Lic. en Cs. de la Computación

Simulación de robot desmalezador y planificación de trayectorias en entornos agrícolas

---

Ismael Ait
ismaelaitd@gmail.com

Director

Taihú Pire
pire@cifasis-conicet.gov.ar

Co-Director

Ernesto Kofman
kofman@cifasis-conicet.gov.ar

Facultad de Ciencias Exactas, Ingeniería y Agrimensura
Universidad de Rosario
Av. Pellegrini 250 - Planta Baja - (S2000BTP)
Rosario - Rep. Argentina.
Teléfono: +54 - 341 - 4802649/52 - interno 112
http://www.fceia.unr.edu.ar

Simulación de robot desmalezador y planificación de trayectorias en entornos agrícolas

En los últimos años el uso de robots autónomos móviles ha experimentado un crecimiento considerable en diferentes sectores de la industria. La inevitable necesidad de aumentar la producción de alimentos debido al crecimiento en la población mundial, ha despertado en la industria agrícola un singular interés sobre estas nuevas tecnologías. Una de las partes principales de la navegación autónoma corresponde a la planificación de los caminos y trayectorias que el robot deberá seguir. Los algoritmos de planificación de caminos buscan generar una ruta puramente geométrica, libre de la colisión con obstáculos, que guíe al robot desde un punto inicial a un punto meta, mientras que la planificación de trayectorias añade la variable de tiempo a dicha ruta indicando la forma en la que el robot deberá recorrerla. En entornos agrícolas, un robot terrestre con ruedas debe recorrer el campo siguiendo las hileras de sembrado, para que de esta manera evite pisar los cultivos. Este problema puede extenderse buscando la optimización de alguna característica como ser la ruta de longitud mínima, la de menor tiempo de ejecución, la que pase lo más lejos posible de los obstáculos o la más suave, solo por nombrar algunas.

En este trabajo se presenta un sistema de navegación para el robot desmalezador de cultivos de soja desarrollado en el Centro Internacional Franco-Argentino de Ciencias de la Información y de Sistemas (CIFASIS), poniendo foco en la etapa de planificación de trayectorias. Para esto se utiliza el *framework* ROS (*Robot Operating System*) y el planificador local TEB (*Timed Elastic Band*). Dicho planificador optimiza las trayectorias del robot de forma iterativa partiendo de las rutas generadas por el planificador global y adaptándolas a las restricciones dinámicas y de maniobrabilidad del robot. Por otro lado, se desarrolla una simulación en Gazebo con un modelo virtual del robot desmalezador y varias configuraciones de campos agrícolas. Dicha simulación permite la prueba de las distintas partes del sistema de navegación, como ser la localización, mapeo, planificación de trayectorias y control del robot. Los resultados obtenidos muestran que el sistema es capaz de trabajar en tiempo real y con suficiente precisión para evitar el pisado de los cultivos. Adicionalmente, se estudia la generación de diferentes recorridos que le permiten al robot cubrir con sus rociadores la totalidad de las hileras de cultivos del campo de forma eficiente. Se obtienen recorridos que logran un ahorro de alrededor de un 20 % del tiempo de ejecución comparados con recorridos triviales.

Agradecimientos

Agradezco profundamente a todos los profesores y compañeros que tuve a lo largo de la carrera. Sinceramente siempre encontré la buena predisposición y el apoyo que necesitaba en todos y cada uno de ellos. Especialmente quiero darle las gracias a mis directores Taihú y Ernesto que estuvieron para ayudarme y acompañarme en todo momento a lo largo de la realización de esta tesina. Asimismo, quiero darle las gracias a todo el equipo que trabajó y trabaja actualmente en el proyecto del robot desmalezador del CIFASIS por su asistencia. Por otro lado, no quiero dejar de nombrar y agradecer a Raúl Kantor y Mauro Jaskelioff que me incentivaron a continuar en el mismo plan de estudios y me ayudaron en los trámites que fueron necesarios para así poder hacerlo. Quisiera agradecer también a todas las personas fuera del ámbito académico que me aguantaron y acompañaron a lo largo la carrera, en especial a mis padres Hugo y Olga, a quienes les dedico esta tesina. Sin su constante apoyo y palabras de motivación nunca podría haber logrado esto.

Índice general

Capítulo 1

Introducción

La investigación en robótica móvil goza actualmente de gran interés práctico en una amplia variedad de aplicaciones que pueden ir desde la exploración espacial, tareas de búsqueda y rescate de personas en territorios peligrosos, manufactura, construcción, trabajo submarino, asistencia a personas con capacidades diferentes, y hasta la realización de actividades domésticas o de servicio. El desafío principal en este campo consiste en lograr la completa autonomía del agente, es decir, dotar al robot de la capacidad de llevar a cabo sus tareas sin la necesidad de la intervención de un operador humano. Particularmente, la capacidad de una navegación autónoma es sumamente importante, ya que por definición un robot realiza sus tareas moviéndose e interactuando con el mundo real. El problema de la navegación autónoma se puede dividir en tres partes: construcción de un mapa del ambiente, localización del agente en el mapa y planificación de los caminos a seguir. El problema de la planificación de caminos, conocido como *path planning* en inglés, ha sido estudiado ampliamente desde hace varias décadas. El objetivo de los algoritmos de planificación de caminos consiste en determinar una ruta libre de obstáculos desde un punto inicial hasta un punto meta, intentando minimizar alguna variable predefinida como puede ser la distancia total recorrida o el tiempo empleado en su ejecución. Dicho plan de camino puede extenderse con información temporal que permite establecer la manera en que deberá ser ejecutado, lo cual convierte al camino puramente geométrico en lo que se conoce como trayectoria. Aún cuando *camino* y *trayectoria* son conceptos diferentes, en la tesina se hará uso mayormente del término *trayectoria* dado que el tiempo será una variable de relevancia.

Este trabajo se enfoca en el estudio de los algoritmos de planificación de caminos y trayectorias particularmente en un ámbito que cada vez más busca aprovechar los beneficios de la automatización y utilización de robots: la industria agrícola. Como objetivo primordial, se busca desarrollar un método adecuado de planificación de trayectorias para el uso en el robot desmalezador de cultivos de soja desarrollado en el CIFASIS (ver sección 1.2). Dado que el robot deberá funcionar en el campo agrícola, se estudiarán solo métodos de planificación compatibles con dicho entorno, adecuados para trabajar en el plano bidimensional y al aire libre. A su vez, debido a que la validación de los distintos métodos y algoritmos de navegación en entornos reales conlleva elevados costos en logística y ejecución, también se aborda el desarrollo de un entorno virtual de simulación para el robot, el cual se utiliza para la prueba de los métodos de planificación cubiertos en este trabajo.

1.1. Agricultura de precisión

El constante incremento en la población mundial conduce inevitablemente a la necesidad de generar un aumento en la producción de alimentos y biocombustibles de forma eficiente y sustentable en términos tanto económicos como ecológicos [1][2]. Se estima que para el año 2050, la población mundial trepará a 9 mil millones de personas, lo cual requerirá doblar la producción agrícola [3]. Es fundamental que este incremento se realice, en todo momento, buscando la reducción del impacto humano sobre el medio ambiente y los niveles de contaminación. Sumado a esto, el sector agrícola se enfrenta actualmente al problema de la escasez de

mano de obra debido a la migración de personas del campo a las grandes ciudades [4].

