de odometría LiDAR se puede dividir en dos fases principales:

- **Inicio de un nuevo barrido**: Si se detecta que comienza un nuevo barrido, el algoritmo inicializa la transformación de la pose $T^L_{k+1}$ a cero. A medida que se reciben puntos durante el barrido, el algoritmo extrae los **puntos en bordes** y **puntos en superficies planas**, asignándolos a los conjuntos $E_{k+1}$ y $H_{k+1}$, respectivamente.
- **Optimización iterativa**: Para cada punto característico en $E_{k+1}$ y $H_{k+1}$, se encuentran las correspondencias geométricas con los puntos del barrido anterior, ya sean **líneas** o **superficies**. Posteriormente, se calculan las distancias entre los puntos característicos y sus correspondencias y se realiza una **optimización no lineal** mediante el método de Levenberg-Marquardt, ajustando $T^L_{k+1}$. El proceso continúa hasta que la optimización converge o se alcanza el número máximo de iteraciones.

Finalmente, al terminar un barrido, los puntos percibidos se re-proyectan al tiempo del próximo barrido para garantizar la consistencia del modelo de movimiento. La transformación final $T^L_{k+1}$ se devuelve como salida, lista para el siguiente ciclo de estimación (Zhang & Singh, 2014). En la Tabla 1 se ilustra una descripción general del esquema de un algoritmo A* Híbrido estándar:

**Tabla 1.** Algoritmo de Odometría LiDAR. **Fuente:** Adaptado de: (Zhang & Singh, 2014)

**Algoritmo 1: Odometría LiDAR**

**Entrada:** $\bar{P}_k$, $P_{k+1}$, $T^L_{k+1}$de la última recursión
**Salida :** $\bar{P}_{k+1}$, nuevo $T^L_{k+1}$ calculado
**Comienzo**
  | **Si** al comienzo de un barrido **entonces**
  | |  $T^L_{k+1} \leftarrow 0$;
  | **Fin**
  | Detectar puntos de borde y puntos planos en $P_{k+1}$, poner los puntos en
  | $E_{k+1}$ y $H_{k+1}$, respectivamente;
  | **Para** un número de iteraciones **entonces**
  |  **Para** cada punto de borde en $_{Ek+1}$ **entonces**
  |  | Encontrar una línea de borde como la correspondencia, luego
  |  | calcular la distancia de punto a línea basada en
  |  | $f_E\left(X^L_{(k+1,i)}, T^L_{k+1}\right) = d_E, i \in E_{k+1}$ y apilar la ecuación en $f(T^L_{k+1}) = d$;
  |  **Fin**
  |  **Para** cada punto plano en $_{Hk+1}$ **entonces**
  |  | Encontrar un parche plano como la correspondencia, luego
  |  | calcular la distancia de punto a plano basada en
  |  | $f_E\left(X^L_{(k+1,i)}, T^L_{k+1}\right) = d_E, i \in E_{k+1}$ y apilar la ecuación en $f(T^L_{k+1}) = d$;
  |  **Fin**
  |  Calcular un peso bisquare para cada fila de $f(T^L_{k+1}) = d$;

```
17:  |     Actualizar T_{k+1}^L para una iteración no lineal basada en (7);
18:  |     Si la optimización no lineal converge entonces
19:  |     |   Salir;
20:  |     Fin
21:  | Fin
22:  | Si al final de un barrido entonces
23:  | |    Re-proyectar cada punto en P_{k+1} a t_{k+2} y formar P̄_{k+1};
24:  | |    Retornar T_{k+1}^L y P̄_{k+1};
25:  | Fin
26:  | Sino
27:  | |    Retornar T_{k+1}^L;
28:  | Fin
29: Fin
```

### 5.3.3. Mapeo con LiDAR.

Mientras que la odometría se encarga de estimar el movimiento del sensor en tiempo real, el mapeo refina y registra las nubes de puntos del LiDAR para generar un mapa preciso del entorno. El objetivo principal de esta sección es integrar los datos de los barridos para construir un mapa global del entorno explorado por el sensor.

#### 5.3.3.1. Procesamiento de las nubes de puntos.

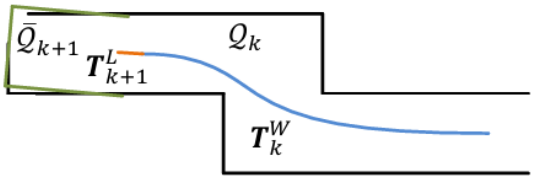



**Figura 16.** Ilustración del proceso de mapeo. **Fuente: (Zhang & Singh, 2014)**

Al final de cada barrido, el algoritmo de odometría genera una nube de puntos corregida por la distorsión de movimiento ($\bar{P}_{k+1}$) y la transformación de pose estimada ($T_{k+1}^L$) .El algoritmo de mapeo toma esta nube de puntos y la proyecta en un sistema de coordenadas global (denotado como $\{W\}$), donde se construye un mapa acumulativo del entorno explorado por el lidar.

Supongamos que, hasta el barrido $k$, el mapa está representado por la nube de puntos $Q_k$, y la pose del LiDAR en ese mapa es $T_k^W$. El objetivo del algoritmo de mapeo es actualizar $T_k^W$ y generar $T_{k+1}^W$, la nueva pose del LiDAR, integrando la nube de puntos $\bar{P}_{k+1}$ en el mapa existente $Q_k$. Este proceso se realiza mediante la **correspondencia de características** y la **optimización no lineal**.

#### 5.3.3.2. Extracción y Correspondencia de Puntos Característicos.

El proceso de extracción de características es similar al realizado en la odometría: se extraen puntos en bordes y superficies planas. Sin embargo, en el mapeo, se utilizan *diez veces más puntos característicos* para mejorar la precisión del registro de las nubes de puntos.

Para encontrar las correspondencias, se divide el espacio del mapa en cubos de 10 metros. Estos cubos permiten realizar una búsqueda rápida en el entorno inmediato de los puntos característicos. Para cada punto característico extraído de $\bar{P}_{k+1}$, el algoritmo busca los puntos

en el mapa $\boldsymbol{Q_k}$ que estén cercanos, y realiza un análisis de los vectores propios de la matriz de covarianza $M$ de los puntos circundantes:

- Si el análisis indica que los puntos están alineados a lo largo de una línea, se establece una correspondencia de borde.
- Si los puntos están distribuidos en un plano, se establece una correspondencia de superficie plana.

Una vez que se encuentran las correspondencias, el algoritmo calcula la **distancia** entre los puntos actuales y las estructuras (líneas o planos) del mapa acumulado. Utilizando la misma metodología que en la odometría, se minimizan estas distancias mediante optimización no lineal (Levenberg-Marquardt) para ajustar la nueva nube de puntos en el mapa existente.

El algoritmo también utiliza un **filtro de rejilla de vóxel** (5 cm de tamaño de vóxel) para reducir la densidad de puntos en el mapa, asegurando una representación eficiente sin sacrificar precisión.

#### 5.3.3.3. Integración de las transformaciones de pose

Finalmente, la pose del LiDAR con respecto al mapa global se actualiza combinando la transformación de pose generada por la **odometría** (que se ejecuta a alta frecuencia) y la transformación producida por el **mapeo** (a baja frecuencia). Este proceso asegura que la pose del LiDAR esté siempre actualizada y optimizada para los fines de navegación y creación de mapas en tiempo real.

La integración de las transformaciones se realiza combinando la pose global del mapeo $T_k^W$ y la pose estimada por la odometría $T_{k+1}^L$, generando una actualización constante de la posición del LiDAR en el mapa global (Zhang & Singh, 2014).

### 5.4. Planificación de movimiento.

La planificación de movimiento representa una problemática computacional que se centra en descubrir una secuencia de acciones encaminada a desplazar un robot o vehículo autónomo de una condición de inicio a una meta predeterminada. Aunque los conceptos de "planificación de movimiento" y "planificación de ruta" se utilizan con frecuencia de manera equivalente, es fundamental entender que no son sinónimos. En concreto, la planificación de movimiento se ocupa de producir el desplazamiento del vehículo en correspondencia con su cambio de ubicación a lo largo del tiempo, a diferencia de la planificación de ruta, cuyo enfoque se restringe a la creación de una trayectoria para el vehículo.

En la práctica, para garantizar el funcionamiento eficaz de la planificación de movimiento de un robot autónomo se requiere una serie de elementos esenciales:

1. **Mapa del Entorno:** generado mediante el uso de un algoritmo de localización y mapeo simultáneos (SLAM).
2. **Estado:** comprende variables tales como la posición y la orientación.
3. **Transformación de Estado:** es la transición del robot de un estado a otro, y es lo que define el movimiento del mismo.
4. **Espacio de Estado:** también conocido como "**espacio de configuración ($C_{space}$)**", es el conjunto de todas las posibles transformaciones que se pueden aplicar al robot.
    - **Espacios Libres:** son aquellas regiones dentro del espacio de configuración donde los estados del robot son considerados válidos.
    - **Obstáculos:** son aquellas regiones dentro del espacio de configuración donde los estados del robot son considerados inválidos.

Por ejemplo, para el problema de planificar el movimiento de un automóvil autónomo en un estacionamiento automatizado, la posición y la orientación del automóvil representan colectivamente su estado. Este automóvil debe contar con un mapa del entorno en el que se identifiquen los espacios libres y los posibles obstáculos. Mientras que su espacio de estados representa el conjunto de todas las maniobras de avance y retroceso posibles (MathWorks, 2021; Xiao, Liu, Warnell, & Stone, 2020).

#### 5.4.1. Clasificación de los algoritmos de planificación de movimiento.

Los algoritmos de planificación de movimiento se pueden clasificar comúnmente según tres enfoques específicos no mutuamente excluyentes:

- **Enfoque basado en búsqueda o muestreo:** en donde se toma en cuenta la forma en cómo se construye la trayectoria, ya sea en forma de árbol o con un gráfico de búsqueda.
- **Enfoque de rutas globales y locales:** que se basan en si la planificación se realiza en todo el mapa o en un subconjunto. Ambos enfoques pueden observarse en la Figura 17.
    - **Planificación de ruta global:** también conocido como planificación basada en mapas, son algoritmos en donde se implica hallar una óptima ruta basada en un pleno conocimiento del entorno creando una ruta inicial que evita los obstáculos conocidos en un ambiente estático.
    - **Planificación de ruta local:** también conocido como replanificación dinámica, son algoritmos que rastrean el plan global y crean trayectorias locales mientras

evitan obstáculos recién incluidos. Son muy usados para la evasión de obstáculos en ambientes dinámicos de constante cambio, como en la navegación autónoma de un automóvil, en donde se utilizan para planificar una trayectoria que permita el cambio de carril para rebasar otro vehículo sin desviarse de la ruta global.

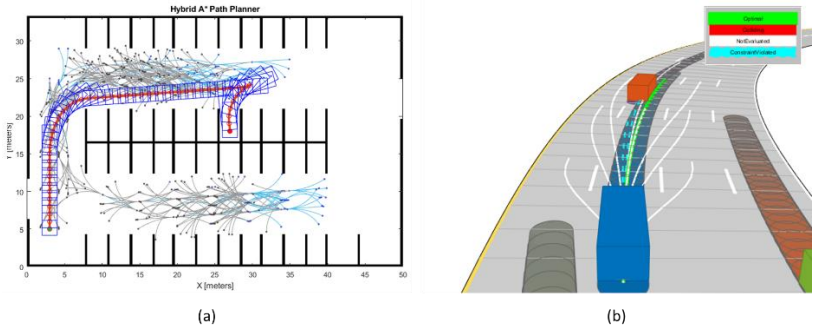



**Figura 17.** Enfoque de rutas globales y locales. (a) Planificación global. (b) Planificación local. **Fuente:** Adaptado de (MathWorks, 2021).

### 5.4.2. Planificación basada en búsquedas.

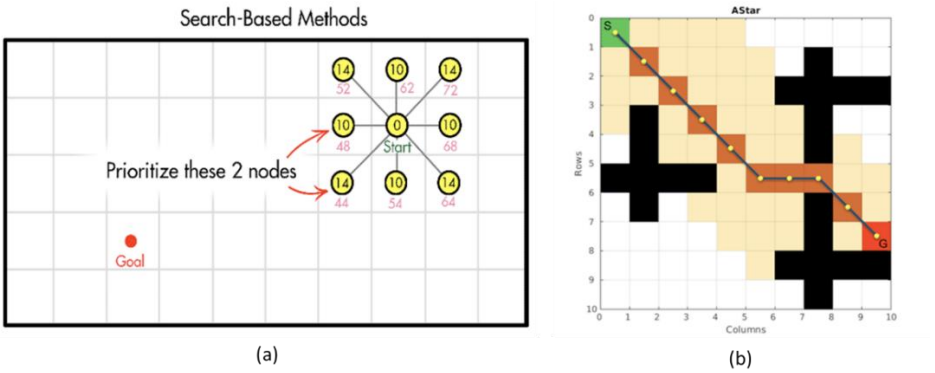



**Figura 18.** Planificación basada en búsquedas. (a) Método basado en búsquedas. (b) Algoritmo A* aplicado a una cuadricula. **Fuente:** (MathWorks, 2021)

La metodología de planificación basada en búsquedas establece un grafo en el que se pueden efectuar búsquedas, identificando cada estado o configuración del vehículo como un nodo específico. Este grafo se desarrolla desde el nodo inicial hasta el nodo de destino, utilizando técnicas centradas en costos y heurísticas para identificar la ruta más eficiente, tal y como puede observarse en la Figura 18. La planificación basada en búsquedas comúnmente se lleva a cabo en un mapa discretizado.

Este modelo suele ser adecuado cuando el robot puede considerarse como un punto y no involucra ninguna ecuación cinemática en la etapa de planificación, luego puede usarse un algoritmo de control para tomar en cuenta dichas restricciones cinemáticas (MathWorks, 2021; Karur, Sharma, Dharmatti, & Siegel, 2021).

Los algoritmos basados en búsqueda más utilizados son:

#### 5.4.2.1. Algoritmo de Dijkstra.

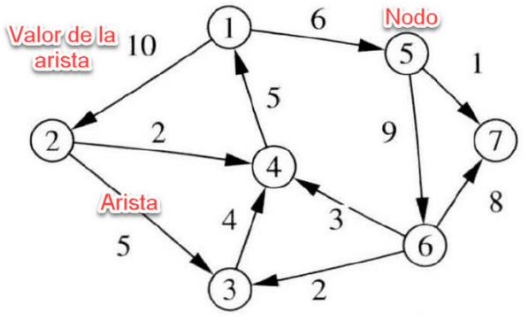



**Figura 19.** Algoritmo de Dijkstra. **Fuente:** (Universidad Nacional de Educación a Distancia, 2023)

El algoritmo de Dijkstra es un algoritmo de búsqueda de ruta más corta introducido por el científico informático neerlandés Edsger W. Dijkstra en 1959. Este algoritmo se basa en la resolución de problemas de los caminos más cortos a partir de un solo nodo origen en un grafo ponderado, donde todas las aristas tienen pesos no negativos (Cormen y otros, 2009). El funcionamiento del algoritmo de Dijkstra puede observarse en la Figura 19.

Su principio es sencillo: el algoritmo mantiene un conjunto de nodos cuya ruta más corta desde el origen es conocida y utiliza este conjunto para encontrar la ruta más corta a otros nodos. El algoritmo selecciona el nodo con la distancia más corta, lo actualiza con las distancias de sus vecinos y repite este proceso hasta que se han explorado todos los nodos.

El algoritmo de Dijkstra, aunque eficiente y útil, tiene limitaciones. Por ejemplo, no funciona bien en grafos con pesos negativos, ya que asume que el camino más corto a un nodo ya visitado no cambiará en el futuro (Kleinberg & Tardos, 2006).

#### 5.4.2.2. Algoritmo A*.

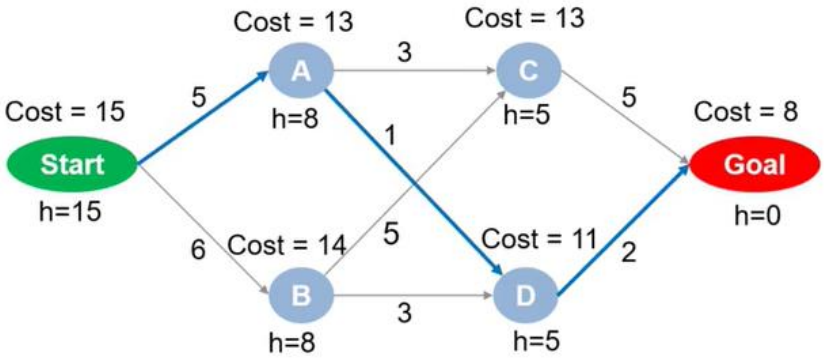



**Figura 20.** Funcionamiento del algoritmo A*. **Fuente: (MathWorks, 2021)**

El algoritmo A* es un planificador de ruta discreto que funciona mediante la creación de gráficos ponderados conectando tanto el nodo de inicio como el nodo objetivo, usando conexiones lineales en *xy* para expandir el árbol de búsqueda. Los nodos se exploran individualmente en función del costo estimado de viajar de un nodo a otro. A* es una extensión del algoritmo de Dijkstra que incorpora heurísticas para mejorar la eficiencia de la búsqueda.

La búsqueda A* utiliza una función de evaluación f(n) para cada nodo en el grafo de búsqueda, que es la suma de dos componentes:

1. g(n): El costo real desde el nodo de inicio hasta el nodo n.
2. h(n): Una estimación heurística del costo mínimo desde el nodo n hasta el nodo de meta.

La función f(n) = g(n) + h(n) entonces guía la búsqueda. En cada paso, el algoritmo expande el nodo en el borde de la búsqueda con el valor mínimo de f(n). Para que A* sea óptimo, es decir, garantice encontrar el camino más corto, la heurística h(n) debe ser admisible, lo que significa que nunca debe sobreestimar el costo real para llegar a la meta. Una elección común para h(n) en la planificación de rutas es la distancia euclidiana desde el nodo n hasta el nodo de la meta (Hart y otros, 1968), como puede observarse en la Figura 20.

#### 5.4.2.3. Algoritmo A* Híbrido.

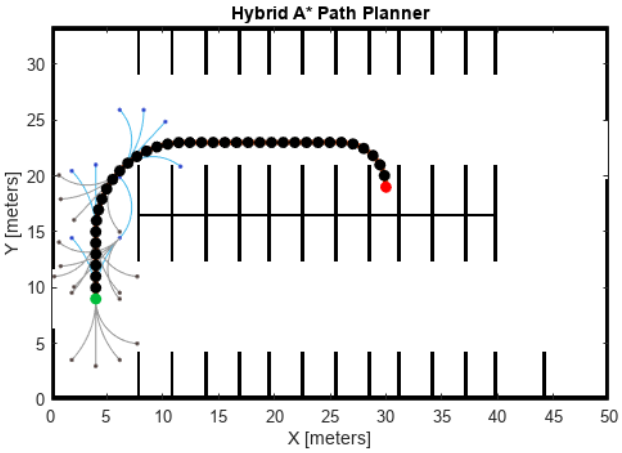



**Figura 21.** Algoritmo A* Híbrido. **Fuente**: (MathWorks, 2019)

El Algoritmo A* Híbrido es una extensión y mejora del Algoritmo A* clásico, diseñado para superar algunas de las limitaciones inherentes a la versión original, específicamente desarrollado para abordar el problema de la alta dimensionalidad. El término "híbrido" se refiere a la combinación de una búsqueda discreta (en la cuadricula) con una planificación de movimiento continua. Funciona en un espacio de búsqueda discretizado, pero asocia cada celda de la cuadrícula con un estado 3D continuo (x, y, θ) de un vehículo. Utiliza un espacio de estado continuo que consta de primitivas de movimiento para generar rutas suaves y manejables en forma de curvas como se puede observar en la Figura 21. La posición continua a la que llega una primitiva de movimiento se almacena junto con la posición discreta en la celda correspondiente.

En lugar de realizar una búsqueda en el espacio de todos los posibles estados del robot como lo hace el algoritmo A* regular, el A* Híbrido realiza una búsqueda en un espacio de configuración reducido utilizando un conjunto de primitivas de movimiento recalculadas para determinar los estados alcanzables y así construir el árbol de búsqueda. En la Figura 22 se puede observar una comparativa en la búsqueda de los posibles vecinos realizados por el algoritmo A* regular y el A* Híbrido. El algoritmo A* Híbrido también comparte sus principales conceptos con la versión regular del algoritmo A*. Utiliza dos conjuntos que mantienen un registro de los estados durante la búsqueda. En cada iteración, se selecciona el nodo con el valor f mínimo (la suma de los costos

reales y los costos estimados hasta la meta) y se expande, es decir, se buscan todos los estados vecinos alcanzables (Petereit y otros, 2012).

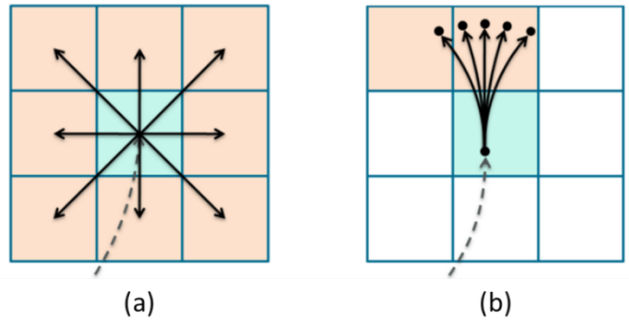



**Figura 22.** Posibles vecinos de una celda. (a) A* regular. (b) A* Híbrido. Fuente: (Petereit y otros, 2012)

A* Híbrido utiliza heurísticas de guía eficiente expandiendo el árbol de búsqueda en la dirección de la meta. Adicionalmente, mejora la precisión y reduce el tiempo de planificación al utilizar la expansión analítica de la ruta, lo que genera una ruta suave y manejable para un vehículo en función de la posición de destino.

### 5.4.3. Planificación de Movimiento con el Algoritmo A* Híbrido.

El problema de planeamiento de trayectorias para un robot móvil puede definirse como encontrar el camino de menor costo desde la posición inicial $x_s$ del vehículo hacia una región objetivo $\boldsymbol{G}$ dentro de un mapa discreto subdividido en celdas, respetando las restricciones impuestas por las cinemáticas del vehículo y las condiciones del terreno. El entorno en el que se desplaza el robot está representado por un mapa tridimensional discreto, con altura 2.5D, compuesto por tres capas:

- $m_o$: que indica si una celda está bloqueada o libre de obstáculos.
- $m_s$: que contiene información sobre la calidad de la superficie.
- $m_h$: que almacena el valor de altura para cada celda, formando un mapa de alturas.

El vehículo se posiciona en el mapa mediante un vector bidimensional $x_s$, acompañado de un ángulo de orientación $\theta_s$ . Para la planificación de la trayectoria, se ignora la altura del vehículo y se asume que esta coincide con la altura de la celda correspondiente $\boldsymbol{m_h(x_s)}$. Las coordenadas discreto $\tilde{x}$ (es decir, los índices de celda) se pueden obtener a partir de coordenadas discretas utilizando la siguiente formula:

$$\tilde{x} = \left\lfloor \frac{x - o_m}{\zeta} \right\rfloor \tag{8}$$

Donde $o_m$ es el origen del mapa y $\zeta$ es el tamaño de las celdas. Por lo tanto, la región objetivo $\boldsymbol{G}$ se define como el conjunto de todas posiciones $\boldsymbol{x}$ dentro de un radio $\boldsymbol{r}$ alrededor de un punto de referencia comandado $\boldsymbol{w}$, descrito por la siguiente ecuación:

$$G = \{ x \mid \|x - w\| < r \} \tag{9}$$

En términos simples, está definiendo una región objetivo $\boldsymbol{G}$ como el área que se encuentra dentro de un radio $\boldsymbol{r}$ alrededor de una posición objetivo o waypoint $\boldsymbol{w}$ (Petereit y otros, 2012).

#### 5.4.3.1. Representación del espacio de búsqueda

El estado continuo de un vehículo que se desplaza en un plano está representado por $(x, \theta)$ con $\boldsymbol{x}$ siendo la posición del vehículo y $\theta$ siendo su dirección. Este espacio de búsqueda tridimensional debe ser discretizado para ser buscable por el algoritmo A* híbrido. La discretización traslacional ya está dada por la representación del mapa, mientras que la orientación del vehículo ($\boldsymbol{\theta_s}$ ) se redondea a su valor discreto más cercano.

El espacio de búsqueda híbrido permite que cada celda almacene tanto la posición discreta $\boldsymbol{x_s}$ como la continua $\boldsymbol{x}$ alcanzada al llegar a esa celda. Esta información es crucial para asegurar que la trayectoria encontrada sea transitable. Además, se utiliza una función de costo que evalúa el costo de moverse desde la posición actual a una adyacente. El costo acumulado se denota como $\boldsymbol{g}$, y cada nodo del gráfico de búsqueda contiene un puntero a su nodo padre $\boldsymbol{n_p}$ para reconstruir el camino final. Los nodos también almacenan una clave de prioridad $f$, que es la suma de los costos reales $\boldsymbol{g}$ y los costos estimados $\boldsymbol{h}$ hasta la meta (Petereit y otros, 2012). De esta manera, un nodo $\boldsymbol{n}$ queda completamente definido por:

$$n = (x_s, \theta_s, x, g, f, n_p) \tag{10}$$

#### 5.4.3.2. Construcción de las primitivas de movimiento.

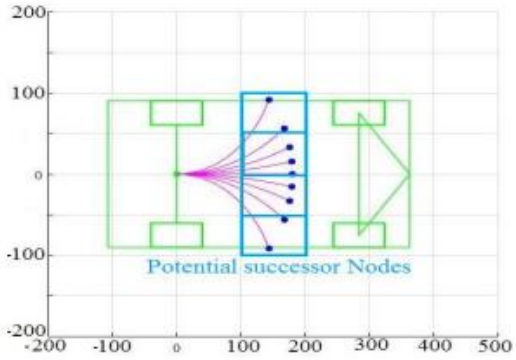



**Figura 23.** Posibles primitivas de movimiento para un robot móvil de 4 ruedas. **Fuente:** (Sedighi y otros, 2019).

Las primitivas de movimiento representan las unidades básicas de desplazamiento en el algoritmo A* Híbrido y están definidas por arcos. Estas primitivas son trayectorias predefinidas que el vehículo puede seguir, y deben cumplir con ciertas restricciones que garantizan la factibilidad de la trayectoria. En la Figura 23 podemos observar todas las posibles primitivas de movimiento que puede realizar un robot móvil de 4 ruedas partiendo de un punto de origen ubicado en el centro de masa del robot hasta la siguiente celda disponible.

Específicamente, una primitiva de movimiento debe satisfacer las siguientes tres condiciones:

- La distancia recorrida debe ser suficiente para abandonar la celda actual. Por lo tanto, $l > \sqrt{2}\,\zeta$, donde $l$ es la longitud del arco.
- La curvatura está limitada por el ángulo de dirección máximo.
- El cambio de dirección debe ser un múltiplo del tamaño del paso de discretización de la dimensión de dirección del espacio de búsqueda (Petereit y otros, 2012).

#### 5.4.3.3. Proceso de búsqueda del algoritmo A* Híbrido.

Durante el proceso de búsqueda, el algoritmo utiliza dos conjuntos de nodos:

- El **conjunto abierto** ($O$) contiene los nodos vecinos de aquellos ya expandidos, pero que aún no han sido procesados.
- El **conjunto cerrado** ($C$) contiene los nodos que ya han sido completamente procesados.

El objetivo es encontrar la trayectoria de menor costo desde el nodo inicial hasta la región objetivo, siguiendo un esquema de búsqueda por prioridad, basado en los costos acumulados $g$ y los costos estimados $h$ hacia el objetivo. En la Tabla 2 se ilustra una descripción general del esquema de un algoritmo A* Híbrido estándar:

**Tabla 2.** Algoritmo A* Híbrido estándar. **Fuente:** Adaptado de (Petereit y otros, 2012)

**Algoritmo 1: Versión estándar del A* Híbrido**

```
Procedimiento PLANIFICAR_RUTA(m, μ, x_s, θ_s, G)
    n_s ← (x̃_s, θ̃_s, x_s, 0, h(x_s, G), −)
    O ← {n_s}
    C ← ∅
    Mientras O ≠ ∅ hacer
        n ← nodo con valor mínimo de f en O
        O ← O \ {n}
        C ← C ∪ {n}
        Si n_x ∈ G entonces
            Retornar la ruta reconstruida empezando en n
        Si no
            ACTUALIZAR_VECINOS(m, μ, O, C, n)
        Fin si
    Fin mientras
    Retornar no se encontró ruta
Fin procedimiento

Procedimiento ACTUALIZAR_VECINOS(m, μ, O, C, n)
    Para todo δ hacer
        n' ← estado sucesor de n usando μ(n_θ, δ)
        Si n' ∉ C entonces
            Si m_o(n'_x̃) = obstáculo entonces
                C ← C ∪ {n'}
            Si no
                Si existe ∃n ∈ O : n_x̃ = n'_x̃ entonces
                    Calcular nuevos costos g'
                    Si g' < valor de g del nodo existente en O
                        entonces
                        Reemplazar el nodo existente en O por n'
                    Fin si
                Si no
                    O ← O ∪ {n'}
                Fin si
            Fin si
    Fin para
Fin procedimiento
```

### 5.4.4. Optimización de la trayectoria usando el algoritmo TEB.

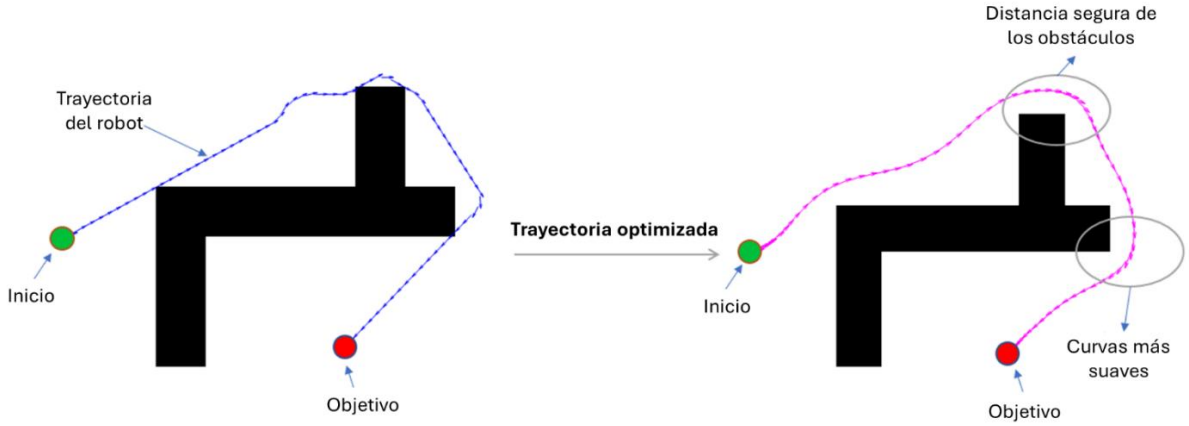



**Figura 24.** Optimización de la trayectoria utilizando el algoritmo TEB. **Fuente:** Adaptado de (MathWorks, 2024)

En la mayoría de entornos reales, un robot móvil autónomo necesita enfrentar diversos desafíos como realizar curvas cerradas o evitar obstáculos, y la trayectoria generada por los planificadores no siempre es la más segura, la más suave o la más estable. Para resolver tales inconvenientes es esencial realizar un proceso de optimización. Para optimizar la trayectoria de manera eficiente en tiempo real, el enfoque del ***Timed-Elastic-Band*** (TEB, por sus siglas en inglés) emerge como una solución robusta, permitiendo la modificación de trayectorias en entornos dinámicos sin comprometer las restricciones físicas del robot.

El algoritmo TEB se fundamenta en el enfoque clásico del "***Elastic Band***" (EB), en donde se deforma la trayectoria modelándola como si fuese una banda elástica. Esta banda está sujeta a fuerzas internas que la contraen y fuerzas externas que la mantienen a una distancia segura de los obstáculos, permitiendo al robot seguir una ruta más directa y libre de colisiones (Quinlan, 1995). Sin embargo, el enfoque EB original no tiene en cuenta las limitaciones dinámicas, como las velocidades y aceleraciones limitadas, adaptando la trayectoria de manera eficiente. Aquí es donde entra en juego el enfoque TEB, una extensión del algoritmo "elastic band" clásico que considera no solo tiene en cuenta la geometría de la trayectoria, sino también las limitaciones dinámicas y temporales del robot.

El objetivo de este enfoque es adaptar esta trayectoria a las condiciones cambiantes del entorno y las limitaciones del robot en tiempo real, utilizando un marco de optimización multiobjetivo ponderado. Este marco permite ajustar tanto las configuraciones espaciales (poses) del robot como los intervalos de tiempo entre ellas. Cada intervalo de tiempo representa el tiempo que el robot necesita para transitar entre dos configuraciones consecutivas. El TEB optimiza simultáneamente estas configuraciones espaciales y los intervalos de tiempo, buscando minimizar una función de costo que suma múltiples objetivos relevantes, dando como resultado una adaptación de la trayectoria original (Roesmann y otros, 2012). Un ejemplo de aplicación de una trayectoria optimizada utilizando el enfoque TEB puede visualizarse en la Figura 24, donde podemos observar como la nueva trayectoria optimizada mantiene una distancia segura frente a los obstáculos generando una curva suave en la trayectoria para así evitarlos sin utilizar cambios abruptos y adaptándose a las limitaciones físicas del robot.

#### 5.4.4.1. Funcionamiento del Algoritmo TEB.

El proceso de optimización de trayectorias mediante TEB puede dividirse en los siguientes pasos:

I. **Inicialización de la trayectoria:** El algoritmo comienza con una trayectoria inicial generada por el planificador global. Esta trayectoria es representada como una secuencia discreta de las poses (posición y orientación) del robot a lo largo del tiempo.

II. **Parámetros de optimización:** se definen un conjunto de parámetros a optimizar. Estos parámetros incluyen las posiciones discretas del robot en la trayectoria y los intervalos de tiempo asociados a cada segmento de la trayectoria. El objetivo es encontrar la mejor combinación de estos parámetros que minimice el tiempo total de recorrido, cumpliendo al mismo tiempo con las restricciones del robot.

El vector de parámetros de optimización se define como:

$$B = \{s_1, \Delta T_1, s_2, \Delta T_2, \dots, s_{n-1}, \Delta T_{n-1}, s_n\} \tag{11}$$

En donde:

- $s_1, s_2, \dots, s_n$ son las configuraciones del robot (su posición y orientación) en puntos discretos a lo largo de la trayectoria.
- $\Delta T_1, \Delta T_2, \dots, \Delta T_{n-1}$ los intervalos de tiempo, los cuales indican el tiempo necesario para transitar entre estas configuraciones.

III. **Planteamiento del problema de optimización:** el problema de optimización en TEB se formula como un procedimiento de programación no lineal (NLP, por sus siglas en inglés) en el que se minimiza la suma de los tiempos de transición al cuadrado, tal y como se puede observar en la siguiente ecuación:

$$\min_{B} \sum_{k=1}^{n-1} \Delta T_k^2 \tag{12}$$

La cual está sujeta a las siguientes restricciones:

- **Restricciones cinemáticas:** estas garantizan que el robot cumpla con las ecuaciones de movimiento que describen su comportamiento, como la velocidad máxima, el radio de giro mínimo y la orientación en cada punto.
- **Restricciones de evitación de obstáculos:** aseguran que el robot mantenga una distancia mínima segura de los obstáculos.
- **Restricciones de velocidad y aceleración:** estas limitan las velocidades y aceleraciones máximas y mínimas, tanto en términos lineales como angulares, para evitar movimientos bruscos o inestables.

El TEB resuelve el problema de optimización utilizando métodos de optimización no lineal, como el algoritmo de Levenberg-Marquardt, el cual es eficiente para este tipo de problemas debido a su capacidad para manejar grandes conjuntos de parámetros y restricciones.

IV. **Actualización y deformación de la trayectoria:** una vez planteado el problema de optimización, el TEB utiliza métodos iterativos para encontrar la solución óptima. Se ajusta continuamente la trayectoria en función de los cambios en el entorno del robot y se recalculan los parámetros de optimización (posiciones y tiempos) en cada iteración. En cada paso, se evalúan las restricciones impuestas y se aplica una penalización a las

soluciones que no las cumplan, lo que asegura que el robot mantenga una trayectoria segura y factible.

V. **Evasión de obstáculos:** el algoritmo introduce fuerzas de repulsión que alejan las configuraciones del robot de cualquier obstáculo presente en su entorno. Estas fuerzas se ajustan dinámicamente a medida que el robot detecta nuevos obstáculos, lo que permite que la trayectoria se adapte continuamente a un entorno cambiante.
La distancia mínima entre el robot y cualquier obstáculo se define mediante una métrica continua, como la métrica euclidiana, y se garantiza que esta distancia no sea inferior a un valor predefinido, asegurando así que el robot se mantenga a una distancia segura de los obstáculos (Rosmann y otros, 2017).

El proceso de optimización se realiza iterativamente hasta obtener una trayectoria optima basada en los parámetros seleccionados previamente. Al reformular la planificación de trayectorias como un problema de optimización con restricciones, el TEB proporciona soluciones viables y eficientes que garantizan la seguridad y la estabilidad del robot en todo momento.

#### 5.4.4.2. Parámetros para la evasión de obstáculos.

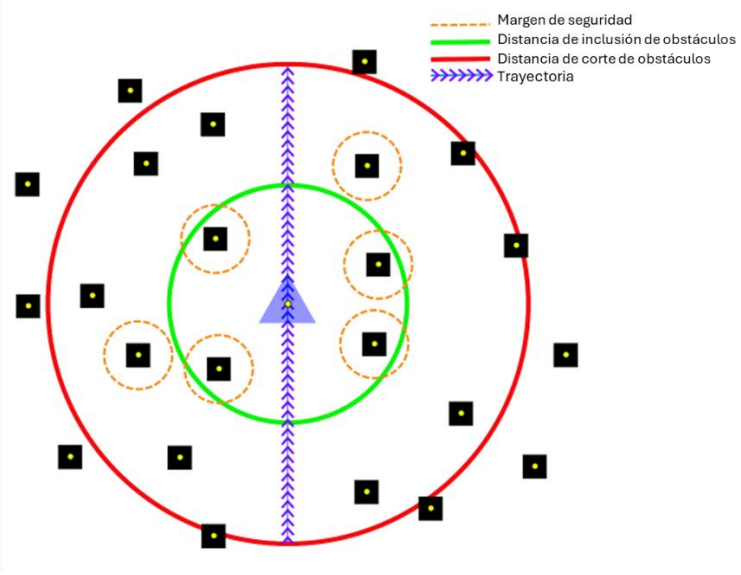



**Figura 25.** Parámetros para la evasión de obstáculos. **Fuente:** Adaptado de (MathWorks, 2024).

Los parámetros considerados para resolver el problema de la evasión de obstáculos son:

- **Margen de seguridad:** especifica una distancia segura a mantener entre las poses de la trayectoria y los obstáculos.
- **Distancia de inclusión de obstáculos:** especifica la distancia desde el robot dentro de la cual se consideran todos los obstáculos mientras se optimiza la ruta en ese caso.
- **Distancia de corte de obstáculos:** especifica la distancia desde el robot más allá de la cual no se consideran todos los obstáculos al optimizar la ruta en ese caso.

El optimizador de ruta también considera el obstáculo más cercano a la izquierda y a la derecha del robot entre la región de inclusión y de corte (MathWorks, 2024). La optimización de estos parámetros son clave para el correcto desempeño de este algoritmo.

## 5.5. Seguimiento de trayectorias

El seguimiento de trayectorias tiene como objetivo controlar el movimiento del robot para que siga una ruta predefinida mientras ajusta su posición y orientación mediante el uso de un conjunto de leyes de control que guíen su movimiento y minimicen los errores respecto a la ruta deseada, esto implica que minimizar las diferencias entre la posición real del robot y la posición deseada en cada instante de tiempo.

El proceso de seguimiento de trayectorias involucra el uso de dos componentes principales:

- **Modelo cinemático:** describe el comportamiento del robot en términos de su posición y orientación, así como las velocidades lineales y angulares que determinan su movimiento.
- **Control de seguimiento**: es el sistema encargado de corregir los errores de seguimiento generados por factores como cambios en la dinámica del robot, imprecisiones en los sensores o perturbaciones externas. Para lograrlo, se emplean diversas estrategias de control que buscan minimizar la distancia entre la trayectoria real y la deseada.

### 5.5.1. Funcionamiento del seguimiento de trayectorias

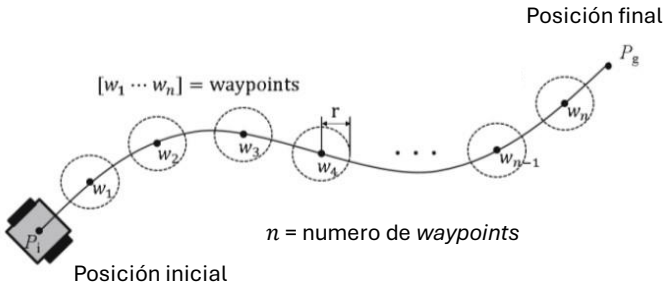



**Figura 26.** Seguimiento de trayectoria de un robot móvil mediante *waypoints.* **Fuente:** Adaptado de **(Yoon y otros, 2014)**

El seguimiento de trayectorias comienza con el resultado de la planificación de la ruta, en donde se definen un conjunto de puntos o curvas (llamados "*waypoints*" o "*puntos de referencia*"), que el robot debe seguir. Durante la ejecución, el controlador de seguimiento recibe información en tiempo real de los sensores de posición para estimar la ubicación actual del robot. Basado en esta información, el controlador compara la posición real con la posición deseada y calcula las acciones correctivas necesarias para minimizar el error.

Las acciones correctivas suelen implicar el ajuste de las velocidades de las ruedas motrices del robot para corregir tanto la orientación (ángulo) como la posición a lo largo de la trayectoria. Por ejemplo, si el robot se desvía hacia la izquierda, el controlador aumentará la velocidad de la rueda derecha o disminuirá la de la izquierda para corregir la dirección del movimiento.

El **error de seguimiento** se define por dos componentes principales:

- **Error lateral**: la distancia entre la posición actual del robot y la línea de la trayectoria deseada.
- **Error angular**: la diferencia entre la orientación actual del robot y la orientación deseada en ese punto de la trayectoria.

La tarea del controlador es minimizar ambos errores de manera continua, asegurando que el robot siga la trayectoria de la manera más precisa posible (Pacheco & Luo, 2015).

### 5.5.2. Métodos principales para el seguimiento de trayectoria

Existen diversos enfoques para abordar el problema del seguimiento de trayectorias, dependiendo de los modelos dinámicos y cinemáticos de los robots, así como de las condiciones en las que operan. Entre los más utilizados se encuentran:

- **Seguimiento con control PID:** En el ámbito del seguimiento de la trayectoria, el controlador PID se emplea principalmente para regular la velocidad de las ruedas de los robots, corrigiendo los errores de la trayectoria usando una combinación de acciones proporcionales $(P)$, integrales $(I)$ y derivativas $(D)$, lo que permite ajustar su orientación y posición de manera precisa respecto a la trayectoria deseada (Nguyen y otros, 2022). Un diagrama de bloques de una configuración de seguimiento de trayectoria utilizando un controlador PID con parámetros variables puede observarse en la siguiente figura:

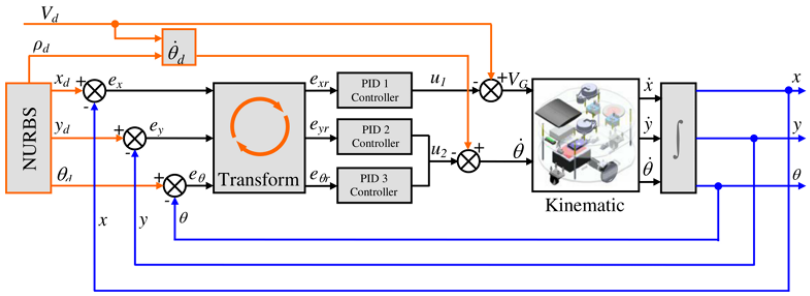



**Figura 27.** Diagrama de bloques de un control PID variable para el seguimiento de trayectoria. **Fuente:** (Nguyen y otros, 2022)

- **Seguimiento con control predictivo (MPC):** El control predictivo basado en modelos (MPC, por sus siglas en inglés) es una técnica avanzada de control radica en utilizar un modelo del sistema para predecir su comportamiento futuro y optimizar las acciones de control en un horizonte de tiempo definido. Esto permite que el controlador ajuste de manera proactiva las entradas para minimizar un criterio de costo, usualmente relacionado con el error de seguimiento de la trayectoria y el esfuerzo de control aplicado. A partir de esta predicción, el controlador genera una secuencia de comandos de control que minimiza la función de costo, garantizando que el robot siga la trayectoria deseada de la mejor manera posible dentro de un horizonte de predicción definido (Shuyou y otros, 2018).

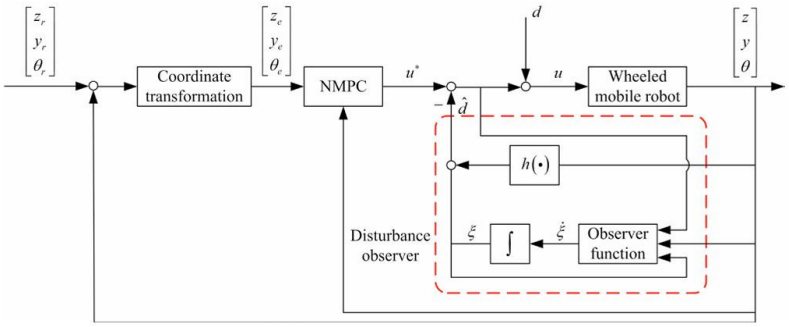



**Figura 28.** Diagrama de bloques de un control MPC para el seguimiento de trayectoria**. Fuente:** (Shuyou y otros, 2018)

- **Seguimiento con control Pure Pursuit:** es un método geométrico de control que se utiliza para guiar la trayectoria de un robot móvil a lo largo de una serie de puntos predeterminados llamados “waypoints”. El robot se desplaza siguiendo un punto que está ubicado a una cierta distancia adelantada, luego ajusta las velocidades angulares que desplaza al robot para alcanzar dicho punto (Coulter, 1990). El funcionamiento completo de este algoritmo se explica a detalle en la siguiente subsección.

### 5.5.3. Algoritmo Pure Pursuit.

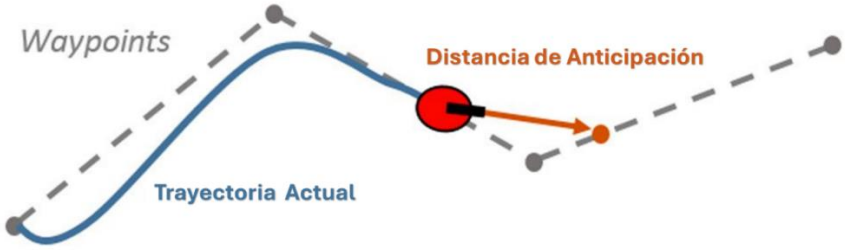



**Figura 29.** Seguimiento de trayectoria con un controlador Pure Pursuit**. Fuente:** Adaptado de **(MathWorks, 2022)**

El algoritmo Pure Pursuit se basa en un enfoque geométrico. Su principio consiste en trazar un arco que conecta la posición actual del robot con un punto objetivo en la trayectoria predefinida. Este punto, conocido como **punto de anticipación** (también conocido como punto objetivo o ***goal point***), está situado a una distancia fija por delante del vehículo, denominada distancia de anticipación $L$ (*lookahead distance*). A partir de esta información, se calcula la curvatura $\kappa$ que define el arco trazado. Este proceso se repite continuamente hasta que el robot alcanza el último punto de la ruta, lo que significa que el robot siempre persigue un punto por delante de él. De ahí proviene su nombre, traducido al español como 'Persecución pura'.

A diferencia de los métodos anteriores, el algoritmo Pure Pursuit no estabiliza el robot en un punto, en su lugar se debe aplicar un umbral de distancia a una ubicación objetivo para detener el robot cerca del objetivo deseado. Adicionalmente, su implementación requiere un bajo coste computacional en comparación con los otros métodos.