En este marco, el desarrollo de dispositivos robóticos propone una solución atractiva, particularmente en el campo de los vehículos autónomos, gracias a sus beneficios de precisión, repetibilidad y durabilidad. Los robots autónomos se pueden utilizar para tareas de siembra, cosecha, fertilización, fumigación y recolección de datos de precisión, entre otras cosas. Particularmente, la recolección masiva de datos a un bajo costo favorece al desarrollo de lo que se conoce en la actualidad como agricultura de precisión [5]. La agricultura de precisión consiste en la obtención a gran escala de datos sobre el campo y la adaptación del proceso de producción basada en dichos datos para aumentar la producción y reducir el desperdicio. Gracias a la robótica móvil tanto el proceso de obtención de datos como la aplicación de los diferentes tratamientos sobre los cultivos puede lograrse de una manera muy eficiente y con una precisión que puede llevarse a niveles muy altos, incluso individualmente en cada planta (*plant-scale husbandry*), algo que resultaría prácticamente imposible de realizar por medio de operadores humanos [6].

Sistemas de navegación semiautomáticos se están utilizando desde hace algunos años en diversas maquinarias agrícolas, fomentado principalmente por los avances en tecnologías de GPS (*Global Positioning System*) y visión por computadora. Este tipo de maquinarías consiste en tractores y cosechadoras de gran porte con asistencia de navegación, pero todavía con supervisión humana. El empleo de robots pequeños completamente autónomos ofrece mayores ventajas, ya que por ejemplo pueden funcionar durante más tiempo de forma ininterrumpida, facilitan el uso de energías renovables como los paneles solares y producen una menor compactación de los suelos. Más aún, los robots de menor escala resultan más adecuados para trabajos a nivel individual de una planta, como el cultivo selectivo, fertilización y desmalezado de precisión que reduce costos en fertilizantes y herbicidas [7]. Sin embargo, la navegación autónoma en entornos agrícolas constituye todavía en la actualidad un problema desafiante para los investigadores, principalmente por tratarse de ambientes complejos y cambiantes con condiciones que varían como el viento, la luz, la temperatura, y el polvo. Los campos agrícolas suelen ser terrenos irregulares y visualmente repetitivos, lo cual puede ser un problema para los sistemas de localización basados en visión. Por otro lado, para evitar el pisado de los cultivos el sistema de navegación necesita poseer un alto grado de precisión.

En este sentido se encuentran en desarrollo diferentes prototipos de robots autónomos móviles para la realización de diversas tareas en el campo. Un caso particular de estos, es el robot desmalezador de cultivos de soja descrito en la siguiente sección.

1.2. Robot desmalezador del CIFASIS

Este trabajo está enmarcado dentro del desarrollo del robot desmalezador (ver Figura 1.1), llevado adelante por el CIFASIS (CONICET-UNR). Dicho robot tiene como objetivo desplazarse a través de los surcos de campos de soja de forma completamente autónoma, al mismo tiempo que detecta, mediante visión por computadora, la presencia de malezas en el cultivo y aplica herbicidas de forma localizada, evitando de esta manera dañar el entorno u otras personas. Este proyecto surge como una alternativa ecológica y de bajo costo a la eliminación de malezas, y en contraposición con el uso indiscriminado de agroquímicos que afecta hoy en día las principales áreas agrícolas de la región e impacta negativamente en el medio ambiente [8].

El robot desmalezador actualmente cuenta con una serie de sensores y actuadores que se describen brevemente a continuación:

- Tres paneles solares y juego de baterías para obtener una autonomía energética.
- Cuatro servo motores eléctricos (*in-wheel*), uno por cada rueda.
- Un motor eléctrico paso a paso para el control de la dirección.
- GPS-RTK, el cual proporciona una estimación de la localización del robot con precisión de centímetros que sirve como referencia para evaluar otros sistemas de localización basados en el resto de los sensores del robot.
- IMU (*Inertial Measurement Unit*), compuesta por acelerómetro y giróscopo.

Figura 1.1: Robot desmalezador desarrollado en el CIFASIS recorriendo un campo de cultivos de soja en la en la Facultad de Ciencias Agrarias de la Universidad Nacional de Rosario, en la localidad de Zavalla.

- Cámara Zed, con el objetivo de ser el sensor principal para la localización, mapeo y detección de las malezas. Al tratarse de una cámara de tipo estéreo permite realizar una reconstrucción 3D de la escena.
- Mini-PC Intel R NUC Kit NUC6CAYH 2 (Intel Celeron J3455 CPU, quad-core 2,3 GHz, 8 GB DDR3 RAM) con Ubuntu 16.04, para poder obtener la información de todos los sensores en forma sincronizada, guardar y procesar la información. A la misma se le incorpora un disco de estado solido con el objetivo de minimizar los tiempos de escritura/lectura.

Para la selección de los sensores utilizados se tuvo en consideración el entorno particular en el que se desplazará el robot, las lentas velocidades de trabajo, la autonomía del robot y, además, proveer de la información que pudiera ser necesaria a futuro para el desarrollo de métodos de localización que combinen distintos sensores o herramientas de adquisición de información.

La decisión de equipar al robot con un GPS-RTK [9] se debe a que será utilizado principalmente para la validación del sistema de SLAM, que solo deberá emplear el resto de los sensores (IMU, odometría y cámara estéreo). Gracias a la tecnología RTK (*Real Time Kinematic*) se puede obtener una precisión de centímetros, mientras que un GPS ordinario posee un error de entre 2 o 3 metros. Esta tecnología, también conocida como posicionamiento diferencial, consiste en la utilización de una estación base cuya posición es conocida de antemano para poder estimar el error de las señales satelitales y así mejorar la precisión de la localización. Se planifica que en el futuro el robot funcione únicamente con un GPS ordinario, ya que el costo de uno del tipo RTK es considerablemente más elevado y requiere de una infraestructura adicional en el campo (estación base).

1.3. Sistema de locomoción y planificación de caminos

Uno de los desafíos más significativos que enfrenta la robótica móvil recae en el área de la planificación de caminos (*path planning*, en inglés). Concretamente, los algoritmos de *path planning* buscan determinar un camino libre de obstáculos desde un punto inicial hasta un punto meta, buscando principalmente minimizar la distancia recorrida o el tiempo total empleado por el robot [10, 11, 12, 13, 14].

El problema de la planificación de caminos puede dividirse en planificación global y planificación local de evasión de obstáculos. La planificación global requiere un modelo conocido de antemano del lugar, que consistirá en una descripción simplificada del mundo real en el que navega el robot. El sistema de navegación

local seguirá los pasos determinados por el plan global, buscando mantenerse cerca del camino planificado al mismo tiempo que intenta evitar colisionar con los obstáculos nuevos que puedan ir apareciendo. Por otro lado, también puede introducir información temporal al plan que indique la manera en que deba ser recorrido dicho camino, convirtiéndolo en lo que se conoce como trayectoria (*trajectory*, en inglés). Mientras que la planificación global requiere un mapa del entorno conocido de antemano, el planificador local requerirá un modelo que refleje el estado más actual únicamente en las cercanías del robot. Es así que el planificador local dependerá principalmente de la información en tiempo real obtenida a través de los distintos sensores que posea el robot. Asimismo, a medida que la trayectoria es ejecutada por el robot se podrá actualizar el modelo global con la información obtenida desde el modelo local. En resumen, el modelo global puede verse como una colección de caminos factibles por los cuales podrá transitar el robot. El planificador global se encargará de seleccionar una serie de caminos para que el robot alcance su meta. Mientras que el planificador local se encargará de los detalles de cómo recorrer esos caminos y evitar obstáculos imprevistos, teniendo también en consideración las restricciones dinámicas y de maniobrabilidad del robot.

Existe una amplia variedad de algoritmos y métodos de planificación de caminos y trayectorias que dependerán de muchos aspectos relacionados con el sistema de locomoción del robot y su entorno de navegación. Una vez analizadas dichas características, se podrán elegir apropiadamente los algoritmos para resolver el problema de navegación. Los robots móviles pueden estar diseñados para desplazarse por ambientes terrestres, aéreos o acuáticos (por debajo del agua o en su superficie). A su vez, los robots terrestres pueden ser robots sobre ruedas o con patas. En este trabajo, solo nos enfocaremos en los robots terrestres sobre ruedas.

Existen principalmente dos mecanismos para el manejo de robots terrestres sobre ruedas. Por un lado, el mecanismo más utilizado en ambientes *indoor* involucra un control independiente de la rotación de las ruedas a cada lado del robot, lo que le otorga su nombre de manejo diferencial (*differential drive*). Por medio de una rotación de la rueda izquierda y derecha en direcciones opuestas a la misma velocidad, el robot puede realizar un giro sobre el lugar. Por otro lado, en ambientes *outdoor* se suele emplear un mecanismo diferente que consiste en la utilización de ruedas de direccionamiento como en las bicicletas y los autos. Este tipo de manejo presenta desafíos interesantes en términos de planificación y control debido a que poseen un radio de giro mínimo distinto de cero, limitado por la longitud del vehículo y el máximo ángulo de dirección de las ruedas. En los vehículos de cuatro ruedas, un correcto ángulo para las ruedas de dirección derecha e izquierda se logra mediante un sistema mecánico conocido como mecanismo de Ackermann. A estos últimos se los suele llamar robots *car-like* por su similitud con los autos. El robot desmalezador es un vehículo *car-like*, por lo que los algoritmos adecuados para el direccionamiento con mecanismo de Ackermann serán el foco de este trabajo.

Otra faceta importante de la navegación autónoma consiste en la localización del robot en el entorno y la generación de un mapa del mismo. Si bien estos aspectos son necesarios para la navegación no serán abordados en este trabajo. En la etapa de experimentación se toma la información que proporciona el simulador de la ubicación exacta del robot con respecto a su entorno simulado para calcular su localización. Por otro lado, el mapeo se realiza utilizando una imagen de mapa de bits que representa fielmente el entorno de simulación. Se adoptó este enfoque para poder aislar el problema de planificación de trayectorias del resto de los problemas que hacen a la navegación autónoma. Quedará para trabajo futuro, el análisis final de la navegación del robot combinando la planificación de trayectorias con métodos de localización y mapeo aplicables en el robot real.

La posibilidad de acceder a computadoras con cada vez mayor poder de cálculo, facilita la creación de entornos de simulación que permiten la validación de los diferentes algoritmos de navegación. Con dichas simulaciones se posibilita la realización de pruebas en diferentes configuraciones del entorno a un costo prácticamente nulo comparado con la realización de las mismas en el campo real. En este trabajo, se utiliza el *framework* ROS junto con el simulador Gazebo para la realización de dichas pruebas. Se eligió ROS ya que permite que todo el software de control y navegación del robot que se desarrolla pueda ser migrado al robot real sin mayores cambios, y el simulador Gazebo se adapta perfectamente al entorno de ROS dado que ambos programas se originaron en la misma compañía, Willow Garage.

Debido a que el objetivo final del robot desmalezador consiste en aplicar herbicida a la totalidad de las malezas del campo, se deberá idear un plan que le permita al robot abarcar cada cultivo existente en el campo, preferentemente sin tener que pasar más de una vez por los mismos sectores. Este problema, conocido como planificación de caminos de cobertura (CPP por sus siglas en inglés: *Coverage Path Planning*) resulta en una

extensión del problema de *path planning* que trata la búsqueda de un camino que pase por todos los puntos de un espacio determinado [15, 16, 17]. La planificación de caminos de cobertura posee especial utilidad no solo en entornos agrícolas sino también en muchas otras aplicaciones de robótica como ser aspiradoras de pisos [16, 18], detectores de minas [19, 17], cortadoras de césped autónomas [20] y limpiadores de ventanas [21], solo por nombrar algunas.

Durante la etapa inicial de esta tesina, se realizó un estudio exhaustivo de los diferentes métodos existentes de planificación de caminos de cobertura y se ideó una combinación de métodos adecuada para las características del robot desmalezador. Dicho trabajo fue presentado en las décima Jornadas Argentinas de Robótica llevadas adelante en 2019, y se encuentra pendiente su publicación [22].

1.4. Objetivos

El objetivo principal de este trabajo consiste en la obtención de un sistema de navegación autónomo y eficiente para el robot desmalezador del CIFASIS, junto con el desarrollo de un entorno de simulación para la validación del mismo. Para esto, se realiza un extenso análisis del estado del arte en navegación autónoma, poniendo especial mirada en los diferentes métodos de planificación de trayectorias existentes. La solución propuesta emplea estándares abiertos y software *open source* como ROS[1] y Gazebo[2]. Se busca que la mayor parte del código pueda llevarse directamente al robot real con la menor cantidad de modificaciones posibles. Adicionalmente, se espera que el entorno de simulación desarrollado permita no solo la prueba y verificación de los algoritmos de navegación sino también la validación en el futuro de otros sistemas involucrados en el robot, como ser el sistema de localización, detección de malezas y detección de surcos.

1.5. Organización del trabajo

El resto de este trabajo se encuentra estructurado de la siguiente manera. El capítulo 2 presenta el estado del arte de los sistemas de navegación autónoma y simulación en robótica. Se define el problema de la planificación de caminos y se analizan los planificadores adecuados para utilizar en el robot desmalezador. Se describe el modelo cinemático del robot y su sistema de control. En el capítulo 3 se presenta el desarrollo de la simulación en la plataforma Gazebo. En esta sección, se detalla la generación del modelo virtual del robot desmalezador y de los campos agrícolas. Adicionalmente, se describe el mecanismo de dirección de Ackermann y el desarrollo de un controlador para su manejo. Por su parte, el capítulo 4 describe los algoritmos de planificación de caminos global y planificación de trayectorias local utilizados con sus respectivas configuraciones. Al final del mismo, se expone el desarrollo de un algoritmo de cobertura que le permitirá al robot recorrer el campo agrícola de forma completa. El capítulo 5 muestra los experimentos realizados con la simulación y los resultados obtenidos con sus respectivas comparaciones cuantitativas y cualitativas. En primera instancia, se realiza un análisis de los diferentes tipos de giros que el robot puede realizar en las cabeceras del campo y la comparación entre ellos en términos de longitud, duración y velocidad. Luego, se plantea el problema de la elección de recorridos óptimos para la cobertura total del campo agrícola y la búsqueda de su solución mediante diferentes algoritmos. Por último, en el capítulo 6 se presentan las conclusiones y los lineamientos de posibles trabajos futuros derivados de la presente tesina.