#### 5.5.3.1. Descripción geométrica del algoritmo.

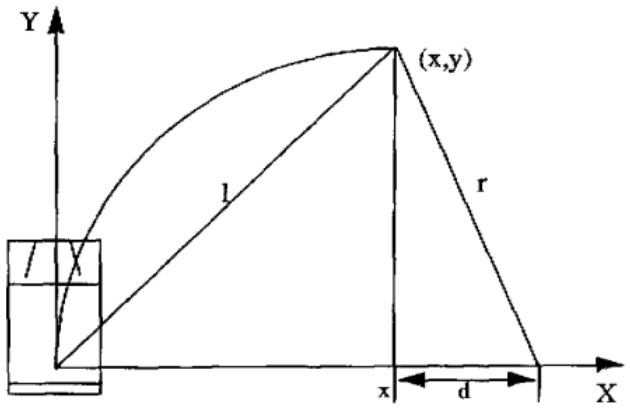



**Figura 30.** Geometría del algoritmo. **Fuente: (Coulter, 1990)**

Considere el escenario planteado en la Figura 30 en donde se representa al robot en un sistema de coordenadas donde el eje X pasa por su eje trasero. En este sistema, el objetivo es calcular la curvatura del arco que une la posición actual del robot (el origen) con el punto objetivo $(\boldsymbol{x}, \boldsymbol{y})$, que está a una distancia $\boldsymbol{L}$ del origen. Este arco puede ser descrito como parte de un círculo, y la curvatura de ese círculo es lo que se busca calcular. Por lo tanto, sabemos que la ecuación del círculo es:

$$x^2 + y^2 = L^2 \tag{13}$$

Igualmente, por geometría la ecuación que describe la relación entre el radio de desplazamiento lateral $d$ del robot y el radio del arco que conecta la posición actual con el punto objetivo es:

$$d = \frac{r^2 - x^2}{2x} \tag{14}$$

Ahora, sabiendo que la curvatura $\kappa$ esta inversamente relacionada con el radio del círculo formado por el arco entre el robot y el punto objetivo, la ecuación que define esta curva es:

$$\kappa = \frac{2y}{L^2} \tag{15}$$

La ecuación $(\mathbf{15})$ describe como la curvatura que el robot debe seguir está directamente relacionada con la posición del punto objetivo. A mayor distancia vertical $\boldsymbol{y}$ del punto objetivo respecto al eje X, mayor será la curvatura requerida para que el robot alcance dicho punto. Además, como se observa en la ecuación, la curvatura $\boldsymbol{\kappa}$ disminuye a medida que aumenta la distancia de anticipación $\boldsymbol{L}$, lo que refleja el comportamiento del algoritmo: cuanto más lejos "mira" el vehículo en el camino, más suave es la trayectoria que sigue (Coulter, 1990).

#### 5.5.3.2. Implementación del algoritmo.

La implementación del algoritmo Pure Pursuit sigue una serie de pasos claramente definidos que permiten realizar el seguimiento de una trayectoria predeterminada, los cuales son:

I. **Determinar la ubicación actual del robot:** en el entorno global. Para esto, el robot utiliza sensores como GPS, LiDAR o cámaras, proporcionando la posición en forma de coordenadas $(x, y, \theta)$.

II. **Encontrar el punto de la trayectoria más cercano al robot:** esto se realiza con el fin de establecer un punto de referencia desde el cual se calculará el punto objetivo que se encuentra a una distancia de anticipación $\boldsymbol{L}$ adelante. El algoritmo busca el punto más cercano al robot mediante la comparación de distancias entre la posición actual y cada punto de la trayectoria.

III. **Encontrar el punto objetivo:** este punto es el objetivo inmediato que el robot intentará alcanzar. Este proceso implica avanzar a lo largo de la trayectoria a partir del punto más cercano encontrado en el paso anterior, midiendo la distancia entre cada nuevo punto y la ubicación actual del robot. La búsqueda continúa hasta que se encuentra un punto cuya distancia del robot es igual o muy cercana a $\boldsymbol{L}$. Si existen múltiples puntos que cumplen esta condición, se selecciona el que minimice el error en la dirección.

IV. **Transformar el punto objetivo a coordenadas del robot:** Esto se realiza porque los cálculos de la curvatura que debe seguir el robot se efectúan en su propio sistema de

referencia, donde el eje X está alineado con la dirección hacia adelante del vehículo, y el eje Y indica el desplazamiento lateral. Esta operación permite expresar la posición del punto objetivo en términos de la distancia hacia adelante y lateral desde el vehículo, lo cual es crucial para el siguiente paso.

V. **Calcular la curvatura $\kappa$:** utilizando la ecuación $(15)$, se determina la curvatura que el robot debe seguir. Este valor luego se traduce en un ángulo de dirección que se le debe aplicar al robot. Este valor de curvatura se transforma en un comando de dirección que se envía al sistema de control del robot, el cual traduce la curvatura en un ángulo de dirección que deben seguir las ruedas, ajustando los actuadores que controlan las ruedas.

VI. **Actualizar la posición del robot:** El algoritmo actualiza continuamente la posición del robot, ajustando la trayectoria en función de los nuevos datos recibidos (Coulter, 1990).

#### 5.5.3.3. Propiedades del Algoritmo.

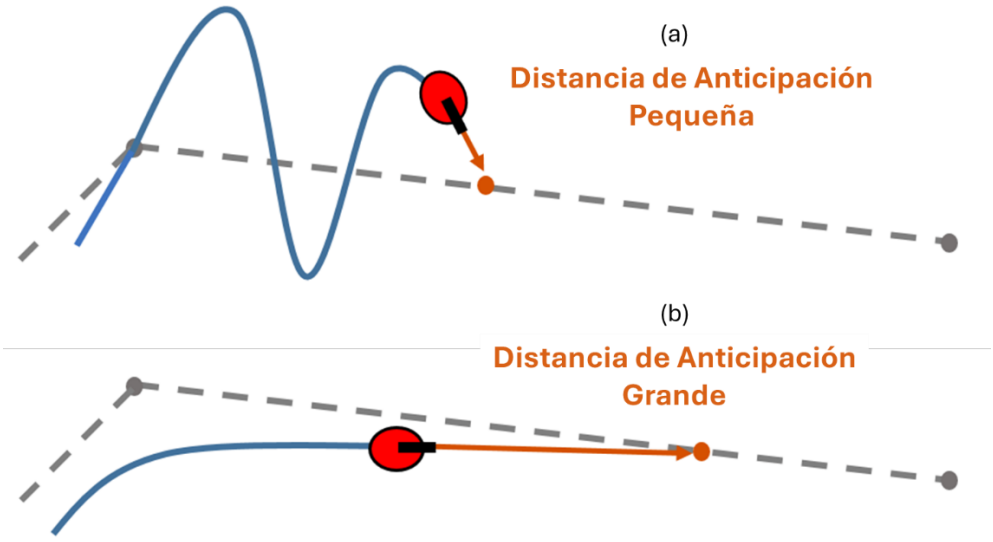



**Figura 31.** Efectos del cambio en la Distancia de Anticipación. (a) Distancia de anticipación pequeña. (b) Distancia de anticipación grande. **Fuente:** Adaptado de **(MathWorks, 2022)**

El algoritmo Pure Pursuit es muy sensible a la elección de la **distancia de anticipación** $L$, la cual tiene un impacto directo en el comportamiento del robot mientras sigue una trayectoria. La influencia de este parámetro puede ser analizada en dos escenarios: cuando el vehículo está **recuperando** la trayectoria y cuando está **manteniendo** la trayectoria.

**Recuperación de la trayectoria.**

Este escenario ocurre cuando el vehículo está alejado del camino deseado y necesita volver a él. El comportamiento del algoritmo en este caso se asemeja al de un sistema dinámico de segundo orden, en el que la distancia de anticipación $L$ actúa como un factor de amortiguamiento.

- **Distancia de anticipación pequeña:** si $L$ es pequeño, el robot tenderá a corregir su curso de manera agresiva para alcanzar rápidamente la trayectoria. Aunque esto puede resultar en un tiempo de convergencia más corto, también puede provocar oscilaciones

indeseadas en el comportamiento del vehículo, generando un seguimiento inestable. Este caso puede observarse en la Figura 31 (a).

- **Distancia de anticipación grande:** Con un $\boldsymbol{L}$ más grande, la corrección del robot es más suave y gradual, con menos oscilación. No obstante, el tiempo que tarda en recuperar la trayectoria puede aumentar debido a la respuesta más lenta del sistema. En general, se observa que una mayor distancia de anticipación tiende a suavizar la respuesta del algoritmo, pero con una convergencia más lenta. Este caso puede observarse en la Figura 31 (b).

**Mantenimiento de la trayectoria.**

Cuando el robot ya está alineado con la trayectoria y desea mantenerse en ella, el efecto de la distancia de anticipación es diferente. En este caso, $\boldsymbol{L}$ afecta directamente la capacidad del vehículo para seguir trayectorias con distintas curvaturas:

- **Trayectorias con curvas cerradas:** Si el camino es muy curvado y el valor de $\boldsymbol{L}$ es grande, el robot tendrá dificultades para seguir correctamente el trayecto. Esto se debe a que el algoritmo Pure Pursuit traza un arco que conecta la posición actual del vehículo con el punto objetivo, pero un valor grande de $\boldsymbol{L}$ hace que el arco calculado no pueda seguir correctamente las curvas cerradas del camino, generando errores de seguimiento.
- **Trayectorias más suaves:** En caminos más rectos o con curvas más amplias, un valor grande de $\boldsymbol{L}$ funciona adecuadamente, ya que el robot puede seguir el camino sin generar grandes errores. Este es el caso ideal para trayectorias con menos cambios abruptos en la dirección.

Por lo tanto, la distancia de anticipación es un parámetro que no puede seleccionarse arbitrariamente, ya que tiene diferentes efectos dependiendo del tipo de trayecto y la distancia del robot respecto a la trayectoria deseada.

## 5.6. Entorno de simulación de escenarios con Simulink.

Simulink es una herramienta de modelado y simulación gráfica creada por la empresa MathWorks comúnmente usado en ingeniería que permite ensamblar un modelo de un sistema ciberfísico conectando una serie de bloques de subsistemas predefinidos utilizando cables virtuales (señales). Los bloques pueden representar elementos de modelo fundamentales como constantes, bloques de ganancia y colas, o algoritmos y subsistemas más complicados como transformadas rápidas de Fourier o filtros de Kalman.

MathWorks ha desarrollado una caja de herramientas de animación 3D para Simulink que se basa en el estándar de lenguaje de modelado de realidad virtual (VRML), pero su conjunto de características y calidad gráfica aún no son competitivos a nivel de simulación con los de una herramienta de desarrollo de entornos virtuales como los empleados en la industria de los videojuegos, ya que estas herramientas cuentan con motores multifísicos capaz de emular condiciones físicas reales con gran precisión (Haley y otros, 2012). Actualmente existen una gran variedad de software de simulación enlazados con Matlab como Gazebo, RoadRunner o Unreal Engine.

### 5.6.1. Motor de videojuego.

Un motor de videojuego es un software especializado en la creación de videojuegos proporcionando métodos para abstraer funcionalidades comunes típicas de este ámbito con el objetivo de que el desarrollador no comience desde cero en su programación, permitiéndole reutilizar activos clave para la ejecución del juego (Andrade, 2015).

Algunas de las funcionalidades que comúnmente se encuentran en los videojuegos son:

- **Motor de renderizado:** el renderizado es el proceso de creación de una imagen a partir de datos gráficos subyacentes. Por lo tanto, el motor de renderizado es responsable de los gráficos generados durante la visualización del videojuego.
- **Manejo de entradas:** permite al usuario interactúa con el juego a través del teclado, el mouse u otro hardware.
- **Bucle del juego:** son las rutinas internas que se actualizan constantemente durante el juego y son responsables de desencadenar eventos.
- **Motor de físicas:** simula el comportamiento físico de los objetos en el mundo virtual, permitiendo simular tanto eventos sencillos como la fuerza de la gravedad, como eventos más complejos como la detección de las colisiones entre los objetos.
- **Gráficos de escenas:** gestiona los elementos gráficos y la disposición espacial de los objetos en la pantalla.
- **Animación:** efecto de movimiento de los objetos que proporciona una jugabilidad vivida y realista.
- **Gestión de recursos:** gestiona eficientemente los recursos limitados por el hardware, como la memoria y el tiempo de procesamiento (Demmel y otros, 2022).

### 5.6.2. Unreal Engine.

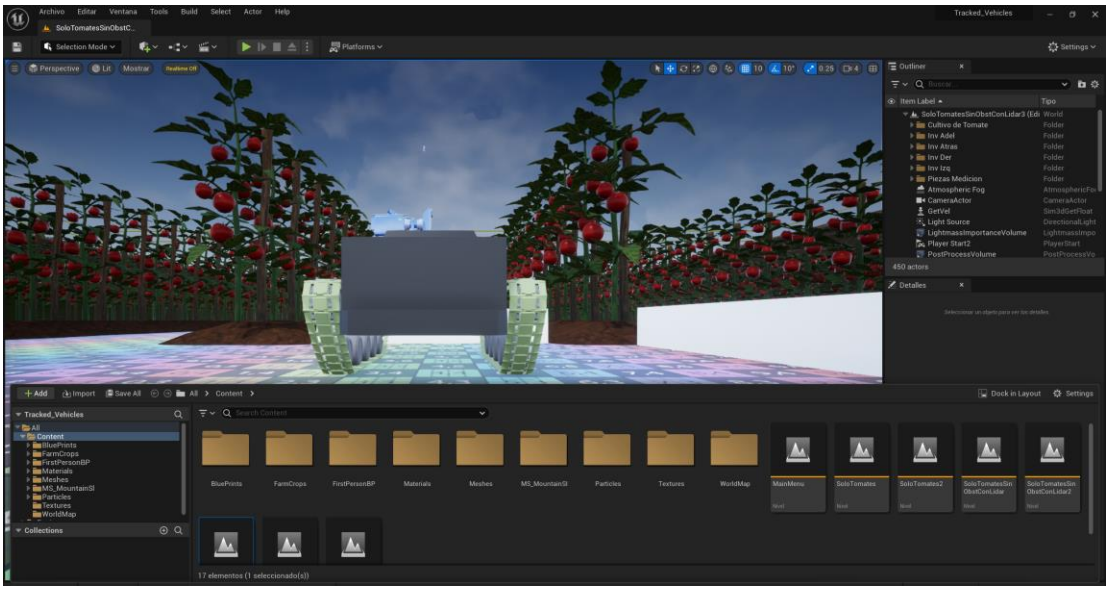


**Figura 32.** Entorno de programación en Unreal Engine**. Fuente:** Autor.

Unreal Engine, desarrollado por Epic Games, es una poderosa serie de motores de videojuegos Open Source especializado en la creación de contenido en 3D de tiempo real. Aunque su uso fue concebido como una herramienta de creación de videojuegos, su excelente potencia computacional y sus múltiples recursos han demostrado su precisión en la representación de físicas realistas (Yeh & Nugroho, 2021), ofreciendo características y funcionalidades que son útiles para estos fines, como se detalla a continuación:

- **Visualización y Simulación en Tiempo Real**: Gracias a sus capacidades avanzadas de renderizado, Unreal Engine permite la visualización de proyectos de ingeniería en tiempo real y con un alto grado de detalle.
- **Blueprint Visual Scripting**: Esta es una característica que permite a los usuarios sin habilidades de programación crear lógica de comportamiento y simulaciones.
- **Simulación Física**: Unreal Engine tiene un sólido sistema de simulación física. Puede simular gravedad, colisiones, fluidos, y más. Esto puede ser útil en la ingeniería para simular y entender cómo un diseño interactuará con las fuerzas físicas en el mundo real (Epic Games, 2021).

Una de las principales ventajas que tiene Unreal Engine frente a otros motores de videojuegos es que permite diferentes modos de programación como la codificación tradicional en el lenguaje C++, como la implementación de programación grafica por medio de scripts visuales llamados Blueprint. Los Blueprint consiste en una interfaz basada en nodos a través de la cual la programación es accesible incluso para los no programadores. De esta manera, los elementos del juego pueden programarse colocando nodos y conectándolos con cables (Demmel y otros, 2022).

# 6. METODOLOGÍA

Para el desarrollo de este proyecto se plantea utilizar la directriz VDI 2206 presentada por la Sociedad Alemana de Ingenieros (2004), la cual es una metodología de modelado funcional basado en el modelo V, orientada al desarrollo de sistemas mecatrónicos, dividiendo el proceso de diseño en cuatro etapas fundamentales, tal y como se presenta en la Figura 33.

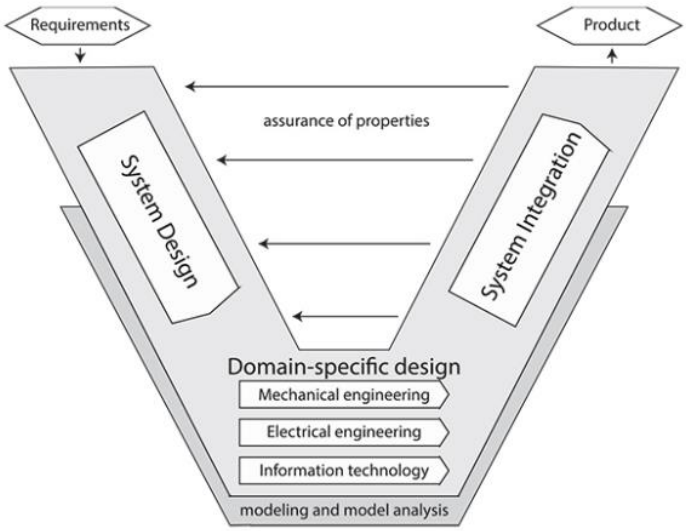



**Figura 33.** Directriz VDI2206 para el diseño Mecatrónico. **Fuente: (Verein Deutscher Ingenieure, 2004)**

**Etapa 1 – Diseño del sistema (System Design):**

Durante esta etapa se investiga y selecciona la parte técnica necesaria para desarrollar el proyecto, enfocándose en los requerimientos y las necesidades del cliente para poder definir plenamente el problema con el fin de plantear una solución a implementar.

**Subetapas:**

a) **Definición de los requerimientos del sistema:** se identifica las necesidades y los requisitos que correspondan a la problemática a solucionar.

   El sistema de navegación autónoma debe cumplir con los siguientes requerimientos:

   - ✓ Desplazamiento autónomo de un robot móvil tipo oruga en un entorno simulado que represente un cultivo de tomate de invernadero.
   - ✓ Localización precisa del robot dentro del cultivo mediante un sistema de mapeo y reconocimiento del entorno.
   - ✓ Planificación de trayectorias óptimas considerando las restricciones cinemáticas del robot y los obstáculos en el entorno.
   - ✓ Seguimiento adecuado de la trayectoria generada y capacidad de reacción ante obstáculos previamente detectados.

b) **Definición de las funciones técnicas globales:** incluyendo las funciones que satisfagan la necesidad, las funciones que las restrinja y las que complementen el servicio. Todo

desde un punto de vista general para el sistema. Las funciones técnicas globales del sistema son:

- ✓ **Funciones que satisfacen la necesidad:**
  - Reconocimiento y mapeo del entorno.
  - Planificación de trayectorias.
  - Seguimiento de trayectorias.
- ✓ **Funciones que restringen:**
  - Limitación de los sensores LiDAR a condiciones simuladas específicas.
  - Restricciones cinemáticas del robot oruga que afectan su capacidad de maniobrar en espacios reducidos.
  - Capacidad computacional del entorno de simulación y hardware asociado.
- ✓ **Funciones que complementan:**
  - Interfaz de simulación en Unreal Engine 5 para la visualización y análisis de los movimientos del robot.
  - Comunicación en tiempo real con Simulink para la transmisión de comandos de control.

c) **Definición de las funciones técnicas especificas:** se descomponen las funciones globales en funciones técnicas específicas que solucionen el problema. Las funciones técnicas especificas son:

- ✓ **Sistema de percepción:** Procesamiento de datos LiDAR para el mapeo del entorno (LOAM).
- ✓ **Sistema de planificación de trayectorias:** Algoritmo A* Híbrido para generar rutas óptimas.
- ✓ **Sistema de control y seguimiento:** Algoritmo Pure Pursuit para seguir las trayectorias planificadas.
- ✓ **Simulación del entorno:** Desarrollo de un invernadero virtual en Unreal Engine 5.
- ✓ **Comunicación:** Interfaz entre Unreal Engine 5 y Simulink para la transmisión de comandos.

d) **Definición de las especificaciones de diseño:** esto incluye la selección de los componentes y la identificación de la zona específica en la que se implementará el sistema. Las especificaciones de diseño incluyen:

- ➢ El robot deberá estar equipado con un sensor LiDAR de alta precisión para el mapeo en 3D del entorno.
- ➢ El algoritmo A* Híbrido deberá ser capaz de generar trayectorias suaves y libres de obstáculos, respetando las restricciones de movimiento del robot oruga.
- ➢ La simulación debe considerar un entorno representativo de un cultivo de tomate en invernadero, incluyendo obstáculos físicos como plantas y estructuras.

**Etapa 2 – Diseño del dominio especifico (Domain-Specific Design):**

Durante se realizan tareas de diseño especificas en paralelo que permitan el pleno desarrollo del proyecto.

**Subetapas Especificas del proyecto:**

**(a) Creación del entorno de simulación:** El entorno de simulación constará de los siguientes elementos:

- **Simulación del robot:** Se modela el robot móvil tipo oruga simplificado en el motor de desarrollo Unreal Engine conectado a Simulink para efectuar las tareas de control. El robot está equipado con un sensor LiDAR virtual que imitará las características y comportamiento del sensor LiDAR real que se planea utilizar.
- **Simulación de la sensórica:** El sensor LiDAR virtual genera datos de rango en un patrón de escaneo y se utiliza para construir un mapa del entorno y para localizar la posición del robot en tiempo real.
- **Simulación del entorno físico:** Modelado tridimensional básico del cultivo de tomate en un invernadero.

**(b) Implementación del sistema de localización (SLAM):** para esta implementación se ha seleccionado el método de odometría y mapeo LOAM para sensores LiDAR. La implementación constará de los siguientes pasos:

- **Seguimiento de una ruta manual y toma de datos:** el robot sigue una ruta creada manualmente del entorno, guardando continuamente las nubes de puntos captadas por el sensor LiDAR durante su desplazamiento. Una vez realizado este procedimiento, el robot se detiene para generar el mapeo offline.
- **Preprocesamiento de datos del LiDAR:** LOAM requiere la manipulación de los datos captados y almacenados por el sensor LiDAR para su procesamiento. Las lecturas del sensor LiDAR se segmentarán en una serie de subconjuntos de barridos planos y de borde, que serán utilizados para calcular la odometría y construir el mapa.
- **Odometría LiDAR:** Utilizando los datos preprocesados, LOAM calcula la odometría LiDAR. Esto implica estimar la posición y orientación del robot en tiempo real utilizando únicamente las lecturas del sensor LiDAR.
- **Mapeo LiDAR:** Paralelamente a la odometría, LOAM construye un mapa 3D del entorno utilizando las lecturas del sensor LiDAR.
- **Optimización y actualización del mapa:** La última etapa de la implementación de LOAM consiste en optimizar y actualizar el mapa 3D. Una vez generado el mapa 3D completo, se procede a convertirlo a un mapa de costos 2D para la planificación de trayectorias bidimensionales.

**(c) Implementación del sistema de generación de trayectorias (A* Híbrido):** la cual constara de los siguientes pasos clave:

- **Creación de la grilla de costos**: En primer lugar, a partir del mapa obtenido mediante LOAM, se crea una grilla de costos, donde cada celda representa un posible punto de la trayectoria.
- **Implementación del algoritmo A* Híbrido:** que busca la ruta más eficiente desde un punto de inicio hasta un punto final, considerando las restricciones de movimiento del robot móvil tipo oruga. Esto implica la generación de un árbol de rutas posibles, y la elección de la ruta óptima basada en una combinación de costo y distancia a la meta.

- **Optimización de la trayectoria:** con la ruta generada por el planificador, se procede a realizar una optimización utilizando el enfoque TEB, de tal forma que la nueva ruta optimizada sea más suave y tenga en cuenta la distancia a los obstáculos.

**(d) Implementación y parametrización del sistema de seguimiento de trayectoria utilizando Pure Pursuit:** el cual consta de los siguientes pasos:

- El **algoritmo Pure Pursuit** se utiliza para ajustar los movimientos del robot a las trayectorias previamente generadas.
- **Parametrización** para optimizar el seguimiento y minimizar errores de desviación en la trayectoria.

**Etapa 3 – Integración del sistema (System Integration):**

Los resultados relacionados a las tareas en paralelo de la etapa 2 se integran en esta etapa para crear el prototipo.

**Subetapas Especificas del proyecto:**

**(a) Integrar diseños de subsistemas en soluciones mecatrónicas:** Los subsistemas implementados (SLAM, generación de trayectorias, seguimiento de trayectorias, simulación) se integrarán para formar un sistema mecatrónico cohesivo. Este sistema será capaz de navegar de manera autónoma en el entorno simulado.

**(b) Identificar los parámetros óptimos de los diseños de los sistemas mecatrónicos:** Se llevarán a cabo ajustes finos en los parámetros de los subsistemas para garantizar un rendimiento óptimo. Esto incluye la precisión de la localización (SLAM), la suavidad de las trayectorias (A* Híbrido) y el seguimiento preciso (Pure Pursuit).

**(c) Seleccionar diseños de soluciones mecatrónicas:** Se evaluarán múltiples configuraciones de diseño, seleccionando las que ofrezcan el mejor rendimiento en términos de precisión, velocidad de cálculo y capacidad de adaptación a obstáculos.

**Etapa 4 – Validación y verificación del sistema (Assurance of properties):**

Esta última etapa tiene como objetivo asegurar que los resultados del sistema cumplan con la solución definida en la fase de diseño. Si el sistema necesita mejoras, el proceso de diseño se repite hasta tener un resultado satisfactorio.

**Subetapas Especificas del proyecto:**

**(a) Diseño y desarrollo de pruebas de validación y verificación:** se desarrollarán pruebas detalladas para validar y verificar cada uno de los subsistemas, así como el sistema mecatrónico completo. Estas pruebas evaluarán si los subsistemas y el sistema en su conjunto cumplen con las especificaciones de diseño y los requisitos funcionales definidos en la etapa 1.

**(b) Realización de las pruebas de validación y verificación:** se realizan las pruebas de validación y verificación en un entorno de desarrollo virtual que simule un invernadero de tomate. Se verifica la precisión del sistema SLAM, la eficiencia del algoritmo de generación de trayectorias, y la efectividad del algoritmo de seguimiento de trayectorias y evasión de obstáculos. Además, se validará la capacidad del sistema para operar de manera autónoma y navegar eficientemente en el entorno simulado.

**(c) Análisis de los resultados de las pruebas:** Se analizarán los resultados de las pruebas para determinar si el sistema cumple con todas las especificaciones y requisitos. Si se identifican problemas o deficiencias, se volverá a las fases anteriores para realizar ajustes y mejoras en el diseño del sistema.

**(d) Divulgación y sustentación.**

# 7. DISEÑO E IMPLEMENTACION.

## 7.1. Diseño del prototipo virtual del Robot Móvil tipo Oruga.

### 7.1.1. Modelo cinemático del Robot Móvil tipo Oruga.

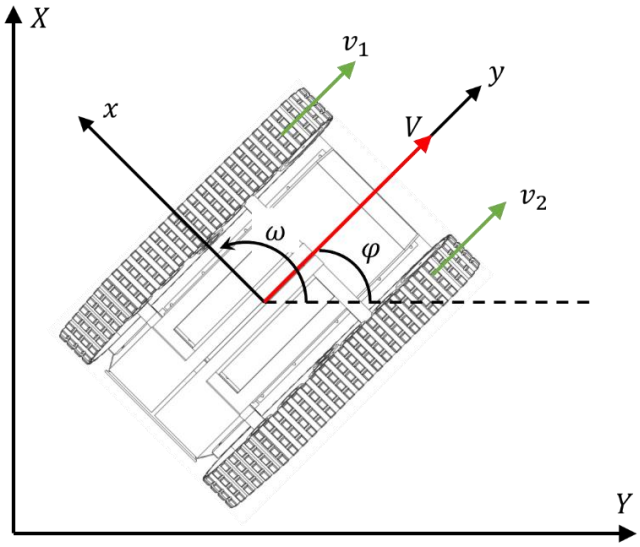



**Figura 34.** Movimiento cinemático de un robot móvil tipo oruga. **Fuente:** Autor.

El modelado de un robot móvil tipo oruga se desarrolla a base de los 3 grados de libertad que presenta el robot y su desplazamiento se define tal y como se muestra en la Figura 34. Para simplificar el modelo se supone que el robot se desplaza a bajas velocidades, por lo que es válido asumir que su desplazamiento lateral es insignificante. De esta forma se ha optado por utilizar un modelo basado en la cinemática de un sistema de locomoción diferencial, pero añadiendo el efecto del deslizamiento longitudinal presente en los robots móviles tipo oruga (González y otros, 2015). Por lo tanto, el modelo cinemático del robot puede escribirse como:

$$\dot{x} = V\,cos\varphi = \frac{v_1 + v_2}{2}\,cos\varphi \tag{16}$$

$$\dot{y} = V\,sin\varphi = \frac{v_1 + v_2}{2}\,sin\varphi \tag{17}$$

$$\omega = \frac{v_2 - v_1}{L} \tag{18}$$

En donde:

- $\dot{x}, \dot{y}$ son las componentes $x$ y $y$ de la velocidad del robot $(V)$ en el centro de masa.
- $v_1$ y $v_2$ son las velocidades de las ruedas izquierda y derecha, respectivamente.
- $\omega$ es la velocidad angular del robot en el centro de masa.
- $L$ es la distancia entre las dos orugas.
- $\varphi$ es el ángulo que hace el robot respecto a la horizontal.

Ahora, la ecuación que define la velocidad de las ruedas es:

$$v_1 = r\omega_1 \tag{19}$$

$$v_2 = r\omega_2 \tag{20}$$

En donde $r$ es el radio del engrane que transmite el movimiento a cada oruga del robot. Al introducir las ecuaciones (**19**) y (**20**), al reemplazarlas en las ecuaciones iniciales (**16**), (**17**) y (**18**), y al despejar las velocidades angulares de las ruedas en términos de la velocidad lineal $V$ y angular del robot $\omega$ en el centro de masa, obtenemos que:

$$\omega_1 = \frac{2V - \omega L}{2r} \tag{21}$$

$$\omega_2 = \frac{2V + \omega L}{2r} \tag{22}$$

### 7.1.2. Modelado del Robot Móvil tipo Oruga en Unreal Engine.

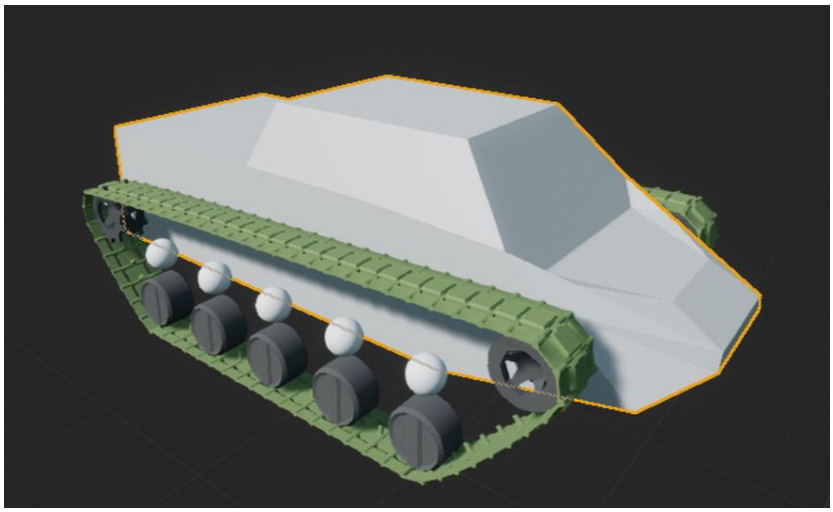


**Figura 35.** Prototipo virtual del Robot Móvil tipo Oruga. **Fuente:** Autor.

Para el desarrollo del prototipo virtual del robot con orugas en el entorno de Unreal Engine, que se puede visualizar en la Figura 35, se optó por hacer uso de la librería de código abierto "UE_Tracked_Vehicles", la cual permite la creación de vehículos tipo orugas personalizados y otros tipos de vehículos dentro de Unreal Engine de una forma realista y fácil de implementar, al contar con blueprints prestablecidos y un kit completo de modelado de vehículos con orugas en donde simplemente se deben integrar correctamente los diferentes elementos disponibles para la creación del vehículo (BoredEngineer, 2015). Los vehículos creados con esta librería tienen las siguientes características:

- Tracción del vehículo basada en la fricción y el torque transmitidos desde el motor, con un conjunto de coeficientes de fricción.
- Sistema de amortiguamiento resorte-amortiguador simple.
- Sistema de frenado simple basado en la fricción.
- Sistema de dirección diferencial.

- Dos conjuntos de fricción de rodadura, “constante”, que representa la pérdida de energía debido a la interacción de las partes mecánicas y “dinámica”, que depende linealmente de la velocidad y es muy específica de los vehículos de orugas.
- Implementación de dos animaciones para los eslabones. La primera utiliza una malla simple con textura animada. La segunda utiliza una malla estática instanciadas para representar los enlaces individuales.

#### 7.1.2.1. Configuración del blueprint maestro y la estructura base.

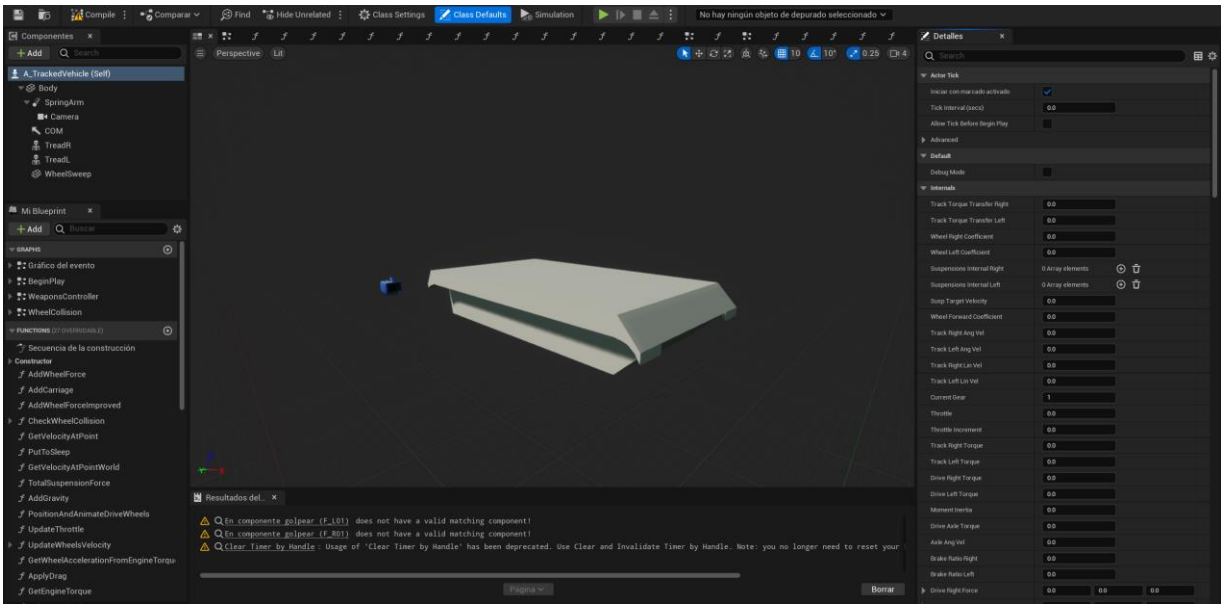


**Figura 36.** Diseño del cuerpo del robot en Unreal Engine. **Fuente:** Autor.

El diseño del robot se basa en el blueprint maestro de la librería “UE_Tracked_Vehicles”, titulado “A_TrackedVehicle”, el cual sirve como la plantilla base de esta librería, ya que contiene todas las funciones y parámetros necesarios para el funcionamiento del vehículo. El blueprint maestro es replicado y personalizado creando un "child blueprint" basado en el blueprint original, lo que permite heredar todas las funciones principales.

La Figura 36 presenta la ventana de gráficos del blueprint maestro, en la cual se visualiza la malla estática correspondiente a un chasis estándar básico proporcionado por la librería, con todos sus parámetros inicializados en blanco. Este blueprint permite la importación de mallas estáticas personalizadas o el uso de una de las tres mallas predeterminadas incluidas en la librería, facilitando así el desarrollo del prototipo robótico. Para esta configuración específica, se seleccionó la malla "Ripsaw_Body" debido a su diseño frontal optimizado para la incorporación de un sensor LiDAR. Las dimensiones de esta malla son 513 x 167 x 143 milímetros, y su estructura puede observarse en la Figura 37, mientras que sus parámetros físicos están detallados en la Tabla 3. Las principales modificaciones realizadas al blueprint maestro fueron las siguientes:

- Se sustituye el cuerpo del vehículo por un modelo estático diferente.
- Se ajusta la masa del vehículo, dividiéndola entre las diferentes partes del mismo, para garantizar que el peso total del vehículo esté correctamente calculado.

- Se cambia el sistema de entrada de datos para permitir el control del vehículo desde Simulink, en vez de la entrada de datos de movimiento a través del teclado.

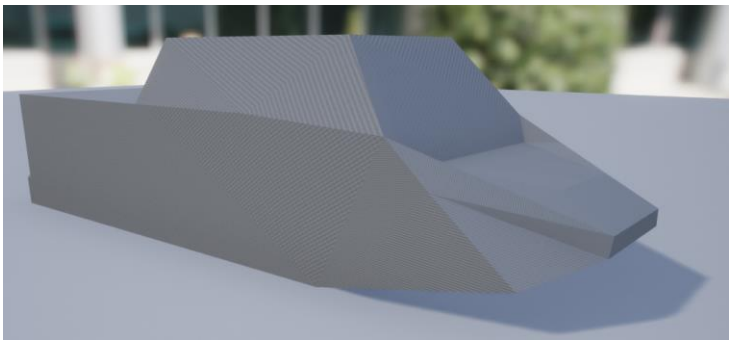


**Figura 37.** Estructura del cuerpo del robot utilizando la malla “Ripsaw_Body”. **Fuente:** Autor.

**Tabla 3.** Parámetros fiscos del chasis del robot. **Fuente:** Autor.

| Parámetros | Valor |
|---|---|
| Masa Chasis | 15 kg |
| Amortiguación Lineal | 0.01 |
| Amortiguación angular | 0 |

### 7.1.2.2. Configuración de las ruedas.

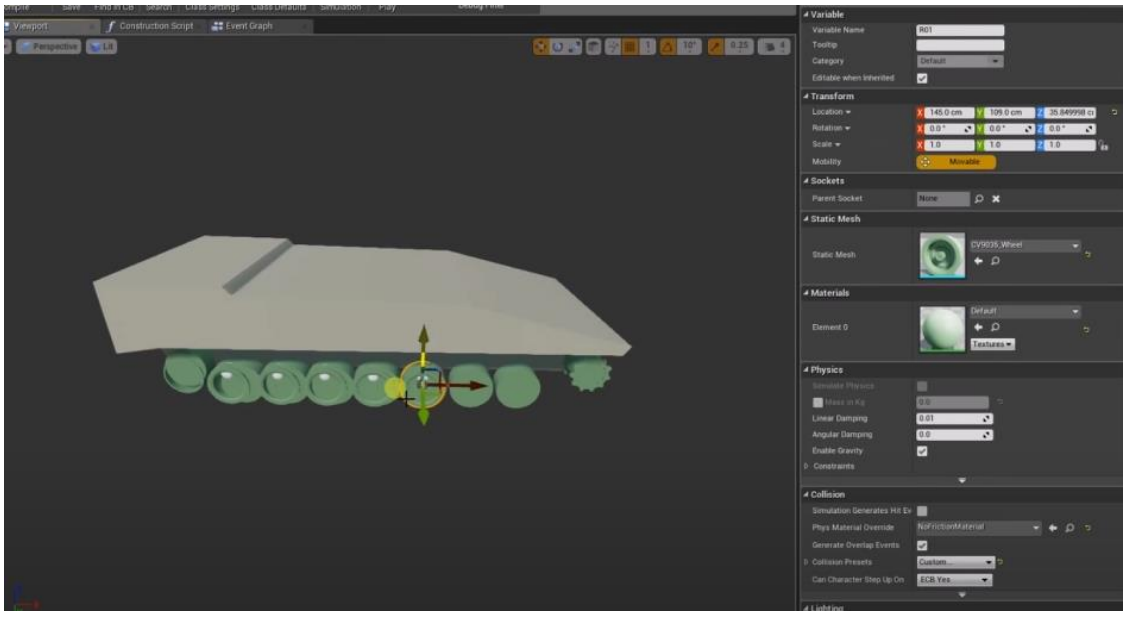


**Figura 38.** Configuración de las ruedas del robot. **Fuente:** Autor.

Una vez configurado el cuerpo del vehículo, se pasa a la configuración de las ruedas. Para ello, se seleccionan los modelos correspondientes a las ruedas y se les asignan propiedades específicas. El proceso incluye:

- Selección y modificación del modelo estático de las ruedas. Para este caso se seleccionó el modelo de ruedas correspondientes al modelo “Ripsaw”, coincidiendo con el tipo de cuerpo seleccionado para el chasis.
- Configuración de la dirección de rotación de las ruedas para asegurar que las del lado izquierdo y derecho giren en direcciones opuestas.
- Escalado de las ruedas para garantizar que encajen con el diseño del vehículo.
- Ajuste de las animaciones y dirección de rotación para sincronizarse correctamente con el movimiento del vehículo.

La librería “UE_Tracked_Vehicles” cuenta con un modelo de referencia de baja resolución para ayudar a alinear las ruedas en las posiciones correctas, como el que se puede observar en la Figura 40.

Un aspecto importante del diseño es asegurarse de que las ruedas tengan configuradas sus colisiones y radios correctamente. El radio de colisión se utiliza para evitar que el vehículo se hunda en el terreno, y este radio se ajusta según el tamaño de las ruedas y la oruga. El radio de colisión debe incluir las dimensiones de la oruga para garantizar que interactúe correctamente con el terreno. El modelo de referencia de la librería implementada cuenta con un radio de colisión ideal, el cual puede observarse en la Figura 39 y el cual sirvió de punto de partida para la asignación de la posición de las ruedas de suspensión.

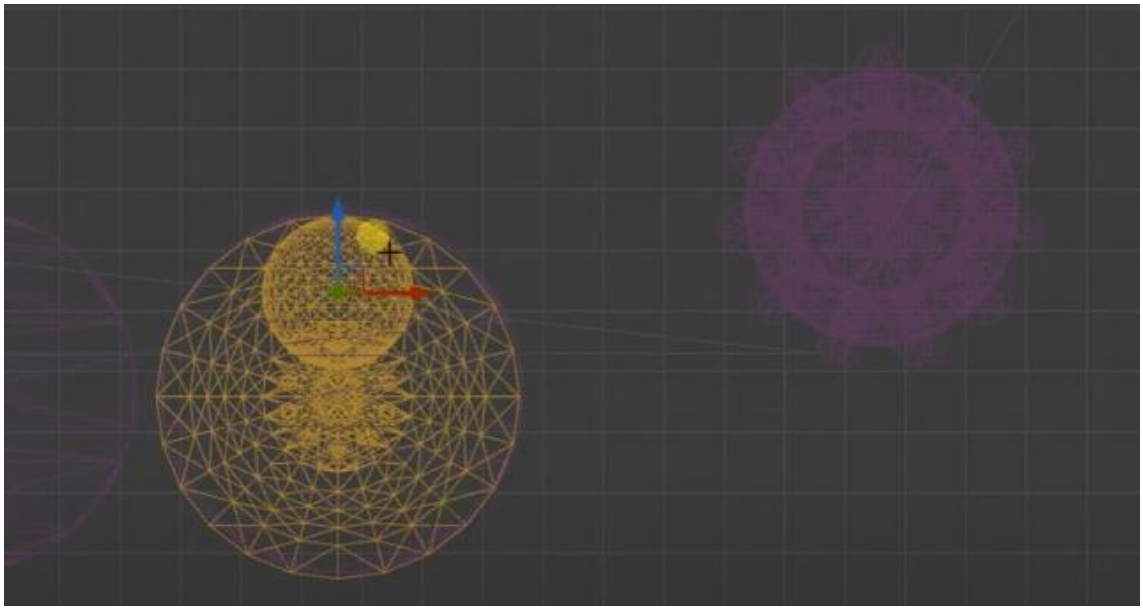
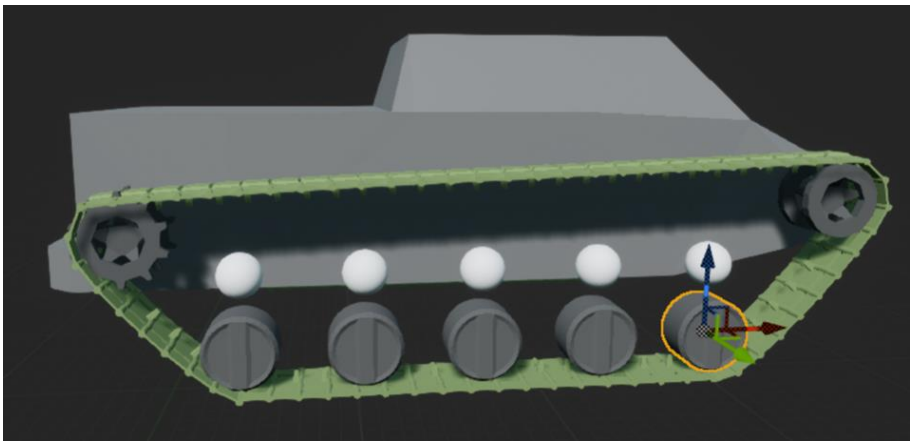


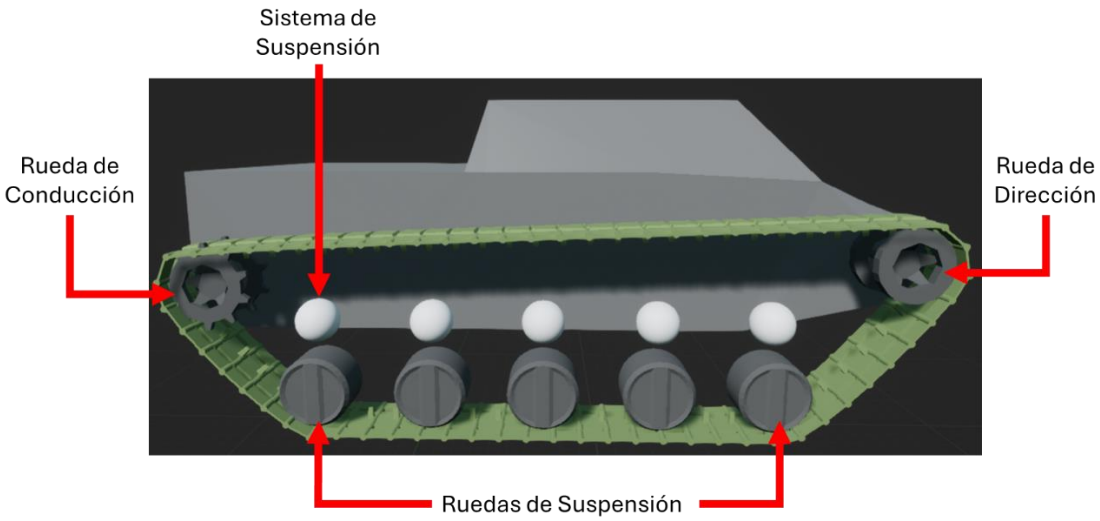
**Figura 39.** Configuración del radio de giro usando el modelo de referencia. **Fuente:** Autor.

En la Figura 40 podemos observar la distribución final de las 7 ruedas que conforman el sistema tipo oruga del robot, el cual se puede subdividir en los siguientes subsistemas:

- **Ruedas de suspensión:** Hay cinco ruedas en la parte inferior que están en contacto directo con la oruga de cada lado. Están distribuidas equidistantemente a lo largo de la base de la estructura del vehículo. Estas ruedas se encargan de distribuir el peso del vehículo y ayudar en el desplazamiento sobre la oruga.



(a)




(b)

**Figura 40.** Configuración del sistema de ruedas y suspensión del robot móvil tipo oruga. (a) Configuración del sistema de rudas. (b) Partes del sistema de suspensión y movimiento de las orugas del robot. **Fuente:** Autor.

- **Rueda de conducción:** Esta rueda está conectada al motor y es responsable de mover la oruga. Al girar, impulsa toda la oruga a lo largo del sistema, generando tracción y propulsión para que el vehículo avance o retroceda.

- **Rueda de dirección:** Aunque no está conectada al motor, su función es guiar y mantener la tensión de la oruga. Ayuda a que la oruga se mantenga en su lugar y facilite el giro y la dirección del vehículo.

- **Sistema de suspensión:** este sistema es representado visualmente por las esferas blancas, como puede observarse a detalle en la Figura 40 (b). Estas esferas representan un resorte-amortiguador simple e incluye toda la física correspondiente a este sistema. Este sistema se explora a detalle en la siguiente subsección.