[1] Robot Operating System, https://www.ros.org.
[2] Gazebo Simulator, http://gazebosim.org.

Capítulo 2

Conceptos preliminares

En los últimos años el uso de robots autónomos móviles ha experimentado un crecimiento considerable en diferentes sectores de la industria. La inevitable necesidad de aumentar la producción de alimentos debido al crecimiento en la población mundial, ha despertado en la industria agrícola un singular interés sobre estas nuevas tecnologías. En la actualidad, ya podemos encontrar el empleo de robots móviles en la asistencia o ejecución de diversas tareas en este campo. La principal ventaja de los vehículos autónomos reside en que pueden navegar el campo recolectando cantidades enormes de información o realizando diversas tareas sobre los cultivos eliminando la necesidad de un operador humano. Se espera que en los próximos años los robots autónomos móviles sean una parte central de las aplicaciones de agricultura de precisión [4].

La planificación de caminos en entornos agrícolas tiene como principal objetivo lograr que el robot pueda recorrer el campo siguiendo las hileras de sembrado, para que de esta manera evite pisar los cultivos con sus ruedas. Por otro lado, se debe tener en cuenta que los robots agrícolas suelen ser vehículos no holonómicos, es decir, que poseen restricciones en sus movimientos, como ser un radio de giro limitado. Todo esto hace que el desarrollo o adaptación de métodos de *path planning* para el uso en este tipo de ambientes constituya un desafío interesante de abordar y con muchas oportunidades para mejoras.

A continuación se presenta el estado actual de los avances en robótica móvil para entornos agrícolas, junto con el estado del arte de los métodos de planificación de caminos y simulación en robótica, lo cual conforman la base de la motivación para este trabajo.

2.1. Robot móvil sobre ruedas

Un sistema robótico consiste en un dispositivo, generalmente complejo, compuesto por sensores y actuadores comandado por un sistema computacional, capaz de recibir una descripción de alto nivel de una tarea encomendada y transformarla en instrucciones de movimiento de bajo nivel para ejecutarla sin intervención humana adicional. La descripción de la tarea indicará **qué** debe hacer el robot y no **cómo** hacerlo [10]. Estos sistemas robóticos pueden clasificarse a grandes rasgos en dos tipos: los robots manipuladores y los robots autónomos móviles.

Un robot manipulador, generalmente está formado por un brazo articulado fijado en uno de sus extremo a una base estática y provisto de dispositivos para la sujeción y ubicación de piezas, o herramientas como cortadoras y soldadoras en su otro extremo. Desde ya hace varias décadas, este tipo de robots se ha aprovechado ampliamente en muchos sectores de la industria manufacturera, de los cuales se puede destacar tanto la fabricación de dispositivos electrónicos como el sector automotriz con sus líneas de ensamblaje automatizadas.

Al mismo tiempo que el uso de los robots manipuladores continúa proporcionando sus beneficios a la industria manufacturera, en los últimos años se ha visto un incremento considerable en la investigación y desarrollo de aplicaciones prácticas también en el campo de la robótica móvil. La posibilidad de emplear robots autónomos móviles posee un gran interés práctico en una amplia variedad de dominios de aplicación

que incluyen ámbitos como la exploración espacial, vigilancia, trabajo submarino, asistencia a personas con capacidades de movimiento reducidas, actividades domésticas y de servicio, entre otras. La principal característica de los robots móviles consiste en la presencia de un sistema de locomoción que le permite al robot moverse libremente por el entorno mientras ejecuta una o varias tareas. Esto proporciona mayor flexibilidad de trabajo que los robots manipuladores y la posibilidad de actuar en ambientes desconocidos. Los robots manipuladores compuestos por brazos fijos generalmente son configurados para la realización de una tarea específica en un punto determinado de una cadena de producción, mientras que los robots móviles pueden utilizarse para múltiples tareas, incluso en ambientes al aire libre, aprendiendo y adaptándose de forma automática al entorno.

Los robot autónomos móviles pueden desenvolverse en ambientes aéreos, acuáticos o terrestres, siendo estos últimos los de mayor utilización en aplicaciones prácticas en la actualidad. A su vez, los robots móviles terrestres pueden subdividirse en diferentes tipos, dentro de los cuales se destacan los robots con patas (bípedos o cuadrúpedos) y los vehículos sobre ruedas. Cabe destacar que la vasta mayoría de robots móviles terrestres actualmente utilizados en aplicaciones reales (no experimentales) están formados por robots sobre ruedas y este trabajo analizará solamente robots de este tipo.

2.1.1. Tipos de ruedas

Los robots móviles sobre ruedas están formados por una base o chasis rígido y un sistema de ruedas que pueden ser de diferentes tipos, como los que se muestran en la Figura 2.1. Los mismos se describen a continuación:

Fija (Figura 2.1a). Una rueda de este tipo solo podrá rotar sobre el eje que atraviesa el centro de la rueda, perpendicular al plano de la misma. Este tipo de rueda está conectada de forma fija al chasis de modo que su orientación con respecto a éste se mantiene constante.

Direccionable (Figura 2.1b). Esta rueda es similar a la anterior, con el agregado de un nuevo eje de rotación vertical. La rotación sobre este nuevo eje permite el cambio de orientación de la misma con respecto al chasis. Ambas rotaciones pueden ser comandadas de forma independiente.

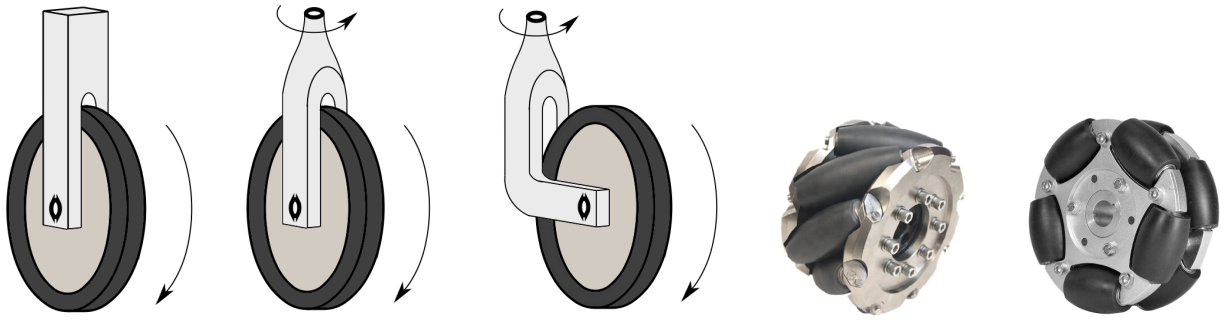
Giratoria (Figura 2.1c). Conocida también como rueda *castor* o *caster*, posee dos ejes de rotación como la anterior pero con la diferencia de que el eje vertical no pasa a través del centro de la rueda sino que está desplazado por medio de un *offset* constante. Esto permite que la rueda pivote automáticamente de manera casi instantánea alineándose con la dirección del chasis. Las ruedas de este tipo se utilizan para proporcionar soporte y estabilidad al robot, y se dejan las rotaciones libres en ambos ejes. Ejemplos de este tipo de ruedas podemos encontrar en los carritos de supermercados o en las sillas de oficina.