#### 7.1.2.3. Configuración del sistema de suspensión.

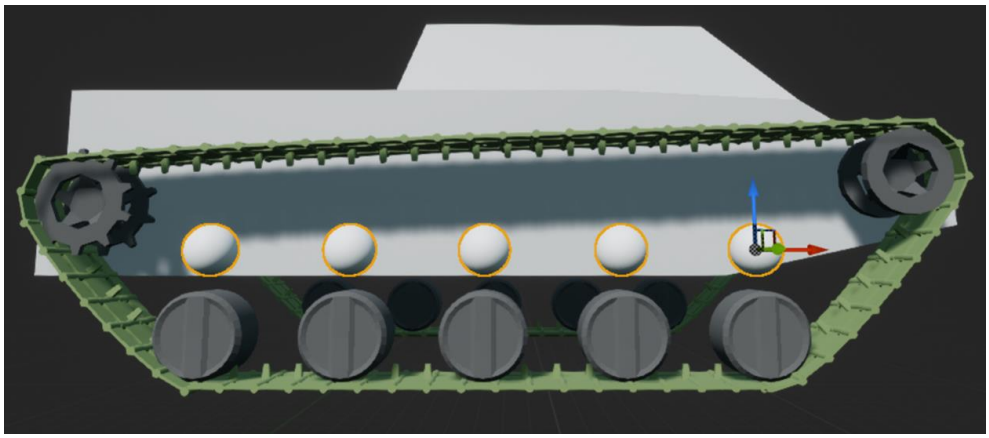


**Figura 41.** Configuración del sistema de suspensión de las orugas**. Fuente:** Autor.

El sistema de suspensión es un elemento crucial en el comportamiento del vehículo oruga, ya que permite que las ruedas se adapten al terreno mientras el vehículo se desplaza. Bajo el modelo de referencia de baja resolución, se alinean las ruedas de suspensión a su respectivo sistema resorte amortiguador, representando por las esferas blancas que se pueden visualizar en la Figura 41. En este tipo de sistemas, la suspensión cumple un papel fundamental para mejorar el rendimiento y la comodidad del vehículo, especialmente en terrenos accidentados. Este sistema sigue un principio de diseño simplificado para simular los mecanismos reales, pero sin mostrar las conexiones visibles a las ruedas. De esta forma se logra optimizar el rendimiento de la simulación evitando sobrecargar el diseño visual, enfocando estos recursos a la animación del movimiento de los eslabones que conforman las orugas, lo que le da más realismo a la simulación.

Las ruedas y suspensiones se duplican para completar el sistema de suspensión del vehículo, asegurando que las configuraciones de cada lado del vehículo sean coherentes. Cada elemento de suspensión tiene parámetros como la fuerza de resorte y la fuerza de amortiguación, que pueden ajustarse para optimizar el comportamiento del vehículo.

Para lograr una suspensión completamente funcional, se deben modificar algunas funciones específicas dentro del blueprint. Se inicia accediendo al blueprint del vehículo y seleccionando la sección de la suspensión. Cada lado (derecho e izquierdo) tiene cinco elementos en el arreglo de la suspensión. Estos cinco elementos corresponden a las cinco ruedas del vehículo, y se deben ajustar otros dos elementos para incorporar completamente las ruedas traseras. Se resalta la importancia de la función de constructor y los "handlers" de la suspensión, que permiten modificar el comportamiento de cada lado del vehículo.

Inicialmente, las ruedas no están configuradas para girar, por lo que se realizó una configuración dentro del blueprint para añadirles esta función, tal y como puede observarse en la siguiente imagen:

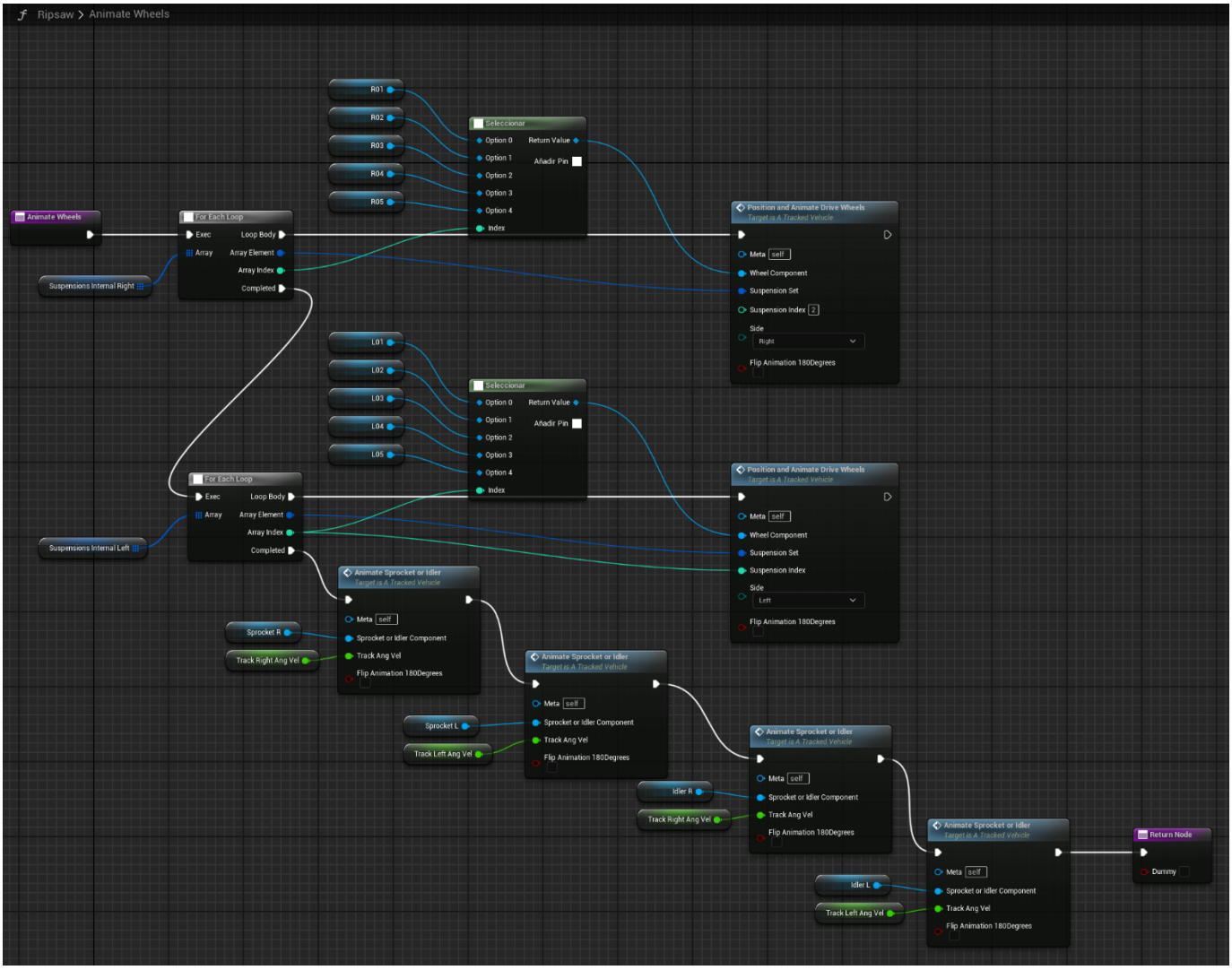



**Figura 42.** Blueprint de la función para añadir la animación del movimiento de las ruedas. **Fuente:** Autor.

En esta función del Blueprint se usan dos bucles separados, uno para la suspensión interna derecha (*Suspensions Internal Right*) y otro para la izquierda (*Suspensions Internal Left*). Esto permite iterar sobre los componentes de suspensión de cada lado del vehículo de manera independiente, lo que es crucial cuando se manejan ruedas en configuraciones asimétricas o donde se requiere un control diferenciado de la animación entre ambos lados del vehículo.

Cada rueda es posicionada y animada a través de nodos que combinan la función de *Position and Animate Drive Wheels.* Estos nodos reciben parámetros como el componente de la rueda, el índice de suspensión y el lado del vehículo (derecha o izquierda), permitiendo que el sistema diferencie el comportamiento según el lado al que pertenece la rueda. Los nodos adicionales para la animación de *Sprocket or Idler* permiten sincronizar la rotación de los engranajes con el movimiento de las ruedas. Estos nodos son ejecutados para ambos lados del vehículo, garantizando una animación fluida en la transmisión de movimiento.

#### 7.1.2.4. Configuración de los eslabones que conforman las orugas.

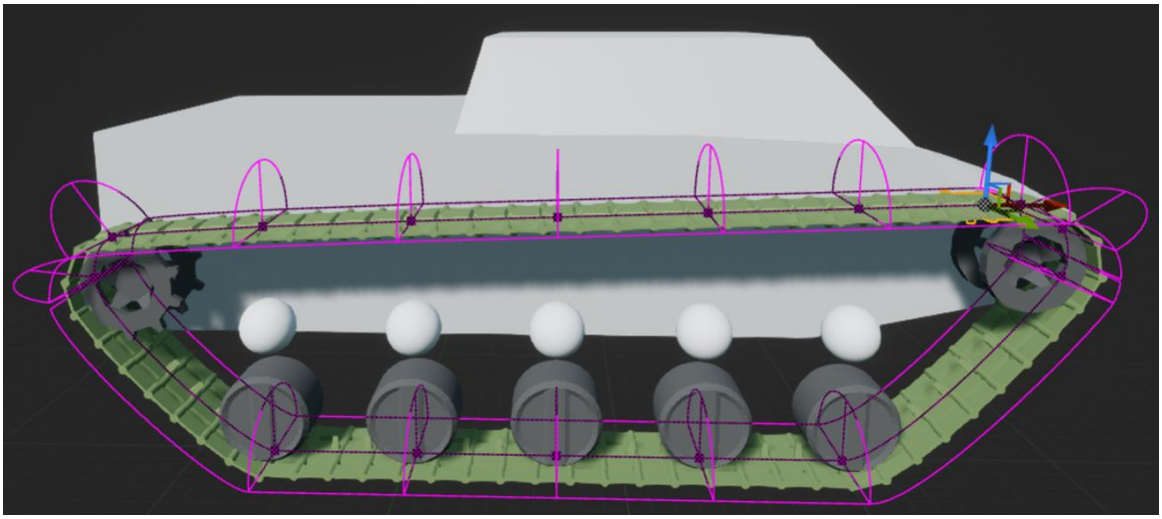


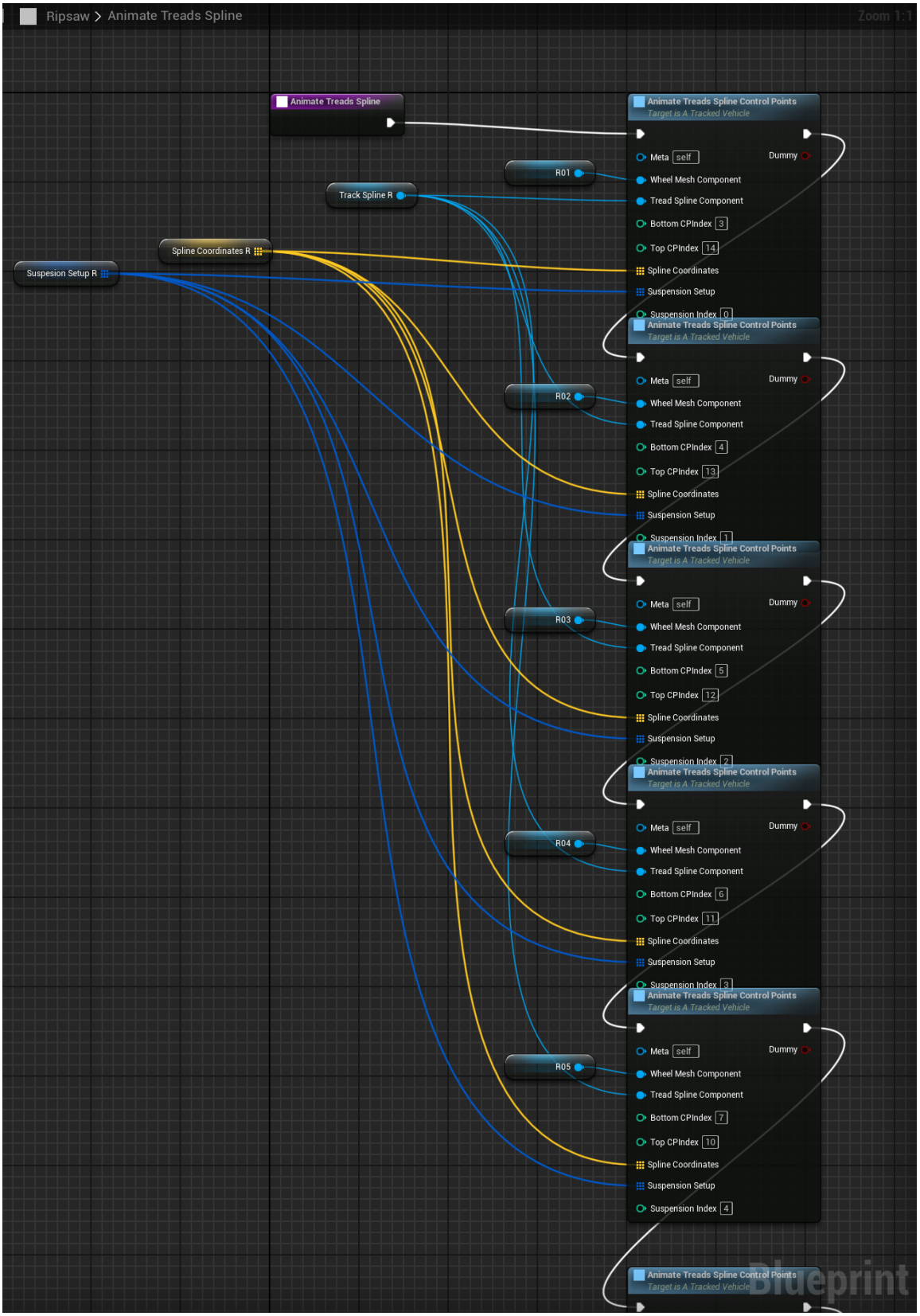
**Figura 43.** Configuración de la animación de los eslabones que conforman la oruga. **Fuente:** Autor.

El proceso de implementación de los eslabones que conforman las ruedas tipo oruga se ha realizado mediante el uso de "*spline*", una herramienta de Unreal Engine que permite modelar de manera precisa las trayectorias geométricas de los eslabones del vehículo.

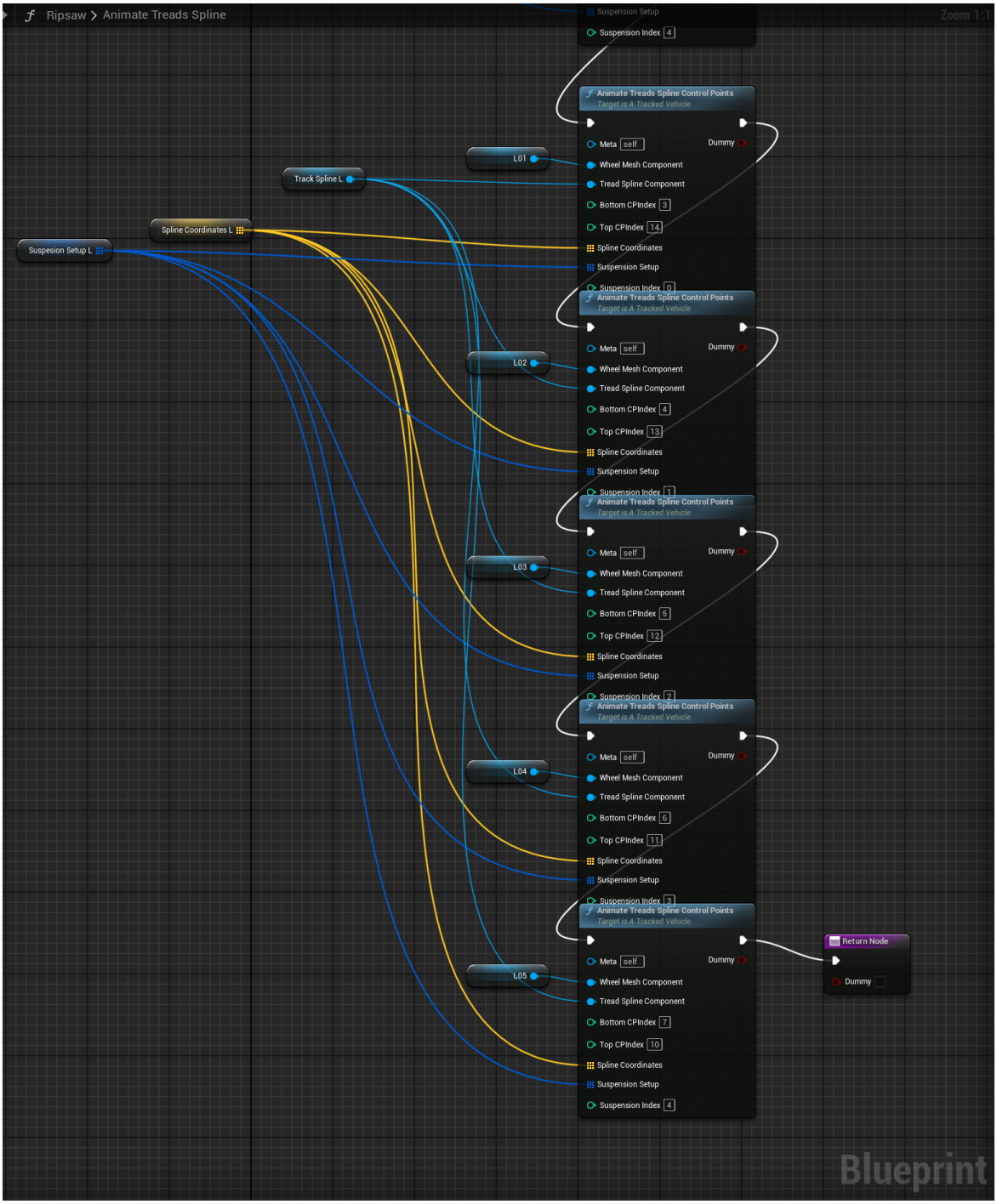
A partir de un blueprint maestro, se procede a configurar los puntos *spline* que definirán la trayectoria de la pista. Para ello, se debe ajustar manualmente la posición de cada punto *spline*, lo que resulta fundamental para alinear correctamente los eslabones con las ruedas del vehículo. La configuración de los puntos *spline* es un proceso preciso, y se puede determinar visualmente la ubicación inicial y final de la *spline* a fin de evitar bucles cerrados.

Una vez identificados los puntos *spline*, se procede a modificar sus coordenadas manualmente, utilizando el blueprint del vehículo. Cada punto de *spline* representa una posición en el espacio tridimensional, la cual se ajusta en función de la ubicación de las ruedas del vehículo. Para facilitar este proceso, se han utilizado objetos de referencia (en este caso, cubos) que permiten medir las coordenadas de los puntos deseados. Posteriormente, estas coordenadas se introducen en el panel de blueprint, ajustando el desplazamiento y la rotación de cada *spline* para asegurar la correcta alineación con las ruedas.

Luego, se configurar el comportamiento de la *spline* en relación con la suspensión del vehículo. Para ello, se ha modificado la función responsable de animar los puntos *spline* en función del movimiento de las ruedas. Cada rueda del vehículo está asociada a un punto *spline* específico, de modo que, cuando la suspensión de la rueda se ajusta, también lo hace el punto *spline* correspondiente. Este vínculo entre los puntos *spline* y las ruedas asegura que los eslabones sigan el movimiento del vehículo de manera fluida y coherente. La función del blueprint que controla el movimiento de los eslabones puede observarse en la siguiente imagen:


Ripsaw > Animate Treads Spline
Zoom 1:1
Animate Treads Spline
Track Spline R
Spline Coordinates R
Suspesion Setup R
R01
R02
R03
R04
R05
Animate Treads Spline Control Points
Target is A Tracked Vehicle
Meta self
Dummy
Wheel Mesh Component
Tread Spline Component
Bottom CPIndex 3
Top CPIndex 14
Spline Coordinates
Suspension Setup
Suspension Index 0
Bottom CPIndex 4
Top CPIndex 13
Suspension Index 1
Bottom CPIndex 5
Top CPIndex 12
Suspension Index 2
Bottom CPIndex 6
Top CPIndex 11
Suspension Index 3
Bottom CPIndex 7
Top CPIndex 10
Suspension Index 4
Blueprint

(a)




(b)

**Figura 44.** Blueprint de la función *Spline* para la animación de los eslabones de la oruga. (a) Parte 1. (b) Parte 2.
**Fuente:** Adaptado de (BoredEngineer, 2015)

El blueprint se estructura en base a dos componentes principales: el *spline* que define la trayectoria de la oruga y los puntos de control que interactúan directamente con la suspensión del vehículo. Se inicia con la definición del *spline* que guía la trayectoria de las orugas y la configuración de suspensión, identificada como *Suspension Setup*. Este setup establece los

parámetros de suspensión, tales como los índices de control correspondientes a cada componente del vehículo, lo cual es clave para sincronizar los puntos de control del *spline* con el movimiento de las ruedas.

Cada segmento de la oruga es asignado a un *spline* específico que está enlazado directamente con la configuración de suspensión. En este sentido, cada instancia del *spline* está conectada a un *Track Spline Component* que recibe las coordenadas correspondientes a las ruedas (índices superiores e inferiores), garantizando una animación fluida y realista del movimiento de las orugas.

**Control de puntos de *Spline* y de suspensión.**

En cada nodo de control de la suspensión, se asignan componentes que representan tanto la malla de las ruedas (*Wheel Mesh Component*) como las coordenadas específicas del *spline* (*Spline Coordinates*). El sistema está diseñado de tal forma que cada rueda, identificada por su propio índice, controla tanto el desplazamiento vertical como la rotación que se transfiere a la oruga. Estos datos son alimentados en el nodo *Animate Treads Spline Control Points*, que se encarga de interpretar los valores y traducirlos en movimiento para el tread (oruga).

Este proceso es repetido para múltiples puntos de control, los cuales están organizados jerárquicamente en el blueprint para asegurar que todos los componentes interactúen de manera sincronizada. A través de este enfoque, el sistema gestiona el comportamiento cinemático de cada rueda, con especial atención en la distribución de la suspensión a lo largo de la oruga, logrando que la animación sea precisa y acorde a las restricciones físicas del modelo.

Ya habiendo configurado correctamente cada uno de los subsistemas personalizables que permite esta librería, el blueprint maestro ya es capaz de simular correctamente el movimiento del robot, incluyendo el sistema de suspensión y la animación de las orugas.

### 7.1.3. Implementación de la sensórica en el robot.

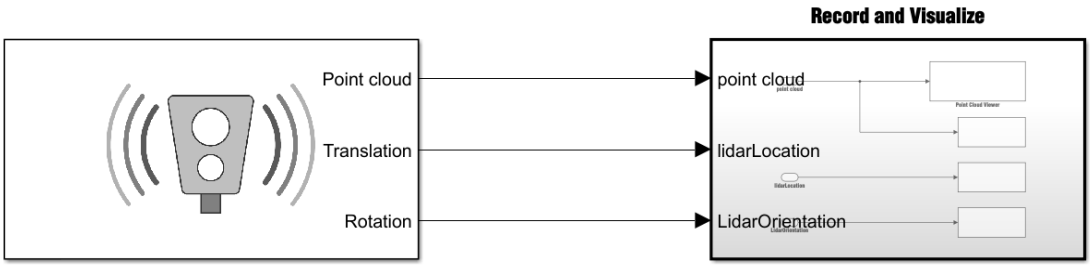
El sistema sensorial del robot está compuesto por dos dispositivos principales: un **sensor LiDAR** y un sistema de navegación global por satélite (GNSS, por sus siglas en inglés) integrado a un sensor de navegación inercial (INS, por sus siglas en inglés), que permiten tanto el mapeo del entorno como la localización precisa del robot en todo momento. La combinación de estos dos sistemas permite que el robot opere en el entorno simulado con datos de alta fidelidad, reproduciendo tanto los procesos de mapeo como la localización en condiciones similares a las de un entorno real. Para la implementación de ambos sensores dentro de Unreal Engine se optó por simular la respuesta de cada sensor utilizando los elementos disponibles en el entorno de desarrollo por medio de una comunicación directa y en tiempo real con Simulink. El proceso de simulación de cada sensor se detalla a continuación.

#### 7.1.3.1. Simulación del sensor LiDAR.

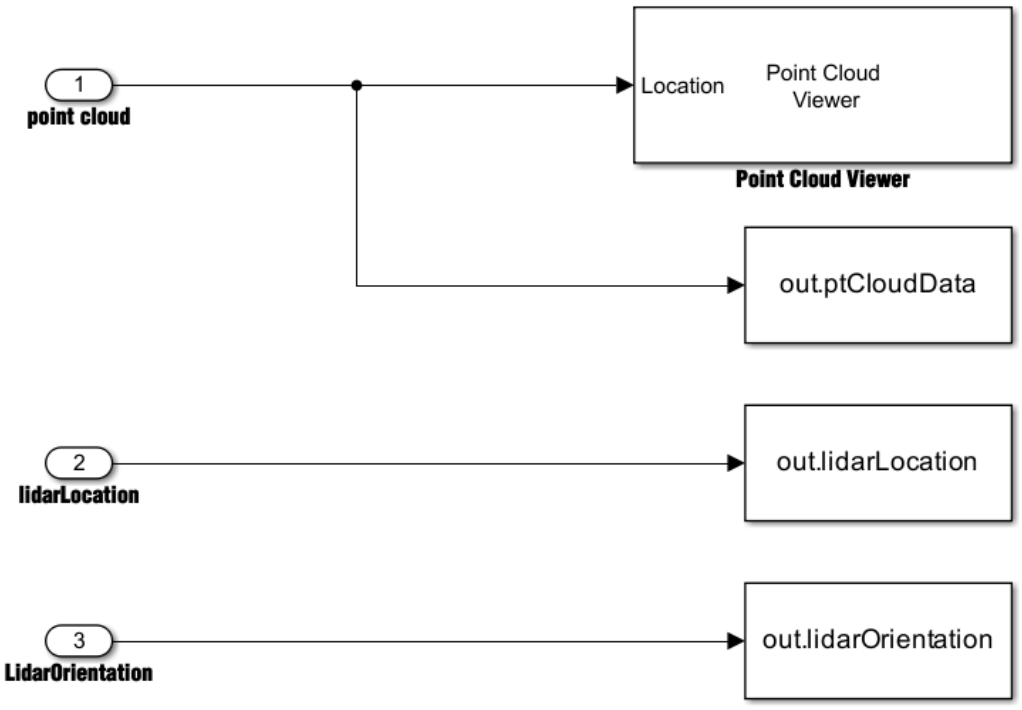
**Figura 45.** Implementación del sensor LiDAR en Unreal Engine. **Fuente:** Autor.

Para la simulación del sensor LiDAR, se implementó el actor predeterminado "*Sim3DLidar*", incluido en la librería "*MathWork Automotive Content*". Este actor emula el comportamiento del sensor en tiempo real, proporcionando información precisa sobre la topografía del entorno virtual y transmitiéndola directamente a Simulink mediante un identificador único asignado al sensor. El actor se posiciona por encima del robot y adopta la forma visual hipotética de una cámara para facilitar la visualización de su ubicación, como se muestra en la Figura 51. Cabe resaltar que esta forma visual será ocultada durante la simulación sin que ello afecte la recopilación de datos.

Durante la simulación, los datos recolectados por el sensor son recibidos en Simulink a través del bloque "*Simulation 3D Lidar*", el cual genera una nube de puntos basada en el campo de visión y la resolución angular especificados, a una tasa de muestreo predefinida. La nube de puntos obtenida se almacena en el espacio de trabajo de Matlab para su posterior uso en el algoritmo LOAM en modo offline y se envía a una función encargada de la visualización continua de la nube de puntos en tiempo real. El diagrama de bloques del proceso de simulación y visualización del sensor LiDAR se ilustra en la Figura 46.




(a)




(b)

**Figura 46.** Diagrama de bloques de simulación del sensor LiDAR en Simulink. (a) Diagrama de bloques principal. (b) Estructura interna del subsistema "*Record and Visualize*". **Fuente:** Autor.

#### 7.1.3.2. Simulación del sensor GNSS/GPS integrado con un INS.

Un sensor GNSS (Sistema Global de Navegación por Satélite) es un conjunto de tecnologías que permiten determinar con precisión la ubicación geográfica de un objeto en la superficie terrestre mediante el uso de satélites, proporcionando un posicionamiento absoluto de dicho objeto. Los sensores GNSS tienen una precisión que varía entre 2 y 5 metros el 95% del tiempo. Sin embargo, los vehículos autónomos suelen requerir una precisión del orden de los 2 decímetros, con la garantía de que el error sea inferior al ancho del carril por el que se desplazan. Para alcanzar este nivel de precisión, es necesario recurrir a fuentes externas de corrección que permitan reducir el margen de error al nivel requerido (Thirsk, 2024).

Un enfoque común para mejorar la precisión es combinar la lectura del sensor de posicionamiento absoluto GNSS con un sensor de posicionamiento relativo, como el INS (Sistema de Navegación Inercial). El sistema INS emplea sensores de movimiento y rotación para calcular la posición, orientación y velocidad relativas de un objeto. Estos sensores tienen una mayor precisión y no dependen de señales externas; sin embargo, tienden a sufrir de deriva acumulativa. Este fenómeno ocurre debido a pequeños errores en la medición de la aceleración y la velocidad angular, que se integran en errores progresivamente mayores en la velocidad y, finalmente, en la posición. Además, el INS es susceptible al ruido blanco, lo que puede introducir más inexactitudes. La fusión de sensores permite combinar la salida de múltiples sensores para limitar la acumulación de errores y mitigar las interrupciones del GNSS, mejorando significativamente la precisión del sistema.

En el entorno simulado, las condiciones ideales permiten conocer la posición absoluta y precisa del robot en todo momento. No obstante, esta situación reduce significativamente el realismo de

la simulación, lo que compromete su validez para pruebas más cercanas a las condiciones del mundo real. Por esta razón, se decidió emular la respuesta de un sistema de navegación inercial (INS) fusionado con un Sistema Global de Navegación por Satélite (GNSS), utilizando el bloque de Simulink denominado "Sensor INS". Este bloque genera señales de posición, velocidad y orientación, las cuales están afectadas por ruido blanco, simulando así las imprecisiones típicas de un sistema real. La magnitud de las perturbaciones se modela a través de un parámetro de precisión definido como la desviación estándar del ruido. El sistema de simulación del sensor de posición y orientación implementado en Simulink puede visualizarse en la Figura 47.

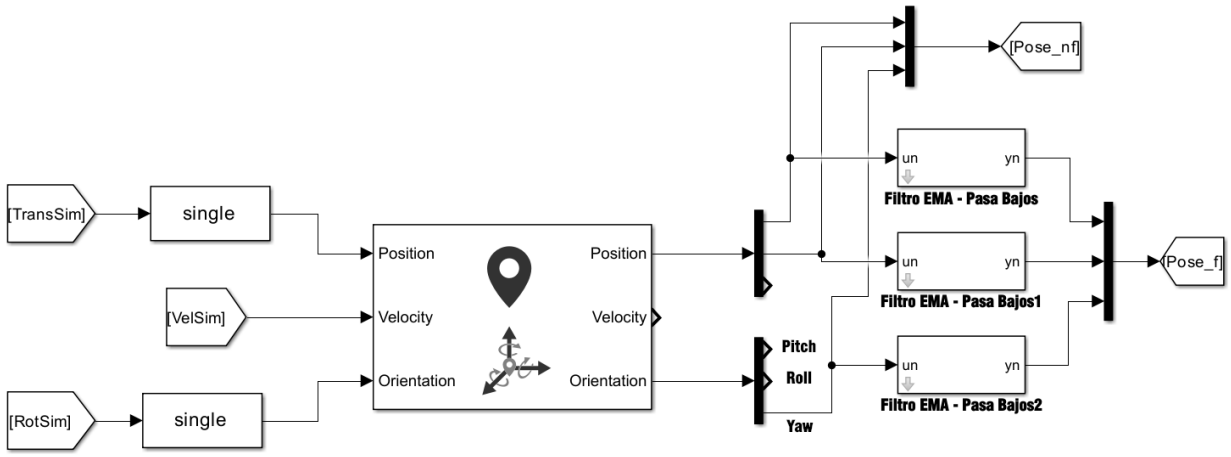



**Figura 47.** Simulación del sensor de posición en Simulink. **Fuente:** Autor.

Para cambiar el nivel de ruido presente en la salida del bloque INS se debe variar los parámetros de precisión para el ángulo de balanceo, cabeceo y guiño, así como la precisión de posición y velocidad. En este bloque, la precisión se define como la desviación estándar del ruido, entre más bajo el nivel de precisión menor ruido tendrá la salida. Los parámetros que se implementaron para la simulación del sensor de posición INS + GNSS están detallados en la Tabla 4.

**Tabla 4.** Parámetros del bloque Sensor INS. **Fuente:** Autor.

| Bloque Sensor INS | | |
|---|---|---|
| **Parámetros** | **Valor** | **Unidades** |
| Precisión de balanceo (eje X): | 0.35 | grados |
| Precisión de cabeceo (eje Y): | 0.75 | grados |
| Precisión de guiñada (eje Z): | 0.02 | grados |
| Precisión de posición: | [0.35, 0.75, 0.02] | m |
| Precisión de velocidad: | 0.005 | m/s |

### Filtrado Digital de los datos del sensor

Dado que el sistema de localización requiere mantener un bajo nivel de error para un funcionamiento óptimo, se implementó un filtro digital de paso bajo en la salida del bloque INS, con el fin de atenuar el ruido inherente generado por el sensor y mejorar la fidelidad de la señal procesada.

Para el diseño del filtro paso bajo se implementó un filtro discreto de *media móvil exponencial* (EMA, por sus siglas en inglés), el cual otorga más peso a los datos recientes al descontar los datos antiguos de manera exponencial, comportándose de manera similar a un filtro RC discreto paso bajo de primer orden (Hunter, 2024). La ecuación diferencial para un filtro de media móvil exponencial es:

$$y(n) = \alpha\, x(n) + (1 - \alpha) y(n - 1) \tag{23}$$

En donde:

- $y(n)$ es la salida del filtro en la $n$-esima muestra.
- $x(n)$ es la señal de entrada no filtrada en la $n$-esima muestra.
- $y(n-1)$ es la salida del filtro de la anterior muestra.
- $\alpha$ es la constante que establece la frecuencia de corta del filtro.

La constante $\alpha$ determina que tan agresivo es el filtro, pudiendo variar entre 0 y 1. A medida que $\alpha$ se aproxima a 0, el filtro se comporta de forma más agresiva, mientras que al acercarse a 1 permite el paso de una mayor cantidad de información. Este filtro se denomina exponencial debido a que la contribución ponderada de las entradas pasadas disminuye exponencialmente en función del tiempo, conforme estas se alejan en la secuencia temporal.

Al aplicar la transformada Z a la ecuación $(23)$ se obtiene la representación del filtro EMA en el dominio discreto.

$$Y(z) = \alpha X(z) + (1 - \alpha) z^{-1} Y(Z) \tag{24}$$

Al llevar a cabo la implementación digital del filtro EMA en un sistema discreto, es necesario derivar la ecuación en diferencias a partir de la ecuación discreta del filtro. Este proceso permite traducir el comportamiento continuo del filtro a un conjunto de ecuaciones que pueden ser programadas y ejecutadas de forma eficiente en un entorno digital, como un microcontrolador.

La ecuación discreta del filtro exponencial móvil se expresa como:

$$y[n] = \alpha x[n] + (1 - \alpha) y[n - 1] \tag{25}$$

Donde:

- $y[n]$ es la salida del filtro en el instante $n$.
- $x[n]$ es la entrada en el instante $n$.
- $y[n-1]$ es la salida del filtro en el instante anterior $n - 1$.

Para simular este filtro en Simulink y acercar la simulación a una implementación real en un microcontrolador, se utiliza un bloque de MATLAB Function, donde la ecuación de diferencias se programa directamente. El diagrama de bloques mostrado en la Figura 48 refleja la estructura del filtro en un sistema discreto, con la retroalimentación necesaria para calcular $y[n]$ a partir de $y[n-1]$.

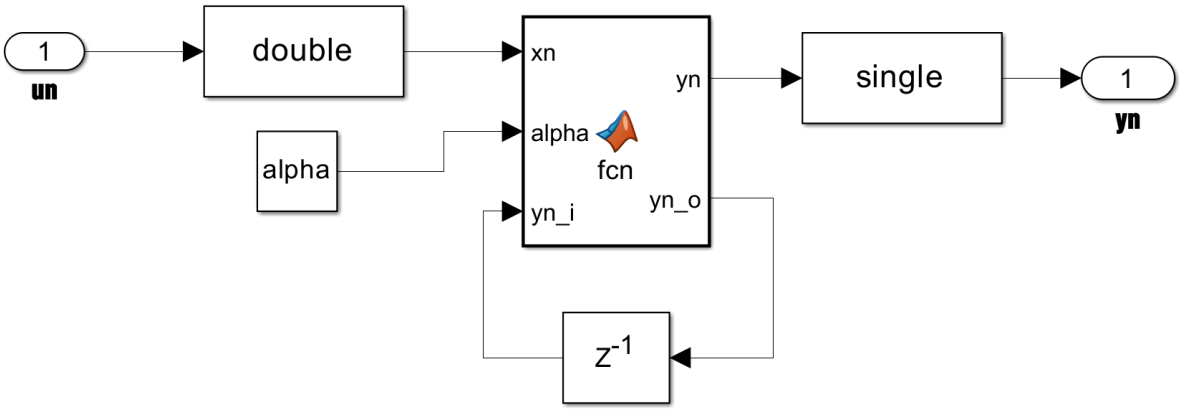



**Figura 48.** Diagrama de bloques del filtro EMA**. Fuente:** Autor.

La estructura mostrada en el diagrama de bloques también refleja fielmente la naturaleza de la retroalimentación inherente a los sistemas de filtrado digital. El código del filtro EMA dentro del bloque de Matlab Function es el siguiente:

```
function [yn, yn_o]  = fcn(xn, alpha, yn_i)
yn_1 = yn_i;                    % Se almacena la salida anterior
yn = xn*alpha + (1-alpha)*yn_1;  % Se calcula la salida actual
yn_1 = yn;                      % Se actualiza la salida anterior
yn_o = yn_1;                    %Se almacena el valor actualizado
```

### Resultados del proceso de filtrado

La Tabla 5 detalla los parámetros elementales de los 3 filtros EMA seleccionados para atenuar el ruido simulado por el bloque “Sensor INS”, incluyendo las condiciones iniciales de las cuales parte el robot. La Figura 49 ilustra la comparación entre la señal ruidosa producida por el sensor INS+GNSS y la salida filtrada mediante el EMA, mientras que la Figura 50 muestra la discrepancia entre la señal filtrada y la pose real del robot, provista por Unreal Engine. En ambas figuras se evidencia una reducción significativa del ruido introducido por el sensor, logrando que el valor de la señal filtrada se aproxime de manera considerable a la pose real del robot. El error promedio entre la pose real y la pose filtrada es de 0.0528% en $X$, 0.0813% en $Y$ y 5.099% en $\theta$.

**Tabla 5.** Parámetros de los filtros EMA. **Fuente:** Autor.

| **Coeficiente de Suavizado** | | |
|---|---|---|
| **Filtro** | **Valor α** | |
| Filtro en X | 0.2 | |
| Filtro en Y | 0.35 | |
| Filtro en θ | 0.65 | |
| **Condiciones Iniciales del Filtro** | | |
| **Parámetro** | **Valor** | **Unidades** |
| $X_0$ | 21.19 | Metros |
| $Y_0$ | 31.96 | Metros |
| $\theta_0$ | -1.571 | Radiantes |

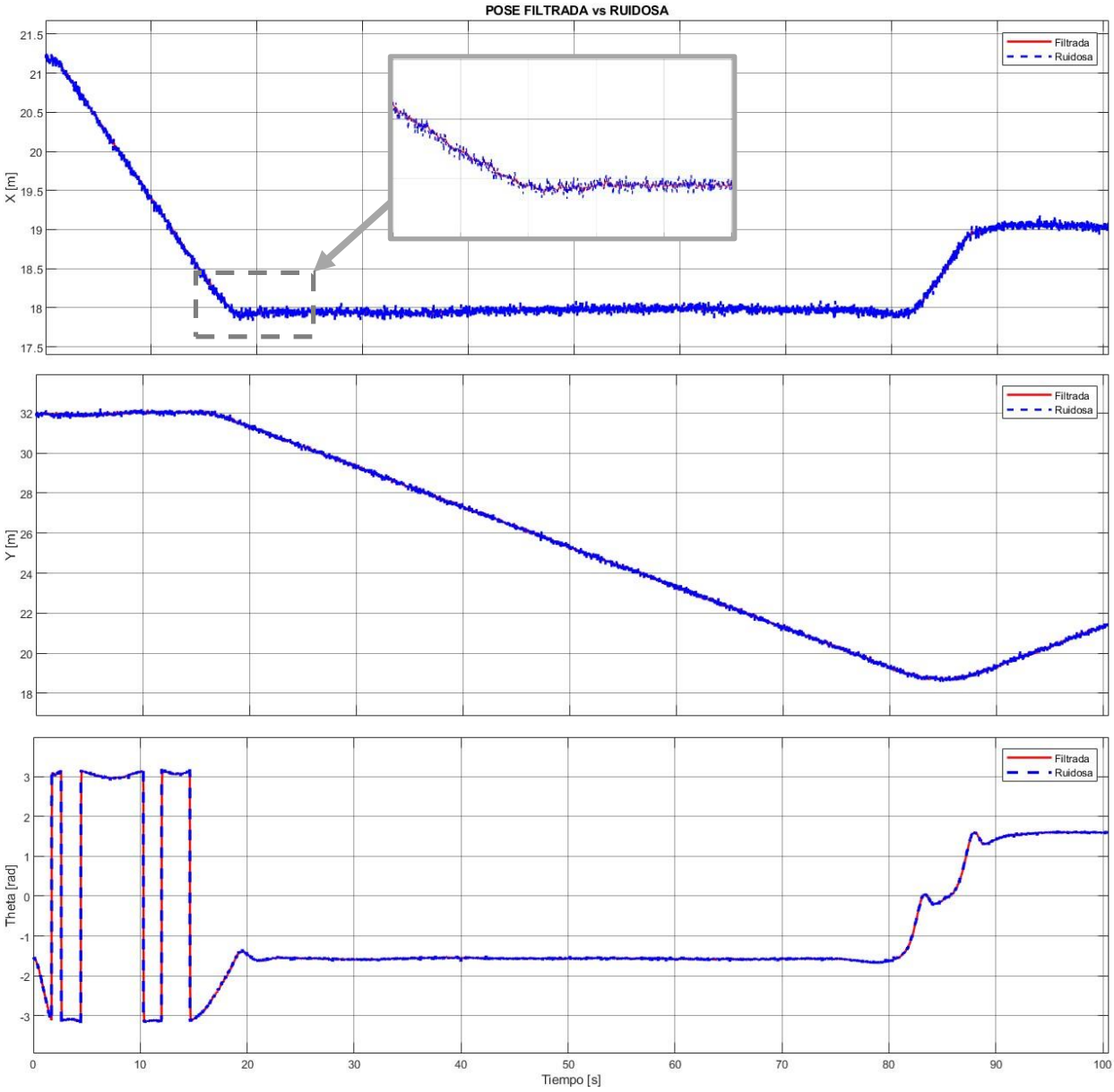



**Figura 49.** Pose $(x, y, \theta)$ Filtrada vs Ruidosa. **Fuente:** Autor.

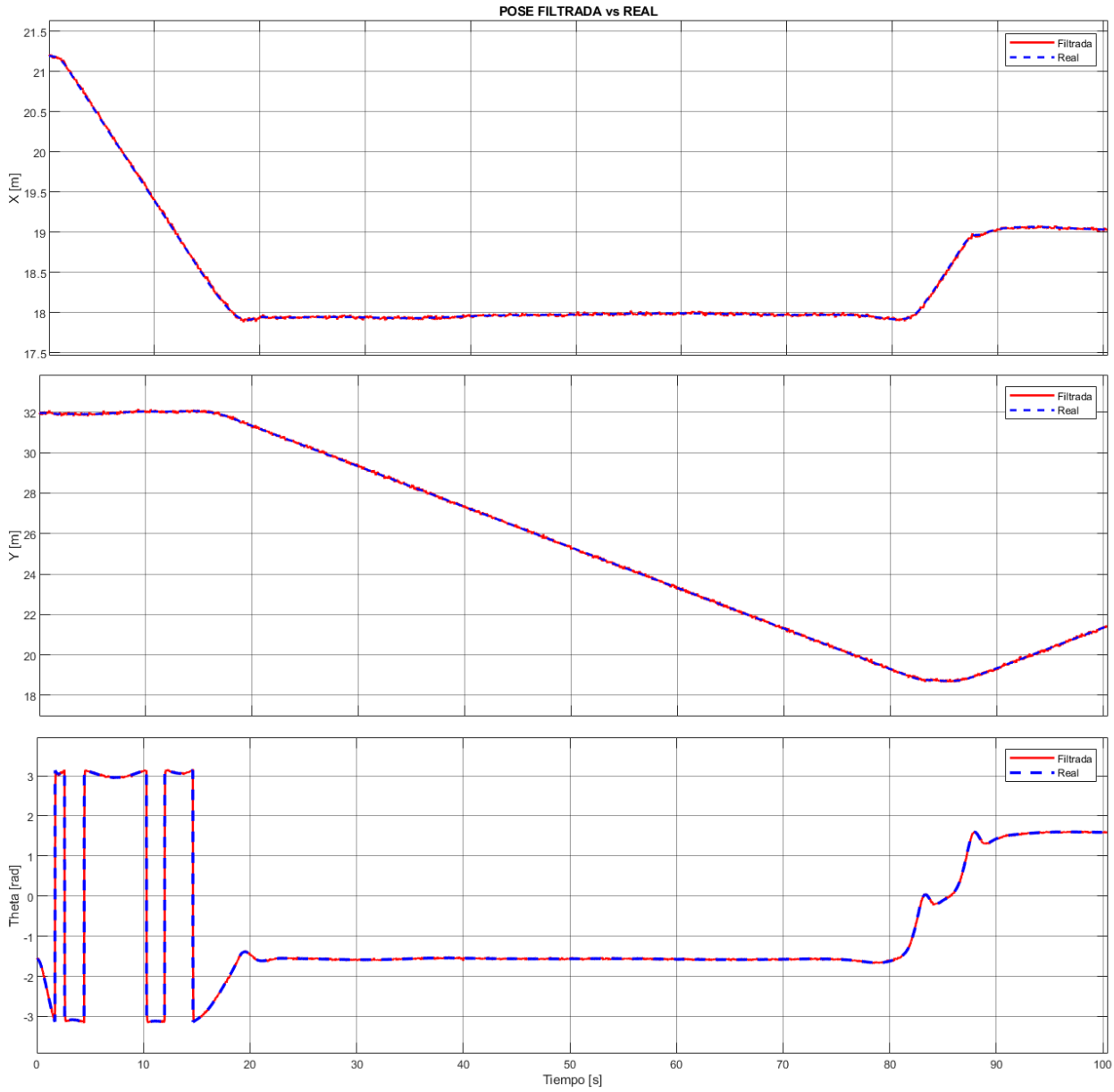



**Figura 50.** Pose $(x, y, \theta)$ Filtrada vs Real. **Fuente:** Autor.

## 7.2. Diseño del invernadero de tomates virtual en Unreal Engine.

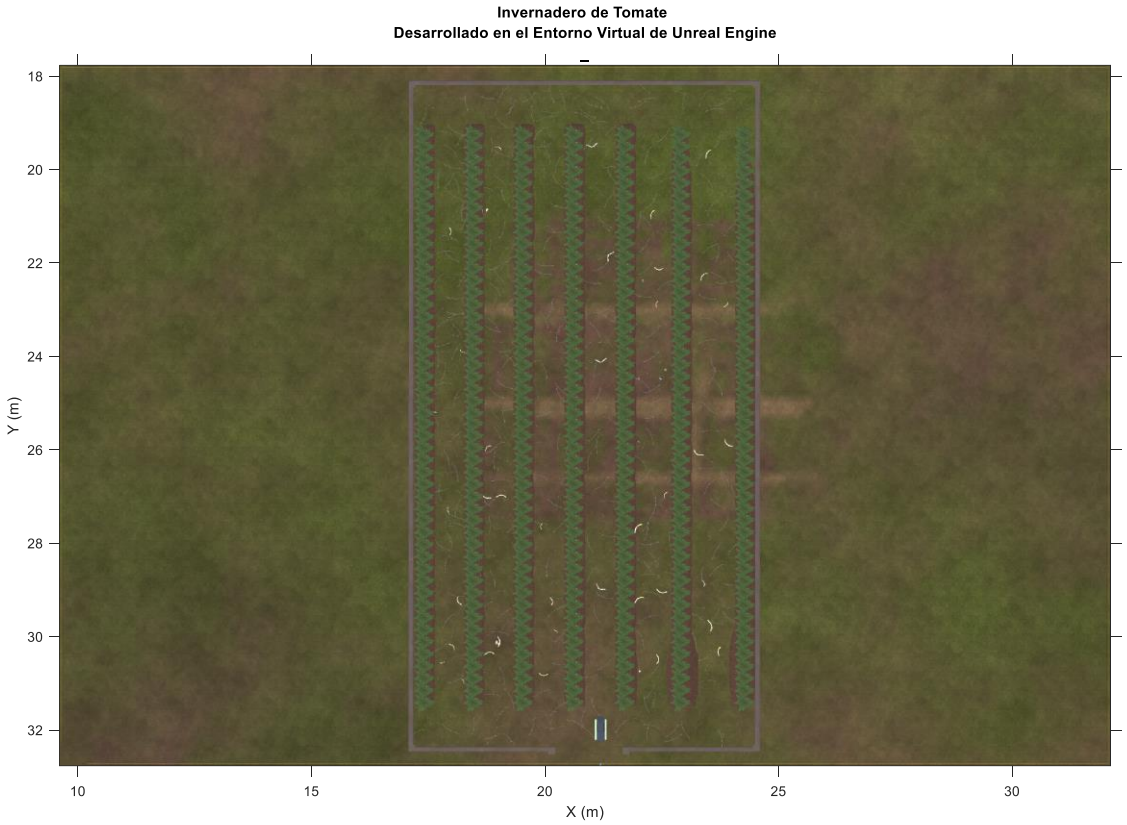
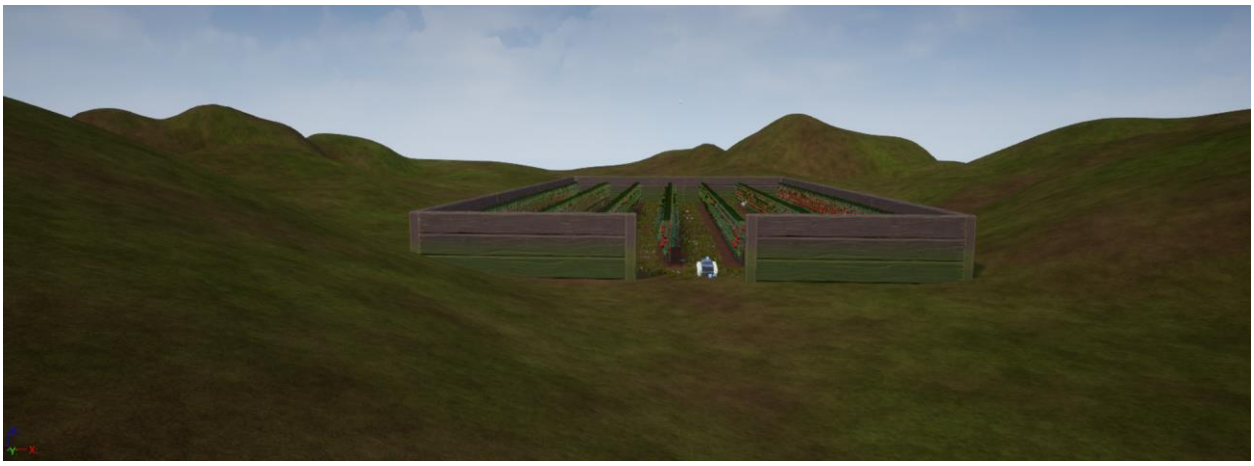



**Figura 51.** Vista satelital del cultivo de tomate de invernadero desarrollado en Unreal Engine. **Fuente:** Autor.

El invernadero destinado al cultivo de tomates ha sido modelado en un entorno virtual utilizando Unreal Engine, siguiendo un conjunto de características específicas de diseño que se detallan a continuación:

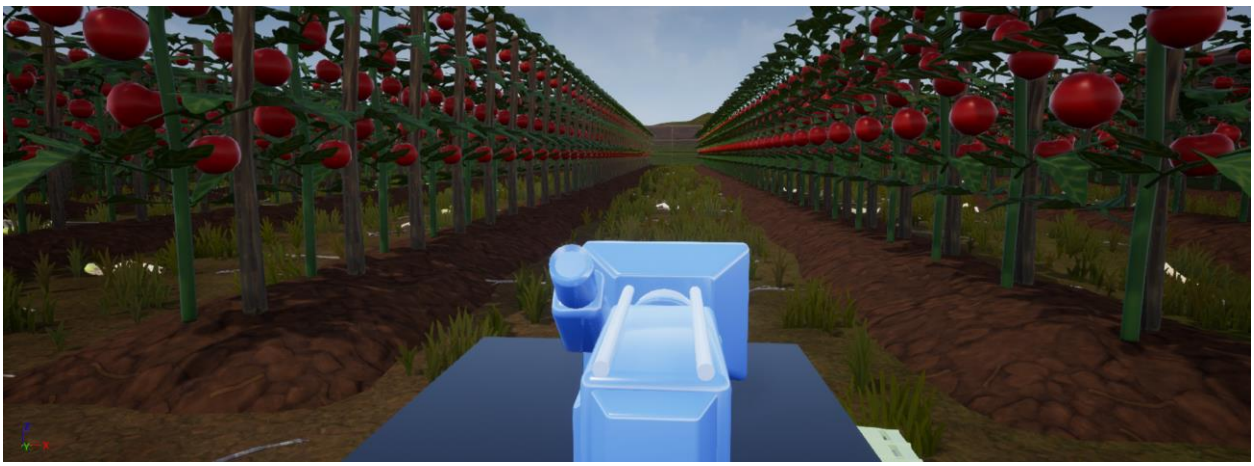
- El invernadero tiene dimensiones de 7.362 metros de ancho y 14.203 metros de largo.
- Dispone de 7 hileras de plantas de tomate, cada una con un total de 40 plantas.
    - Las hileras están separadas entre sí por una distancia de 0.65 metros, y ubicadas a 0.85 metros de las paredes frontal y trasera del invernadero.
- La ubicación del invernadero se encuentra en una semi-planicie con inclinación, situada en un valle.
    - El terreno presenta desniveles en algunas secciones, recreando las irregularidades naturales del relieve.
- En el interior del invernadero se distribuyeron de manera intencional ramas y raíces con el propósito de simular pequeños obstáculos, diseñados para poner a prueba el sistema de amortiguación del robot.