Mecanum (Figura 2.1d). También conocida como rueda sueca, fue ideada para permitir el movimiento de un vehículo en cualquier dirección [23]. Consiste en una rueda convencional, no direccionable, con una serie de rodillos pasivos de goma ubicados a lo largo del borde externo. El eje de rotación de cada rodillo se encuentra generalmente inclinado 45° con respecto al plano de la rueda. Un vehículo equipado con cuatro de estas ruedas montados en pares en dos ejes paralelos resulta en un vehículo omnidireccional.

Omnidireccional (Figura 2.1e). También conocida como *poly-wheel*, es un tipo similar al anterior, conformado por una rueda convencional con pequeños discos o rodillos sobre la circunferencia de la misma pero dispuestos de forma perpendicular a la dirección de giro de la rueda [24]. Con este tipo de ruedas se pueden obtener configuraciones omnidireccionales de la misma forma que con las Mecanum.

2.1.2. Configuraciones de robots móviles

Es posible generar una amplia variedad de estructuras cinemáticas con la combinación de los diferentes tipos de ruedas vistos en la sección anterior. Las imágenes de la Figura 2.2 ilustran algunas de las configuraciones más comúnmente usadas en robots móviles, las cuales se describen brevemente a continuación:



(a) Fija. **(b)** Direccionable. **(c)** Giratoria. **(d)** M*ecanum*. **(e)** Omnidireccional.

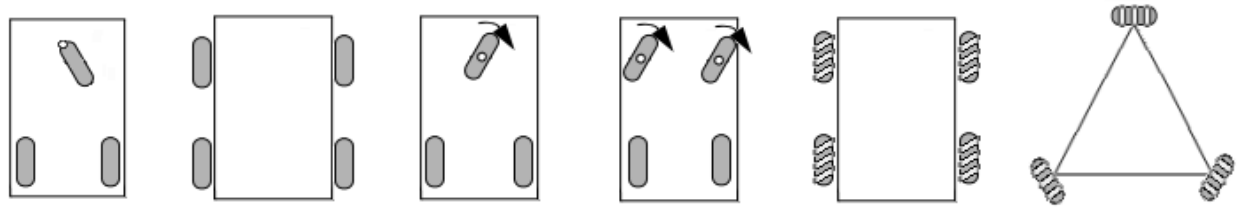
Figura 2.1: Tipos de ruedas más comúnmente utilizadas en robots terrestres.

Differential-drive (Figura 2.2a). Es también conocido como robot de tipo Hilare, por su empleo en el reconocido robot desarrollado por el instituto LAAS de Tolouse en 1977 [25]. Este tipo de vehículos está compuesto por dos ruedas fijas con el mismo eje de rotación pero comandadas independientemente, y una o más ruedas *caster* de menor tamaño para mantener la estabilidad del robot. Para realizar los giros se deben aplicar velocidades de rotación diferentes a cada una de las ruedas laterales, lo que origina su nombre de manejo diferencial. Se utilizan ampliamente en ambientes *indoor*, por ejemplo, en aplicaciones como las aspiradoras automáticas. Si bien no son robots holonómicos, puesto que no pueden moverse lateralmente, poseen buena maniobrabilidad con radio mínimo de giro cero y de fácil control.

Skid-steer (Figura 2.2b). Este tipo de vehículos está compuesto por cuatro ruedas fijas controladas de forma independiente. Los giros se realizan otorgándole diferentes velocidades a los pares de ruedas del lado izquierdo y derecho. Este sistema es popular por su simplicidad mecánica, ya que no posee sistema adicional de direccionamiento, y por su gran maniobrabilidad y capacidad de girar en radios muy pequeños. La desventaja, por otro lado, resulta en el derrapado de las ruedas inherente al modo de operación, lo que torna difícil la predicción precisa de sus movimientos [26]. Se utiliza mucho en aplicaciones *outdoor*.

Triciclo (Figura 2.2c). Compuesto por dos ruedas fijas montadas sobre un eje trasero y una rueda direccionable en el frente. Las ruedas fijas suelen estar controladas por un único motor que provee la tracción al robot. La transmisión del torque en estas ruedas se suele llevar a cabo mediante un mecanismo diferencial para que se distribuya de forma equitativa durante los giros, debido a que la rueda interna tendrá menor velocidad que la externa. La rueda direccionable es manejada por otro motor que cambia su orientación con respecto al chasis. Alternativamente, las ruedas traseras pueden ser pasivas y la tracción ser proporcionada por la rueda delantera. Debido a que tiene un radio de giro mínimo distinto de cero, está limitada su maniobrabilidad. La trayectoria planificada deberá tener en cuenta este radio de giro puesto que el robot no podrá seguir un camino si éste propone giros más pronunciados.

Car-like (Figura 2.2d). Consiste en un vehículo de cuatro ruedas, semejante a un automóvil. Posee dos ruedas fijas montadas sobre un mismo eje trasero y dos ruedas delanteras direccionables montadas en el eje delantero. Generalmente, un motor provee tracción en las ruedas traseras, aunque también puede hacerlo sobre las ruedas delanteras o sobre las cuatro ruedas, o bien existir más de un motor de tracción. En los casos que corresponda, el torque es transmitido mediante un mecanismo diferencial como en el tipo anterior. Cualquiera sea el modelo de tracción, otro motor independiente es el encargado de manejar la orientación de las ruedas delanteras. Cabe destacar que para evitar deslizamientos indeseados, las ruedas delanteras deben tener orientaciones ligeramente diferentes mientras el vehículo realiza un giro. Particularmente, la rueda interna deberá tener un mayor ángulo con respecto a la externa. Esto se consigue siguiendo la geometría de dirección de Ackermann. Al igual que el caso anterior, su radio de



(a) *Diff-drive.* **(b)** *Skid-steer.* **(c)** Triciclo. **(d)** *Car-like.* **(e)** *Mecanum.* **(f)** *Omni-wheel.*

Figura 2.2: Configuraciones de robots terrestres con diferentes implicancias cinemáticas.

giro es distinto de cero, lo que limita su maniobrabilidad. El robot desmalezador analizado en este trabajo, resulta ser un vehículo de este tipo, en el que existe un motor proporcionando tracción a cada rueda de forma independiente y otro para la dirección.

Holonómico. Los robots móviles holonómicos más comunes son dos: uno compuesto por tres ruedas *omni-wheels* dispuestas en un triángulo equilátero (ver Figura 2.2f) y otro compuesto por cuatro ruedas de tipo Mecanum en forma rectangular (ver Figura 2.2e). Los rodillos transfieren una porción de la fuerza de la dirección de rotación de la rueda hacia una fuerza en dirección al giro de los rodillos. Dependiendo del sentido y la velocidad de giro de cada rueda se puede producir un vector resultante de fuerza tal que pueda mover el vehículo en cualquier dirección deseada, sin necesidad de cambiar la orientación de las ruedas. El control de este tipo de robots resulta más complejo que los descritos anteriormente. Estos robots poseen todos los grados de libertad posibles, pudiendo desplazarse hacia adelante y atrás, como también hacia los lados y girar sobre sí mismos. Sin embargo, el deslizamiento indeseado de las ruedas es un problema común en este tipo de vehículos debido a que en todo momento cada rueda tiene un solo rodillo en contacto con el suelo, y estos proporcionan menor fricción (agarre) que las ruedas convencionales. Debido a esto, no son adecuados para ambientes exteriores de terreno irregular y generalmente solo se los emplea en entornos hogareños o industriales *indoor*. Además hay que tener en cuenta que su movimiento lateral y de giro resulta relativamente lento, comparado con el desplazamiento hacia adelante y atrás, y también comparado con los giros de otros robots que poseen ruedas direccionables como los *car-like*.

De las diferentes configuraciones vistas, las que predominan en aplicaciones prácticas son las de manejo diferencial (*differential drive*) y los vehículos de cuatro ruedas con dirección delantera de Ackermann (*car-like*). Los robots de manejo diferencial se utilizan mayormente en ambientes interiores y en tareas domésticas. Por otro lado, los robots de cuatro ruedas se utilizan principalmente en ambientes al aire libre y en aplicaciones más avanzadas, debido a su mayor robustez. Este trabajo se enfoca únicamente en los robots de cuatro ruedas con dirección delantera de Ackermann, debido a que este tipo de robots es adecuado para emplearse en entornos agrícolas.

A excepción de los robots que utilizan ruedas del tipo *omni-wheel* o Mecanum, cualquier otro vehículo sobre ruedas estará sujeto a restricciones cinemáticas que en general reducen la movilidad local instantánea del robot, mientras que dejan intacta la posibilidad de alcanzar cualquier configuración posible mediante las maniobras apropiadas. Por ejemplo, sabemos que a pesar de que es imposible mover instantáneamente un auto de forma lateral, es posible estacionarlo entre otros dos autos siempre que tengamos un margen de maniobrabilidad. Es fundamental entonces considerar el modelo cinemático del robot y analizar en detalle sus restricciones.

2.1.3. Espacio de configuraciones

Al **espacio de trabajo** (*workspace*) de un robot móvil, representado por la letra $\mathcal{W}$, se lo define como el área accesible por el mismo y es potencialmente ilimitado. Por ejemplo, para los robots móviles terrestres

su espacio de trabajo puede ser $\mathbb{R}^2$. Por otro lado, al **espacio de configuraciones** (*configuration space*), representado por la letra $\mathcal{Q}$, se lo define como el conjunto de todas las posiciones y orientaciones posibles que el robot puede tomar dentro de su espacio de trabajo. A partir de una **configuración** particular tomada del espacio de configuraciones, lo cual se nota con $q \in \mathcal{Q}$, se puede determinar la ubicación de todos los puntos del robot dentro de su espacio de trabajo. La noción del espacio de configuraciones fue formulado por primera vez en el contexto de planificación de movimientos en [27] y es ampliamente utilizada en diferentes textos de *path planning*, de los cuales podemos destacar [28, 29].

La configuración de un robot puede ser representada de muchas maneras, siempre y cuando se consiga con ésta una definición completa de la ubicación de todos los puntos del robot. Generalmente, la representación de una configuración se hace mediante una serie de variables de tipo numéricas. Debido a que un robot está formado principalmente por cuerpos rígidos y de tamaño determinado, solo una cantidad limitada de variables es necesaria para representar su configuración. Por ejemplo, la configuración de una puerta puede especificarse mediante un único parámetro $q = \theta$ que determine el ángulo de apertura, mientras que un simple punto en el plano puede especificarse mediante dos parámetros $q = (x, y)$.

La configuración de un robot terrestre se representa habitualmente dando la posición de un punto (x, y) del robot y el ángulo de orientación θ hacia donde apunta el robot. En los robots *car-like*, se toma como punto de referencia para indicar su posición el centro de su eje trasero, que coincide con el punto sobre el cual se especifica también su radio de giro. Dicho esto, la configuración queda determinada con $q = (x, y, \theta)$. Las primeras dos variables indican la ubicación en el plano del centro del eje trasero del robot y la última variable indica su orientación. A esta definición de posición y orientación del robot también se la conoce como **pose**.

El número de **grados de libertad** (DoF, por sus siglas en inglés *Degrees of Freedom*) de un robot consiste en la dimensión de su espacio de configuraciones, o el mínimo número de variables necesarias para especificar su configuración. Si se elige un número mayor de variables, digamos n, para representar la configuración de un robot, entonces existirán k ecuaciones independientes tales que $k = n - \text{DoF}$, como se demuestra en [14]. Expresado de otra manera, tenemos que:

$$\text{grados de libertad} = (\text{número de variables}) - (\text{número de ecuaciones independientes}). \tag{2.1}$$

Para ejemplificar esto, supongamos que elegimos tres puntos A, B y C del robot como se muestra en la Figura 2.3. Dado un marco de coordenadas determinado, la ubicación de los tres puntos en el plano puede ser escrita como (x_A, y_A), (x_B, y_B) y (x_C, y_C). Si dichos puntos pudiesen ser ubicados en cualquier lugar de forma independiente, el robot tendría seis grados de libertad, dos para cada uno de los tres puntos. Pero debido a que el robot está compuesto por cuerpos rígidos, la distancia entre los puntos A y B, notada como d_{AB}, tiene que mantenerse siempre constante. De igual forma, d_{AC} y d_{BC} también tienen que ser constantes. Por lo tanto, la ubicación de los tres puntos deberá satisfacer las siguientes ecuaciones independientes:

$$\begin{aligned}
\sqrt{(x_A - x_B)^2 + (y_A - y_B)^2} - d_{AB} &= 0, \\
\sqrt{(x_A - x_C)^2 + (y_A - y_C)^2} - d_{AC} &= 0, \\
\sqrt{(x_B - x_C)^2 + (y_B - y_C)^2} - d_{BC} &= 0.
\end{aligned} \tag{2.2}$$

Entonces, tenemos seis variables para definir la ubicación de estos puntos pero tres restricciones independientes sobre estas variables, lo que da como resultado que el robot tendrá solo tres grados de libertad. Puesto en otra forma, primero elegimos la ubicación (x, y) del punto A, lo que nos da dos grados de libertad. Luego, el punto B debe ubicarse dentro de la circunferencia de un círculo centrado en A con radio d_{AB}. Este punto puede ser determinado con un solo parámetro que indique el ángulo θ entre el vector $\overrightarrow{AB}$ y el eje x. Por último, el punto C queda determinado por la intersección de los círculos centrados en A y en B con radios d_{AC} y d_{BC} respectivamente. Si bien existen dos intersecciones, se toma la que está a la derecha del vector $\overrightarrow{AB}$ ya que la otra pondría al robot con las ruedas para arriba.