En la Figura 51 se puede observar una vista satelital simulada del invernadero, mientras que en la Figura 52 se puede observar 3 puntos de vista diferentes del cultivo incluyendo: (a) una vista externa en donde se puede observar la posición del invernadero en la semi-planicie, (b) una vista lateral del cultivo y sus hileras, (c) una vista del cultivo desde el punto de visión del robot en su posición inicial.



(a)

(b)



(c)

**Figura 52.** Diferentes vistas del invernadero de tomates. (a) Vista externa. (b) Vista lateral. (c) Vista interna del invernadero desde el punto de visión del robot. **Fuente:** Autor.

### 7.2.1. Modelado de las plantas de tomate.

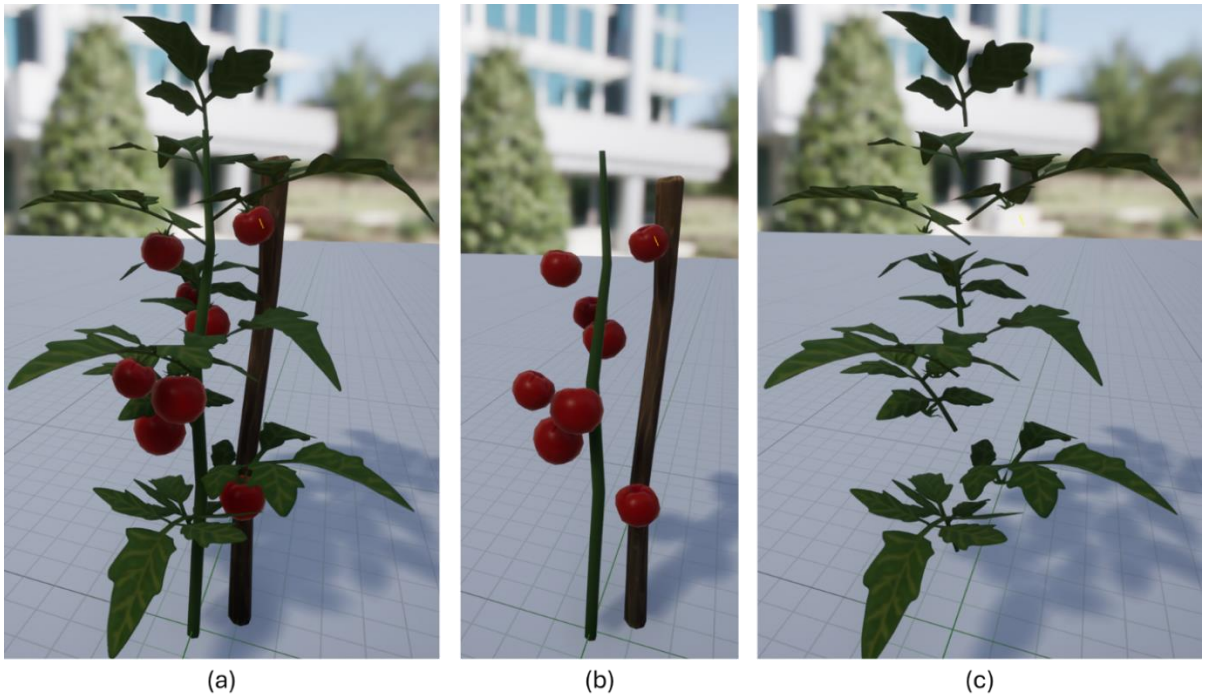



**Figura 53.** Modelado de las plantas de tomate en Unreal Engine. (a) Planta de tomate completa. (b) Malla 1. (b) Malla 2. **Fuente:** Autor.

Para la construcción del invernadero de tomate, se utilizó un modelo uniforme de planta de tomate distribuido a lo largo de todo el cultivo, como se muestra en la Figura 53. Cada planta está equipada con un sistema de amarre sencillo para mantener la planta erguida. El modelo se compone de dos mallas principales: la malla #1, que incluye el tallo, los tomates y un soporte de madera; y la malla #2, que contiene únicamente las ramas y hojas de la planta. Las dimensiones de cada planta de tomate se corresponden con las proporciones naturales, alcanzando una altura máxima de 90 cm.

Cada hilera de cultivo está conformada por una serie de 6 surcos sucesivos en los que se han sembrado las plantas. Los surcos tienen una longitud de 2.4 metros y puede observarse en la Figura 54. Tanto el diseño de las plantas de tomate como el de los surcos fueron extraídos de la librería “FarmCrops” de Unreal Engine y posteriormente adaptados para su implementación en este proyecto.

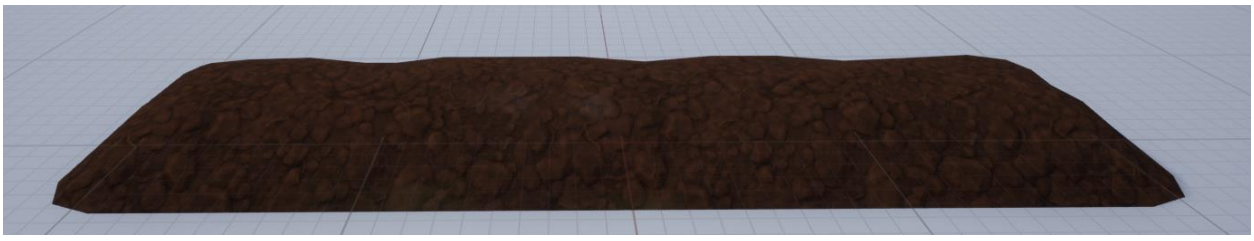


**Figura 54.** Surco de tierra que conforman la base de las plantas. **Fuente:** Autor.

### 7.3. Interfaz de comunicación Unreal Engine – Simulink.

Para implementar una comunicación en tiempo real entre Simulink y Unreal Engine se utilizó los plugins de Unreal Engine: "*MathWorks Interface*" y "*MathWorks Automotive Content*", diseñados para facilitar la integración de simulaciones colaborativas entre ambos entornos. Estos plugins son parte de un esfuerzo para aprovechar el poder visual y de simulación inmersiva de Unreal Engine en el contexto del desarrollo de vehículos autónomos, por lo que han proporcionado una librería de vehículos (automóviles), sensores y otros actores para su uso en estas simulaciones.

El bloque "Simulation 3D Scene Configuration" permite establecer la comunicación directa entre Unreal Engine y Simulink al configurar los parámetros necesarios para realizar esta configuración, ajustando características como los parámetros de la escena, el tiempo de muestreo y el origen de la escena, para así asegurar una integración fluida en la simulación.

#### 7.3.1. Envío y recepción de datos en Unreal Engine.

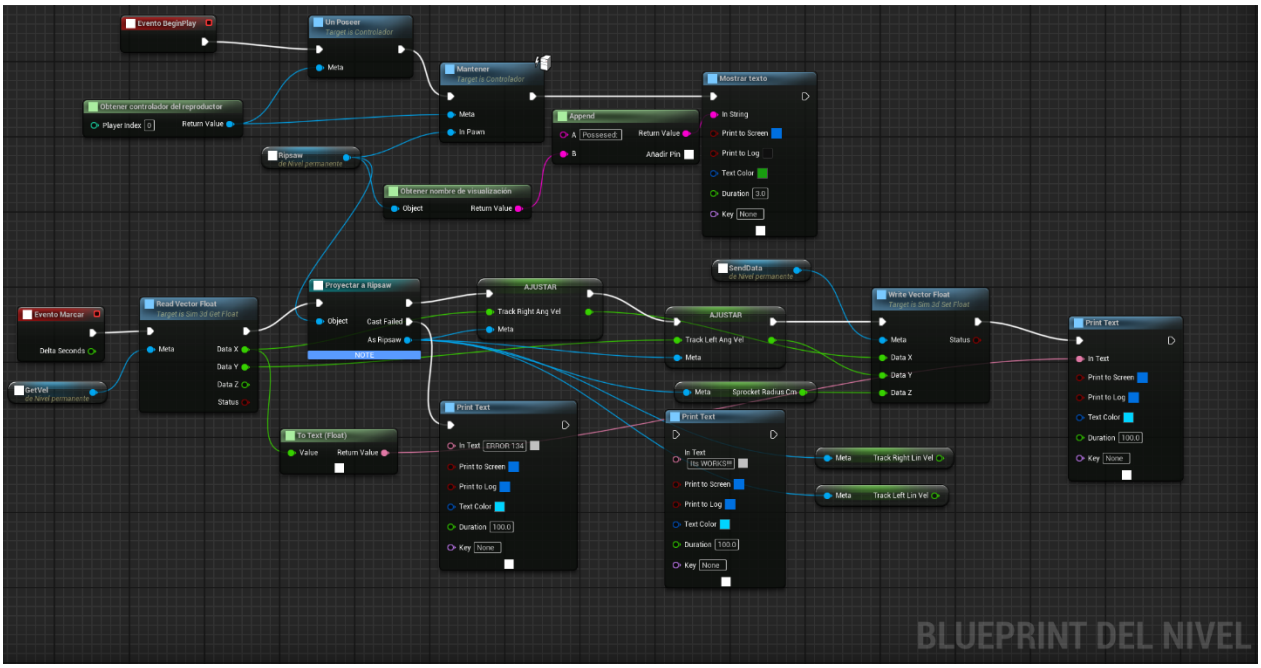



**Figura 55.** Blueprint de nivel estableciendo comunicación directa Unreal Engine – Simulink. **Fuente:** Autor.

La Figura 55 ilustra el blueprint del nivel principal en donde está ubicado el invernadero de tomate. Este blueprint fue diseñado para posicionar al robot en el punto de partida y gestionar la comunicación bidireccional de datos entre Unreal Engine y Simulink durante toda la simulación. El flujo de trabajo comienza con el nodo Evento BeginPlay, el cual inicializa al robot como actor jugable dentro de la simulación.

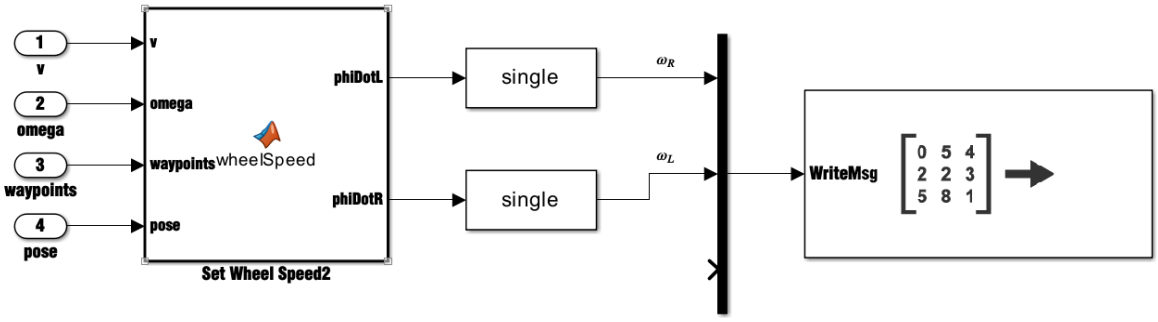
Para establecer una comunicación directa entre ambos entornos se define una arquitectura formada por dos actores principales, "SetVel" y "GetVel," desempeñan las funciones de envío y recepción de datos, respectivamente, utilizando el formato vectorial Float. Ambos actores pertenecen al conjunto de librerías proporcionadas por el plugin "MathWorks Interface." Sus nombres han sido personalizados para facilitar su identificación durante el proceso de programación.

El actor "***SetVel***" ha sido configurado para enviar datos relacionados con la velocidad del robot, especificados mediante el bloque "Write Vector Float." Este bloque toma los valores de velocidad calculados en Unreal y los convierte en un vector de tipo flotante, el cual se transmite posteriormente hacia Simulink. La transmisión de estos valores permite que Simulink reciba en tiempo real la información sobre la velocidad del robot en sus componentes $\boldsymbol{X}$, $\boldsymbol{Y}$ y $\boldsymbol{Z}$, posibilitando una sincronización precisa de los estados del sistema entre ambas plataformas.

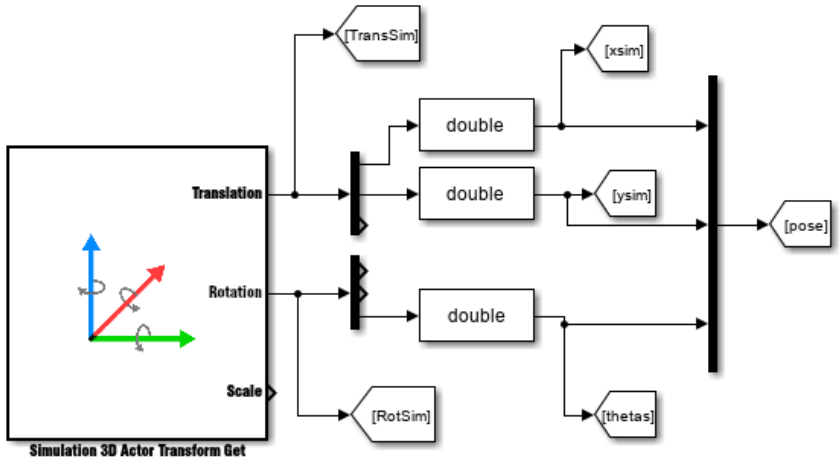
En paralelo, el actor "***GetVel***" está configurado para recibir datos provenientes de Simulink, específicamente las velocidades angulares que son empleadas para el control del movimiento del robot. Estos valores se reciben mediante el bloque "Read Vector Float," que interpreta el vector recibido y distribuye sus componentes hacia los nodos correspondientes que procesan los valores de las velocidades angulares en las ruedas izquierda y derecha del robot. Las señales de velocidad angular se emplean en los nodos "AJUSTAR" (track angular velocity nodes) para ajustar el comportamiento del robot en función de las entradas recibidas desde Simulink.

El blueprint del nivel no solo gestiona la transmisión de información de velocidad, sino que también controla el flujo de datos para asegurar la estabilidad y precisión en el movimiento del robot simulado.

### 7.3.2. Envío y recepción de datos en Simulink.




(a)




(b)

**Figura 56.** Envío y recepción de datos en Simulink. (a) Envió de las señales de velocidad angular de cada una de las ruedas del robot. (b) Recepción de la pose del robot desde Unreal Engine. **Fuente:** Autor.

La Figura 56 ilustra el proceso de envío y recepción de datos desde Simulink. Esta estructura de comunicación permite que los algoritmos de Simulink en envíen comandos precisos de velocidad y reciban información en tiempo real sobre la posición y orientación del robot, garantizando así un control de movimiento coherente y coordinado en el espacio simulado.

El proceso de envío de las velocidades angulares de cada una de las ruedas del robot móvil tipo oruga se puede observar en la Figura 56 (a). El bloque "*Set Wheel Speed*" calcula las velocidades angulares para las ruedas izquierda y derecha basándose en una lógica diseñada para adaptar la velocidad del robot según su proximidad al destino. Al acercarse al punto final, se reduce gradualmente la velocidad lineal para garantizar una parada suave, deteniéndose completamente cuando está dentro de un umbral específico. Este ajuste gradual de la velocidad optimiza la precisión de la maniobra y minimiza la inercia que podría afectar la estabilidad del robot en trayectorias complejas. Las velocidades angulares de las ruedas se derivan de la combinación de la velocidad lineal ajustada y la velocidad angular de dirección, utilizando las constantes como el radio de las ruedas para convertir estas magnitudes en comandos de velocidad para cada rueda. Finalmente, las velocidades angulares de las ruedas se envían a Unreal Engine mediante un mensaje empaquetado, el cual es recibido por el actor "*GetVet".*

En la Figura 56 (b), se detalla el proceso inverso, en el cual Simulink recibe datos de estado desde Unreal Engine. Este procedimiento se lleva a cabo a través del bloque "*Simulation 3D Actor Transform Get*," que proporciona información detallada sobre la translación, rotación y escala del actor correspondiente al robot en el entorno de simulación. Estas salidas de datos se separan en componentes específicas: las translaciones en los ejes $\boldsymbol{X}$ y $\boldsymbol{Y}$, y las rotaciones alrededor del eje $\boldsymbol{Z}$ (correspondiente al ángulo de guiñada o $yaw$), que representa la orientación del robot en el plano. En cambio, en la Figura 57 muestra el proceso de recepción de la velocidad del robot, transmitida por el actor "*SetVel*" en Unreal Engine mediante un mensaje empaquetado. En este caso, se emplea un bloque específico para la recepción de dichos datos.

La comunicación bidireccional descrita en ambas figuras (56a y 56b) proporciona una estructura de control cerrada, en la cual las decisiones tomadas en Simulink, basadas en los datos de estado de Unreal Engine, se traducen en comandos de movimiento precisos.

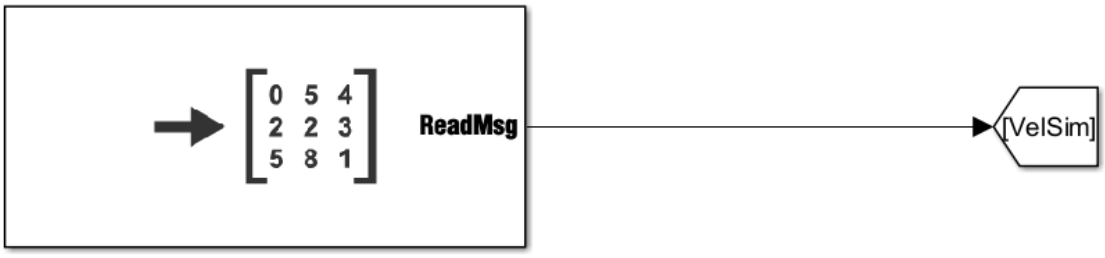



**Figura 57.** Recepción de la velocidad del robot. **Fuente:** Autor.

## 7.4. Diseño del sistema de navegación autónoma.

### 7.4.1. Implementación del algoritmo Pure Pursuit para el seguimiento de trayectorias.

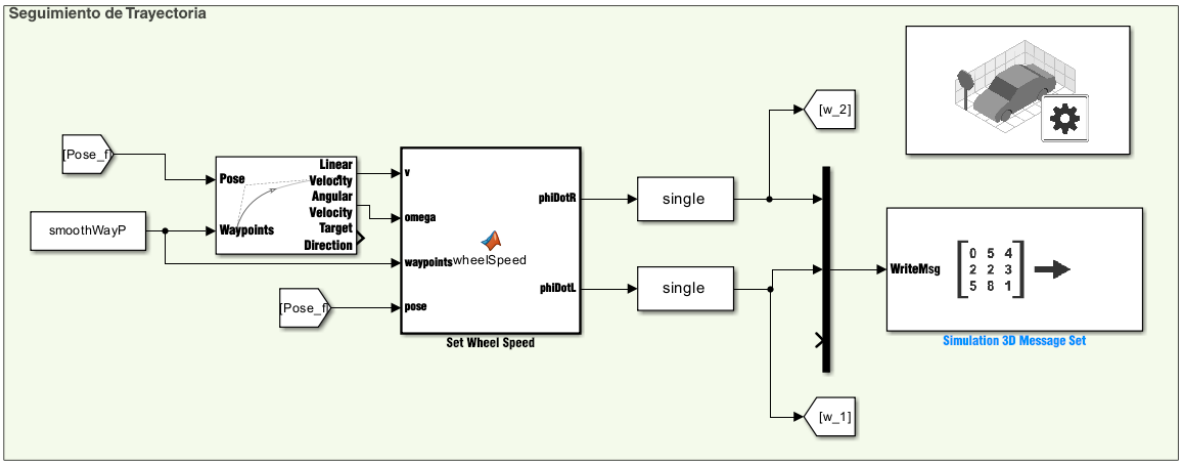



**Figura 58.** Diagrama de bloques del algoritmo de seguimiento de trayectorias Pure Pursuit. **Fuente:** Autor.

El algoritmo Pure Pursuit está diseñado para optimizar el seguimiento de trayectorias en escenarios dinámicos, asegurando que el robot ajuste su orientación y velocidad en función de su posición relativa a los puntos de referencia (o *waypoints*) que componen la ruta objetivo. El diagrama de bloques mostrando en la Figura 58 ilustra la estructura completa del seguimiento de trayectorias, integrando el bloque Pure Pursuit con los bloques de procesamiento y envió de datos para el control preciso del movimiento.

- **Bloque Pure Pursuit:**

En la entrada del sistema se proporciona la posición y los puntos de referencia con el fin de brindarle al robot la información esencial sobre su ubicación actual y trayectoria deseada. Estos datos se introducen en el bloque "**Pure Pursuit**," que calcula la velocidad lineal y angular necesarias para que el robot mantenga su curso en dirección a los puntos de referencia. En este contexto, el algoritmo Pure Pursuit calcula una dirección objetivo basándose en la posición relativa del robot respecto a los waypoints, ajustando dinámicamente las velocidades de manera que el robot avance de manera suave y precisa hacia el siguiente punto de la trayectoria. Los parámetros implementados en el bloque “Pure Pursuit” se encuentran detallados en la Tabla 6.

**Tabla 6.** Parámetros del bloque Pure Pursuit. **Fuente:** autor.

| **Bloque Pure Pursuit** | | |
|---|---|---|
| **Parámetro** | **Valor** | **Unidades** |
| Velocidad lineal deseada | 2 | m/s |
| Velocidad angular máxima | 30 | rad/s |
| Distancia de anticipación | 0.75 | m |

- **Bloque Set Wheel Speed:**

Una vez calculadas las velocidades necesarias, estas son procesadas en el bloque **"S*et Wheel Speed*"**. Este bloque tiene como objetivo ajustar la velocidad lineal y angular del robot en función de su posición actual y su cercanía a un punto objetivo, representado por waypoints. En su código interno se definen parámetros de umbral de parada y desaceleración (*stopThreshold* y slowThreshold, respectivamente), de manera que el robot disminuye su velocidad cuando se aproxima a su destino. Para ello, si la distancia al objetivo es menor que el umbral de desaceleración, la velocidad lineal se reduce proporcionalmente a dicha distancia; si la distancia es menor que el umbral de parada, el robot detiene su movimiento. Esto permite una desaceleración gradual cuando el robot se acerca al destino, evitando la limitación del algoritmo Pure Pursuit al no poder estabilizar al robot en un punto determino por sí solo.

La conversión de la velocidad lineal y angular del robot a la velocidad angular de cada oruga se realiza mediante las ecuaciones (21) y (22), previamente definidas. Los parámetros empleados en este bloque se encuentran detallados en la Tabla 7, mientras que el código correspondiente se presenta en el ANEXO A1.

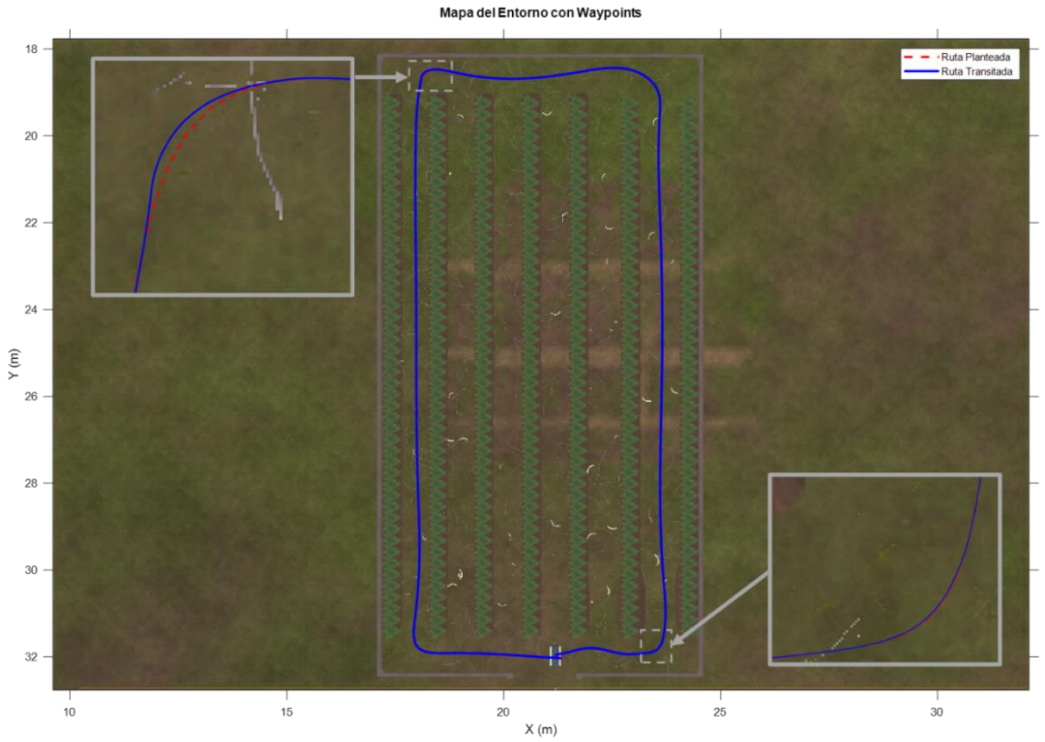
**Tabla 7.** Parámetros del bloque "Set Wheel Speed". **Fuente:** Autor.

| **Parámetros del bloque "Set Wheel Speed"** | | |
|---|---|---|
| **Parámetro** | **Valor** | **Unidades** |
| stopThreshold | 0.5 | - |
| slowThreshold | 2 | - |
| trackWidth | 0.0295 | m |
| wheelRadius | 0.024 | m |

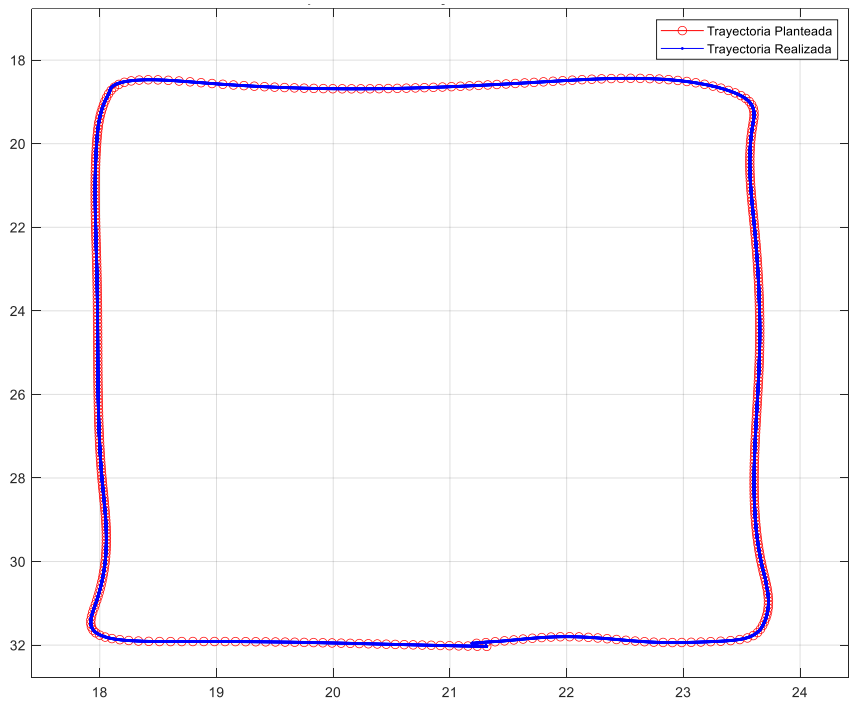
- **Bloque "*Simulation 3D Message Set*":**

Posteriormente, las velocidades angulares de las ruedas son enviadas a Unreal Engine mediante el bloque "***Simulation 3D Message Set***" que encapsula estos valores en un mensaje de datos enviándolos como un vector. Esta estructura de comunicación permite que Unreal Engine reciba en tiempo real las velocidades de cada rueda, permitiendo que el robot responda de acuerdo a las acciones de control generadas en Simulink.

En la Figura 59 (a) se presenta una vista satelital del cultivo que compara la trayectoria planteada manualmente con la trayectoria efectivamente seguida por el robot móvil tipo oruga en el entorno de simulación. Por su parte, la Figura 59 (b) muestra una representación simplificada de ambas trayectorias en un gráfico cartesiano, permitiendo un análisis cuantitativo de la precisión en el seguimiento. A simple vista, la diferencia entre ambas trayectorias es prácticamente imperceptible, destacando la precisión del robot al seguir la ruta en las secciones rectas, aunque se observan ligeras desviaciones en las curvas, atribuibles a las limitaciones del sistema de control en respuesta a cambios bruscos de dirección. Esta comparación visual permite evaluar el desempeño del algoritmo en la adaptación del robot a la trayectoria propuesta en un entorno complejo, representativo de un cultivo de tomate en invernadero, logrando un error promedio de 0.0184 metros, equivalente a un 0.05% de error en el seguimiento de la trayectoria total.




(a)




(b)

**Figura 59.** Comparación de la trayectoria planteada vs la trayectoria realiza. (a) Comparación de trayectorias desde una vista satelital del cultivo. (b) Comparación de trayectorias en un gráfico simplificado. **Fuente:** autor.

### 7.4.2. Implementación del algoritmo LOAM.

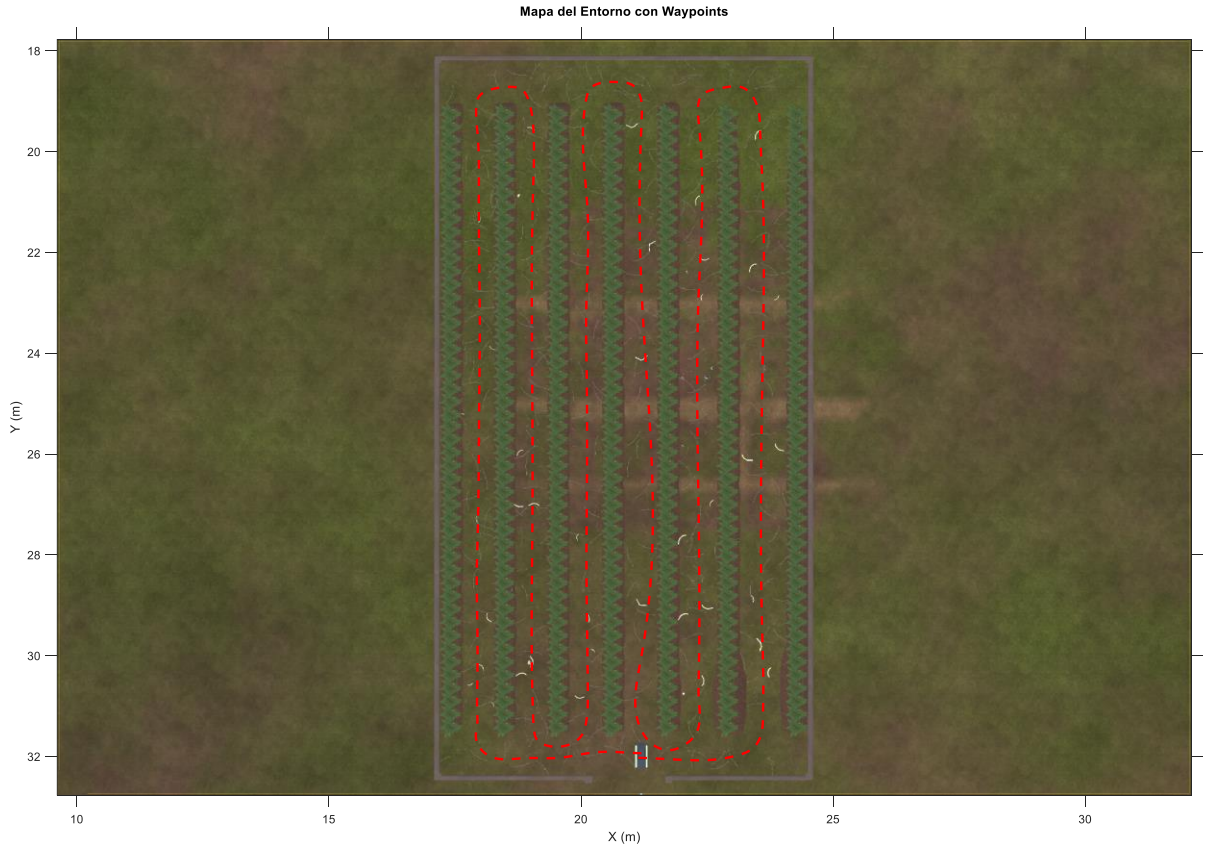



**Figura 60.** Trayectoria realizada para el escaneo LiDAR del cultivo. **Fuente:** autor.

La implementación del algoritmo LOAM tiene como propósito generar un mapa tridimensional del entorno a partir de los datos capturados previamente por un sensor LiDAR durante un recorrido inicial. El robot móvil sigue una trayectoria predefinida, trazada manualmente con base en una vista satelital del terreno, como se ilustra en la Figura 60. A lo largo de este trayecto, el robot registra nubes de puntos del entorno, que son almacenadas en un arreglo para su posterior procesamiento. Una vez concluido el recorrido, estas nubes de puntos se integran y analizan de manera *off-line* en MATLAB utilizando el algoritmo LOAM. Este proceso permite consolidar la información capturada para construir un mapa tridimensional del entorno con alta precisión, representando fielmente las características espaciales del cultivo, al tiempo que reduce significativamente la carga computacional en el robot durante su operación.

El algoritmo inicia con una verificación inicial de dos nubes de puntos consecutivas. Estas se registran utilizando la función *pcregisterloam*, que implementa una transformación inicial basada en una resolución de cuadrícula específica, controlada por el parámetro *gridStep*. En este paso, se seleccionan áreas de interés en las nubes de puntos dentro de un cilindro virtual definido por los parámetros *egoRadius* y *cylinderRadius*. Esto permite centrar el análisis en regiones representativas y relevantes del entorno, optimizando los cálculos subsiguientes. Los valores específicos de estos parámetros están detallados en la Tabla 8.

**Tabla 8.** Parámetros del algoritmo LOAM. **Fuente:** autor.

| Algoritmo LOAM | |
|---|---|
| **Parámetro** | **Valor** |
| gridStep | 0.05 |
| egoRadius | 3 |
| cylinderRadius | 30 |
| maxPlanarSurfacePoints | 5 |
| numSkip | 2 |
| viewId | 1 |
| voxelSize | 0.1 |

Durante el proceso iterativo del algoritmo LOAM, cada nueva nube de puntos capturada por el LiDAR es procesada y alineada con las anteriores para actualizar el mapa global. Este proceso comienza con la selección de puntos relevantes dentro de un cilindro virtual, definido por parámetros que optimizan la representación del entorno inmediato del robot. Las características distintivas de estas nubes, como planos y esquinas, son extraídas utilizando técnicas específicas que permiten reducir la densidad de puntos irrelevantes sin comprometer la precisión. Estas características se registran entre vistas consecutivas mediante transformaciones iniciales estimadas, que se refinan posteriormente para calcular la posición absoluta del robot en el mapa. Este proceso se integra en un marco de referencia global mediante un objeto dinámico de tipo *pcmaploam*, que acumula las características y actualiza las relaciones espaciales en tiempo real. De esta forma, el sistema genera una representación tridimensional precisa del entorno, cuyo resultado final se visualiza gráficamente para evaluar tanto la trayectoria como la calidad del mapeo. En la Figura 61 se muestra la comparación entre la trayectoria real y la estimada por el algoritmo, donde el error promedio es del 3.26%.

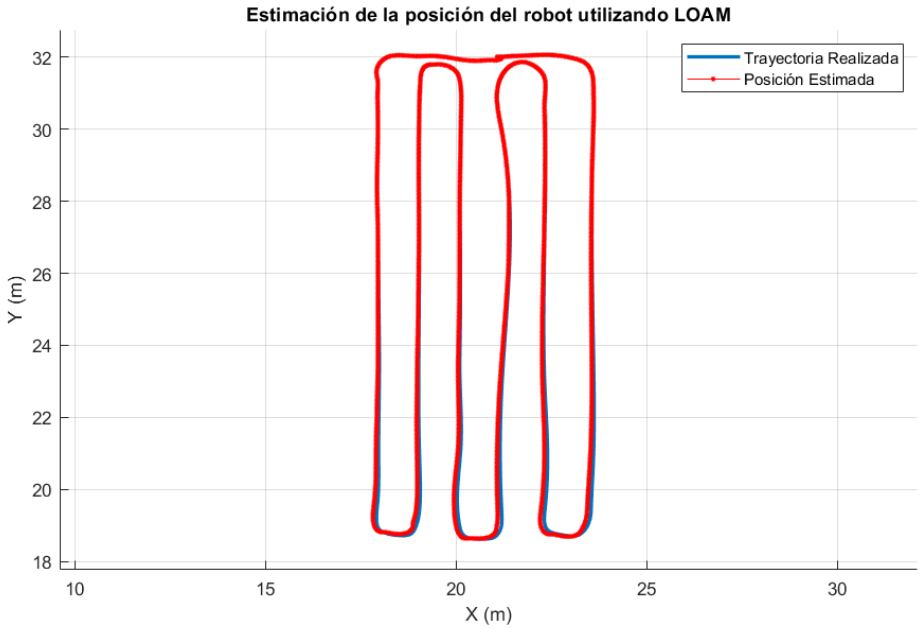
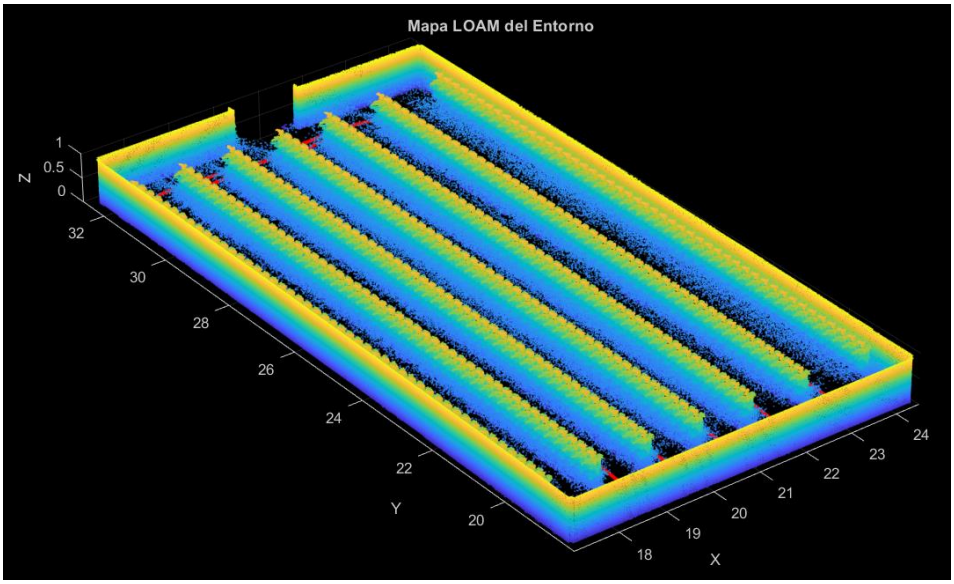
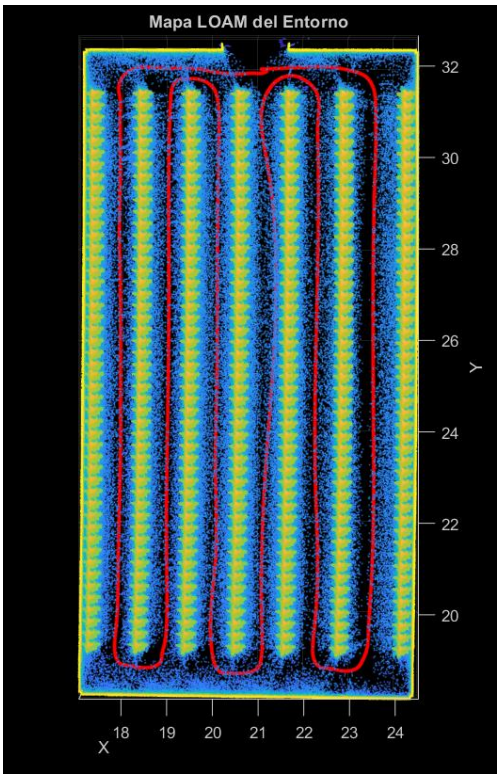



**Figura 61.** Trayectoria transitada vs posición estimada por el algoritmo LOAM. **Fuente:** autor.

Adicionalmente, las representaciones tridimensional y superior del mapa generado se muestran en las Figura 62a y 62b, respectivamente, destacándose la reconstrucción detallada del entorno y la trayectoria estimada del robot dentro del cultivo, representada por la línea roja.




(a)




(b)

**Figura 62.** Mapa tridimensional del entorno generado por el algoritmo LOAM. (a) Vista isométrica. (b) Vista de planta. **Fuente:** autor.

#### 7.4.2.1. Conversión del mapa LOAM a un mapa de costos.

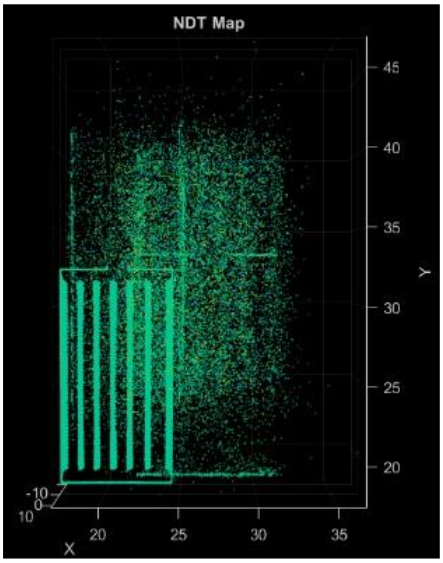



**Figura 63.** Mapa NTD del entorno. **Fuente:** autor.

Una vez obtenido el mapa tridimensional completo del cultivo, este se transforma en un mapa de costos vehicular, utilizado para generar trayectorias que permitan al robot desplazarse de manera eficiente en el entorno. Para ello, la nube de puntos previamente construida del entorno se convierte en un mapa de Transformación de Distribuciones Normales (NDT, por sus siglas en ingles), una representación comprimida del mapa tridimensional (MathWorks, 2024). En la Figura 63 se presenta el mapa NDT del entorno.

A partir del mapa NDT, se genera un mapa de costos vehicular bidimensional (2D), que define el espacio de búsqueda para la planificación de trayectorias alrededor del vehículo. Este mapa de costos incluye información relevante del entorno, como la ubicación de obstáculos y áreas inaccesibles para el vehículo. Para detectar posibles colisiones, los obstáculos en el mapa se inflan mediante un radio de inflación, creando una representación más realista de las zonas prohibidas. El mapa de costos se almacena como una cuadrícula bidimensional, comúnmente denominada cuadrícula de ocupación. Cada celda de esta cuadrícula tiene un valor en el rango [0, 1], que indica el costo asociado a la navegación por dicha celda (MathWork, 2024). Además, las celdas se clasifican como libres, ocupadas o desconocidas, dependiendo de su estado, lo cual se ilustra mediante diferentes colores en la Figura 64.

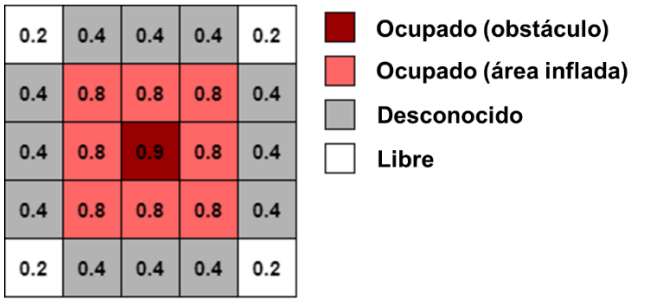



**Figura 64.** Representación de los estados de cada celda de un mapa de costos según su color. **Fuente:** adaptado de (MathWork, 2024)

En la Figura 65 se presenta el mapa de costos correspondiente al cultivo, mientras que el código completo del algoritmo LOAM y el proceso detallado para la generación del mapa de costos se encuentran en el ANEXO A2.