A estas ecuaciones independientes que reducen la dimensión del espacio de configuraciones se las conoce

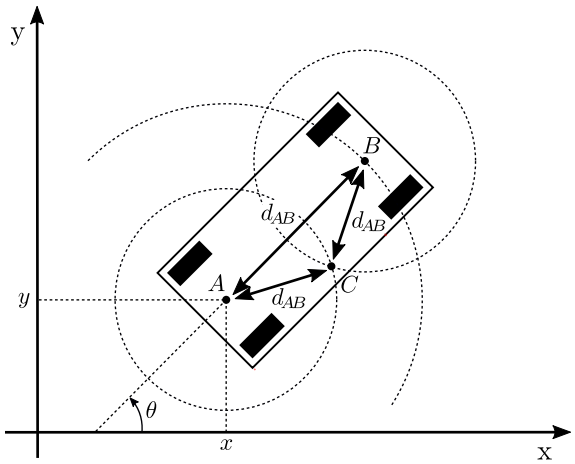
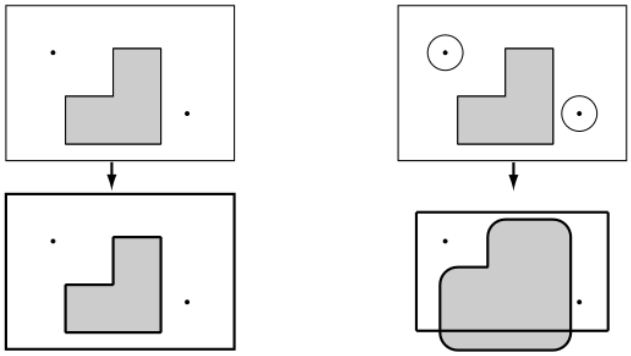


Figura 2.3: Grados de libertad de un robot. Una vez elegido el punto A, el punto B debe estar ubicado en la circunferencia de un círculo con radio d_{AB} centrado en A. Una vez elegida la ubicación de B, el punto C debe ubicarse en la intersección de los círculos centrados en A y B. Con la ubicación de un solo punto del robot (x, y) y su ángulo de orientación θ, se tiene información suficiente para determinar la ubicación de cualquier punto del robot.

como **restricciones holonómicas**. Formalmente, una restricción holonómica es aquella que puede ser representada por una función g sobre variables del espacio de configuraciones o el tiempo, de la forma $g(q, t) = 0$. Por el contrario, una restricción que aplica sobre la velocidad del robot, de la forma $g(q, \dot{q}, t) = 0$, y que además no es integrable, se la conoce como **restricción no holonómica**. Este tipo de restricciones reducen la dimensión de las posibles velocidades del sistema pero no la dimensión del espacio de configuraciones accesible. Un ejemplo, de este tipo de restricciones veremos en la siguiente sección cuando analizaremos el modelo cinemático del robot *car-like*.

Un robot omnidireccional no poseerá restricciones de velocidad, por lo cual también se lo conoce como robot holonómico. Un robot de este tipo puede moverse de forma instantánea en cualquiera de sus tres grados de libertad. Por el contrario, la movilidad local de un robot no omnidireccional siempre será reducida. Por ejemplo, un robot del tipo *car-like* no podrá moverse instantáneamente en dirección paralela al eje de sus ruedas traseras, por lo que resulta en un robot no holonómico. A pesar de esto, el robot podrá maniobrar de tal forma que al final su desplazamiento sea en dicha dirección. En otras palabras, este tipo de robots está sujeto a restricciones en cuanto a sus posibles movimientos instantáneos, sin que esto limite la posibilidad de conseguir cualquier posición y orientación en su espacio de trabajo. Los métodos de *path planning* necesitan considerar estas restricciones para poder generar caminos que el robot pueda físicamente seguir.

Como se mencionó anteriormente, llamaremos al entorno en el cual navega el robot como su **espacio de trabajo** (*workspace*) y lo notaremos con $\mathcal{W}$. En nuestro caso, el espacio de trabajo consistirá en el plano euclídeo, $\mathcal{W} = \mathbb{R}^2$. Dicho espacio incluye a los obstáculos. El i-ésimo obstáculo se lo notará con $\mathcal{WO}_i$. Luego, el espacio libre de trabajo será especificado por $\mathcal{W}_{\text{free}} = \mathcal{W} \setminus \bigcup_i \mathcal{WO}_i$. La completa definición de la posición y orientación del robot en su espacio de trabajo resulta la **configuración** del robot (*configuration*) y se lo nota con la letra q. El conjunto de todas las configuraciones del robot en su espacio de trabajo se lo conoce como **espacio de configuraciones** (*configuration space*) y se lo nota con $\mathcal{Q}$. Esto abarca todas las posiciones y orientaciones *diferentes* que el robot puede tener dentro de su *workspace* sin tener en cuenta la colisión con los obstáculos. Dos configuraciones del robot se consideran *diferentes* si necesitan ser distinguidas por el planificador. Por ejemplo, si el robot es circular y omnidireccional, no tendrá sentido especificar la orientación del mismo para definir una configuración ya que esta característica no tendrá importancia en la planificación, solo bastará con definir la posición del centro del robot, $q = (x, y) \in \mathbb{R}^2$, como puede verse en la Figura 2.4b. En este caso, el espacio de configuraciones se ve reducido por las dimensiones del robot. En el ejemplo mostrado en la Figura 2.4a, el robot consiste en un punto en el plano y su espacio de configuraciones resulta igual a



(a) Robot como un punto. **(b)** Robot circular omnidireccional.

Figura 2.4: Diferencia entre *workspace* y *configuration* space. Arriba, el *workspace* con un obstáculo y dos disposiciones del robot. Abajo, el *configuration space* correspondiente. En el ejemplo **(a)**, el espacio de trabajo y de configuraciones resultan idénticos, pero en el ejemplo **(b)**, el espacio de configuraciones se ve reducido por las dimensiones del robot. Imagen adaptada de [11].

su espacio de trabajo. En ambos casos, el espacio de configuraciones posee dos dimensiones pero como vimos anteriormente, para el caso de un robot *car-like*, se deberá especificar tanto la posición como la orientación, por lo que una configuración del mismo quedará determinada por la tripleta $q = (x, y, \theta) \in \mathbb{R}^2 \times S^1$, resultando en un espacio de tres dimensiones.