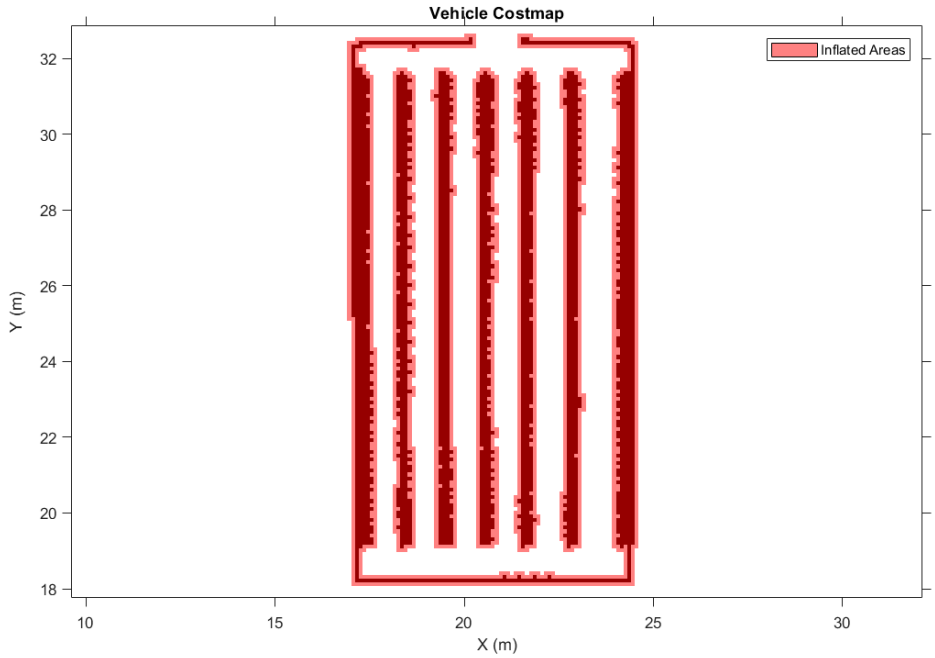



**Figura 65.** Mapa de costos vehicular del cultivo de tomate. **Fuente:** autor.

### 7.4.3. Generación de trayectorias óptimas.

#### 7.4.3.1. Implementación del algoritmo A* Híbrido.

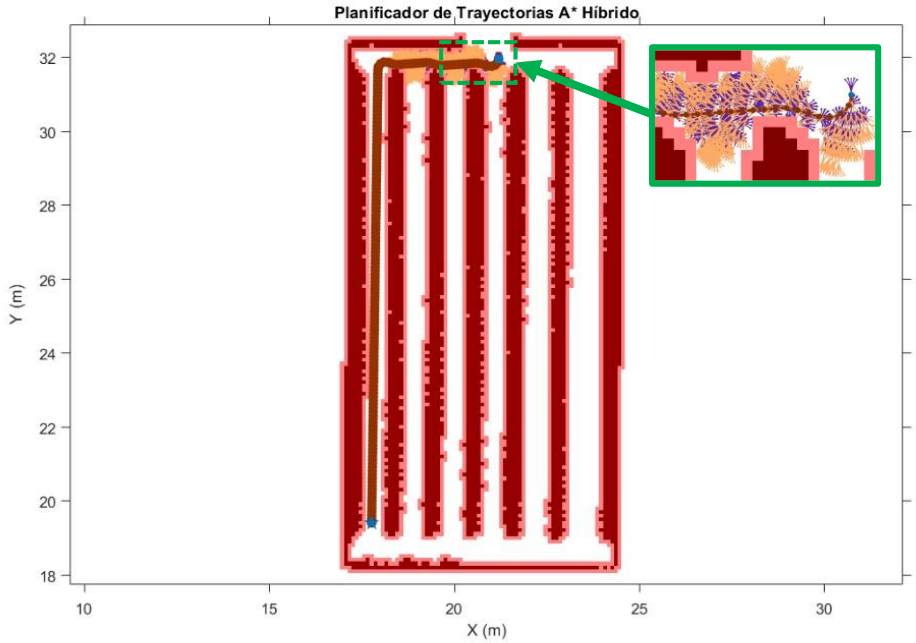



**Figura 66.** Trayectoria planificada con el Algoritmo A* Híbrido. **Fuente:** autor.

El algoritmo A* Híbrido utiliza el mapa de costos desarrollado en la sección anterior como base para generar trayectorias óptimas y seguras que permitan al robot desplazarse en el entorno delimitado. En la Figura 66 se presenta un ejemplo de una trayectoria planificada, destacando un detalle ampliado que muestra las primitivas de movimiento generadas por el algoritmo. Su propósito es garantizar que la trayectoria sea continua, dinámica y físicamente realizable, adaptándose a las capacidades del robot y evitando colisiones en zonas de alta densidad de obstáculos.

La configuración del planificador se fundamenta en tres parámetros clave: el ***radio mínimo de giro*** (*MinTurningRadius*), que determina las restricciones de maniobrabilidad del vehículo; ***la longitud de los primitivas de movimiento*** (*MotionPrimitiveLength*), que define la resolución del espacio de búsqueda y la suavidad de las trayectorias generadas; y ***el costo asociado a los movimientos en reversa*** (*ReverseCost*), el cual prioriza los desplazamientos hacia adelante para reducir maniobras complejas. Estos parámetros, definidos en la Tabla 9, desempeñan un rol esencial en la planificación, ya que ajustan la solución a las capacidades físicas del vehículo y las características del entorno.

**Tabla 9.** Parámetros del planificador A* Híbrido. **Fuente:** autor.

| **Planificador A* Híbrido** | |
|---|---|
| **Parámetro** | **Valor** |
| MinTurningRadius | 2 |
| MotionPrimitiveLength | 1.5 |
| ReverseCost | 10 |

### 7.4.3.2. Optimización de la trayectoria.

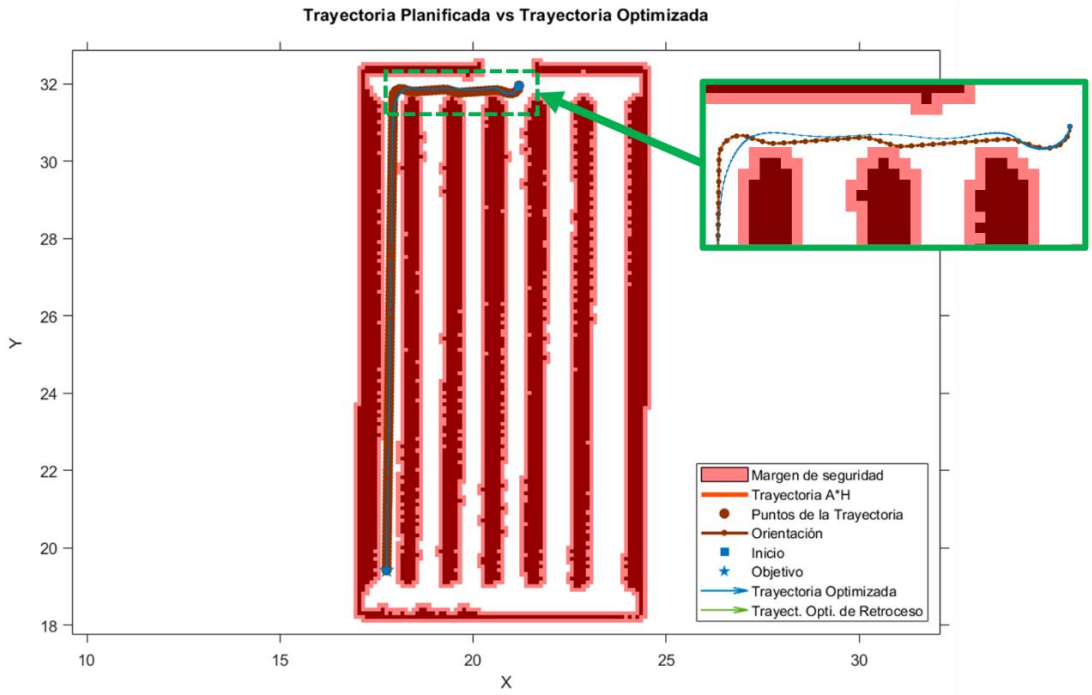



**Figura 67.** Trayectoria planificada por el algoritmo A* Híbrido vs trayectoria optimizada. **Fuente:** autor.

La trayectoria generada inicialmente por planificadores como el A* Híbrido, aunque factible y eficiente en términos de costos, no siempre cumple con los requisitos de suavidad, estabilidad o seguridad óptimos. Para abordar estas limitaciones, se emplea un proceso de optimización basado en el algoritmo de Banda Elástica Temporal (TEB), cuyo objetivo es generar una nueva trayectoria más suave y con una distancia de seguridad adecuada respecto a los obstáculos previamente identificados. En la Figura 67 se presenta una comparación entre la trayectoria original generada por el planificador y la trayectoria optimizada por el algoritmo TEB. La trayectoria resultante presenta una menor variabilidad en las curvas, con una mayor continuidad en sus segmentos y un distanciamiento prudente de los obstáculos, evidenciando la mejora en términos de seguridad y adaptabilidad al entorno.

La configuración del optimizador TEB está definida por tres conjuntos de parámetros clave: los parametros de la trayectoria, los parametros para la evasion de obstaculos y los parametros del solucionador. Estos parametros se detallan en la Tabla 10.

**Tabla 10.** Parámetros para la optimización usando el algoritmo TEB. **Fuente:** autor.

| **Optimización de la trayectoria** | | |
|---|---|---|
| **Parámetro** | **Valor** | **Unidades** |
| MinTurningRadius | 3 | m |
| MaxVelocity | 5 | m/s |
| MaxAcceleration | 1 | m/s/s |
| ReferenceDeltaTime | 0.2 | s |
| MaxPathStates | 800 | - |
| **Optimización de la evasión de obstáculos** | | |
| **Parámetro** | **Valor** | **Unidades** |
| ObstacleSafetyMargin | 2 | m |
| ObstacleInclusionDistance | 0.75 | m |
| ObstacleCutOffDistance | 3 | m |
| **Parámetros del solucionador** | | |
| **Parámetro** | **Valor** | **Unidades** |
| NumIteration | 4 | - |
| MaxSolverIteration | 15 | - |

- **Parámetros de la trayectoria:** se utilizan para especificar las limitaciones del robot mientras se mueve a lo largo de la trayectoria. Los parámetros a ajustar son:
  - ***MinTurningRadius***: es el radio de giro mínimo del robot. Aumentar este parámetro dará lugar a curvas más grandes en la trayectoria.
  - ***MaxVelocity:*** es la velocidad máxima que puede alcanzar el robot.
  - ***MaxAcceleration:*** es la aceleración máxima posible para el robot.
  - ***ReferenceDeltaTime:*** es la duración del recorrido entre dos poses consecutivas. Aumentar este parámetro aumenta la velocidad del robot.

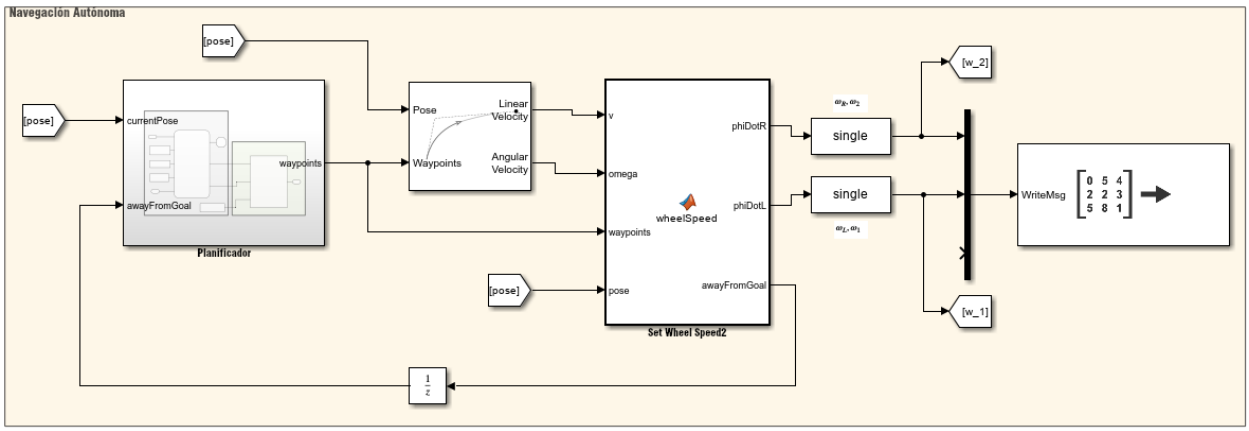
- ***MaxPathStates:*** es el número máximo de poses permitida en la trayectoria. Aumentar este parámetro puede generar una trayectoria más suave, pero puede aumentar el tiempo de ejecución de la optimización.

- **Parámetros para la evasión de obstáculos:** especifican la influencia de los obstáculos en la trayectoria. Los parámetros a ajustar son:
  - ***ObstacleSafetyMargin:*** es el margen de seguridad que debe mantener la trayectoria respecto a los obstáculos. Aumentar este margen hará que la trayectoria sea más segura y aumentará la distancia de que se deja respecto a los obstáculos.
  - ***ObstacleInclusionDistance:*** los obstáculos dentro de esta distancia se incluirán para la optimización.
  - ***ObstacleCutOffDistance:*** los obstáculos más allá de esta distancia se ignorarán durante la optimización. Este valor siempre debe ser mayor que la distancia de inclusión de los obstáculos.

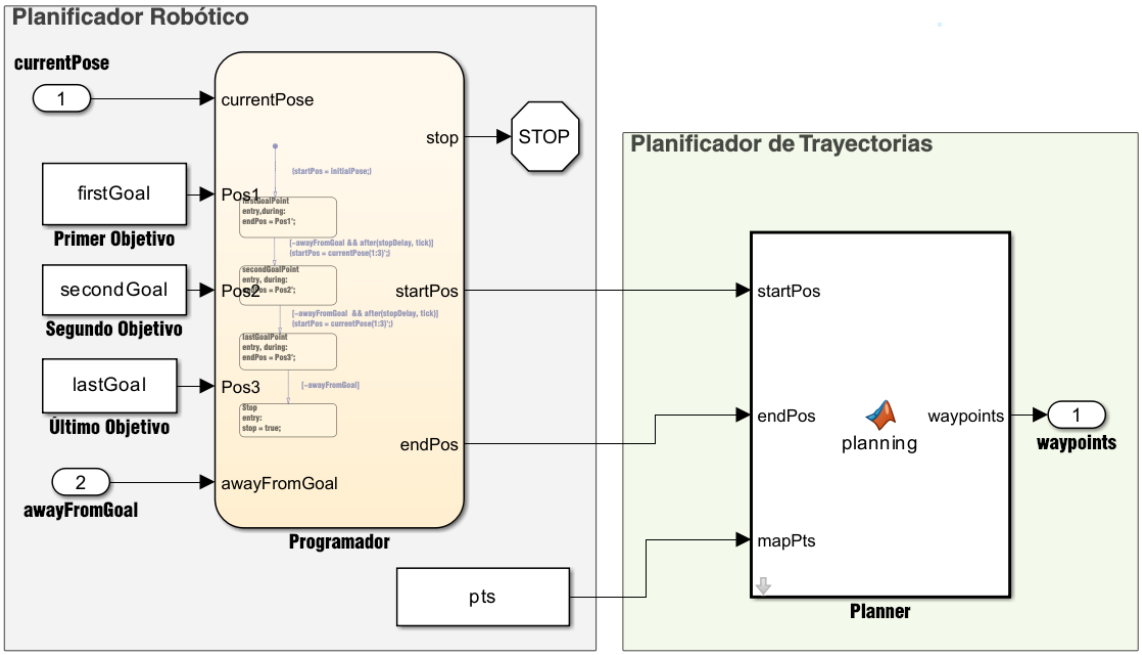
- **Parámetros del solucionador:** especifican las opciones que el solucionador utiliza para optimizar la trayectoria. Los valores más altos de estos parámetros mejoran los resultados de la optimización, pero también afectan al tiempo de optimización. Los parámetros a ajustar son:
  - ***NumIteration:*** es el número de veces que se invoca al solucionador durante la optimización.
  - ***MaxSolverIteration:*** es el número máximo de iteraciones por invocar dentro del solucionador.

#### 7.4.3.3. Diseño del planificador en Simulink.

El planificador diseñado para el sistema de navegación autónoma, implementado en Simulink, se encarga de gestionar el movimiento secuencial del robot móvil a través de tres puntos objetivos, generando y siguiendo una trayectoria hacia cada uno de ellos. El diagrama de bloques correspondiente al planificador se muestra y se desglosa de manera detallada en la Figura 68, donde se ilustra de forma clara y precisa los componentes que conforman el funcionamiento completo del sistema de navegación.




(a)




(b)

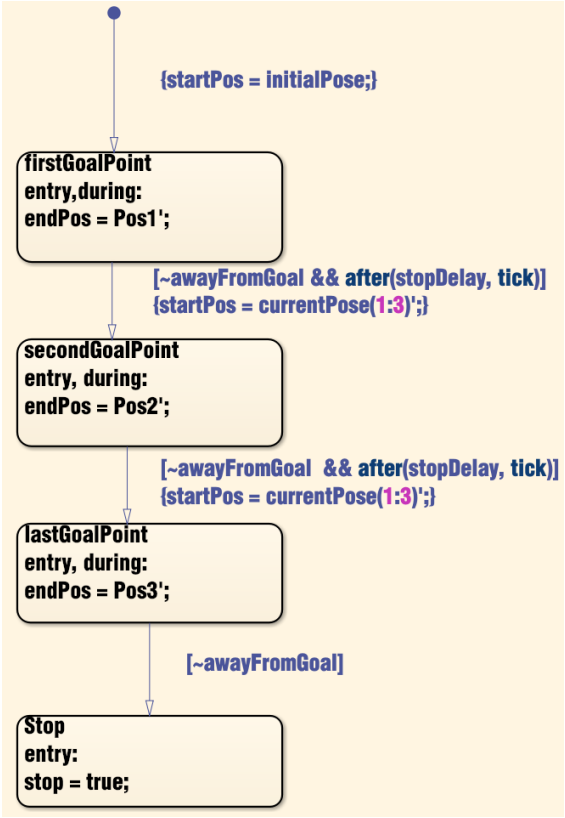



**Figura 68.** Diseño del planificador de trayectorias multiobjetivo en Simulink. (a) Diagrama de bloques principal. (b) Bloque planificador. (c) Máquina de estados finitos del bloque programador. **Fuente:** autor.

La Figura 68a presenta una vista global del sistema de navegación autónoma. Este sistema integra módulos que trabajan de manera conjunta para planificar trayectorias, generar velocidades y controlar el movimiento del robot. En la Figura 68b se detallan los componentes internos del bloque “*Planificador Robótico*”, compuesto por dos submódulos principales:

- **El bloque programador:** constituye el núcleo de la lógica secuencial para el planificador robótico, encargándose de gestionar la transición entre los diferentes objetivos del sistema de navegación autónoma. Este bloque opera mediante un modelo de máquina de estados finitos, ilustrado en la Figura 68c, el cual organiza y controla el flujo lógico del sistema desde la posición inicial hasta la culminación del recorrido, pasando secuencialmente por tres puntos objetivos designados.
- **Bloque Planner:** Este submódulo se encarga de calcular los puntos intermedios o "*waypoints*" que definen la trayectoria desde la posición inicial (startPos) hasta el punto final (endPos). Además, recibe un conjunto de puntos mapeados (mapPts) que permiten ajustar la trayectoria dentro del entorno previamente conocido.

## 7.5. Validación del sistema en simulación.

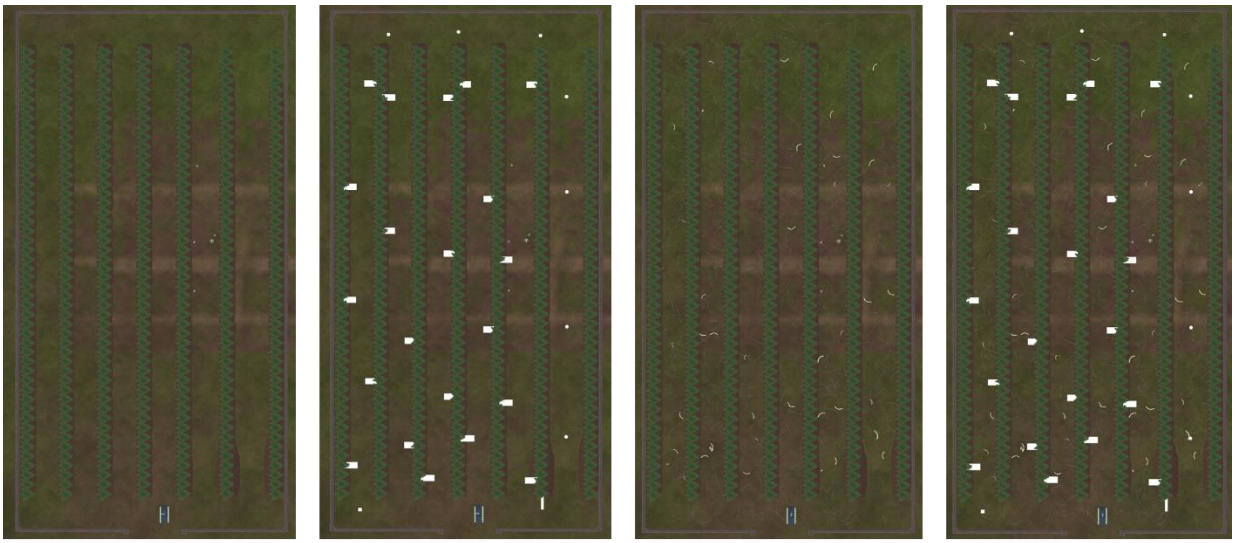



**Figura 69.** Escenarios de prueba para la validación del sistema de navegación. (a) Escenario 1: normal sin obstáculos. (b) Escenario 2: normal con obstáculos. (c) Escenario 3: irregular sin obstáculos. (d) Escenario 4: irregular con obstáculos. **Fuente:** autor.

La validación del sistema de navegación autónoma propuesto se llevó a cabo mediante una serie de pruebas diseñadas con el objetivo de evaluar su desempeño en condiciones variadas de terreno y obstáculos. La finalidad de estas pruebas fue examinar la capacidad del robot para seguir trayectorias predefinidas con precisión, así como su aptitud para responder de manera efectiva a obstáculos conocidos en el entorno.

Las pruebas se realizaron en cuatro escenarios diferentes, los cuales se ilustran en la Figura 69. Cada uno de estos escenarios fue diseñado para replicar situaciones específicas que el robot podría encontrar en un entorno agrícola realista. En cada escenario, se evaluó el desempeño del robot mediante cuatro conjuntos de trayectorias, cada uno compuesto por tres puntos objetivo secuenciales y se registraron de manera sistemática las poses alcanzadas por el robot a medida que se desplazaba por el entorno. Estos puntos fueron estratégicamente distribuidos con el fin de asegurar que el conjunto total de trayectorias cubriera de manera integral el área del cultivo simulado. Este enfoque permitió evaluar no solo la capacidad del robot para desplazarse por el espacio agrícola, sino también su habilidad para mantener la trayectoria planificada en condiciones de terreno variable y frente a la presencia de obstáculos.

A continuación, se describen los diferentes escenarios utilizados para la simulación de las pruebas, así como las características de cada uno y su propósito en la evaluación del comportamiento del sistema de navegación autónoma.

### 7.5.1. Diseño de los escenarios de prueba.

#### 7.5.1.1. Escenario 1: Normal sin obstáculos.

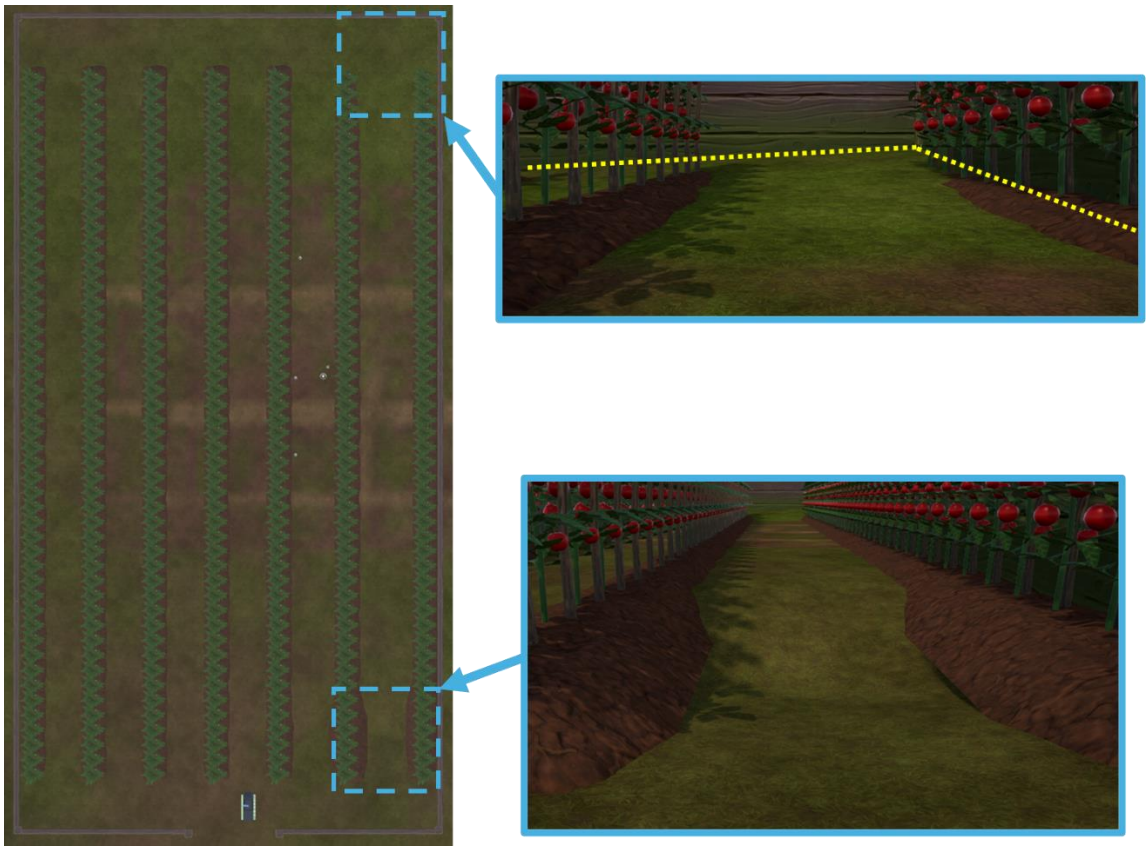
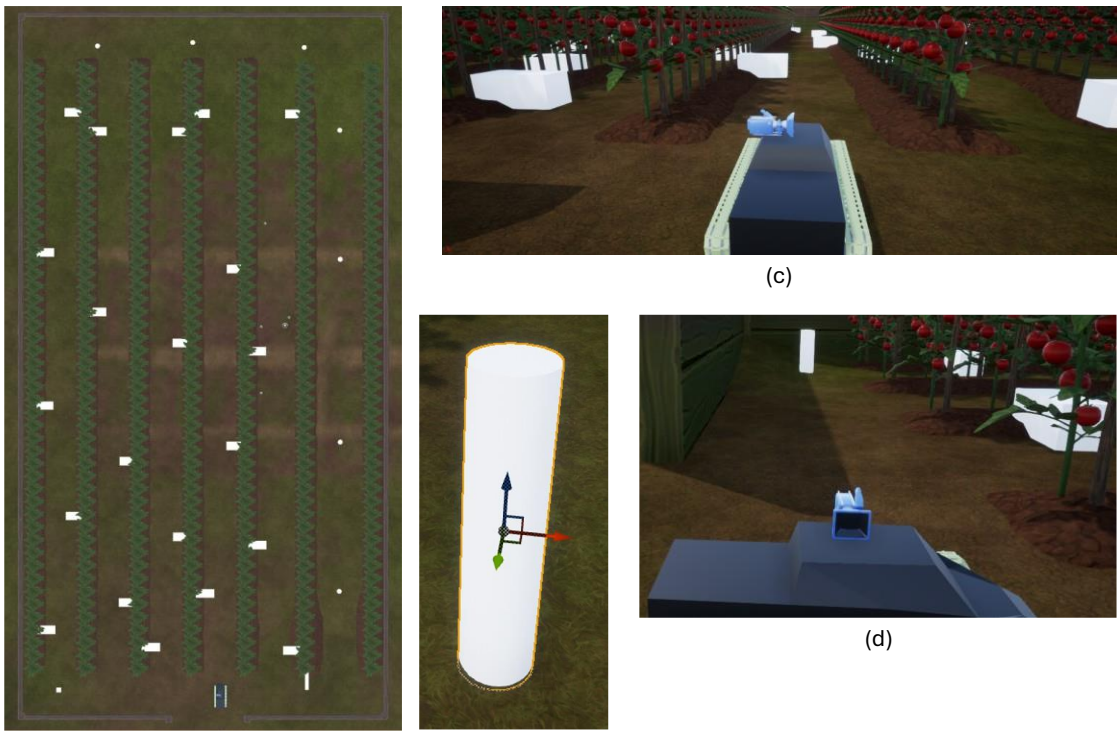


**Figura 70.** Escenario 1: normal sin obstáculos, con vista a detalle de dos desniveles principales del terreno. **Fuente:** autor.

El primer escenario simula un invernadero de tomate en estado óptimo, sin obstáculos externos. Este entorno se caracteriza por tener una distribución regular de las hileras de cultivo, con algunos desniveles mínimos en el terreno, como se puede observar en la Figura 70. Este escenario se considera como el escenario ideal para que el robot transite sin dificultades, sirviendo como referencia para evaluar el rendimiento del sistema de navegación en condiciones controladas. Aquí, el robot puede seguir las trayectorias sin interferencias externas, permitiendo evaluar su capacidad para mantener el rumbo y alcanzar los puntos objetivos sin desviaciones significativas.

#### 7.5.1.2. Escenario 2: Normal con obstáculos.

El segundo escenario es similar al primero, pero incluye obstáculos externos ubicados en el entorno de prueba. Estos obstáculos, de forma rectangular y cilíndrica, están diseñados con una altura suficiente para evitar que el robot pase por encima de ellos, generando así colisiones significativas. Los obstáculos representan posibles objetos que podrían encontrarse en las proximidades de las hileras de cultivo, como herramientas o elementos dejados accidentalmente durante el proceso de trabajo. La disposición de estos obstáculos puede verse en la Figura 71a, y en las figuras 71b, 71c y 71d se presentan diferentes vistas del escenario, mostrando las posiciones relativas de los obstáculos desde el punto de partida del robot.



(a) (b) (c) (d)

**Figura 71.** Escenario 2: normal con obstáculos. (b) Obstáculo en forma de cilindro. (c) Vista frontal desde el punto de partida del robot. (d) Vista lateral izquierda desde el punto de partida del robot. **Fuente:** autor.

Este escenario tiene como propósito evaluar la capacidad del robot para detectar y evadir obstáculos, poniendo a prueba los algoritmos de detección y evasión en un entorno más realista y dinámico. El robot debe ser capaz de identificar los obstáculos y realizar maniobras de evasión para continuar su trayecto sin colisiones.

El mapeo correspondiente a los terrenos, tanto con obstáculos como sin ellos, fue llevado a cabo en los Escenarios 1 y 2. Esta elección se fundamenta en que, al no alterar la disposición de las hileras del cultivo y al mantener constante la ubicación de los obstáculos, ambos mapas resultan extrapolables a los Escenarios 3 y 4, respectivamente. Esta metodología garantiza la coherencia y validez de los resultados obtenidos en los distintos escenarios analizados.

#### 7.5.1.3. Escenario 3: Irregular sin obstáculos.

El tercer escenario mantiene la distribución estándar de las hileras de tomate del primer escenario, pero incorpora irregularidades en el terreno. Estas irregularidades están representadas por pequeños obstáculos como rocas y ramas, las cuales el robot debe poder atravesar sin generar colisiones, tal como se observa en la Figura 72.

Como se observa en la Figura 73, las irregularidades son de tal tamaño y forma que el robot puede superarlas sin que se produzca una colisión, lo que permite evaluar su capacidad para navegar en terrenos irregulares y pone aprueba el sistema de amortiguamiento característico de los robots móviles tipo oruga. Las irregularidades no representan un desafío insuperable para el robot y no son detectadas por el sensor LiDAR en la generación del mapa, pero sirven para poner a prueba su capacidad para manejar superficies no completamente planas, como las que podrían encontrarse en un cultivo real.

Este escenario es crucial para evaluar cómo el sistema maneja pequeñas variaciones en el terreno, lo cual es común en ambientes agrícolas. La simulación de estos obstáculos mínimos permite observar cómo el robot ajusta su navegación ante condiciones de terreno imperfecto, sin perder precisión en su trayectoria.

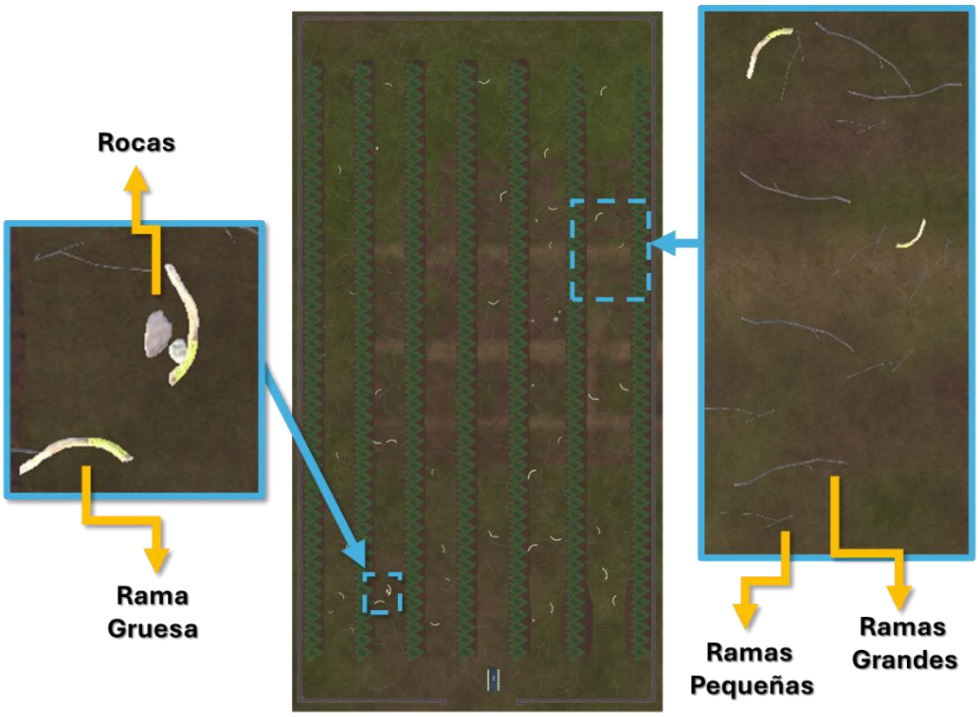



**Figura 72.** Escenario 3: irregular sin obstáculos, con vista a detalle de las rocas y las ramas que conforman las irregularidades del terreno. **Fuente:** autor.

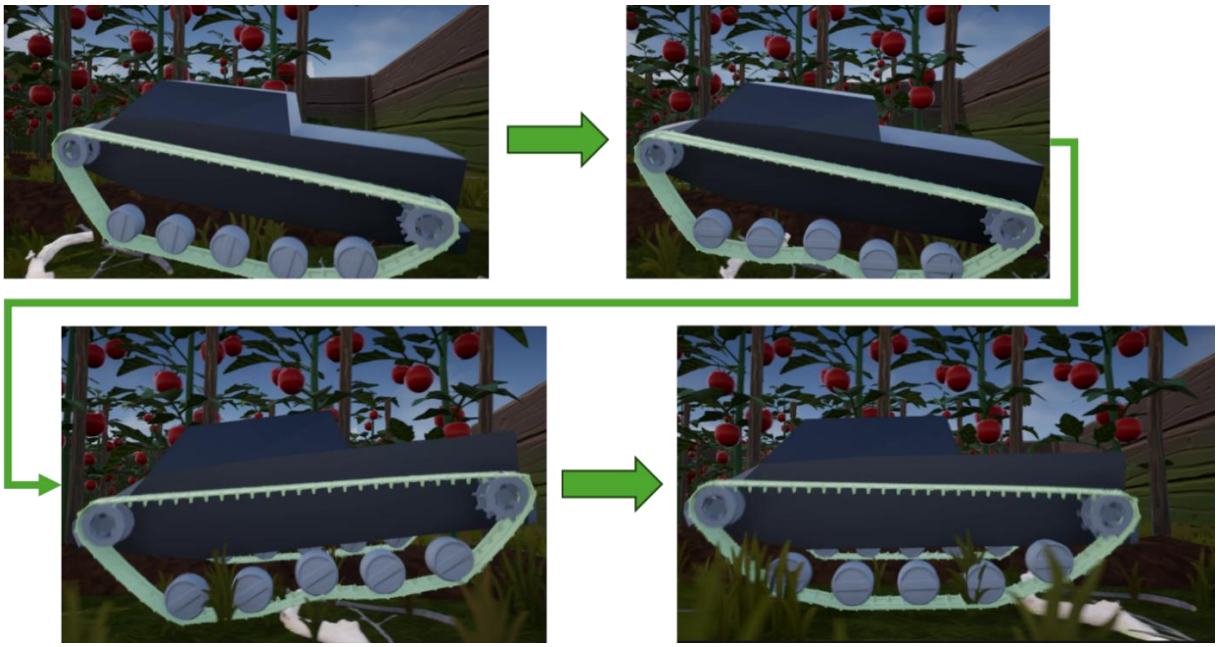


**Figura 73.** Proceso de evasión de las irregularidades en el terreno del Escenario 3. **Fuente:** autor.

#### 7.5.1.4. Escenario 4: Irregular con obstáculos.

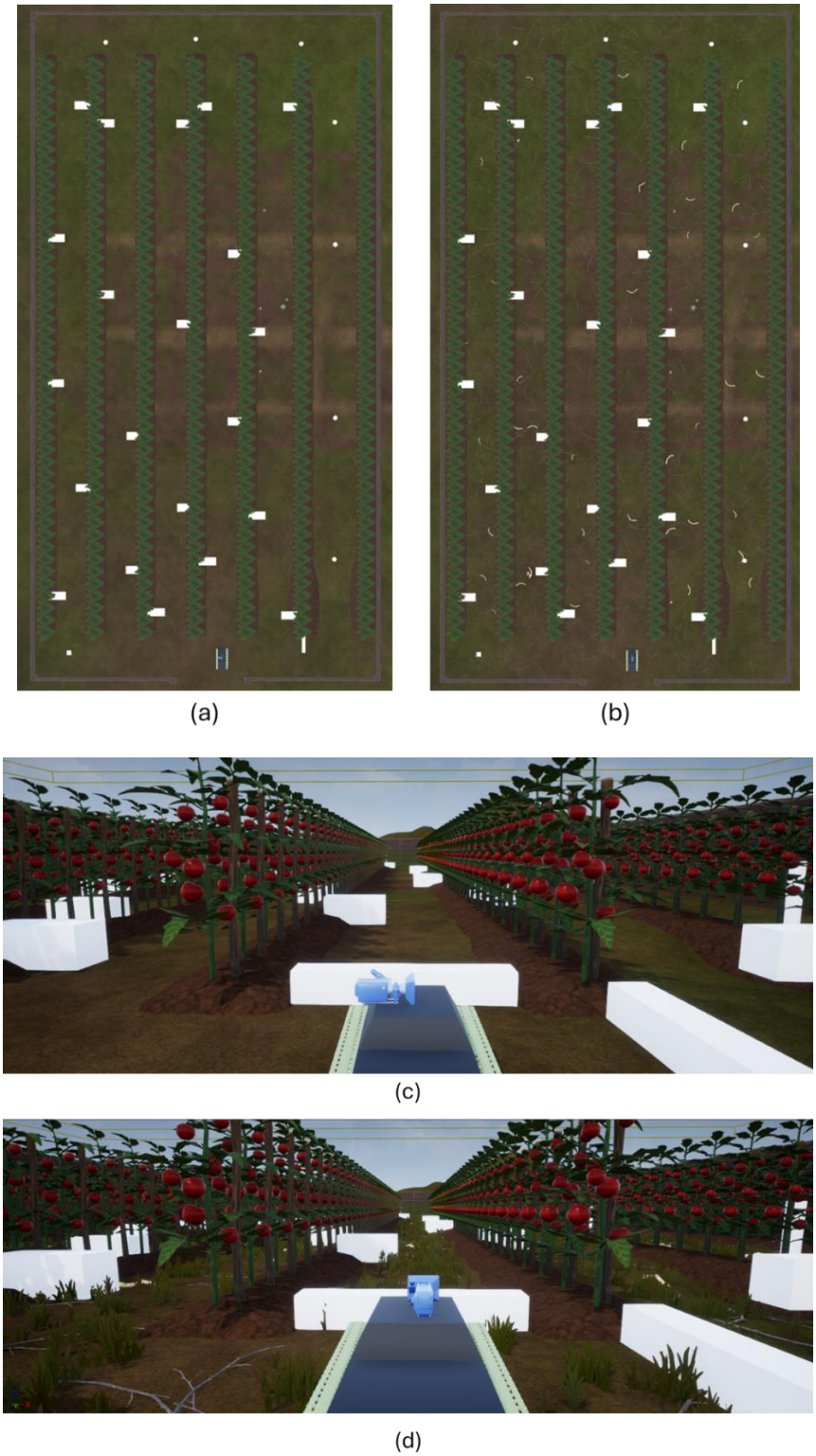



**Figura 74.** Comparativa entre el Escenario 2 y el Escenario 4. (a) Vista de planta del Escenario 2: normal con obstáculos. (b) Vista de planta del Escenario 4: irregular con obstáculos. (c) Vista frontal desde punto de partida del robot en el Escenario 2. (d) Vista frontal desde punto de partida del robot en el Escenario 4. **Fuente:** autor.

El cuarto escenario combina las características de los dos escenarios anteriores, integrando tanto irregularidades en el terreno como obstáculos externos. De esta manera, se crea un entorno más desafiante para el robot, que debe enfrentar tanto obstáculos que impiden el paso como irregularidades que afectan la estabilidad y el comportamiento de navegación. Las irregularidades del terreno son las mismas que en el escenario 3, mientras que los obstáculos son los mismos que en el escenario 2, tal como se observa en la Figura 74. Las Figura 74a y 74b muestran una vista de planta de ambos escenarios, permitiendo comparar cómo las irregularidades y los obstáculos afectan la distribución del entorno. Las Figura 74c y 74d muestran vistas frontales desde el punto de partida del robot en cada escenario, facilitando la comparación de las condiciones del terreno y la proximidad de los obstáculos.

Este escenario fusiona los desafíos del entorno realista con la necesidad de maniobras evasivas, evaluando de manera integral la capacidad del sistema de navegación autónoma para adaptarse a condiciones variadas y complejas. Se busca verificar que el robot pueda realizar un seguimiento preciso de las trayectorias, mientras evita obstáculos y ajusta su navegación según las irregularidades del terreno.

### 7.5.2. Selección de los sets de trayectorias.

Con el objetivo de abordar el cultivo de manera sistemática y evaluar la capacidad del robot para seguir diferentes trayectorias, se han establecido cuatro conjuntos de rutas. Todos los conjuntos tienen un punto de partida común. Desde este punto, el planificador genera la trayectoria hacia el primer objetivo, denominado P1. Una vez alcanzado P1, este punto se convierte en el nuevo punto de partida, permitiendo que el algoritmo replantee la ruta hacia el segundo objetivo, P2. El proceso se repite al alcanzar P2, estableciendo P3 como el siguiente destino. De este modo, cada conjunto de trayectorias comprende tres puntos objetivos secuenciales, lo que implica que el sistema genera tres trayectorias diferentes para cada conjunto.

**Tabla 11.** Coordenadas de cada uno de los sets de trayectorias implementados. **Fuente:** autor.

| | | X [m] | Y [m] | Θ [rad] |
|---|---|---|---|---|
| Punto de Partida | | 21.191 | 31.958 | $-\pi/2$ |
| Set 1 | P1 | 17.58 | 19.416 | $-\pi/2$ |
| | P2 | 21.202 | 25.227 | $\pi/2$ |
| | P3 | 23.566 | 31.967 | 0 |
| Set 2 | P1 | 20.58 | 25.616 | $-\pi/2$ |
| | P2 | 22.258 | 22.616 | $\pi/2$ |
| | P3 | 23.558 | 28.216 | $-\pi/2$ |
| Set 3 | P1 | 23.458 | 19.416 | $-\pi/2$ |
| | P2 | 22.358 | 29.716 | $\pi/2$ |
| | P3 | 21.158 | 21.216 | $-\pi/2$ |
| Set 4 | P1 | 18.958 | 19.816 | $-\pi/2$ |
| | P2 | 23.558 | 18.516 | 0 |
| | P3 | 21.258 | 32.016 | $\pi$ |

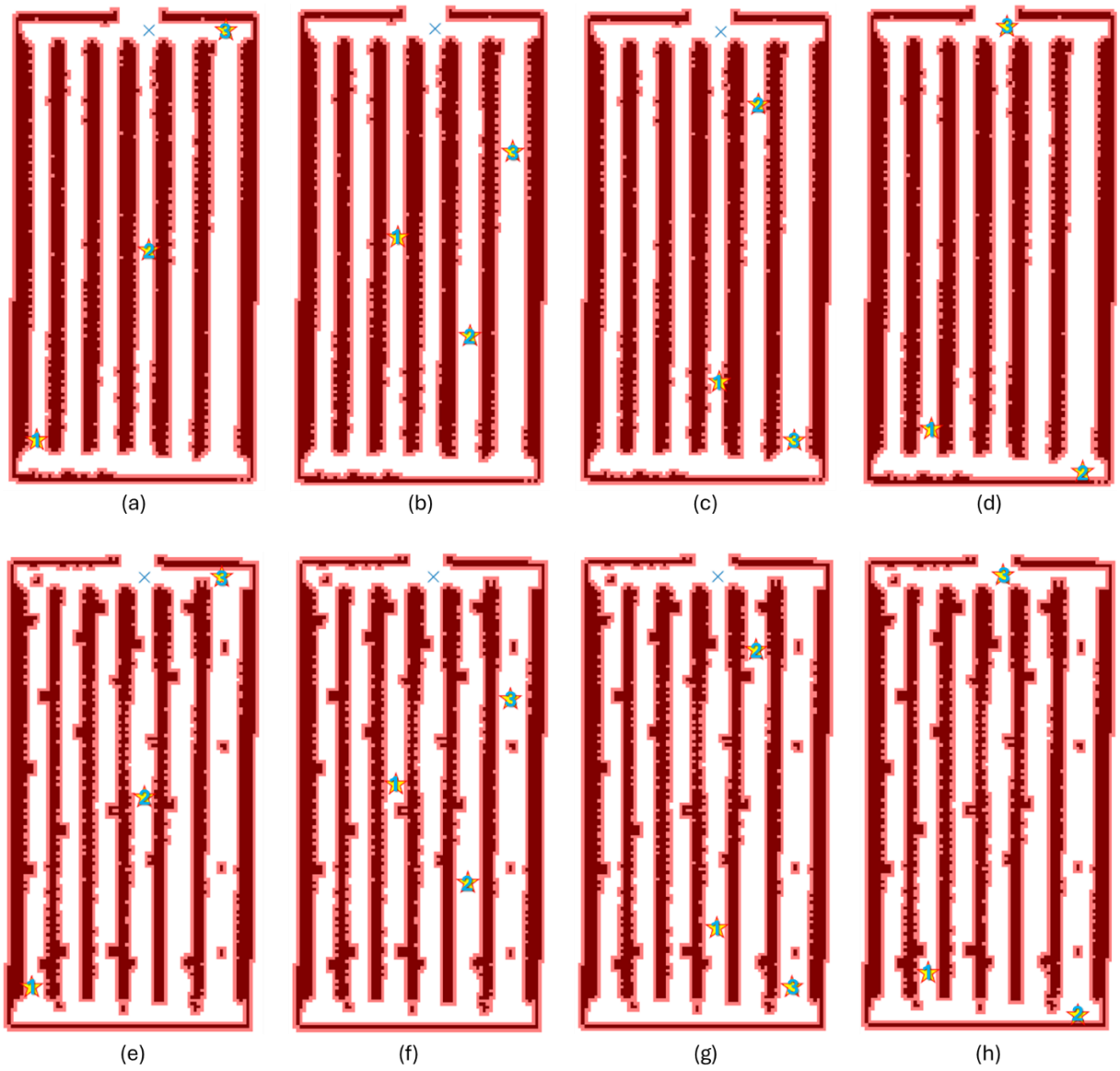



**Figura 75.** Posición en el mapa de los todos sets de trayectorias en los Escenarios 1 y 2. (a) Esc 1 – Set 1. (b) Esc 1 – Set 2. (c) Esc 1 – Set 3. (d) Esc 1 – Set 4. (e) Esc 2 – Set 1. (f) Esc 2 – Set 2. (g) Esc 2 – Set 3. (h) Esc 2 – Set 4.
**Fuente:** autor.

La Tabla 11 proporciona una descripción detallada de cada conjunto, especificando tanto los puntos objetivos como el punto de partida común. Asimismo, la Figura 75 ilustra las posiciones en el mapa de todos los puntos objetivos; la X representa el punto de inicio, mientras que las estrellas indican los objetivos P1, P2 y P3, cada uno de ellos identificado con un número diferenciador. Esta figura se divide en secciones (a) hasta (h): los mapas de 75(a) a 75(d) corresponden al Escenario 1, en el que el invernadero se encuentra libre de obstáculos adicionales, más allá de las hileras de tomate. Por otro lado, las imágenes de 75(e) a 75(h) representan la disposición de los mismos conjuntos en el Escenario 2, que incluye obstáculos de mayor tamaño. En todos los casos, los conjuntos de puntos objetivos permanecen constantes, lo que permite una comparación directa del desempeño del robot en cada escenario.

### 7.5.3. Diseño de las pruebas de validación.

Con el propósito de evaluar el desempeño del robot móvil tipo oruga en la generación de trayectorias óptimas, el seguimiento de las mismas y la evasión oportuna de obstáculos, se han definido una serie de pruebas de validación que abarcan los cuatro escenarios descritos en secciones previas. Este enfoque busca dar cumplimiento al cuarto objetivo específico, al comparar el comportamiento del robot utilizando dos versiones del algoritmo de planificación de trayectorias: A* Híbrido en su forma pura y A* Híbrido optimizado mediante el algoritmo TEB (*Timed Elastic Band*). De este modo, se pretende corroborar si la optimización introducida por TEB conlleva mejoras significativas en términos de precisión y eficiencia de la ruta.

Cada prueba consiste en situar al robot en el entorno virtual, configurado con uno de los cuatro escenarios, y ordenarle recorrer cada uno de los cuatro sets de trayectorias definidos. Para cada set, se generan internamente tres trayectorias (una hacia cada punto objetivo) y se supervisa el desempeño del robot en tiempo real. A lo largo del recorrido, se registra la trayectoria real seguida por el robot para comparar posteriormente dicho recorrido con la trayectoria ideal o planificada. Este análisis comparativo permite obtener métricas de desempeño, tales como el error promedio y el error RMS (*Root Mean Squared Error*), entre otras que serán descritas en la subsección.

En lo que respecta a la respuesta ante obstáculos, se considera que una prueba finaliza correctamente siempre y cuando el robot haya recorrido la totalidad de la trayectoria evitándolos de manera adecuada. No obstante, pueden ocurrir imprevistos externos al sistema que deriven en una colisión. Entre estos se incluyen inconsistencias en el mapeo, perturbaciones externas o una distancia insuficiente entre el límite del obstáculo y el centroide del robot, ocasionando el choque con alguna de sus partes. En caso de registrarse una colisión, la prueba se da por fallida y se descartan los datos asociados, pues no se habrá completado la ruta planificada. Con ello se asegura que únicamente se analicen trayectorias culminadas exitosamente, lo cual permite comparar de forma rigurosa el desempeño de ambas versiones del algoritmo de planificación.

#### 7.5.3.1. Métricas para el análisis de la trayectoria.

Con el objetivo de evaluar rigurosamente el desempeño del robot en términos de seguimiento de trayectoria, optimización de la ruta y evasión de obstáculos, se han seleccionado diversas métricas que permiten cuantificar la calidad de la navegación en cada uno de los escenarios planteados. A continuación, se clasifican y describen las métricas en tres grupos, detallando su definición y justificación dentro del proceso de validación.

**Indicadores de precisión en el seguimiento de la trayectoria:**

Esta categoría agrupa las métricas que permiten valorar cuán cercanas han sido las posiciones y orientaciones reales del robot respecto a la ruta ideal generada por el planificador. Su objetivo es medir la fidelidad con que el robot reproduce la trayectoria planificada, considerando tanto los errores lineales como los angulares a lo largo del recorrido.

**(a) Error promedio (m):** describe la distancia media entre la posición real del robot y la posición ideal en cada instante de la trayectoria. Para su cálculo, se toman todas las posiciones reales registradas $p_{real,i}$ y se comparan con sus contrapartes ideales $p_{ideal,i}$, obteniendo así un valor global que refleja la desviación típica del robot respecto a la ruta prevista, tal y como se puede observar en la siguiente ecuación:

$$Error\ Promedio = \frac{1}{N}\sum_{i=1}^{N}\left\|p_{real,i} - p_{ideal,i}\right\| \quad (26)$$

En donde $N$ es el total de puntos o muestras de la trayectoria. El error promedio brinda una medida global del desempeño del robot en la fidelidad de seguimiento de la ruta. Un valor reducido indica un seguimiento estable y uniforme.

**(b) Error RMSE (m):** el *Root Mean Squared Error* (RMSE, por sus siglas en inglés) mide la magnitud cuadrática media del error en la posición del robot en relación con la trayectoria ideal, expresada en metros y se calcula usando la siguiente ecuación:

$$Error\ RMSE = \sqrt{\frac{1}{N}\sum_{i=1}^{N}\left\|p_{real,i} - p_{ideal,i}\right\|^2} \quad (27)$$

A diferencia del error promedio, el RMSE penaliza en mayor medida las desviaciones grandes, lo cual permite identificar situaciones en las que el robot se desvía notablemente, aunque sea de forma esporádica.

**(c) Error promedio y error promedio normalizado (%):** es el error dividido entre la longitud total de la trayectoria planificada, multiplicado por 100 para expresarlo en porcentaje. De esta forma se facilita la comparación entre trayectorias al expresar el error en forma porcentual, obteniendo una visión proporcional de la desviación respecto a la extensión total del recorrido.

**(d) Error Máximo (m):** corresponde a la mayor distancia puntual entre la ruta real y la ruta ideal, medida en metros, a lo largo de todo el recorrido. Aun cuando el promedio y el RMSE proporcionan una visión global del error, resulta fundamental conocer la magnitud de las desviaciones extremas que el robot haya podido experimentar en un instante determinado.

**(e) Error de orientación (rad):** es la diferencia angular entre la orientación real del robot y la orientación ideal de la trayectoria planteada, medidas en radianes y se calcula usando la siguiente ecuación:

$$Error\ Orientación = \frac{1}{N}\sum_{i=1}^{N}\left\|\theta_{real,i} - \theta_{ideal,i}\right\| \quad (28)$$

Un bajo error de orientación suele correlacionarse con una navegación estable y precisa.

**(f) Máxima desviación angular (rad):** es el máximo valor de diferencia angular observada en cualquier instante. Esta métrica permite identificar momentos en los que el robot pudo haber realizado giros bruscos o perdido la orientación deseada, siendo un factor indicador de la suavidad del movimiento y la eficacia del planificador.

**Indicadores de eficiencia y optimización de la ruta:**

Este conjunto de métricas está orientado a determinar qué tan eficiente resulta la trayectoria generada y recorrida, tanto en términos de tiempo como de distancia. Su análisis resulta clave para valorar el efecto positivo de la optimización introducida por el algoritmo TEB en comparación con el A* Híbrido puro.

a) **Tiempo empleado (s):** se refiere al lapso total necesario para que el robot complete la ruta designada. Resulta de particular interés al juzgar la eficiencia de la navegación, pues una trayectoria que minimiza la distancia no siempre implica el menor tiempo de ejecución, especialmente si el robot debe reducir su velocidad para sortear obstáculos o negociar giros complejos.
b) **Distancia total real (m):** este indicador se basa en la suma de la longitud de todos los segmentos efectivamente recorridos por el robot durante la prueba. Su comparación con la distancia total planificada revela el grado de apego a la ruta ideal y la presencia de desvíos o rodeos no previstos.
c) **Distancia total planificada (m):** es la longitud de la trayectoria propuesta por el planificador antes del inicio del movimiento. Sirve como referencia para analizar la eficiencia de la ruta real y, en conjunto con la distancia real, permite medir el ajuste global entre lo planificado y lo ejecutado.
d) **Diferencia relativa entre distancia (%):** es la diferencia entre la distancia total real y la distancia total planificada, normalizada por esta última. El resultado, expresado en forma de porcentaje, permite apreciar la magnitud de la desviación de manera uniforme, independientemente de la longitud absoluta de la ruta. Un valor cercano a 0 % indica una elevada correspondencia entre lo planificado y lo recorrido, mientras que valores superiores señalan la existencia de desvíos considerables que podrían responder a correcciones de maniobra, errores de mapeo o perturbaciones externas.

# 8. RESULTADOS.

## 8.1. Resultados de las pruebas con el planificador A* Híbrido Puro.

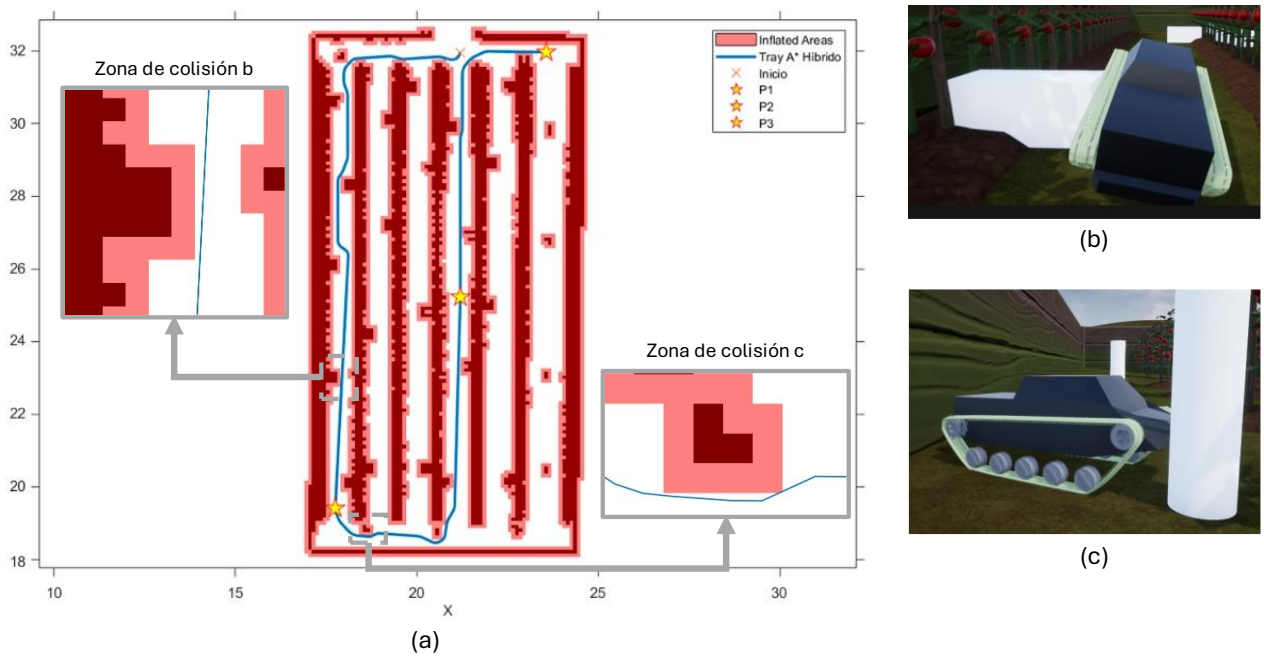



**Figura 76.** Demostración de los puntos de colisión en la prueba fallida con el algoritmo A* Híbrido Puro. (a) Mapa del Escenario 2 con la ruta planificada y los dos puntos de colisión respectivamente señalados. (b) Captura de la colisión en la zona b. (c) Captura de la colisión en la zona c. **Fuente:** autor.

Se llevaron a cabo las primeras pruebas de validación utilizando el algoritmo A* Híbrido Puro, seguidas de las pruebas con su versión optimizada mediante el algoritmo TEB, con el objetivo de comparar el desempeño de ambos planificadores. No obstante, las trayectorias generadas por el **A* Híbrido Puro** provocaron colisiones con los obstáculos externos previstos en los Escenarios 2 y 4 para los cuatro conjuntos de trayectorias evaluados. Como resultado, estas pruebas fueron consideradas fallidas y sus datos fueron descartados.

En la Figura 76 se presenta un mapa del Escenario 2 con una trayectoria planificada por el algoritmo A* Híbrido Puro para el primer conjunto de trayectorias. En dicho mapa se resaltan dos zonas específicas, denominadas ***zona de colisión b*** y ***zona de colisión c***, donde se registraron colisiones antes de detener la simulación. Estas denominaciones facilitan la identificación de las zonas en la Figura 76, la cual incluye capturas de las colisiones observadas durante la simulación.

El análisis de estas zonas reveló que el algoritmo A* Híbrido generó trayectorias demasiado próximas a los obstáculos, incluso considerando el margen de seguridad incorporado (representado en color rosado). Debido a las dimensiones del robot y su proximidad a los obstáculos, se produjeron colisiones que afectaron negativamente la navegación. En la **zona de colisión b**, el impacto ocurrió en uno de los laterales del robot (Figura 76b), lo que provocó una desviación de la trayectoria planificada; sin embargo, el robot logró retomar su curso. En contraste, en la **zona de colisión c**, el impacto fue frontal, lo que dejó al robot inmovilizado frente al obstáculo, incapaz de regresar a su trayectoria planificada (Figura 76c). Situaciones similares

se observaron en los demás conjuntos de trayectorias evaluados tanto en el Escenario 2 como en el Escenario 4.

Por otro lado, en los Escenarios 1 y 3, donde los únicos obstáculos presentes corresponden a las hileras del cultivo, el algoritmo A* Híbrido Puro no presentó colisiones directas con ningún elemento del entorno. Por lo tanto, las pruebas en estos escenarios se completaron exitosamente.

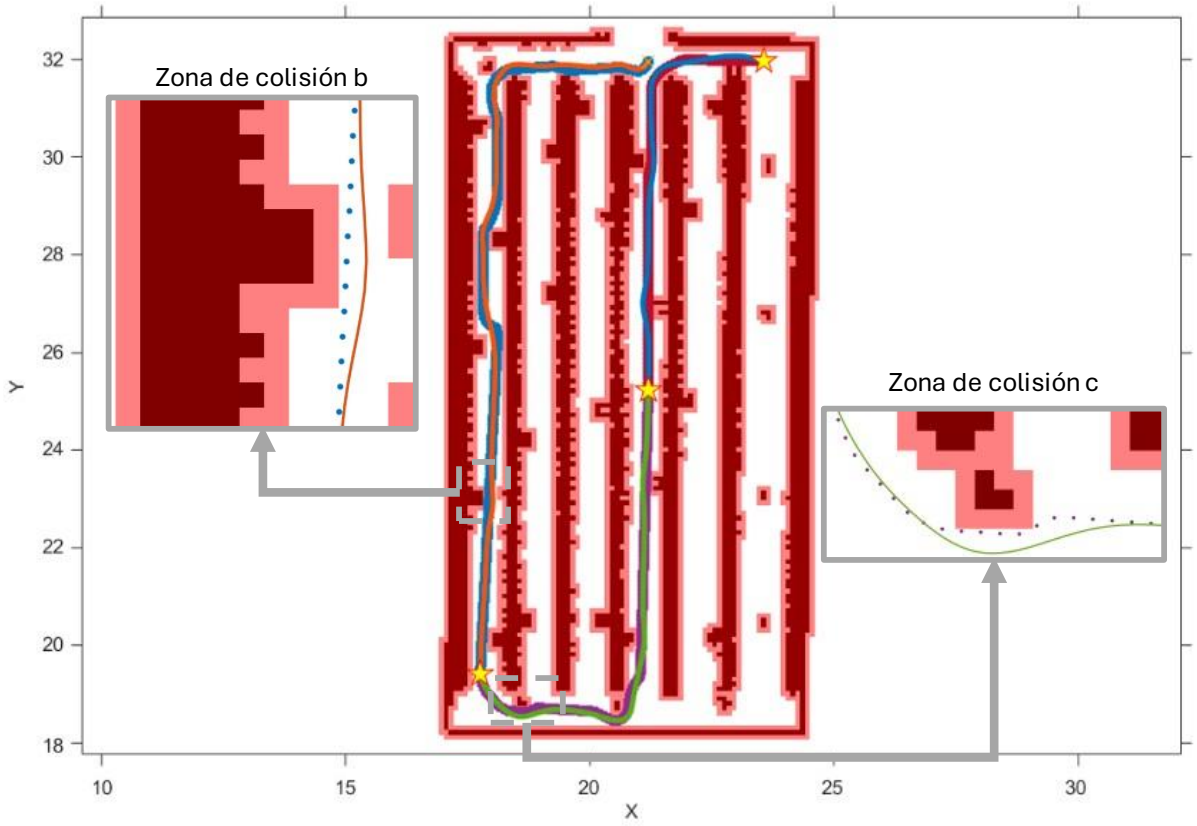



**Figura 77.** Comparación entre las trayectorias generadas por el algoritmo A* Híbrido Puro y A* Híbrido optimizado por TEB, en el Escenario 2. **Fuente:** autor.

En contraste, el planificador **A* Híbrido optimizado mediante el algoritmo TEB** incluye un margen de seguridad adicional frente a los obstáculos, además del margen ya incorporado a los bordes del mapa. Esta característica le permitió completar satisfactoriamente todas las pruebas en los cuatro escenarios planteados para los cuatro conjuntos de trayectorias.

La Figura 77 ilustra una comparación entre las trayectorias generadas por ambos planificadores. Las trayectorias del algoritmo A* Híbrido Puro se representan con líneas punteadas, mientras que las del algoritmo A* Híbrido optimizado se muestran con líneas continuas. Aunque a primera vista las diferencias parecen mínimas, un análisis detallado de las zonas b y c, donde el A* Híbrido Puro presentó colisiones, evidencia que el planificador optimizado genera trayectorias con un mayor distanciamiento respecto a los obstáculos. Esta mejora significativa permitió al A* Híbrido Optimizado navegar con éxito en todos los escenarios evaluados.

A continuación, se presentan los resultados del algoritmo A* Híbrido Puro en los únicos escenarios donde las pruebas pudieron completarse.

### 8.1.1. Resultados del A* Híbrido Puro en los Escenarios 1 y 3.

- **Escenario 1:**

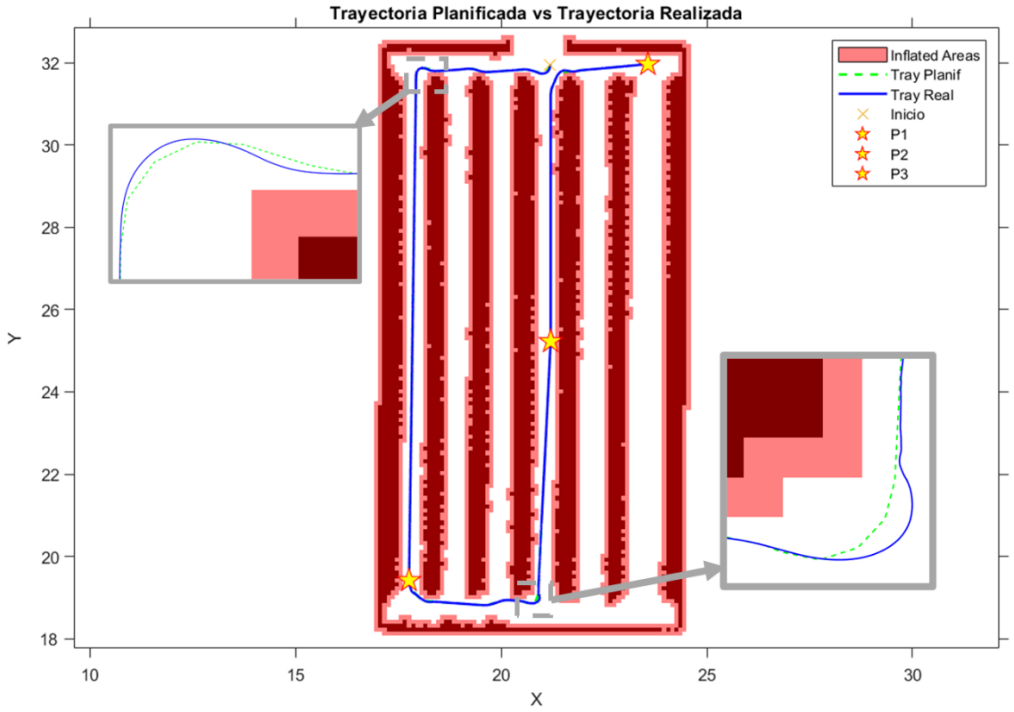



**Figura 78.** A* Híbrido Puro: Escenario 1 – Set 1. **Fuente:** autor.

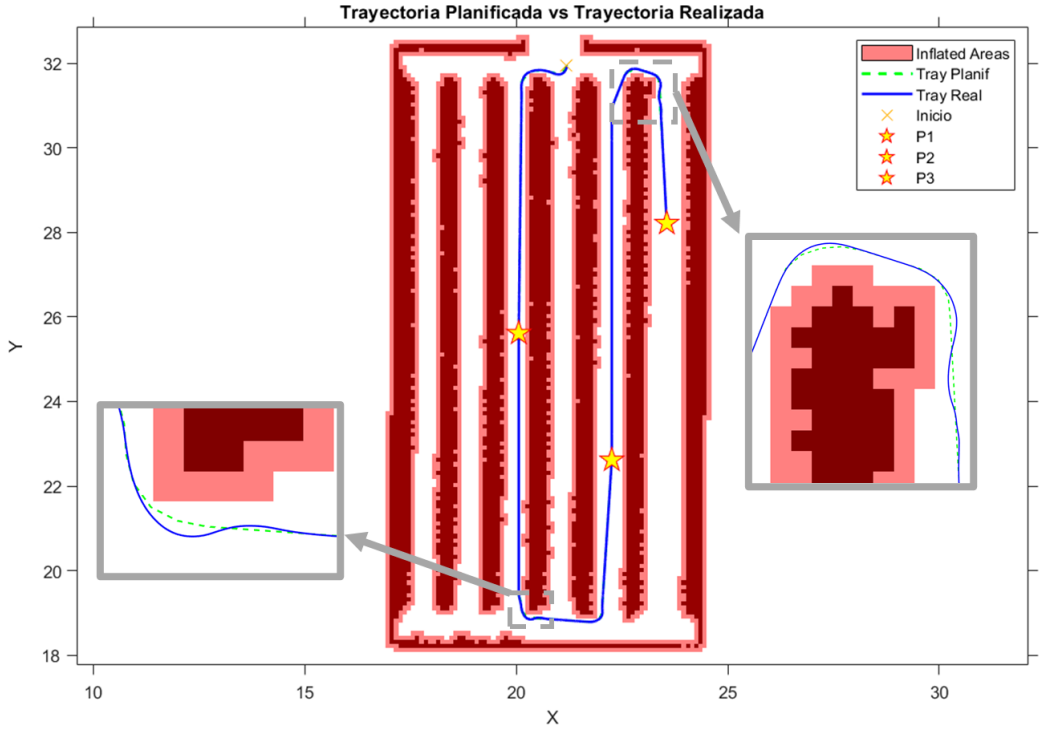



**Figura 79.** A* Híbrido Puro: Escenario 1 – Set 2. **Fuente:** autor.

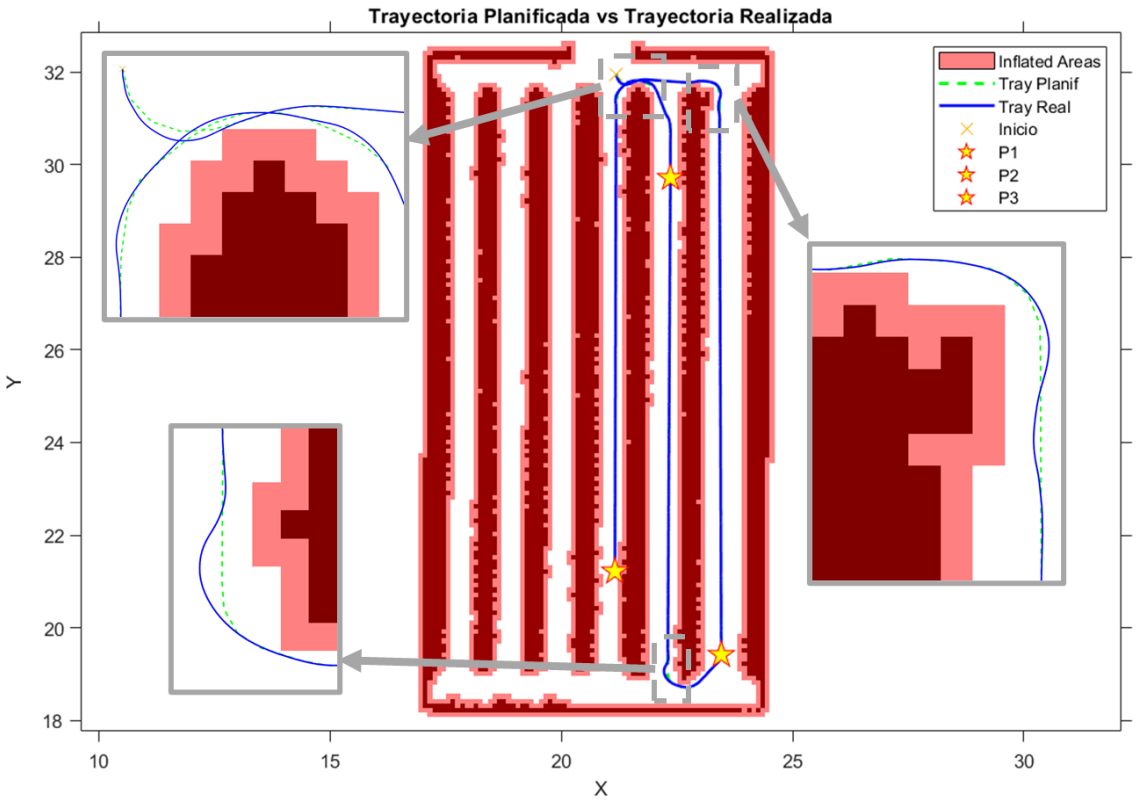



**Figura 80.** A* Híbrido Puro: Escenario 1 – Set 3**. Fuente:** autor.

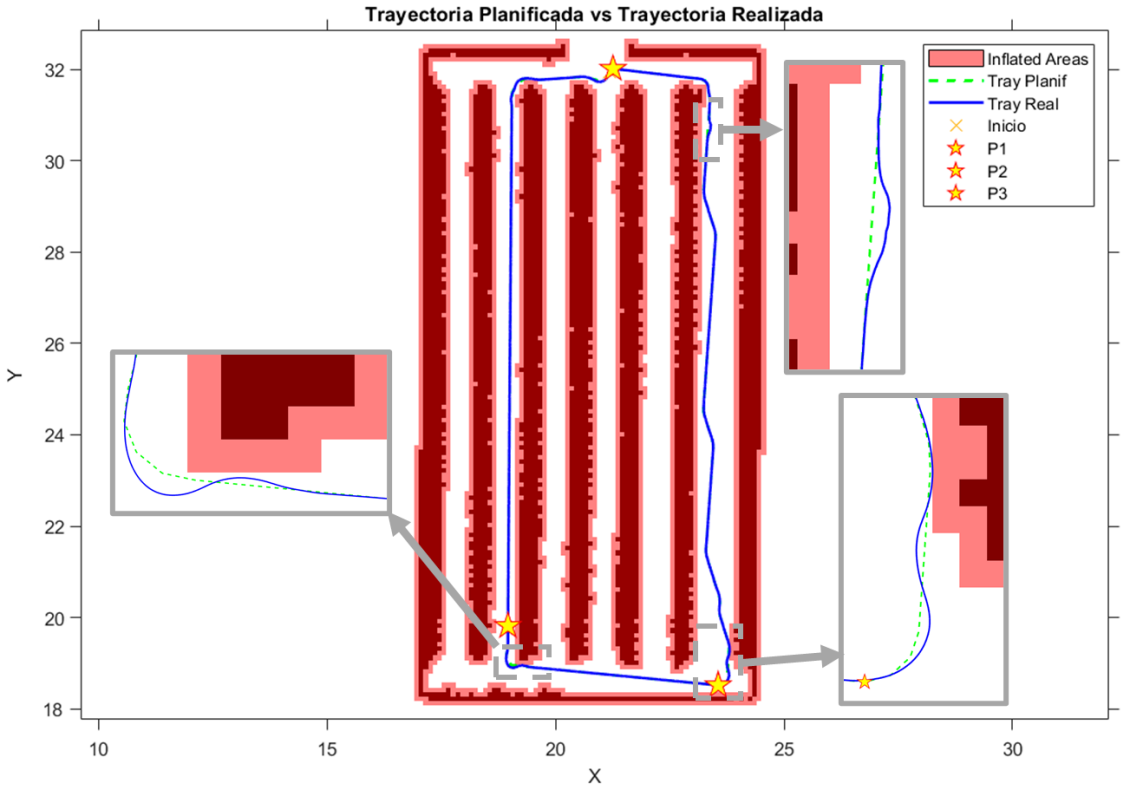



**Figura 81.** A* Híbrido Puro: Escenario 1 – Set 4**. Fuente:** autor.

- **Escenario 3:**

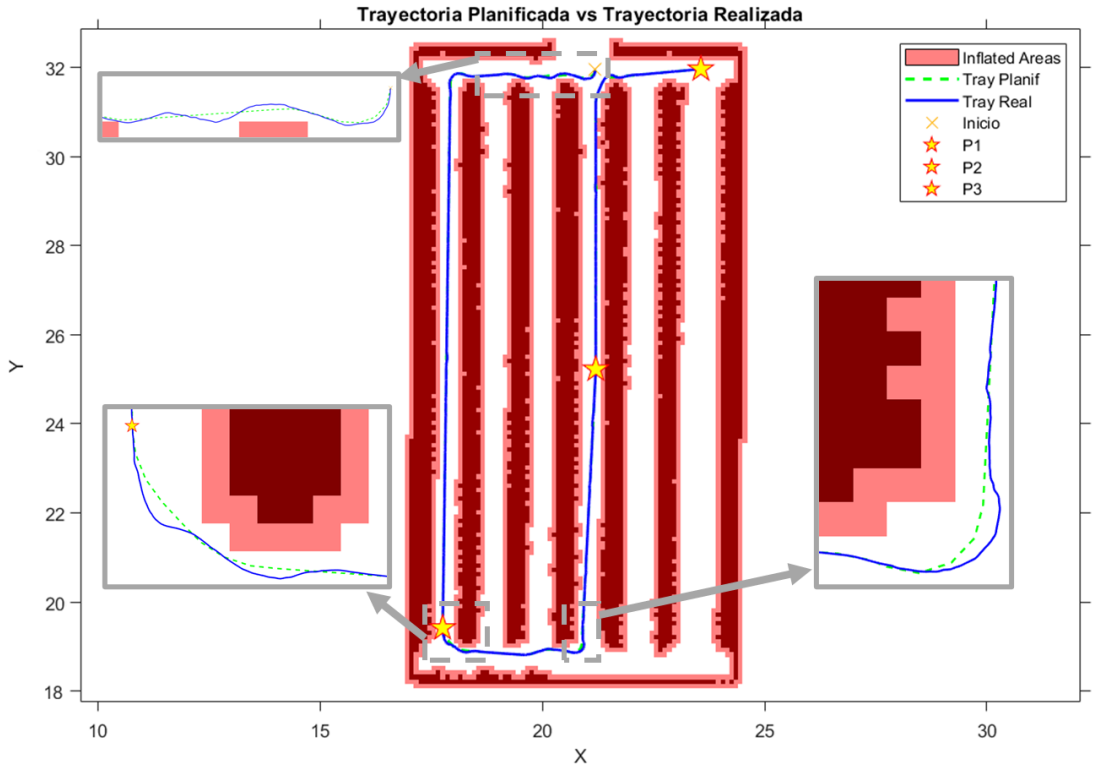



**Figura 82.** A* Híbrido Puro: Escenario 3 – Set 1. **Fuente:** autor.

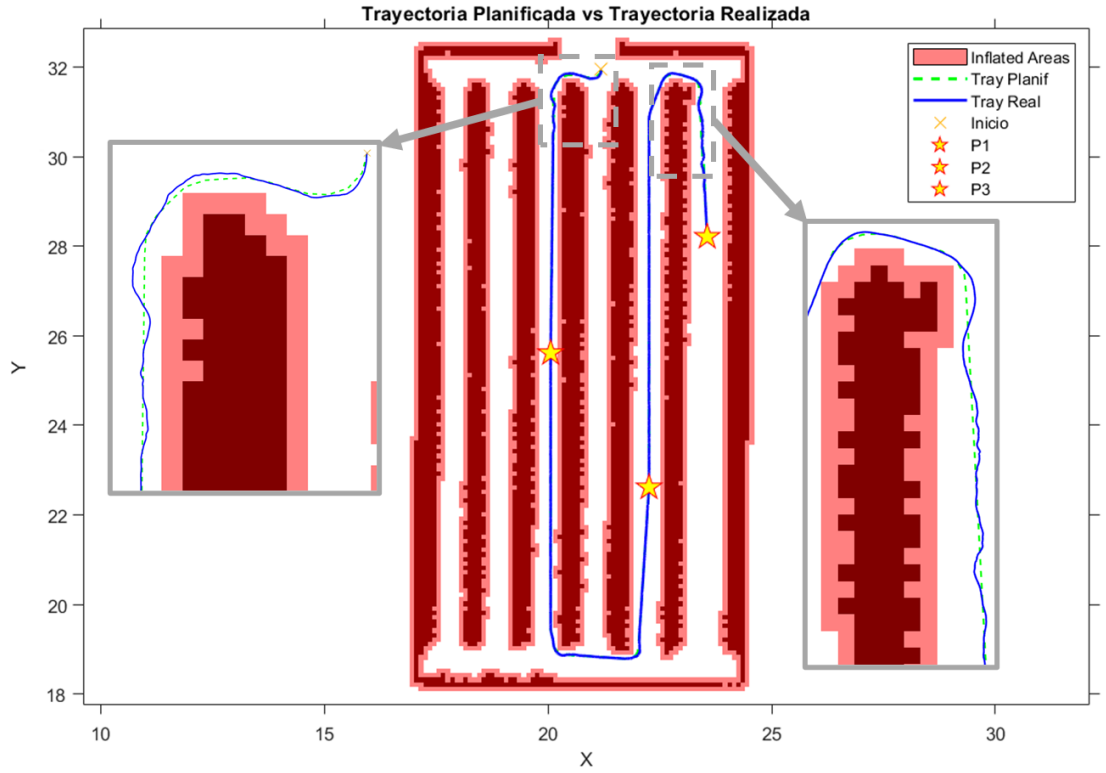



**Figura 83.** A* Híbrido Puro: Escenario 3 – Set 2. **Fuente:** autor.

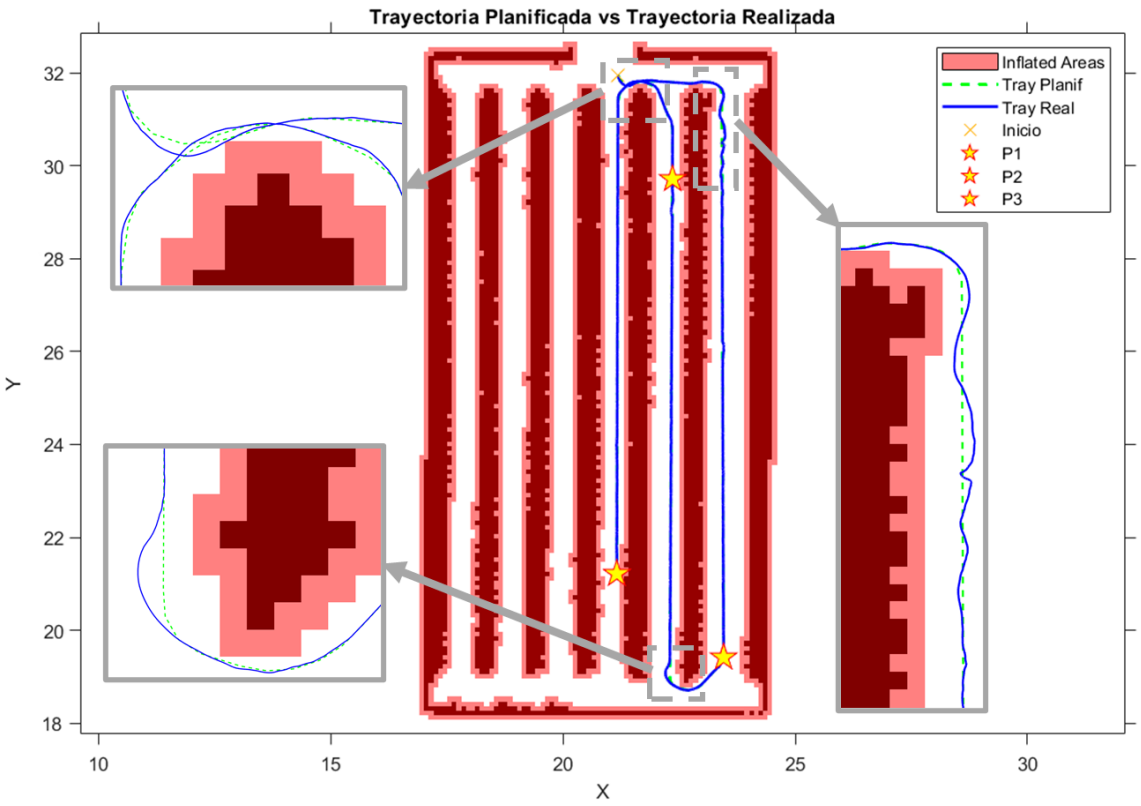



**Figura 84.** A* Híbrido Puro: Escenario 3 – Set 3. **Fuente:** autor.

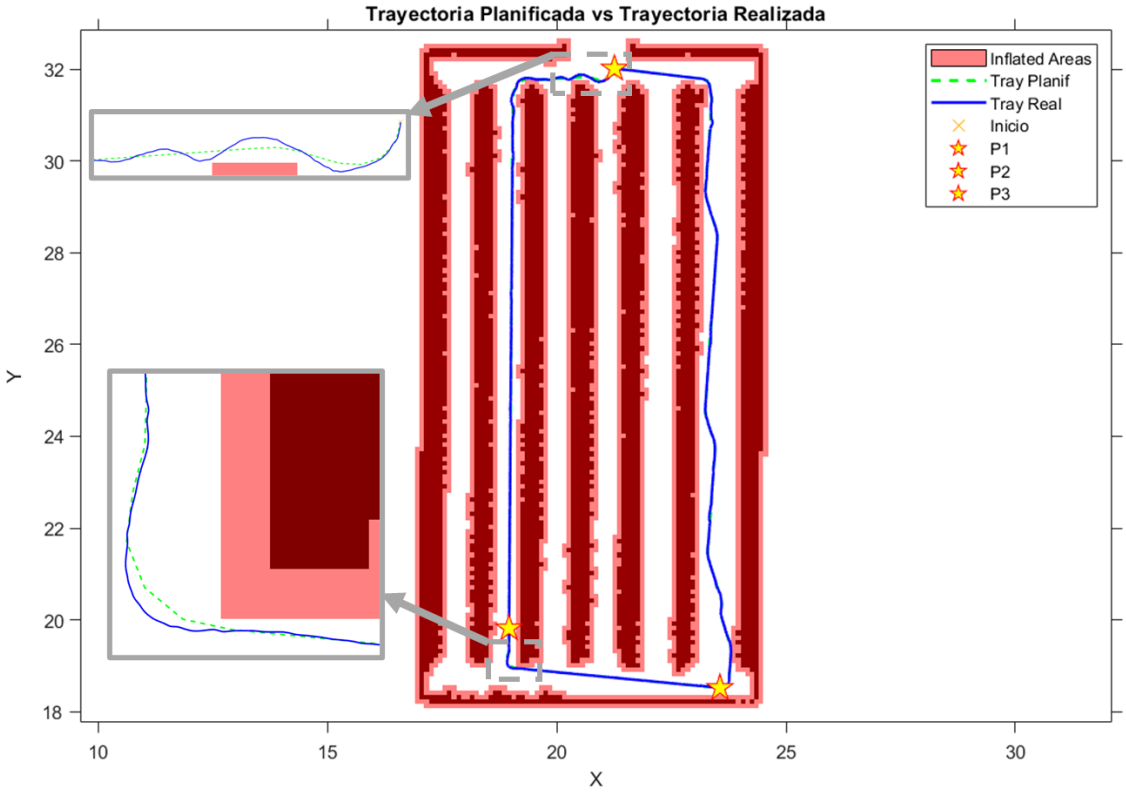



**Figura 85.** A* Híbrido Puro: Escenario 3 – Set 4. **Fuente:** autor.

- **Resultados y métricas obtenidas con el planificador A* Híbrido Puro:**

**Tabla 12.** Indicadores de precisión en el seguimiento de la trayectoria para el planificador A* Híbrido Puro. **Fuente:** autor.

| A* Híbrido Puro | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| Escenario | Set | Error Promedio [m] | Error Promedio [%] | Error RMSE [m] | Error RMSE [%] | Error MAX [m] | Error Orien [rad] | Máx. Desv. Angular [rad] |
| E1: normal sin obstáculos | S1 | 0.0209 | 0.06 | 0.0251 | 0.07 | 0.0719 | 0.0824 | 0.6188 |
| | S2 | 0.0146 | 0.04 | 0.018 | 0.05 | 0.0476 | 0.0846 | 0.3142 |
| | S3 | 0.0335 | 0.08 | 0.0354 | 0.09 | 0.0887 | 0.1009 | 0.6193 |
| | S4 | 0.0313 | 0.09 | 0.0364 | 0.1 | 0.071 | 0.0749 | 0.5895 |
| E3: Irregular sin obstáculos | S1 | 0.0261 | 0.08 | 0.0314 | 0.09 | 0.0706 | 0.0829 | 0.6198 |
| | S2 | 0.0334 | 0.1 | 0.0414 | 0.13 | 0.0903 | 0.0855 | 0.3626 |
| | S3 | 0.0931 | 0.23 | 0.1002 | 0.23 | 0.1567 | 0.1038 | 0.5919 |
| | S4 | 0.0678 | 0.19 | 0.0731 | 0.21 | 0.1209 | 0.0755 | 0.619 |

**Tabla 13.** Indicadores de eficiencia y optimización de la ruta para el planificador A* Híbrido Puro. **Fuente:** autor.

| A* Híbrido Puro | | | | | |
|---|---|---|---|---|---|
| Escenario | Set | Tiempo Empleado [s] | Dist Total REAL [m] | Dist Total Ideal [m] | Diferencia relativa [%] |
| E1: normal sin obstáculos | S1 | 179.533 | 34.3976 | 34.2337 | 0.478767998 |
| | S2 | 173.183 | 33.1359 | 33.0189 | 0.354342513 |
| | S3 | 207.033 | 40.3787 | 40.2716 | 0.265944239 |
| | S4 | 184 | 35.2622 | 35.1062 | 0.444365952 |
| E3: Irregular sin obstáculos | S1 | 179.533 | 34.5107 | 34.2337 | 0.809144206 |
| | S2 | 187.667 | 33.3149 | 33.0189 | 0.896456272 |
| | S3 | 211 | 40.6034 | 40.2716 | 0.82390568 |
| | S4 | 185.167 | 35.3936 | 35.1062 | 0.818658812 |

Las Figura 78 a Figura 85 presentan una comparación de las trayectorias generadas por el algoritmo A* Híbrido para cada uno de los conjuntos de trayectorias seleccionados en los dos únicos escenarios donde las pruebas pudieron completarse satisfactoriamente. En estas figuras, la línea punteada verde representa la trayectoria original generada por el planificador, mientras que la línea continua azul indica la trayectoria efectivamente seguida por el robot en el entorno virtual. Los indicadores de precisión y eficacia óptima obtenidos durante estas pruebas se resumen en las Tabla 12 y Tabla 13, respectivamente.

A pesar de que las trayectorias generadas por el algoritmo A* Híbrido presentan un leve zigzagueo, especialmente notable en el conjunto 4, el planificador logró completar las pruebas de manera eficiente. Las trayectorias no mostraron movimientos bruscos significativos y los errores en el seguimiento se mantuvieron en niveles mínimos, incluso ante las irregularidades del terreno en el Escenario 3. En este último, se observó una ligera variación en el tiempo requerido para completar cada conjunto de trayectorias en comparación con el Escenario 1. Aunque la diferencia relativa en las distancias recorridas se duplicó en la mayoría de los conjuntos, ningún valor superó el 1%.

Las irregularidades del terreno, como hojas y rocas, no afectaron significativamente el desempeño del robot en el seguimiento de la trayectoria ni en la estabilidad de la ruta. Aunque estas condiciones generaron ligeras ondulaciones en las trayectorias realizadas y un incremento mínimo en el error, este se mantuvo por debajo del 0.3%, sin impacto relevante en el rendimiento general.

## 8.2. Resultados de las pruebas con el planificador A* Híbrido optimizado mediante el algoritmo TEB.

- **Escenario 1:**

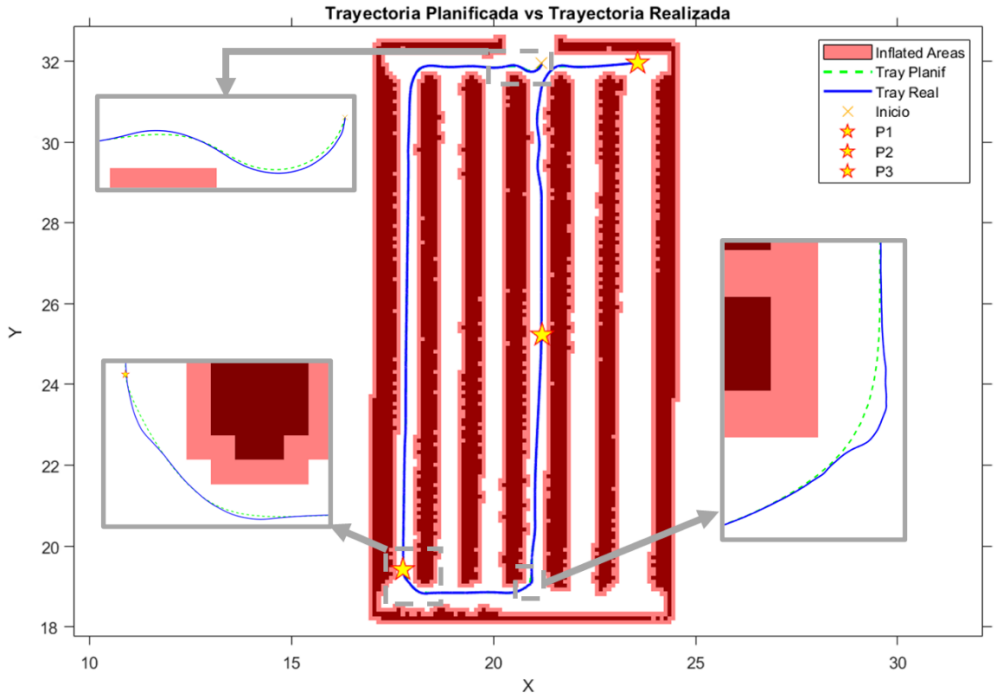



**Figura 86.** A* Híbrido + Optimización TEB: Escenario 1 – Set 1. **Fuente:** autor.

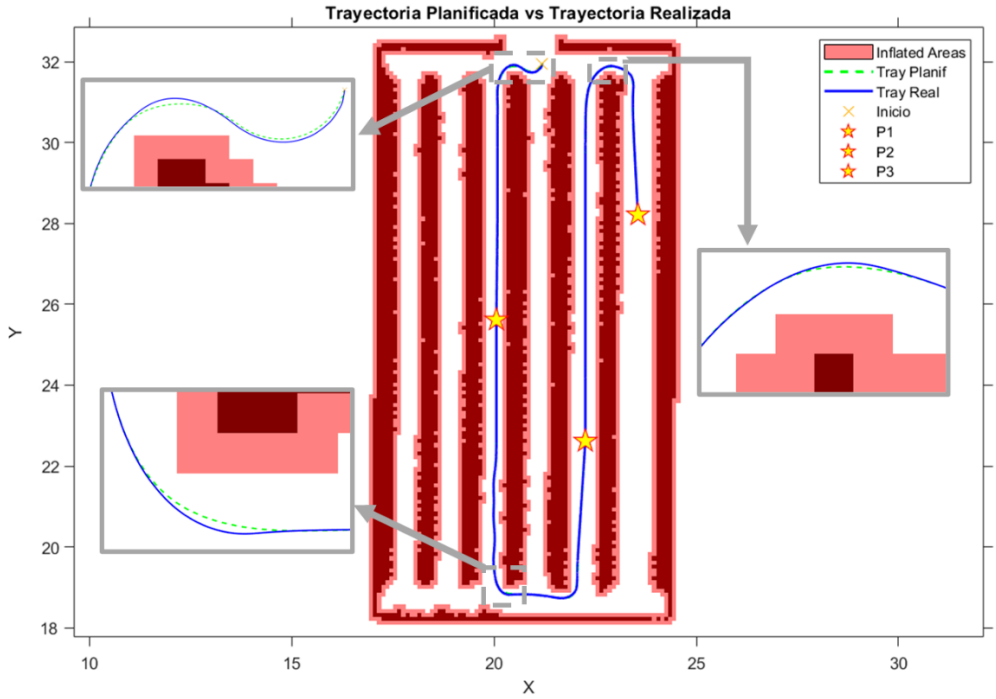



**Figura 87.** A* Híbrido + Optimización TEB: Escenario 1 – Set 2. **Fuente:** autor.

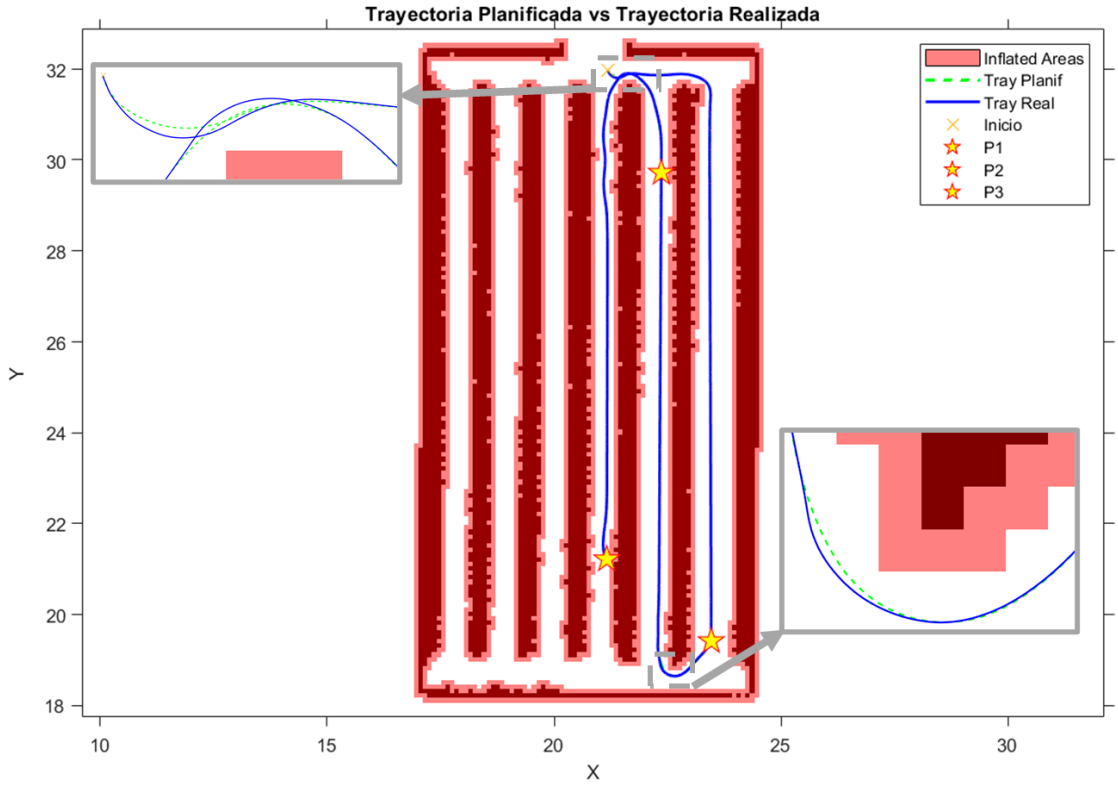



**Figura 88.** A* Híbrido + Optimización TEB: Escenario 1 – Set 3. **Fuente:** autor.

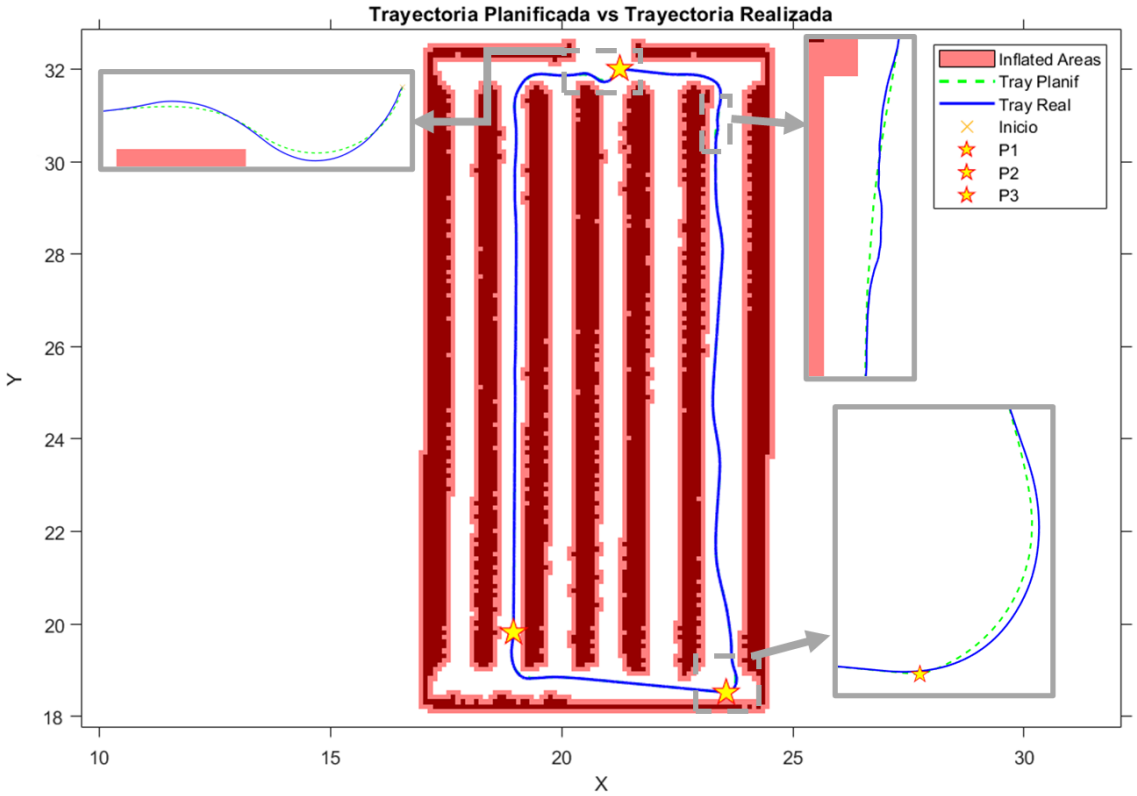



**Figura 89.** A* Híbrido + Optimización TEB: Escenario 1 – Set 4. **Fuente:** autor.

- **Escenario 2:**

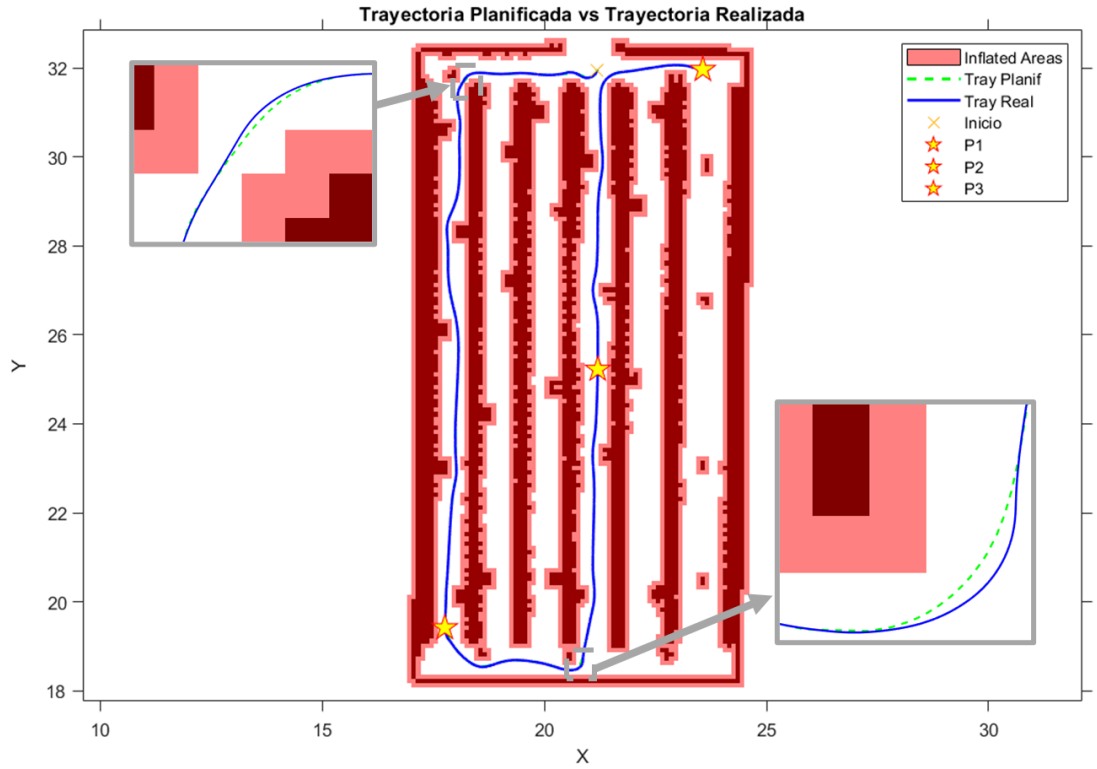



**Figura 90.** A* Híbrido + Optimización TEB: Escenario 2 – Set 1. **Fuente:** autor.

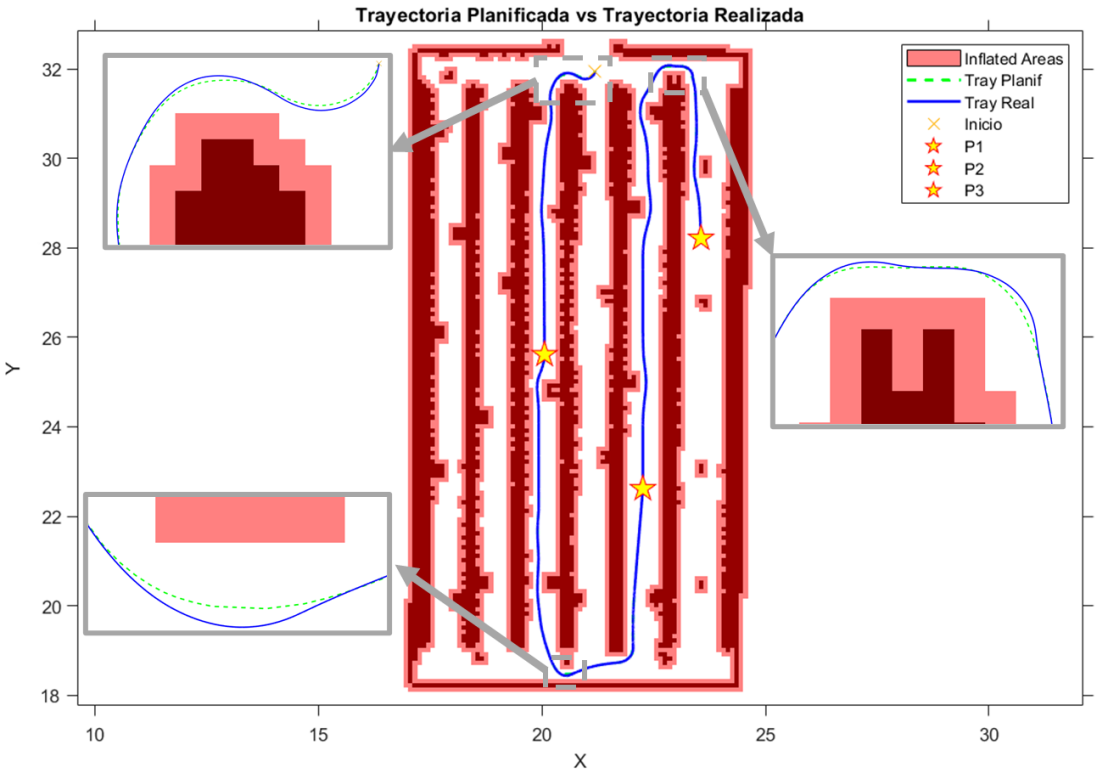



**Figura 91.** A* Híbrido + Optimización TEB: Escenario 2 – Set 2. **Fuente:** autor.

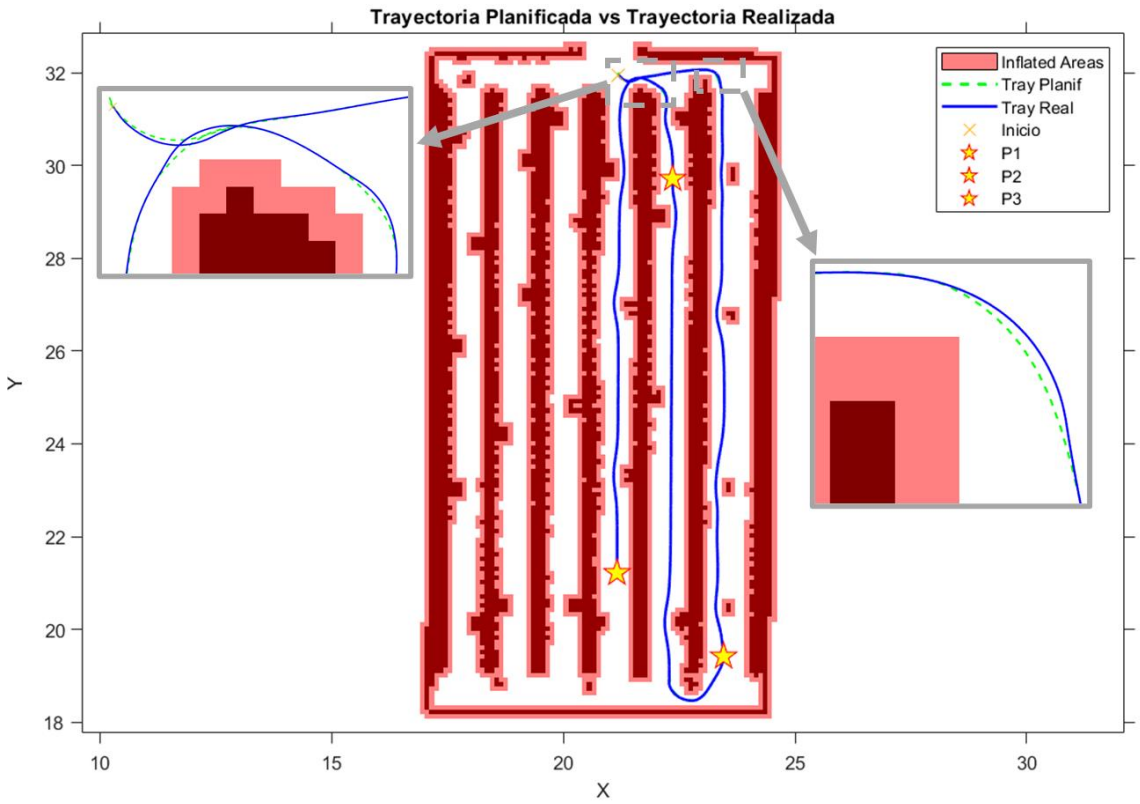



**Figura 92.** A* Híbrido + Optimización TEB: Escenario 2 – Set 3. **Fuente:** autor.

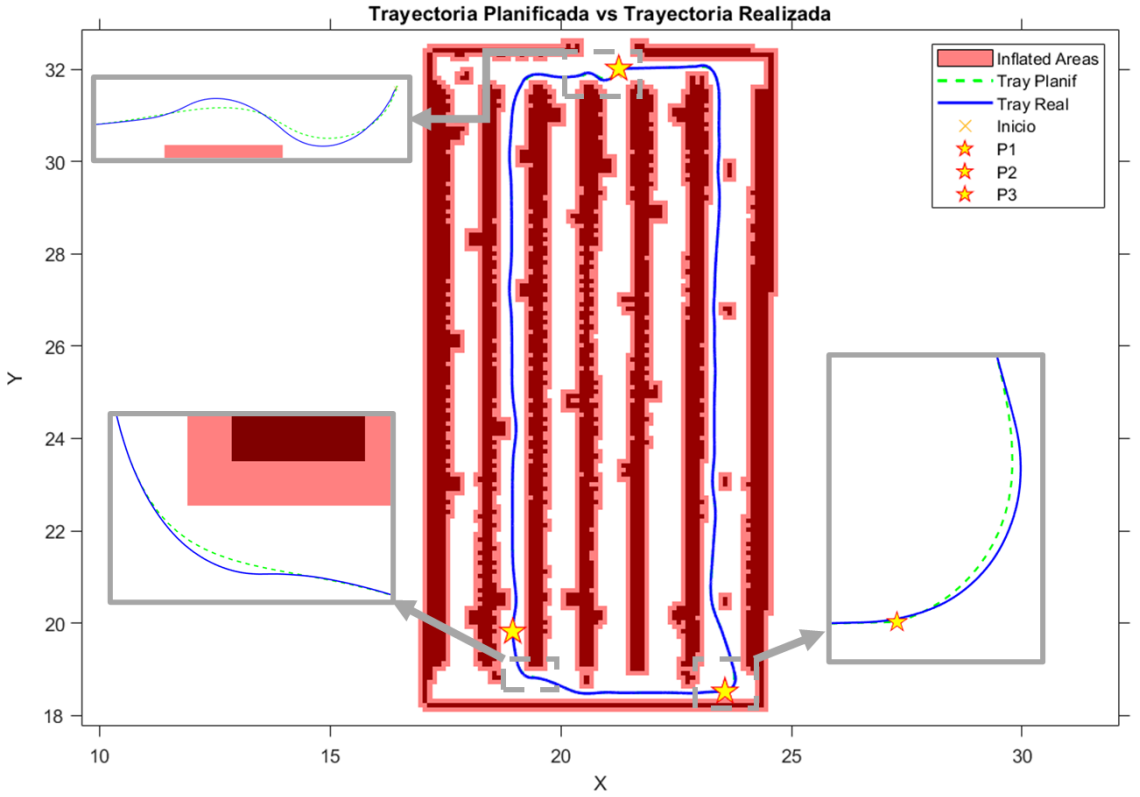



**Figura 93.** A* Híbrido + Optimización TEB: Escenario 2 – Set 4. **Fuente:** autor.

- **Escenario 3:**

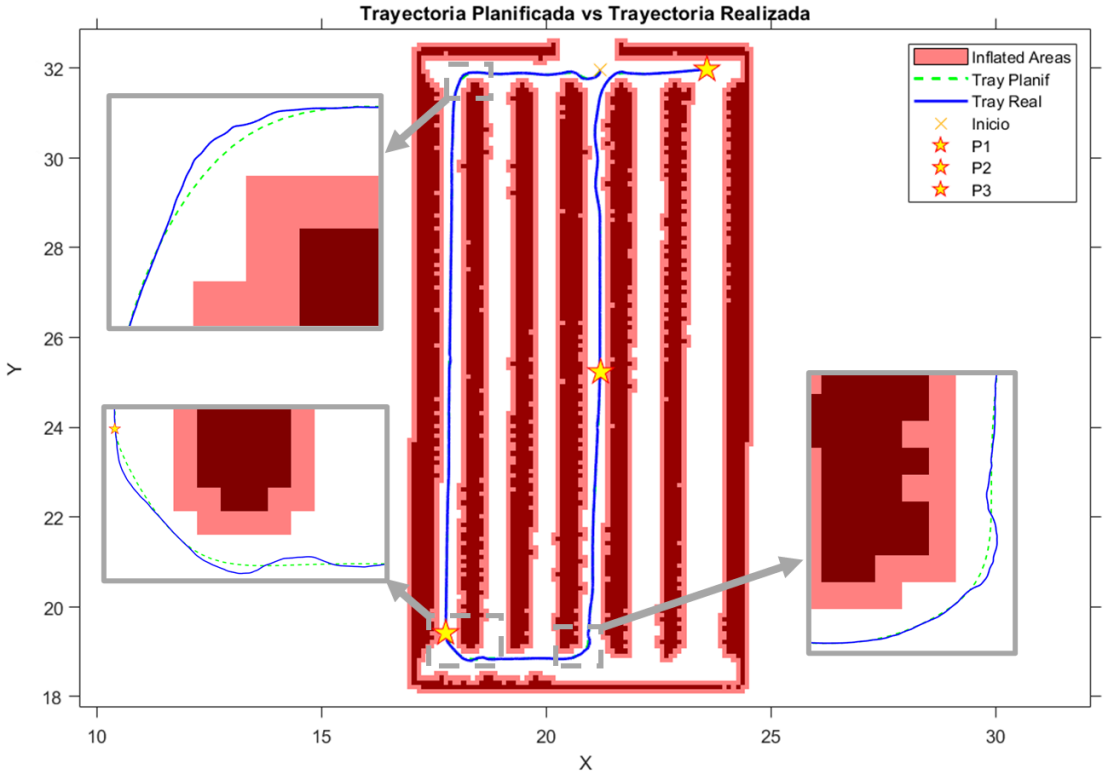



**Figura 94.** A* Híbrido + Optimización TEB: Escenario 3 – Set 1. **Fuente:** autor.

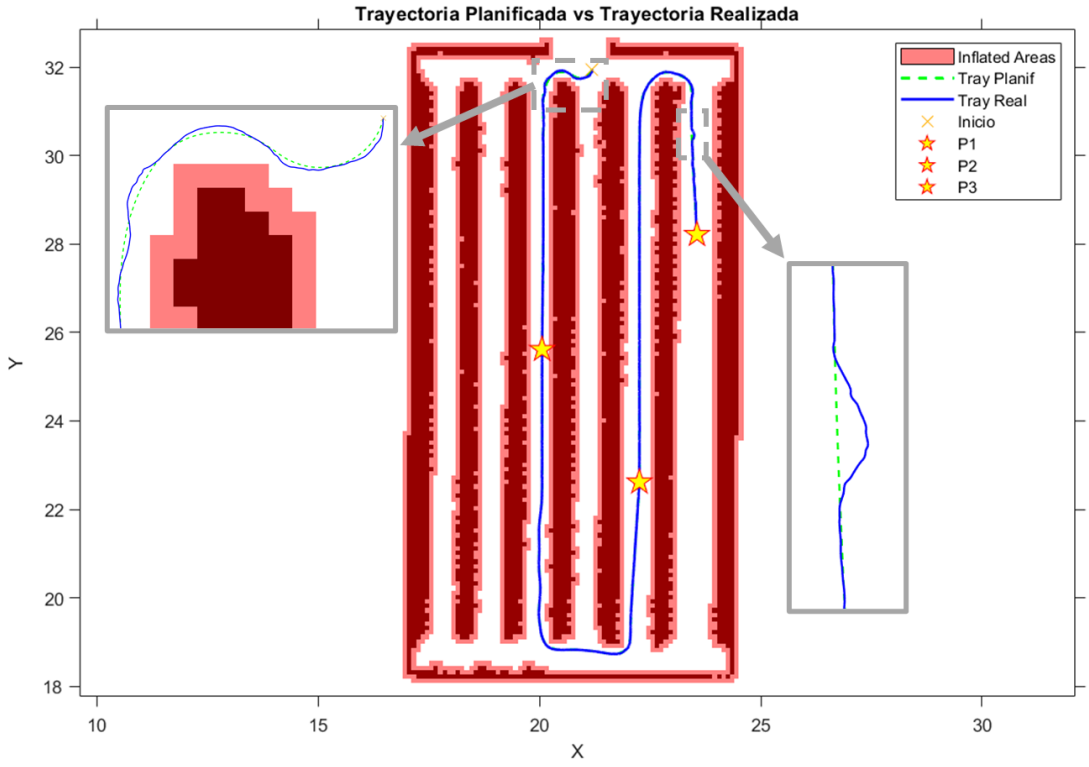



**Figura 95.** A* Híbrido + Optimización TEB: Escenario 3 – Set 2. **Fuente:** autor.

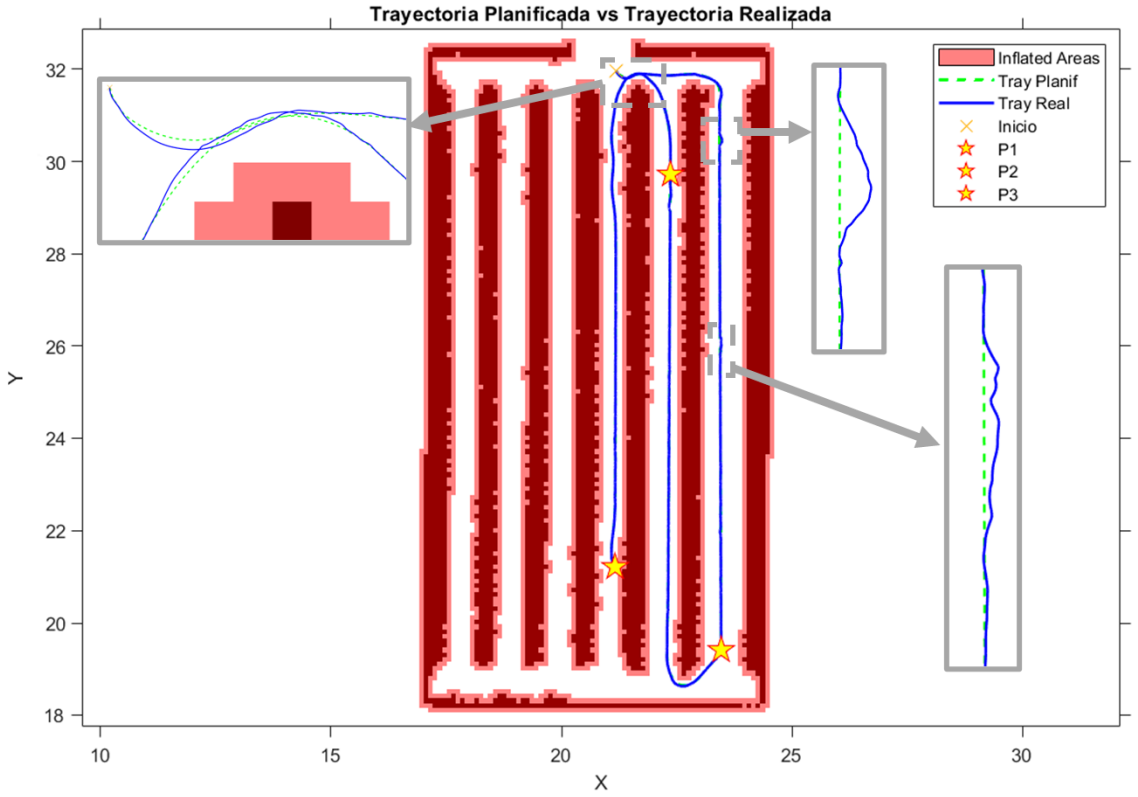



**Figura 96.** A* Híbrido + Optimización TEB: Escenario 3 – Set 3. **Fuente:** autor.

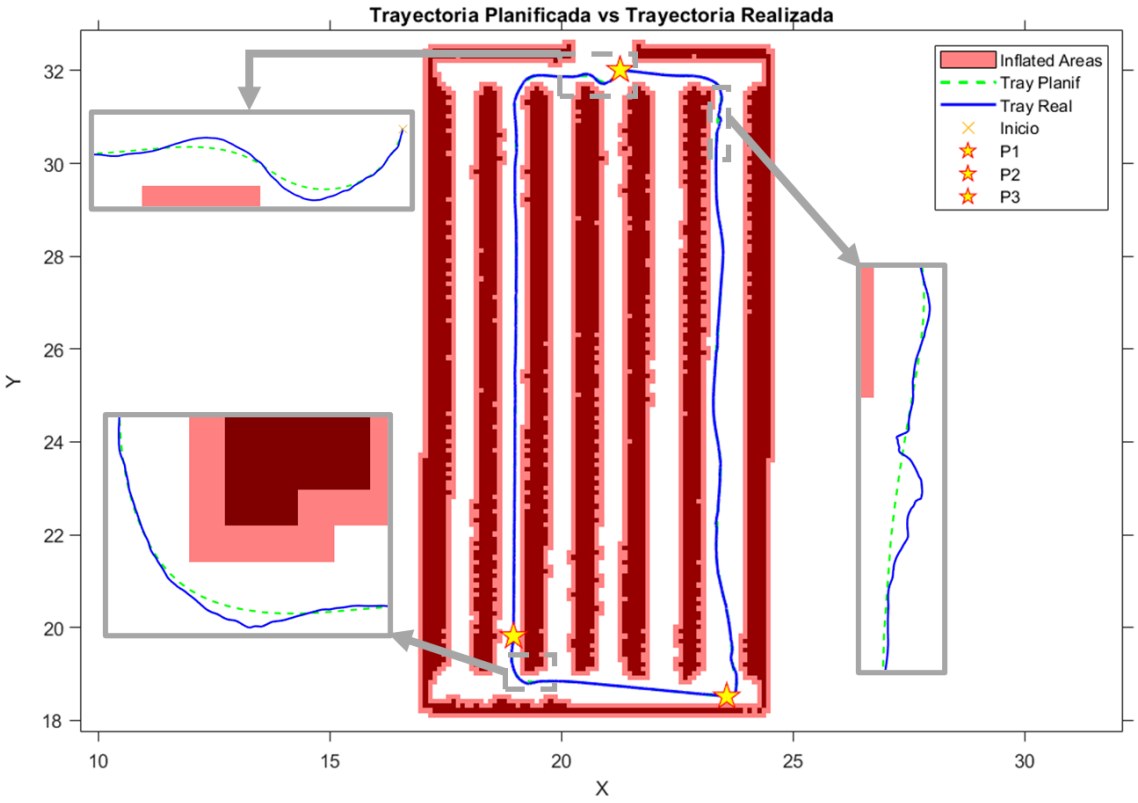



**Figura 97.** A* Híbrido + Optimización TEB: Escenario 3 – Set 4. **Fuente:** autor.

- **Escenario 4:**

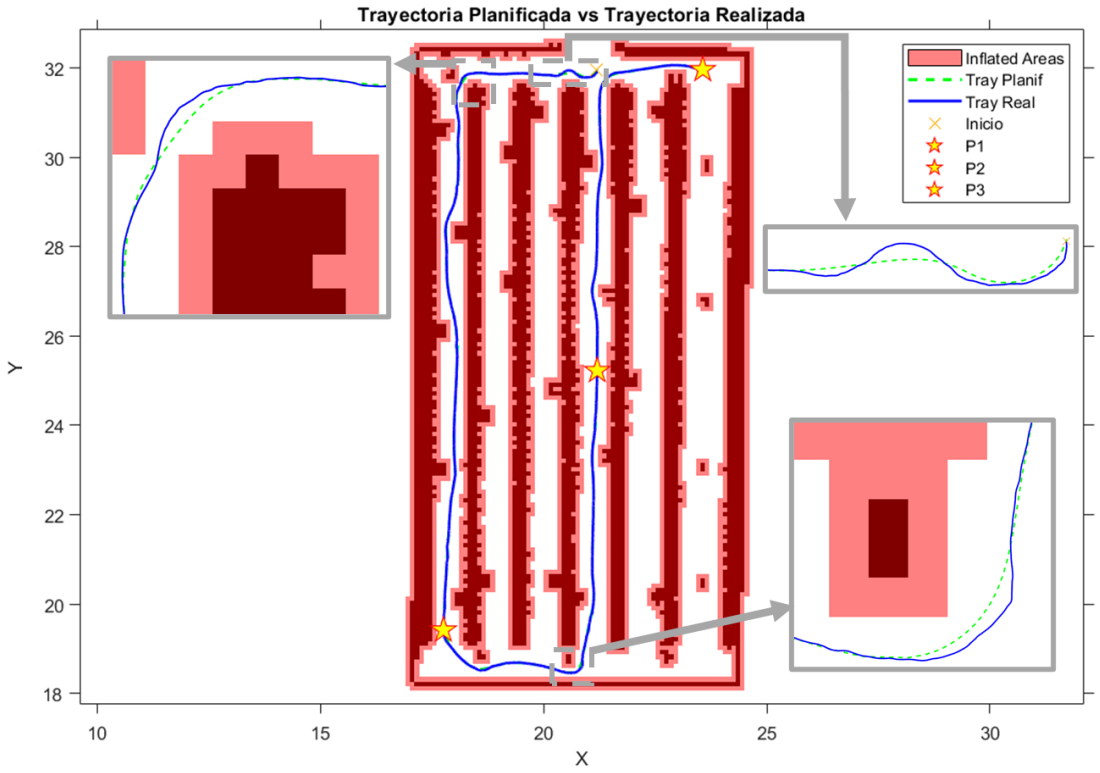



**Figura 98.** A* Híbrido + Optimización TEB: Escenario 4 – Set 1. **Fuente:** autor.

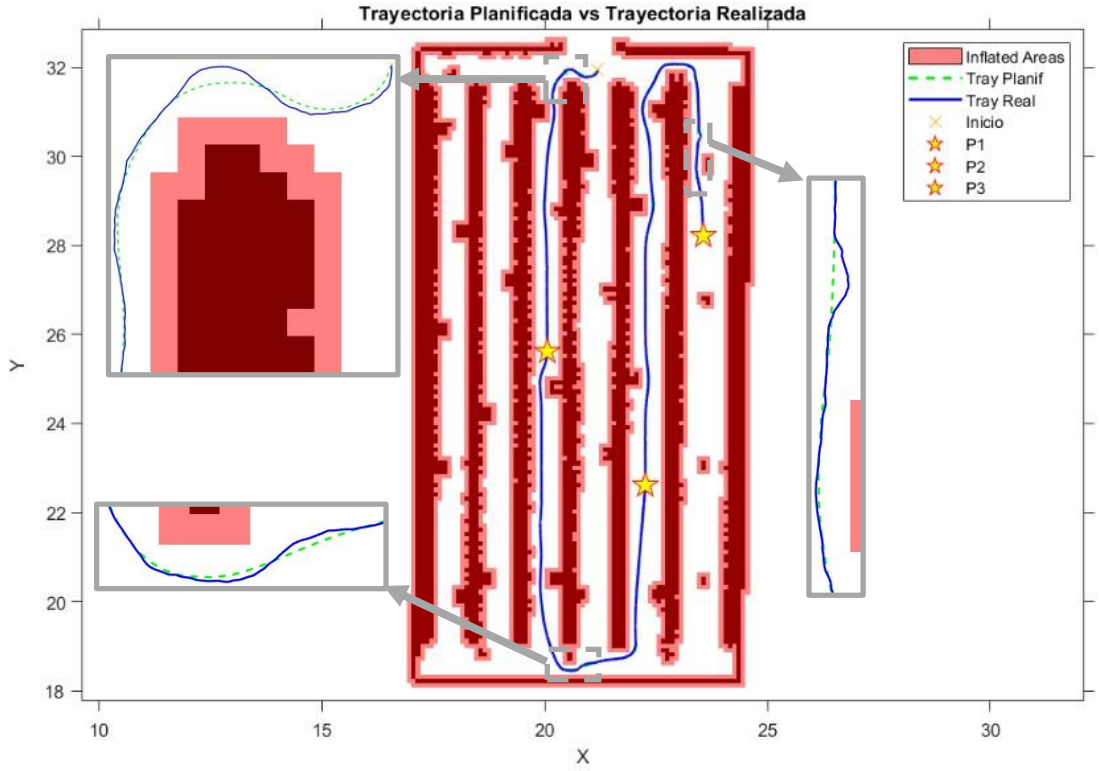



**Figura 99.** A* Híbrido + Optimización TEB: Escenario 4 – Set 2. **Fuente:** autor.

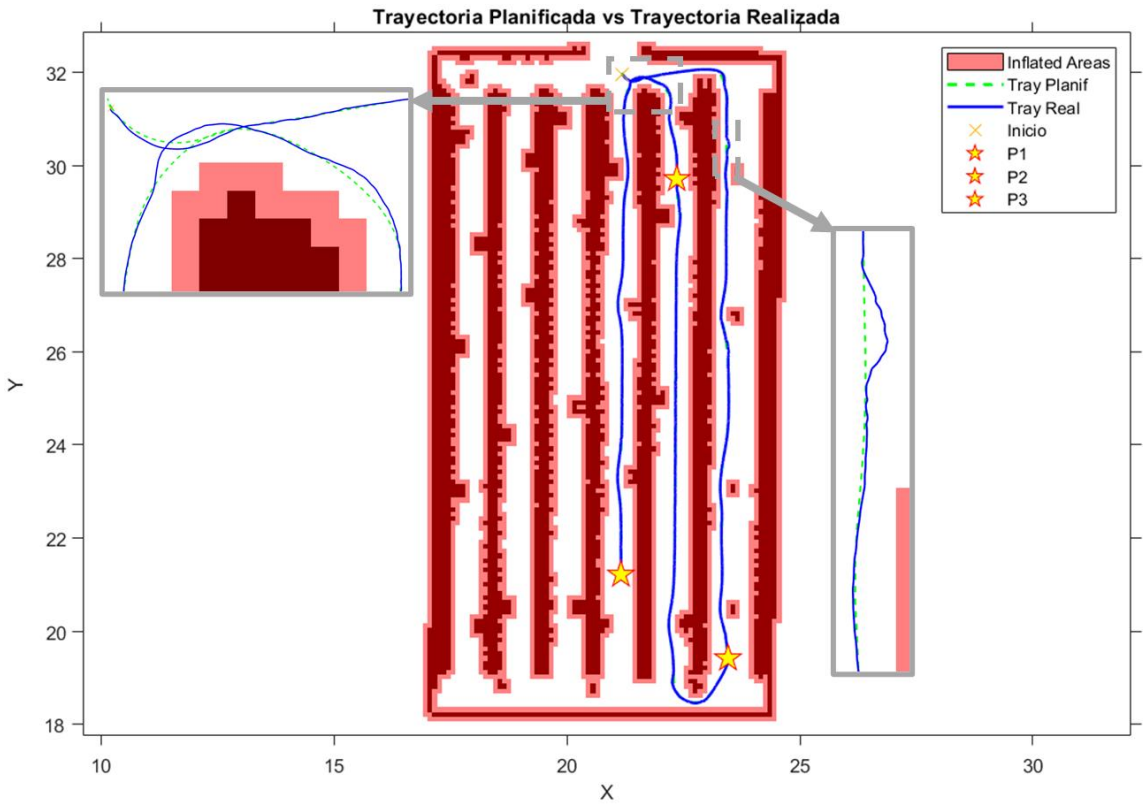



**Figura 100.** A* Híbrido + Optimización TEB: Escenario 4 – Set 3. **Fuente:** autor.

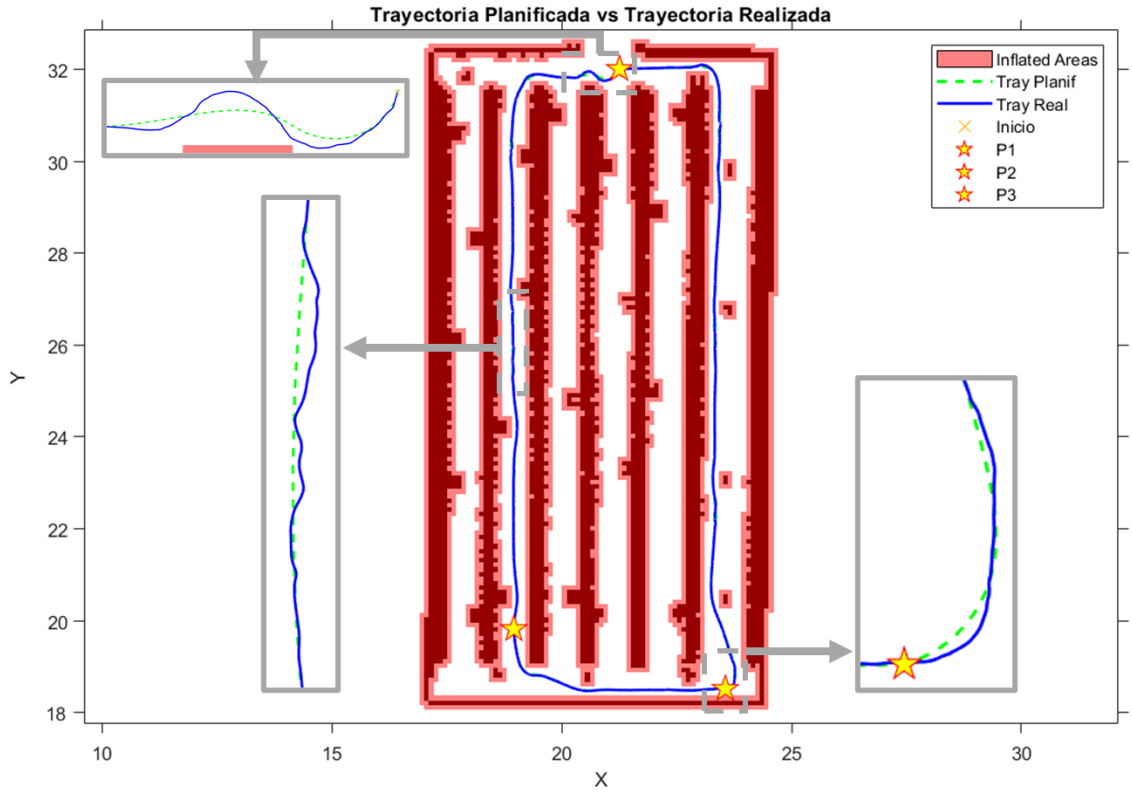



**Figura 101.** A* Híbrido + Optimización TEB: Escenario 4 – Set 4. **Fuente:** autor.

- **Resultados y métricas obtenidas con el planificador A* Híbrido + Optimizador TEB:**

Desde la Figura 86 hasta la Figura 101 se pueden observar una comparación de las trayectorias que el A* Híbrido generó para cada uno de los sets de trayectorias seleccionados en los cuatro escenarios planteados. La línea punteada verde representa la trayectoria original generada por el planificador y la línea continua azul representa la trayectoria realizada por el robot en el entorno virtual. Los indicadores de precisión y los de eficacia óptima obtenidos en estas pruebas se encuentran almacenados en la Tabla 14 y Tabla 15, respectivamente.

**Tabla 14.** Indicadores de precisión en el seguimiento de la trayectoria para el planificador A* Híbrido + Optimizador TEB. **Fuente:** autor.

| A* Híbrido + Optimizador TEB | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| **Escenario** | **Set** | **Error Promedio [m]** | **Error Promedio [%]** | **Error RMSE [m]** | **Error RMSE [%]** | **Error MAX [m]** | **Error Orien [rad]** | **Máx. Desv. Angular [rad]** |
| **E1: normal sin obstáculos** | **S1** | 0.01462 | 0.04 | 0.01714 | 0.05 | 0.04653 | 0.009 | 0.6565 |
| | **S2** | 0.0226 | 0.07 | 0.0247 | 0.07 | 0.0408 | 0.0021 | 0.4642 |
| | **S3** | 0.0156 | 0.04 | 0.0166 | 0.04 | 0.0312 | 0.0024 | 0.6185 |
| | **S4** | 0.0213 | 0.06 | 0.022 | 0.06 | 0.304 | 0.0079 | 0.6289 |
| **E2: normal con obstáculos** | **S1** | 0.0131 | 0.04 | 0.0153 | 0.04 | 0.0376 | 0.0052 | 0.6105 |
| | **S2** | 0.0165 | 0.05 | 0.0196 | 0.06 | 0.06 | 0.0036 | 0.6172 |
| | **S3** | 0.0361 | 0.09 | 0.0421 | 0.1 | 0.093 | 0.0029 | 0.5557 |
| | **S4** | 0.022 | 0.06 | 0.0227 | 0.06 | 0.0346 | 0.0072 | 0.6205 |
| **E3: irregular sin obstáculos** | **S1** | 0.0323 | 0.09 | 0.0355 | 0.1 | 0.063 | 0.013 | 0.6272 |
| | **S2** | 0.032 | 0.09 | 0.0375 | 0.11 | 0.0797 | 0.0042 | 0.6098 |
| | **S3** | 0.0432 | 0.11 | 0.0453 | 0.11 | 0.0741 | 0.0056 | 0.6164 |
| | **S4** | 0.024 | 0.07 | 0.0296 | 0.08 | 0.0704 | 0.0107 | 0.6261 |
| **E4: Irregular con obstáculos** | **S1** | 0.0475 | 0.14 | 0.0519 | 0.15 | 0.0875 | 0.0109 | 0.6248 |
| | **S2** | 0.0349 | 0.1 | 0.0438 | 0.13 | 0.098 | 0.0045 | 0.6176 |
| | **S3** | 0.0233 | 0.06 | 0.0289 | 0.07 | 0.0897 | 0.0047 | 0.6162 |
| | **S4** | 0.051 | 0.14 | 0.0592 | 0.17 | 0.104 | 0.0173 | 0.6249 |

Los resultados obtenidos en los indicadores de precisión evidencian una alta efectividad del optimizador TEB en condiciones ideales, con un rendimiento decreciente conforme aumentan la complejidad del terreno y la presencia de obstáculos. El Escenario 1 se observó que las trayectorias generadas presentan una alta precisión, con errores promedio que oscilan entre 0.01426 metros y 0.0226 metros y un error RMSE máximo de 0.0247 metros en el conjunto S2. Estos valores demuestran la capacidad del planificador para mantener la trayectoria deseada en condiciones ideales, destacándose también por los bajos errores de orientación, el cual obtuvo un error de 0.0024 radianes en S2, el más bajo que se presentó en comparación con los demás escenarios. Este resultado es coherente con las condiciones ideales de este escenario, donde la ausencia de irregularidades y obstáculos permitió al sistema operar con mayor estabilidad.

En el Escenario 2, los valores generales de error promedio y RMSE se incrementaron ligeramente debido a la necesidad de evitar obstáculos, siendo el set S3 el más afectado con un error promedio de 0.0361 metros y un RMSE de 0.0384 metros. Esto sugiere que la presencia de obstáculos externos impacta la precisión del seguimiento de trayectoria, aunque el planificador mantiene una adecuada capacidad de ajuste, llegando incluso a obtener el error promedio más bajo de todos los escenarios, siendo este de 0.0131 metros correspondientes a S1, a pesar de la presencia de los obstáculos, resultando ligeramente menor al error encontrado en el escenario

sin obstáculos S1. Este resultado indica que el sistema mantiene un alto grado de exactitud en la trayectoria planificada, incluso en presencia de restricciones ambientales.

Los Escenarios 3 y 4 evidenciaron un aumento significativo en los errores, siendo el Escenario 4 el que presentó los errores más altos, alcanzando un error promedio de 0.051 metros en S4, mientras que el Escenario 3 presentó un error promedio de 0.0432 metros y un error RMSE de 0.0453 en S3. Este resultado pone de manifiesto el desafío que representan las condiciones combinadas de irregularidades y obstáculos, lo que implica una mayor necesidad de ajustes en la planeación y ejecución de las trayectorias. Además, la desviación angular máxima en el Escenario 4 para S4 fue de 0.6249 rad, uno de los valores más altos registrados, lo que sugiere una mayor dificultad en el control de orientación en terrenos complejos.

**Tabla 15.** Indicadores de eficiencia y optimización de la ruta para el planificador A* Híbrido + Optimizador TEB.
**Fuente:** autor.

| A* Híbrido + Optimizador TEB | | | | | |
|---|---|---|---|---|---|
| **Escenario** | **Set** | **Tiempo Empleado [s]** | **Dist Total REAL [m]** | **Dist Total Ideal [m]** | **Diferencia relativa [%]** |
| **E1: normal sin obstáculos** | **S1** | 174.133 | 34.1855 | 34.1144 | 0.208416387 |
| | **S2** | 168.783 | 33.3429 | 33.2859 | 0.17124368 |
| | **S3** | 204.75 | 40.3901 | 40.3422 | 0.118734229 |
| | **S4** | 178.65 | 35.2033 | 35.1472 | 0.159614422 |
| **E2: normal con obstáculos** | **S1** | 176.183 | 34.7149 | 34.6614 | 0.154350372 |
| | **S2** | 172.767 | 33.9884 | 33.9001 | 0.260471208 |
| | **S3** | 208.217 | 41.1969 | 41.1333 | 0.15461925 |
| | **S4** | 179.283 | 35.3617 | 35.3098 | 0.14698469 |
| **E3: irregular sin obstáculos** | **S1** | 174.8 | 34.31 | 34.1144 | 0.573364913 |
| | **S2** | 171.5 | 33.4966 | 33.2859 | 0.63300076 |
| | **S3** | 207.267 | 40.5503 | 40.3422 | 0.515837014 |
| | **S4** | 181.5 | 35.4593 | 35.1472 | 0.887979697 |
| **E4: Irregular con obstáculos** | **S1** | 178.35 | 34.9149 | 34.6614 | 0.731361111 |
| | **S2** | 175.1 | 34.16 | 33.9001 | 0.766664405 |
| | **S3** | 211.333 | 41.3038 | 41.1333 | 0.414506009 |
| | **S4** | 182.667 | 35.5917 | 35.3098 | 0.798361928 |

Los resultados obtenidos en los indicadores de eficiencia muestran una tendencia clara: los escenarios irregulares (E3 y E4) demandan mayores tiempos de desplazamiento en comparación con los escenarios normales (E1 y E2). Por ejemplo, en el Escenario 3, el Set S3 requirió 207.267 segundos, mientras que en el Escenario 1, el mismo Set completó la trayectoria en 204.75 segundos. Esta diferencia puede atribuirse al impacto de las irregularidades del terreno en las dinámicas de desplazamiento del robot, lo que implica una mayor carga computacional y ajustes en tiempo real para mantener la trayectoria planificada.

En términos de eficiencia en la distancia recorrida, las diferencias relativas entre la distancia total real y la ideal fueron más bajas en los Escenarios 1 y 2. En el Set S3 del Escenario 1, la diferencia relativa fue de apenas 0.1187%, mientras que en el Set S3 del Escenario 2 se registró un valor similar de 0.1546%. Por el contrario, en el Escenario 4, estas diferencias aumentaron significativamente, alcanzando un 0.7984% en el Set S4. Este incremento refleja las dificultades del sistema para optimizar las trayectorias en entornos más desafiantes, donde las irregularidades del terreno y la presencia de obstáculos obligan al robot a desviarse más de la trayectoria ideal.

# 9. CONCLUSIONES.

## 9.1. Conclusiones y observaciones generales.

El sistema de navegación autónoma desarrollado logró un excelente desempeño en entornos previamente conocidos. Durante las pruebas, el robot móvil tipo oruga completó con éxito su desplazamiento a través de un cultivo simulado, alcanzando tres puntos objetivos mediante un movimiento secuencial. Las trayectorias generadas fueron seguidas con alta precisión, garantizando la llegada a los puntos designados sin desviaciones significativas. Además, el sistema demostró su capacidad para sortear irregularidades en el terreno, como rocas y ramas a nivel del suelo, sin comprometer el desplazamiento ni la estabilidad del robot.

De igual forma, la simulación desarrollada en Unreal Engine 5 permitió incorporar un alto nivel de detalle gráfico y complejidad computacional, logrando una representación realista de las condiciones de un invernadero. Este entorno virtual emuló con precisión sistemas físicos complejos, como el amortiguamiento del robot y el movimiento de sus orugas sobre superficies ideales e irregulares. Gracias a estas características, fue posible validar el modelo de navegación bajo condiciones controladas, con resultados que reflejan escenarios reales. Los desniveles leves y los obstáculos presentes en el entorno no impidieron el libre movimiento del robot, demostrando su capacidad para adaptarse a terrenos ligeramente diferentes al ideal.

La implementación del sistema de localización y mapeo basado en el algoritmo LOAM (Lidar Odometry and Mapping**)** en modo offline resultó clave para disminuir la complejidad computacional durante las simulaciones. Este enfoque permitió que el robot se enfocara exclusivamente en el cálculo de la ruta óptima durante las pruebas. Aunque la generación inicial del mapa requirió un tiempo considerable, el resultado fue altamente satisfactorio, con un error promedio del 3% en la estimación de la trayectoria. Este margen de error es óptimo considerando que el mapeo se realizó exclusivamente con datos obtenidos de un sensor LiDAR. Adicionalmente, el sistema logró un cierre de bucle efectivo y produjo un mapa preciso y confiable del entorno. No obstante, se identificaron pequeñas irregularidades en el contorno del mapa 2D, atribuibles posiblemente al ruido del sensor o a errores en la conversión de la nube de puntos al mapa de costos. Estas irregularidades no impactaron significativamente la generación ni el seguimiento de las trayectorias, confirmando la robustez del sistema.

El algoritmo A* Híbrido Optimizado demostró ser una herramienta eficaz para la planificación de rutas óptimas en un entorno simulado. Las trayectorias generadas respetaron las restricciones cinemáticas del robot, aseguraron una curvatura suave y mantuvieron una distancia segura respecto a los obstáculos identificados. Además, el planificador respondió adecuadamente a los requerimientos del mapa generado por el sistema LOAM, lo que garantizó un desplazamiento eficiente y sin colisiones.

Por su parte, el algoritmo de seguimiento de trayectoria Pure Pursuit tuvo un desempeño extraordinario dentro del sistema de navegación. A pesar de las perturbaciones causadas por los obstáculos externos y las irregularidades del terreno, el error promedio en el seguimiento de la trayectoria no superó el 0.3%. Este resultado supera las limitaciones previstas por el propio algoritmo y demuestra su idoneidad para el control preciso del movimiento del robot en entornos complejos. La combinación del planificador A* Híbrido y el seguidor de trayectorias Pure Pursuit resultó ser una solución robusta y efectiva para garantizar un desplazamiento autónomo confiable en escenarios simulados.

Las pruebas realizadas en los diferentes escenarios permitieron evaluar el error de seguimiento y la respuesta del robot frente a obstáculos conocidos. Los resultados confirman que el sistema es capaz de operar de manera autónoma en un entorno agrícola simulado, con un desempeño aceptable en términos de precisión y eficiencia.

Este proyecto ofrece una contribución significativa al diseño y validación de sistemas de navegación autónoma para robots móviles en entornos agrícolas. Destaca especialmente por su enfoque innovador al integrar un motor de videojuegos, Unreal Engine 5, como plataforma de simulación avanzada. Este método, poco explorado en proyectos similares, permitió conectar el poder analítico de MATLAB con las capacidades computacionales y gráficas de un motor de videojuegos. Esta combinación facilitó la emulación de sistemas físicos complejos y la validación de algoritmos bajo condiciones controladas, estableciendo un precedente para futuras investigaciones que busquen aprovechar tecnologías interdisciplinarias.

A continuación, se exponen conclusiones adicionales que abordan aspectos clave como la precisión en el seguimiento de la trayectoria, la optimización de los procesos y la capacidad de respuesta ante obstáculos, según las métricas obtenidas en las pruebas de validación. Posteriormente, se presentarán recomendaciones orientadas a posibles líneas de investigación futura.

### 9.2. Precisión en el seguimiento de trayectorias

- **Mejora significativa con el uso del optimizador TEB**: El planificador A* Híbrido Optimizado con TEB demostró una mejora notable en la precisión del seguimiento de trayectorias en comparación con el A* Híbrido Puro. En el Escenario 1 (normal sin obstáculos), el error promedio de seguimiento se redujo en un 30% en promedio para todos los sets de trayectorias (S1-S4). Por ejemplo, en S1, el error promedio disminuyó de 0.0209 m a 0.01462 m, lo que representa una reducción del 30.05%. Esta mejora se atribuye a la optimización dinámica de trayectorias que realiza TEB, ajustando continuamente la ruta para minimizar desviaciones.
- **Desempeño en terrenos irregulares**: En el Escenario 3 (irregular sin obstáculos), el A* Híbrido Optimizado con TEB mantuvo un error promedio de seguimiento significativamente menor que el A* Híbrido Puro, en casi todos los escenarios. Por ejemplo, en S3, el error promedio fue de 0.0432 m con TEB, frente a 0.0931 m con el planificador puro, lo que representa una reducción del 53.6%. Esto indica que TEB es más robusto frente a irregularidades del terreno, ajustando la trayectoria para compensar las variaciones en la superficie.
- **Error Máximo:** En todos los escenarios donde se realizó una comparación directa, el planificador optimizado con TEB presentó un Error Máximo considerablemente menor en relación al A* Híbrido Puro. Por ejemplo, en el escenario 1, el Error Máximo disminuyó de 0.0719 m con A* Híbrido Puro a 0.04653 m con TEB, representando una reducción del 35%. Esto confirma que la optimización introducida por TEB mejora notablemente la capacidad del robot para adherirse a la trayectoria planteada, especialmente en escenarios simples.
- **Error RMSE:** El RMSE también se redujo significativamente con el uso del optimizador TEB en casi todos los escenarios. En el escenario 1, el RMSE pasó de 0.0251 m con A* Híbrido Puro a 0.01714 m con TEB, lo que representa una mejora del 31.7%. Esta reducción indica que el optimizador TEB no solo minimiza los errores puntuales, sino que también mejora la consistencia general del seguimiento de la trayectoria.

- **Reducción del error de orientación y desviación angular:** En todos los escenarios, el A* Híbrido Optimizado mostró un menor error de orientación en comparación con el A* Híbrido Puro, lo que indica una mayor precisión en la orientación del robot durante el seguimiento de la trayectoria. Esto es particularmente relevante en escenarios con irregularidades en el terreno, donde la orientación correcta del robot es crucial para evitar desviaciones. A pesar de esto, el optimizador TEB obtuvo mayores desviaciones angulares máximas en casi todos los escenarios, pero esto no afecto significativamente el error de orientación.

### 9.3. Eficiencia y optimización en la trayectoria.

- **Reducción en la distancia recorrida:** El A* Híbrido Optimizado con TEB mostró una mayor eficiencia en la optimización de rutas, reduciendo la distancia total recorrida en comparación con el A* Híbrido Puro. En el Escenario 1, la diferencia relativa entre la distancia real e ideal fue menor en todos los sets de trayectorias. Por ejemplo, en S1, la diferencia relativa fue de 0.2084% con TEB, frente a 0.4787% con el planificador puro. Esto indica que TEB genera rutas más cercanas a la trayectoria ideal, optimizando el desplazamiento del robot.
- **Tiempo de ejecución:** Aunque el tiempo de ejecución no mostró una mejora significativa en todos los casos, en algunos escenarios, TEB logró reducir el tiempo de planificación y ejecución. Por ejemplo, en S2 del Escenario 1, el tiempo empleado fue de 168.783 s con TEB, frente a 173.183 s con el planificador puro. Esto sugiere que TEB no solo optimiza la ruta, sino que también puede mejorar la eficiencia temporal en ciertas condiciones.

### 9.4. Capacidad de respuesta frente a obstáculos y terrenos irregulares.

- **Limitaciones del A Híbrido Puro:** El A* Híbrido Puro no pudo completar las pruebas en los Escenarios 2 y 4 (con obstáculos), lo que indica una limitación significativa en su capacidad para manejar entornos con obstáculos no superables. Esto resalta la importancia de la optimización dinámica que ofrece TEB, que sí logró operar en estos escenarios sin colisiones.
- **Robustez de TEB en entornos complejos:** En los Escenarios 2 y 4, el A* Híbrido Optimizado con TEB demostró una capacidad robusta para evitar obstáculos y adaptarse a irregularidades del terreno. Por ejemplo, en S1 del Escenario 4, el error promedio fue de 0.0475 m, lo que, aunque superior a los escenarios sin obstáculos, sigue siendo aceptable para un entorno agrícola. Esto confirma que TEB es capaz de manejar tanto obstáculos como irregularidades del terreno de manera efectiva.

### 9.5. Recomendaciones para trabajos futuro.

- Ampliar la validación del sistema en entornos reales para evaluar su desempeño bajo condiciones no controladas.
- Implementar algoritmos de detección y evasión de obstáculos dinámicos en tiempo real, de esta forma se aumentaría el nivel de autonomía del robot.
- Aunque el algoritmo *Pure Pursuit* mostró un buen desempeño en esta aplicación, su limitación al operar en una sola dirección puede ser un obstáculo al enfrentar obstáculos complejos que requieran retroceso. En este sentido, el algoritmo *TEB* permitiría la generación de rutas con retrocesos, pero no pudo ser implementado debido a que el *Pure Pursuit* no admite estas trayectorias.
- Optimizar el tiempo de procesamiento del algoritmo A* Híbrido para mejorar su rendimiento en escenarios más complejos o de mayor escala.

## BIBLIOGRAFIA


Andrade, A. (2015). Game engines: a survey, EAI Endorsed Transactions on Game-Based Learning.

Arad, B., Balendonck, J., Barth, R., Ben-Shahar, O., Edan, Y., Hellström, T., Hemming, J., Kurtser, P., Ringdahl, O., & Tielen, T. (2020). Development of a sweet pepper harvesting robot. *Field Robot, 37*, 1027-1039.

Aravind, K., Raja, P., & Pérez-Ruiz, M. (2017). Task-based agricultural mobile robots in arable farming: A review. *Spanish Journal of Agricultural Research, 15*, 10.

Asociación Nacional de Industriales. (2019). *Bancos de alimentos, claves en la lucha frente al hambre*. Asociación Nacional de Empresarios de Colombia. https://www.andi.com.co/Home/Noticia/15572-bancos-de-alimentos-claves-en-la-lucha

Bac, C., Henten, E., Hemming, J., & Edan, Y. (2014). Harvesting Robots for High-value Crops: State-of-the-art Review and Challenges Ahead. *Journal of Field Robotics, 31*(6), 888–911.

Biber, P., & Strasser, W. (2003). The normal distributions transform: a new approach to laser scan matching. *2003 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), 3*, 2743-2748.

Binbin, X., Jizhan, L., Meng, H., Jian, W., & Zhujie, X. (2021). Research progress on Autonomous Navigation Technology of Agricultural Robot. *IEEE 11th Annual International Conference on CYBER Technology in Automation, Control, and Intelligent Systems (CYBER)* (págs. 881-898). Jiaxing, China.: IEEE.

Blackmore, S., Stout, B., Wang, M., & Runov, B. (2005). Robotic agriculture—the future of agricultural mechanisation? *5th European Conference on Precision Agriculture.*

BoredEngineer. (2015). *UE4_Tracked_Vehicles*. Github. https://github.com/BoredEngineer/UE4_Tracked_Vehicles

Braunl, T. (2006). *Embedded Robotics, Mobile Robot Design and Applications with Embedded Systems* (2 ed.). Springer.

Cadena, C., Carlone, L., Carrillo, H., Latif, Y., Scaramuzza, D., Neira, J., Reid, I., & Leonard, J. (2016). Past, Present, and Future of Simultaneous Localization and Mapping: Toward the Robust-Perception. *IEEE*. https://doi.org/10.1109/TRO.2016.2624754

Ceres, R., Pons, J., Jiménez, A., Martín, J., & Calderón, L. (1998). Design and implementation of an aided fruit-harvesting robot (Agribot). *Ind. Robot Int. J.*, 337-346.

Chen, J., Qiang, H., Wu, J., & Xu, G. W. (2021). Navegation path extraction for greenhouse cucumber-picking robots using the prediction-point Hough transform. *Comput. Electron. Agr*, 180. https://doi.org/10.1016/S1881-8366(13)80017-1

Chen, Y., Zhang, X., & Sun, D. (2018). Learning Navigation Behaviors End-to-end With Autoencoder-like Deep Networks. *IEEE Transactions on Neural Networks and Learning Systems, 29*(11), 5484–5498.

Cormen, T. H., Leiserson, C. E., Rivest, R. L., & Stein, C. (2009). *Introduction to Algorithms.* The MIT Press.

Coulter, R. (1990). Implementation of the Pure Pursuit Path Tracking Algorithm. *Carnegie Mellon University*.

Cui, Y., Qiao, Z., Zou, X., & Wang, B. (2015). Study on the method of visual navigation baseline identification and extraction of agricultural machinery. *2015 6th IEEE International Conference on Software Engineering and Service*, (págs. 766-769). https://doi.org/10.1109/ICSESS.2015.7339169

Dailey, M., & Parnichkun, M. (2006). Simultaneous Localization and Mapping with Stereo Vision. *IEEE*. https://doi.org/10.1109/ICARCV.2006.345269

Demmel, B., Hölle, M., Dietmüller, T., Leber, P., & Kibler, T. (2022). Applying game engine in robot simulation. *Duale Hochschule Baden-Wuttemberg (DHBW)*. https://doi.org/10.12903/DHBW_RV_FN_01_2022_DEMMEL_HOELLE_DIETMUELLER_LEBER_KIBLER

Departamento Administrativo Nacional de Estadística. (2014). El cultivo del tomate de mesa bajo invernadero, tecnología que ofrece mayor producción, calidad e inocuidad del producto. *Boletin mensual INSUMOS Y FACTORES ASOCIADOS A LA PRODUCCIÓN AGROPECUARIA*. https://www.dane.gov.co/files/investigaciones/agropecuario/sipsa/insumos_factores_de_produccion_dic_2014.pdf

Departamento Nacional de Planeación. (2020). *Política para la prevención y reducción de las pérdidas y desperdicios de alimentos.* Departamento Nacional de Planeación. https://dnp.gov.co/Crecimiento-Verde/Documents/Comite%20Sostenibilidad/Presentaciones/Sesi%C3%B3n%205/1_Avances_Politica_para_prevencion_reduccion_de_perdidas_desperdicios_alimentos.pdf

Dirección de Investigaciones Económicas. (2020). La recolección de café en Colombia. En *Ensayos sobre Economía Cafetera* (32 ed., págs. 35-65). Bogotá: Federación Nacional de Cafeteros de Colombia. https://federaciondecafeteros.org/static/files/ECC32.pdf

Dolgov, D., Thrun, S., Montemerlo, M., & Diebel, J. (2008). Path planning for autonomous vehicles in unknown semi-structured environments. *The International Journal of Robotics Research, 29*(5), 485-501.

Droeschel, D., & Behnke, S. (2018). Efficient Continuous-Time SLAM for 3D Lidar-Based Online Mapping. *Proceedings of the International Conference on Robotics and Automation (ICRA)*. https://doi.org/10.1109/ICRA.2018.8461000

Epic Games. (2021). *Unreal Engine 4 Documentation*. Epic Games. https://docs.unrealengine.com/4.27/en-US/

Escobar, H., & Lee, R. (2009). *Manual de Producción de Tomate Bajo Invernadero.* Bogotá: Fundación Universidad de Bogotá Jorge Tadeo Lozano.

FAO. (2017). The future of food and agriculture - Trends and challenges. Rome.

FAOSTAT. (2021). Crops. FAOSTAT Database.

Gonzalez, C., & Adams, M. (2019). An improved feature extractor for the Lidar Odometry and Mapping (LOAM) algorithm. *International Conference on Control, Automation and Information Sciences (ICCAIS)*.

González, R., Rodríguez, F., & Guzmán, J. (2015). Robots Móviles con Orugas Historia, Modelado, Localización y Control. *Revista Iberoamericana de Automática e Informática industrial 12*, 3-12.

González, R., Rodríguez, F., Sánchez-Hermosilla, J., & Donaire, J. (2009). NAVIGATION TECHNIQUES FOR MOBILE ROBOTS IN GREENHOUSES. *the American Society of Agricultural and Biological Engineers, 25*(2), 153-165.

Hajjaj, S., & Sahari, K. (2014). Review of Research in the Area of Agriculture Mobile Robots. *The 8th International Conference on Robotic, Vision, Signal Processing & Power Applications. 291*, págs. 107-117. Springer.

Haley, R., Coe, J., & J., K. (2012). 3-D Visualization of Simulink Physics Models Using Unreal Engine. *Computer Science*.

Hart, P. E., Nilsson, N. J., & Raphael, B. (1968). A Formal Basis for the Heuristic Determination of Minimum Cost Paths. *IEEE Transactions on Systems Science and Cybernetics, 4*(2), 100-107.

Hayashi, S., Yamamoto, S., Saito, S., Ochiai, Y., Kamata, J., Kurita, M., & al., e. (2014). Field operation of a movable strawberry-harvesting robot using a travel platform. *Jpn. Agric. Res.*, 307-316. https://doi.org/10.6090/jarq.48.307

Henten, E., Bac, C. W., Hemming, J., & Edan, Y. (2013). Robotics in protected cultivation. *IFAC Proceed, 46*, 170-177. https://doi.org/10.3182/20130828-2-SF-3019.00070

Hou, J., Pu, W., Li, T., & Ding, X. (2020). Development of dual-lidar navigation system for greenhouse transportation robot. *Trans. Chin. Soc. Agric.*, 80-88. https://doi.org/10.11975/j.issn.1002-6819.2020.14.010

Hunter, G. (2024, Feb 26). *Exponential Moving Average (EMA) Filters*. https://blog.mbedded.ninja/programming/signal-processing/digital-filters/exponential-moving-average-ema-filter/

Ibaduiza Burgos, D. A., & Barrero Perez, J. G. (2006). PLANEAMIENTO DE TRAYECTORIAS DE UN ROBOT MOVIL. *UIS*.

Jasinski, M., Mączak, J., Radkowski, S., Korczak, S., Rogacki, R., Mac, J., & Szczepaniak, J. (2016). Autonomous Agricultural Robot - Conception of Inertial Navigation System. *Automation*.

Jiang, S. .., Wang, S. .., Yi, Z. .., Zhang, M. .., & Lv, X. (2022). Autonomous Navigation System of Greenhouse Mobile Robot Based on 3D Lidar and 2D Lidar SLAM. *Frontiers in Plant Science*.

Juan, R., Pablo, G. A., Mario, G., Jorge, D. L., Jaime, D. C., & Antonio, B. (2016). Heterogeneous multi-robot system for mapping environmental variables of greenhouses. *Sensors*. https://doi.org/10.3390/s16071018

Karaman, S., & Frazzoli, E. (2011). Sampling-based algorithms for optimal motion planning. *The International Journal of Robotics Research*.

Karur, K., Sharma, N., Dharmatti, C., & Siegel, J. (2021). A Survey of Path Planning Algorithms for Mobile Robots. *MDPI*. https://doi.org/10.3390/vehicles3030027

Khan, N., Medlock, G., Graves, S., & Anwar, S. (2018). GPS Guided Autonomous Navigation of a Small Agricultural Robot with Automated. *SAE Technical Paper, 1*.

Kleinberg, J., & Tardos, É. (2006). *Algorithm Design.* Pearson Education.

LaValle, S. M. (1998). Rapidly-Exploring Random Trees: A New Tool for Path Planning. *Computer Science Dept., Iowa State University*.

Lee, I., Lee, K., Lee, J., & You, K. (2015). Autonomous greenhouse sprayer navigation using automatic tracking algorithm. *Appl. Eng. Agric., 31*, 17-21. https://doi.org/10.13031/aea.3110448

Lehnert, C., McCool, C., Sa, I., & Perez, T. (2020). Performance improvements of a sweet pepper harvesting robot in protected cropping. *Field Robot, 37*, 1197-1223.

Li, L., Zhang, Q., & Huang, D. (2010). A review of imaging techniques for plant phenotyping. *Sensors (Switzerland), 14*(11), 20078–20111.

Mao, J., Cao, Z., Wang, H., Zhang, B., & Niu, W. (2019). Agricultural robot navigation path recognition based on k-means algorithm for large-scale image segmentation. *2019 14th IEEE conference on industrial electronics and applications (ICIEA)* (págs. 1233-1237). IEEE 2019. https://doi.org/10.1109/ICIEA.2019.8834296

Massino, P. (2008). *Una introducción a los Robots Móviles.* IL Bambino.

MathWorks. (2019). *plannerHybridAStar*. MathWorks. https://la.mathworks.com/help/nav/ref/plannerhybridastar.html

MathWorks. (2021). *Motion Planning with MATLAB*. MathWorks. https://la.mathworks.com/campaigns/offers/next/getting-started-with-motion-planning-in-matlab-ebook.html#2

MathWorks. (2022). *Developing Autonomous Mobile Robots Using MATLAB and Simulink.* MathWorks. Mathworks. https://la.mathworks.com/campaigns/offers/next/autonomous-mobile-robots.html

MathWorks. (2024). *Optimization Based Path Smoothing for Autonomous Vehicles*. Mathworks. https://la.mathworks.com/help/nav/ug/optimization-based-path-smoothing-for-autonomous-vehicles.html?lang=en

MathWorks. (s.f.). *SLAM (localización y mapeo simultáneos)*. MathWorks. https://la.mathworks.com/discovery/slam.html

Mohd Basri, M., & Adnan, M. (2022). Autonomous Agriculture Robot for Monitoring Plant using Internet of Things. *ELEKTRIKA- Journal of Electrical Engineering*.

Mulla, D. J. (2013). Twenty five years of remote sensing in precision agriculture: Key advances and remaining knowledge gaps. *Biosystems Engineering, 114*(4), 358-371.

Muñoz Martinez, V. F. (1997). *Planificación de Trayectorias para Robots Móviles.* Universidad de Málaga. Departamento de Ingeniería de Sistemas y Automática de la Universidad de Málaga.

Navas Gomez, O. D., & Ortiz Ortega, J. N. (2006). ALGORITMOS GENETICOS APLICADOS AL PLANEAMIENTO DE TRAYECTORIAS DE UN ROBOT MOVIL. *UIS*.

Nguyen, H., Trinh, T., Hoang, T., & Le, Q. (2022). Trajectory Tracking Control for Differential-Drive Mobile Robot by a Variable Parameter PID Controller. *International Journal of Mechanical Engineering and Robotics Research, 11*(8), 614-621.

Niño Rodriguez, S., & Salazar Triana, J. D. (2018). DISEÑO Y CONSTRUCCION DE UN SISTEMA DE NAVEGACION AUTONOMA PARA UN VEHICULO DE CAMPOS AGRICOLAS DE MAIZ. *UIS*.

Noguchi, N., Ishii, K., & Terao, H. (2016). Agricultural Robots -Mechanisms and Practice-. *Japan Society of Mechanical Engineers.*

Obregón, D., Arnau, R., Campo-Cossio, M., Arroyo-Parras, J. G., Pattinson, M., & Tiwari, S. (2019). Precise positioning and heading for autonomous couting robots in a harsh environment. *International Work-Conference on the Interplay Between Natural and Artificial Computation* (págs. 82-96). Springer, Cham. https://doi.org/10.1007/978-3-030-19651-6_9

Octavio, Á. (2018). *ANTIOQUIA Y NORTE DE SANTANDER SON LOS DEPARTAMENTOS LÍDERES EN LA PRODUCCIÓN DE TOMATE*. Agronegocios. https://www.agronegocios.co/agricultura/cuales-son-las-regiones-que-mas-producen-tomate-2728689

Oliveira, L., Moreira, A., & Silva, M. (2021). Advances in Agriculture Robotics: A State-of-the-Art Review and Challenges Ahead. *Robotics, 10*(52).

OLLERO, A. (2001). *Robótica Manipuladores y robots móviles.* Marcombo S.A.

Pacheco, L., & Luo, N. (2015). Testing PID and MPC Performance for Mobile Robot Local Path-following. *International Journal of Advanced Robotic Systems*.

Pedersen, S., Fountas, S., & Blackmore, S. (2008). *Agricultural robots-Applications and economic perspectives, service robot applications*. http://www.intechopen.com/books/service_robot_applications/agricultural_robots_-_applications_and_economic_perspectives

Petereit, Janko, Emter, T., Frey, C. W., Kopfstedt, T., & Beutel, A. (2012). Application of Hybrid A* to an Autonomous Mobile Robot for Path Planning in Unstructured Outdoor Environments. *ROBOTIK 2012: 7th German Conference on Robotics*, 1-6.

Pilli, S., Nallathambi, B., George, S., & Diwanji, V. (2015). eAGROBOT—A robot for early crop disease detection using image processing. *2nd International Conference on Electronics and Communication Systems (ICECS)*, (págs. 1684-1689). Coimbatore, India.

Pineda Quijano, D. F., & Prada Largo, F. A. (2009). DISENO Y CONSTRUCCION DE UN PROTOTIPO DE ROBOT MOVIL AUTONOMO DE EXPLORACION. *UIS*.

Quinlan, S. (1995). *Real-Time Modification of Collision-Free Paths.* Stanford, CA: Stanford University.

Roesmann, C., Feiten, W., Woesch, T., Hoffmann, F., & Bertram, T. (2012). Trajectory modification considering dynamic constraints of autonomous robots. En *ROBOTIK 2012; 7th German Conference on Robotics* (págs. 1-6). Munich, Germany.

Rosmann, C., Hoffmann, F., & Bertram, T. (2017). Kinodynamic Trajectory Optimization and Control for Car-Like Robots. *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. https://doi.org/10.1109/IROS.2017.8206458

Santhi, P., Kapileswar, N., Chenchela, V., & Prasad, C. (2017). Sensor and vision based autonomous AGRIBOT for sowing seeds. *2017 International Conference on Energy, Communication, Data Analytics and Soft Computing (ICECDS)*, (págs. 242-245).

Sedighi, S., Nguyen, D.-V., & Kuhnert, K.-D. (2019). Guided Hybrid A-star Path Planning Algorithm for Valet Parking Applications. *5th International Conference on Control, Automation and Robotics (ICCAR).*

Shuyou, Y., Yang, G., Lingyu, M., Ting, Q., & Hong, C. (2018). MPC for Path Following Problems of Wheeled Mobile Robots. *IFAC (International Federation of Automatic Control)*.

Siciliano, B., Sciavicco, L., Villani, L., & Oriolo, G. (2009). *Robotics: Modelling, Planning and Control.* Londres: Springer-Verlag.

Siegwart, R., & Nourbakhsh, I. R. (2004). *Introduction to Autonomous Mobile Robots.* A Bradford Book.

Sukkarieh, S. (2017). Mobile on-farm digital technology for smallholder farmers. *the 2017 Crawford Fund Annual Conference on Transforming Lives and Livelihoods: The Digital Revolution in Agriculture.* Canberra, Australia.

The Raussendorf Company. (s.f.). *Autonomous system for agricultural purposes such as spraying, tillage, fertilization, contour cut, harvest, and transportation*. Raussendorf. https://www.raussendorf.de/en/fruit-robot.html

Thirsk, R. (2024). An Introduction to GNSS: A primer in using Global Navigation Satellite Systems for positioning and autonomy. HEXAGON. https://novatel.com/an-introduction-to-gnss

Tiwari, S. .., Zheng, Y. .., Pattinson, M. .., Campo-Cossio, M. .., Arnau, R. .., Obregón, D. .., Ansuategi, A. .., Tubío, C. .., Lluvia, I. .., Rey, O. .., Verschoore, J. .., Adam, V. .., & Reyes, J. (2020). Approach for Autonomous Robot Navigation in Greenhouse Environment for Integrated Pest Management. *2020 IEEE/ION Position, Location and Navigation Symposium (PLANS)*, (págs. 1286-1294).

Tiwari, S., Zheng, Y., Pattinson, M., Campo-Cossio, M., & Reyes, J. (2020). Approach for autonomous robot navigation in greenhouse environment for integrated pest management. 2020 ieee/ion position, location and navigation symposium (PLANS. *IEEE 2020*, 1286-1294. https://doi.org/10.1109/PLANS46316.2020.9109895

Universidad Nacional de Educación a Distancia. (2023). *El Algoritmo de Dijkstra o de Caminos mínimos*. Atlas UNED. http://atlas.uned.es/algoritmos/voraces/dijkstra.html#:~:text=El%20algoritmo%20de%20Dijkstra%2C%20tambi%C3%A9n,por%20primera%20vez%20en%201959.

Verein Deutscher Ingenieure. (2004). VDI 2206 - Entwicklungsmethodik für mechatronische Systeme (design methodology for mechatronic systems).

Xiao, X., Liu, B., Warnell, G., & Stone, P. (2020). Motion planning and control for mobile robot navigation using machine learning: a survey. https://doi.org/10.1007/s10514-022-10039-8

Yeh, L., & Nugroho, H. (2021). Design of Hardware-in-the-loop Simulation Approach for Slip-Compensated Odometry Tracked Mobile Robot Platform. *2021 8th International Conference on Computer and Communication Engineering (ICCCE)*, 355-360. https://doi.org/10.1109/ICCCE50029.2021.9467181

Yoon, J.-S., Min, B.-C., Shin, S.-O., Jo, W.-S., & Kim, D.-H. (2014). GA-Based Optimal Waypoint Design for Improved Path Following of Mobile Robot. *Robot Intelligence Technology and Applications 2*.

Zhang, J., & Singh, S. (2014). LOAM: Lidar Odometry and Mapping in Real-time. *Robotics: Science and Systems Conference (RSS)*.

Zhang, N., Wang, M., & Wang, N. (2002). Precision agriculture—a worldwide overview. Computers and Electronics in Agriculture. *36*(2-3), 113-132.

Zhang, X., Lai, J., Xu, D., Li, H., & Fu, M. (2020). 2D Lidar-Based SLAM and Path Planning for Indoor Rescue Using Mobile Robots. *Hindawi Journal of Advanced Transportation*. https://doi.org/10.1155/2020/8867937

Zou, X., Zou, H., & Lu, J. (2012). Virtual manipulator-based binocular stereo vision positioning system and errors modelling. *Mach. Vis. Appl.*, 43-63. https://doi.org/10.1007/s00138-010-0291-y

# ANEXO A

Código de programación

## A1. Código del bloque "*Set Wheel Speed*".

```
function [phiDotR, phiDotL] = wheelSpeed(v, omega, waypoints, pose)

% Path constants
stopThreshold = 0.5; %0.1
slowThreshold = 2; %0.3

% Robot constants r = 0.24; l = 2.95;
trackWidth = 2.95;
wheelRadius = 0.24;

% Slow the robot down when it's near the threshold
distanceToEndpoint = norm(waypoints(end, :) - pose(1:2)');
if (distanceToEndpoint < slowThreshold)
        v = distanceToEndpoint/slowThreshold*v;

    % Stop the robot if it's inside the target threshold
    if distanceToEndpoint < stopThreshold
        v = 0;
    end
end

% Convert velocity and heading angular velocity to wheel speeds
phiDotR = (v - trackWidth/2*omega)/wheelRadius; %R
phiDotL = (v + trackWidth/2*omega)/wheelRadius; %L

end
```

## A2. Código del algoritmo LOAM.

```
%% Paso 1: Cargar Imagen de Referencia y Configuración Inicial
clc; clear all; close all;
%%
clc; clearvars -except ptCloudArr groundTruthPosesLidar smoothWayP
sceneImage = imread("D:\PROYECTO DE GRADO\UE4_Tracked_Vehicles-master
5.1\Saved\Screenshots\Windows\HighresScreenshot00000.png");
imageSize = size(sceneImage);
xlims = [96.08 321.08];
ylims = [177.66 327.66];

% Importamos la Trayectoria
if exist('smoothWayP') == 0
data = load('D:\PROYECTO DE GRADO\Matlab Code\waypoints.mat');
assignin('caller', 'wayPoints', {data.wayPoints});
assignin('caller', 'refPoses', {data.refPoses});
wayPoints = wayPoints{1,1};
refPoses = refPoses{1,1};
numPoses = size(refPoses{1}, 1);
refDirections = ones(numPoses,1);
numSmoothPoses = 20 * numPoses;
[smoothRefPoses,~,cumLengths] = smoothPathSpline(refPoses{1}, refDirections,
numSmoothPoses);
```

```
smoothWayP = [smoothRefPoses(:,1),smoothRefPoses(:,2)];
clearvars data wayPoints refPoses numPoses refDirections numSmoothPoses
smoothRefPoses cumLengths
end
%%
if exist('ptCloudArr') == 0 && exist('groundTruthPosesLidar') == 0
load('LidarCuadraDataINS_PLANO.mat', 'out');
simOut = out; clearvars out;
ptCloudArr = helperGetPointClouds(simOut);
groundTruthPosesLidar = helperGetLidarGroundTruth(simOut);
end
%% Paso 2: Verificación Inicial
ptCloud = ptCloudArr(1);
nextPtCloud = ptCloudArr(2);
gridStep = 0.05;% 0.01 % Ajuste del valor para aumentar la precisión
tform = pcregisterloam(ptCloud, nextPtCloud, gridStep);

egoRadius = 3.0; % 2.0 Ajuste del valor para captar un área más representativa
cylinderRadius = 30; % 25 Ajuste del valor para captar un área más representativa
selectedIdx = findPointsInCylinder(ptCloud, [egoRadius cylinderRadius]);
ptCloud = select(ptCloud, selectedIdx, OutputSize="full");
selectedIdx = findPointsInCylinder(nextPtCloud, [egoRadius cylinderRadius]);
nextPtCloud = select(nextPtCloud, selectedIdx, OutputSize="full");

maxPlanarSurfacePoints = 5; % 200 Ajuste del valor para aumentar la precisión
points = detectLOAMFeatures(ptCloud, MaxPlanarSurfacePoints=maxPlanarSurfacePoints);
points = downsampleLessPlanar(points, gridStep);
nextPoints = detectLOAMFeatures(nextPtCloud,
MaxPlanarSurfacePoints=maxPlanarSurfacePoints);
nextPoints = downsampleLessPlanar(nextPoints, gridStep);

[~, rmseValue] = pcregisterloam(points, nextPoints);
numSkip = 2; % Ajuste del valor para mejorar la precisión
%% Lidar Odometry and Mapping (LOAM)
absPose = groundTruthPosesLidar(1);
relPose = rigidtform3d;
vSetMapping = pcviewset;
ptCloud = ptCloudArr(1);
selectedIdx = findPointsInCylinder(ptCloud,[egoRadius cylinderRadius]);
ptCloud = select(ptCloud,selectedIdx,OutputSize="full");
points = detectLOAMFeatures(ptCloud,'MaxPlanarSurfacePoints',maxPlanarSurfacePoints);
points = downsampleLessPlanar(points,gridStep);

viewId = 1;
vSetMapping = addView(vSetMapping,viewId,absPose);

voxelSize = 0.1;
loamMap = pcmaploam(voxelSize);
addPoints(loamMap,points,absPose);

% Display the parking lot scene with the reference trajectory
hScene = figure(Name="Invernadero de Tomates2", NumberTitle="off");
hold on
plot(smoothWayP(:,1), smoothWayP(:,2),LineWidth=2,DisplayName="Reference Path");
xlim(xlims)
```

```
ylim(ylims)
hScene.Position = [100 100 1000 500];

numSkip = 2; % Ajuste del valor para mejorar la precisión

for k = (numSkip+1):numSkip:numel(ptCloudArr)
    prevPtCloud = ptCloud;
    prevPoints = points;
    viewId = viewId + 1;
    ptCloud = ptCloudArr(k);

    % Select a cylindrical neighborhood of interest.
    selectedIdx = findPointsInCylinder(ptCloud,[egoRadius cylinderRadius]);
    ptCloud = select(ptCloud,selectedIdx,OutputSize="full");

    % Detect LOAM points and downsample the less planar surface points
    points =
detectLOAMFeatures(ptCloud,MaxPlanarSurfacePoints=maxPlanarSurfacePoints);
    points = downsampleLessPlanar(points,gridStep);

    % Register the points using the previous relative pose as an initial
    % transformation
    relPose = pcregisterloam(points,prevPoints,MatchingMethod="one-to-
one",InitialTransform=relPose);

    % Find the refined absolute pose that aligns the points to the map
    absPose = findPose(loamMap,points,relPose);
    % absPose = rigidtform3d(absPose.A * relPose.A); %*

    % Store the refined absolute pose in the view set
    vSetMapping = addView(vSetMapping,viewId,absPose);

    % Add the new points to the map
    addPoints(loamMap,points,absPose);

    % Visualize the absolute pose in the parking lot scene
    plot(absPose.Translation(1), -
absPose.Translation(2),Color="r",Marker=".",MarkerSize=8)
    xlim(xlims)
    ylim(ylims)
    title("Build a Map Using Lidar Mapping")
    legend(["Ground Truth","Estimated Position"])
    pause(0.001)
    view(2)
end
figure(2)
show(loamMap,MarkerSize=20)
%%
numPoses = numel(vSetMapping.Views.AbsolutePose);
absTformOut = zeros(4,4,numPoses,'single');
% Display the parking lot scene with the reference trajectory
% hScene = figure(Name="Invernadero de Tomates2", NumberTitle="off");
% hold on
% plot(smoothWayP(:,1), smoothWayP(:,2),LineWidth=2,DisplayName="Reference Path");
% xlim(xlims)
```

```
% ylim(ylims)
% hScene.Position = [100 100 1000 500];

for i = 1:numPoses
    % plot(vSetMapping.Views.AbsolutePose(i,1).Translation(1),...
    %     -
vSetMapping.Views.AbsolutePose(i,1).Translation(2),Color="r",Marker=".",MarkerSize=8)
    % xlim(xlims)
    % ylim(ylims)
    % title("Build a Map Using Lidar Mapping")
    % legend(["Ground Truth","Estimated Position"])
    % pause(0.001)
    % view(2)
    absTformOut(:,:,i) = vSetMapping.Views.AbsolutePose(i).A;
end

numFrames = size(ptCloudArr,2);
for i = 1:numFrames
    ptCloudIn = ptCloudArr(i);
    % gtc = helperProcessPointCloud2(ptCloudIn);
    gtc = helperPreProcessPointCloud(ptCloudIn,3.0,30,0.2); %* NEW PLANO2 0.05
    location(:,:,:,i) = gtc.Location;
end

[ptCloudMap, vSet] = helperSLAMVisualization(location,absTformOut);
%%
% Crear una matriz de transformación para invertir el eje Y
% Esto es básicamente una matriz de escalado con -1 en la segunda dimensión (Y)
invertY = affine3d([1 0 0 0; 0 -1 0 0; 0 0 1 0; 0 0 0 1]);

% Aplicar la transformación a la nube de puntos
ptCloudMap = pctransform(ptCloudMap, invertY);

% Visualizar la nube de puntos corregida
figure;
pcshow(ptCloudMap);
xlabel('X'); ylabel('Y'); zlabel('Z');
title('Nube de Puntos Corregida');
hold on
plot(smoothWayP(:,1), smoothWayP(:,2),LineWidth=2,DisplayName="Reference Path");
for i = 1:numPoses
    plot(vSetMapping.Views.AbsolutePose(i,1).Translation(1),...
        -
vSetMapping.Views.AbsolutePose(i,1).Translation(2),Color="y",Marker=".",MarkerSize=8)
end
hold off
%%
% pts = ndtMap.VoxelMean;
pts = sMap.VoxelMean;

mapLocation = [xlims(1), ylims(1)];

% Length and width of costmap in meters.
mapWidth = xlims(2)-xlims(1);
mapLength = ylims(2)-ylims(1);
```

```
% Set the cost value for empty space to zero.
costVal = 0;
% Side length of each square cell
cellSize = 1;

costmap =
vehicleCostmap(mapWidth,mapLength,costVal,CellSize=cellSize,MapLocation=mapLocation);

costmap.CollisionChecker.InflationRadius = 0.25;
xyPoint = round([pts(:,1),pts(:,2)]);
costVal = 1;
setCosts(costmap,xyPoint,costVal)

figure
plot(costmap)
title("Vehicle Costmap")
xlabel("X (m)")
ylabel("Y (m)")

%% Funciones Auxiliares
function ptCloudArr = helperGetPointClouds(simOut)
    ptCloudData = simOut.ptCloudData.signals.values;
    ptCloudArr = pointCloud(ptCloudData(:,:,:,1));
    for n = 2:size(ptCloudData, 4)
        ptCloudArr(end+1) = pointCloud(ptCloudData(:,:,:,n));  %#ok<AGROW>
    end
end

function gTruth = helperGetLidarGroundTruth(simOut)
    numFrames = size(simOut.lidarLocation.time, 1);
    gTruth = repmat(rigidtform3d, numFrames, 1);
    for i = 1:numFrames
        gTruth(i).Translation = squeeze(simOut.lidarLocation.signals.values(:,:,i));
        yaw = double(simOut.lidarOrientation.signals.values(:,3,i));
        gTruth(i).R = [cos(yaw) -sin(yaw) 0;
                       sin(yaw)  cos(yaw) 0;
                          0         0     1];
    end
end


function ptCloud = helperProcessPointCloud2(ptCloudIn)
%helperProcessPointCloud Processes the pointCloud to remove ground and ego vehicle
%   ptCloud = helperProcessPointCloud(ptCloudIn) processes
%   ptCloudIn by removing the ground plane and the ego vehicle.

egoRadius = coder.const(3.0);
cylinderRadius = coder.const(30);
selectedIdx = findPointsInCylinder(ptCloudIn, [egoRadius cylinderRadius]);
ptCloud = select(ptCloudIn, selectedIdx, OutputSize="full");

end

function [ptCloudMap, vSet] = helperSLAMVisualization(location,absTformOut)
%helperSLAMVisualization function takes location data of point clouds
```

```
% and the absolute transformation for each point cloud as input
% arguments, and visualizes the point cloud map with view set
% connections overlaid on the map.
%
% This is a helper function for example purposes and may be removed or
% modified in the future.

% Copyright 2023 The MathWorks, Inc.

% Initialize the view set.
vSet = pcviewset;
skipFrames = 2;
viewId = 1;

for idx = 1:skipFrames:size(location,4)
    ptCloud = pointCloud(location(:,:,:,idx));
    absTform = rigidtform3d(absTformOut(:,:,viewId));
    vSet = addView(vSet,viewId,absTform,PointCloud=ptCloud);
    if viewId > 1
        vSet = addConnection(vSet,viewId-1,viewId);
    end
    viewId = viewId + 1;
end

% Create point cloud map based on point clouds and absolute transformations
% present in the view set.
ptClouds = vSet.Views.PointCloud;
absPoses = vSet.Views.AbsolutePose;
mapGridSize = 0.2;
ptCloudMap = pcalign(ptClouds,absPoses,mapGridSize);

hFigAfter = figure(Name="Lidar SLAM");
hAxAfter = axes(hFigAfter);
pcshow(ptCloudMap,Parent=hAxAfter)

% Overlay view set display.
hold on
plot(vSet,Parent=hAxAfter)
hold off
end

function ptCloud = helperPreProcessPointCloud(ptCloud,egoRadius,cylinderRadius,
ElevTh)

% Select the points inside the cylinder radius and outside the ego radius.
cylinderIdx = findPointsInCylinder(ptCloud,[egoRadius cylinderRadius]);
ptCloud = select(ptCloud,cylinderIdx,'OutputSize','full');

% Remove ground.
[~,ptCloud] = segmentGroundSMRF(ptCloud,'ElevationThreshold',ElevTh); %0.05

end
```

## A3. Código del bloque planificador.

```
function waypoints = planning(startPos, endPos, mapPts, MinTurnRad, MotPrimLeg,
ReverseC)

persistent waypointsInternal
persistent drivDirInternal
persistent prevStartPos
persistent prevEndPos

% Set constant parameters
maxNumNodes = 5000;
xlims = [96.08 321.08];
ylims = [177.66 327.66];

% Initialize persistent variables that are used downstream
if isempty(prevStartPos)
    prevStartPos = startPos;
end

if isempty(prevEndPos)
    prevEndPos = endPos;
end

isNewPath = (~isequal(prevStartPos, startPos) || ~isequal(prevEndPos, endPos));

if isempty(waypointsInternal) || isNewPath
    % Two conditions in which waypoints must be set: they are not yet
    % initialized, or they are initialized, but the current start and end
    % do not match those used to generate the previous set of waypoints
    waypointsInternal = planPath(mapPts, maxNumNodes, startPos, endPos);
end

% Update the outputs
waypoints = waypointsInternal;

% Update the persistent variables that indicate previous state
prevStartPos = startPos;
prevEndPos = endPos;

%% Helper Function

    function wayPoints = planPath(mapPts, maxNodes, startPos, endPos)

% Crea un Mapa de Costos para el robot
    costmap = vCostMap(mapPts);

    ss = stateSpaceSE2;
    ss.StateBounds = [170 245; 180 326; 0 2*pi];
    sv = validatorVehicleCostmap(ss,Map=costmap);
    sv.ValidationDistance = 0.1;
    % MinTurnRad = 2; % 4
    % MotPrimLeg = 1.5; %1.8
    % ReverseC = 10; %10
    planner = plannerHybridAStar(sv,MinTurningRadius=MinTurnRad,...
                            MotionPrimitiveLength=MotPrimLeg,...
```

```
                                ReverseCost=ReverseC); % 5 4 2 ReverseCost=10

    refPath = plan(planner,startPos,endPos,SearchMode="exhaustive");
    path = refPath.States(:,1:2);
    goalP = endPos(:,1:2);
    wayPoints = repmat(goalP(:)', maxNodes, 1);

    % Conversion a occupancyMap
    resolution = 1/costmap.CellSize;
    origin = [costmap.MapExtent(1,1),costmap.MapExtent(1,3)];
    occMap = occupancyMap(costmap.Costmap, resolution);
    occMap.GridLocationInWorld = origin;

    optimPath = optPath(refPath,path,occMap);

    % Number of nodes that are actually used
    numNodes = length(optimPath);
    wayPoints(1:numNodes,:) = optimPath;

end

function map = vCostMap(mapPts)
    mapLocation = [xlims(1), ylims(1)];
    % Length and width of costmap in meters.
    mapWidth = xlims(2)-xlims(1);
    mapLength = ylims(2)-ylims(1);
    % Set the cost value for empty space to zero.
    costVal = 0;
    % Side length of each square cell
    cellSize = 1;

    ccConfig = inflationCollisionChecker('InflationRadius',0.25);

    costmap =
vehicleCostmap(mapWidth,mapLength,costVal,CellSize=cellSize,MapLocation=mapLocation,.
..
    CollisionChecker=ccConfig);

    % costmap.CollisionChecker.InflationRadius = 0.25;
    xyPoint = round([mapPts(:,1),mapPts(:,2)]);
    costVal = 1;
    setCosts(costmap,xyPoint,costVal);
FixZ = [200.58, 311.16;200.58, 310.16;200.58, 309.16;
        211.58, 300.16;211.58, 299.16;211.58, 298.16; 191.58, 199.16;
        201.58, 288.16;201.58, 287.16;190.58, 301.16; 190.58, 272.16;
        179.58, 202.16;179.58, 201.16;234.58, 205.16;];
FixO = [215.58, 190.16;216.58, 190.16;217.58, 190.16; 226.58, 189.16;
        231.58, 315.16;231.58, 314.16;231.58, 313.16];
setCosts(costmap,FixZ,0); setCosts(costmap,FixO,1);

    map = costmap;
end

    function optimPath = optPath(refPath,path,occMap)
    %Parametros de la optimizacion
```

```
    options = optimizePathOptions;
    options.MinTurningRadius = 3; % meters
    options.MaxVelocity = 5; % m/s
    options.MaxAcceleration = 1; % m/s/s
    options.ReferenceDeltaTime = 0.2; % second
    separationBetweenStates = 0.2; % meters
    numStates = refPath.pathLength/separationBetweenStates;
    options.MaxPathStates = round(numStates);
    options.ObstacleSafetyMargin = 2; % meters
    options.ObstacleInclusionDistance = 0.75; % meters
    options.ObstacleCutOffDistance = 3; %i meters2.5
    %
    % options.ObstacleSafetyMargin = 1.8; % meters 2 2
    % options.ObstacleInclusionDistance = 0.75; % 0.75meters
    % options.ObstacleCutOffDistance = 2.5; %i meters 2.5 3

    options.NumIteration = 4;
    options.MaxSolverIteration = 15;

    options.WeightTime = 10;
    options.WeightSmoothness = 1000;
    options.WeightMinTurningRadius = 10;
    options.WeightVelocity = 10;
    options.WeightObstacles = 50;

    inpath = [path(:,1), path(:,2)];
    [optimizedPath,kineticInfo] = optimizePath(inpath,occMap,options);
    optimPath = [optimizedPath(:,1), optimizedPath(:,2)];

end


end
```