Notaremos con $R(q)$ al conjunto de los puntos en el workspace que el robot ocupa en su configuración q. Notamos con $\mathcal{QO}_i$ al conjunto de configuraciones que el robot no puede tomar debido a su colisión con el obstáculo $\mathcal{WO}_i$, es decir, $\mathcal{QO}_i = \{q \mid R(q) \bigcap \mathcal{WO}_i \neq \emptyset\}$. Ahora podemos definir el **espacio libre de configuraciones** (*free configuration space*) de la siguiente manera:

$$\mathcal{Q}_{\text{free}} = \mathcal{Q} \setminus \bigcup_i \mathcal{QO}_i. \tag{2.3}$$

Sean q_{start} la configuración inicial del robot y q_{end} su configuración final objetivo, un camino (*path*) consiste en una función continua en el intervalo unitario $[0, 1]$ y dominio en el espacio libre de configuraciones.

$$c : [0, 1] \to \mathcal{Q} \mid c(0) = q_{\text{start}}, c(1) = q_{\text{end}}. \tag{2.4}$$

Cuando la función está parametrizada por el tiempo t, se dice que es una trayectoria. Luego, la velocidad y aceleración pueden obtenerse con la primera y segunda derivada de $c(t)$. Lo cual implica que c debe ser al menos dos veces diferenciable, $c \in C^2$. Formalmente, el problema de la planificación de caminos consiste en encontrar una función continua $c : [0, 1] \to \mathcal{Q}$, de modo que ninguna configuración en el recorrido cause una colisión entre el robot y un obstáculo, es decir, $c([0, 1]) \subseteq \mathcal{Q}_{\text{free}}$.

2.1.4. Modelo Cinemático

Un robot *car-like* puede moverse hacia adelante y atrás, y puede realizar giros modificando la orientación de sus ruedas delanteras. Sin embargo, no puede moverse lateralmente y en la realización de los giros tiene restringido su radio de giro mínimo. A pesar de estas restricciones, el robot puede tomar cualquier posición y orientación en el plano realizando las maniobras adecuadas. Este tipo de restricciones pueden ser representadas con ecuaciones que afectan la velocidad del robot, por lo que se las conoce como restricciones no holonómicas.

A la relación entre las variables de configuración y los controles se la conoce como modelo cinemático del robot. Para obtener estas relaciones asumimos condiciones ideales para el robot en las que sus ruedas no patinan ni se deslizan lateralmente. Consideremos un robot *car-like* en movimiento como el de la Figura 2.5.

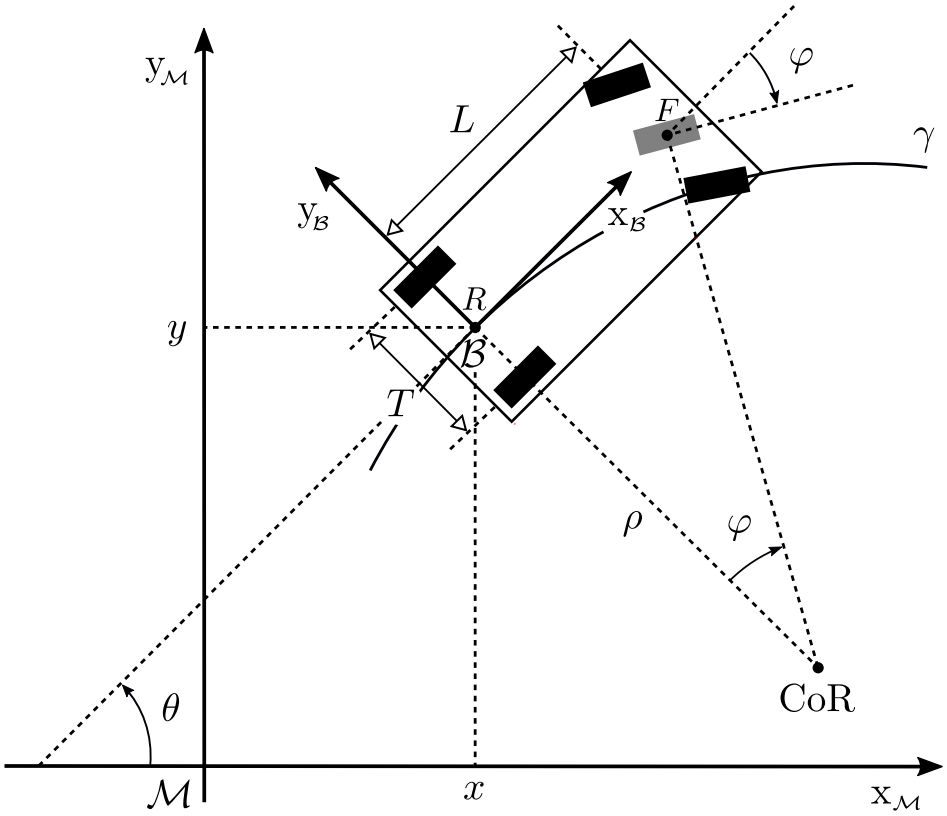


Figura 2.5: Modelo de un robot *car-like* navegando en el plano. Su configuración está formada por (x, y, θ). Los componentes x, y corresponden a las coordenadas del punto medio del eje trasero R en el marco de referencia $\mathcal{M}$. El componente θ corresponde al ángulo formado entre el marco de referencia $\mathcal{M}$ y el marco de referencia determinado por la orientación del robot $\mathcal{B}$. Los marcos de referencia se nombran convencionalmente $\mathcal{M}$ por *map* y $\mathcal{B}$ por *base*. El ángulo φ entre la orientación del robot y el vector velocidad del punto medio del eje delantero F corresponde al ángulo de direccionamiento del robot. Notar que en este caso el ángulo φ es negativo, ya que los ángulos son positivos en sentido antihorario. La intersección con la prolongación del eje trasero y la línea perpendicular a este vector velocidad, determina el centro de rotación del robot, notado con CoR. De este punto derivan el radio de giro ρ del robot y la curva γ que realiza el punto R.

El sistema de coordenadas de referencia $\mathcal{B}$ alineado al cuerpo del robot tiene su origen en el punto medio de su eje trasero R. Su eje longitudinal $x_{\mathcal{B}}$ apunta hacia el frente del robot y su eje transversal $y_{\mathcal{B}}$ apunta hacia la rueda izquierda. A medida que el robot se mueve describe la curva γ que será tangente vector $x_{\mathcal{B}}$ en el origen de coordenadas del marco de referencia $\mathcal{B}$. Se puede obtener la velocidad lineal instantánea del punto R como el vector tangente a la curva γ en el origen del marco $\mathcal{B}$, que tiene como dirección θ.

El ángulo de direccionamiento del robot φ consiste en el ángulo entre la orientación del robot $x_{\mathcal{B}}$ y el vector velocidad del punto F. Para asegurarse que las ruedas no tengan deslizamiento lateral, cada una de las ruedas delanteras deben estar perpendiculares a la línea que va desde la rueda hasta el centro de rotación, notado como CoR por sus siglas en inglés de *Center of Rotation*. Esto se consigue con la relación de Ackermann. Se puede simplificar el modelo del robot con dirección de Ackermann por un modelo de triciclo en el cual se establece una rueda imaginaria en el centro del eje delantero con el ángulo de direccionamiento φ, resultando en las mismas ecuaciones cinemáticas que el original. Luego, teniendo en cuenta las dimensiones T y L del robot se puede determinar los ángulos de cada una de las ruedas delanteras con la relación de Ackermann, como se verá en la Sección 3.2. Dicho ángulo de dirección φ está restringido por un ángulo máximo $\varphi_{\max}$ cuando la dirección llega a su tope.

$$|\varphi| \leq \varphi_{\max} < \frac{\pi}{2}. \tag{2.5}$$

Esto trae como consecuencia que el radio ρ de la curva γ queda limitado por

$$\rho \geq \rho_{\min} = \frac{L}{\tan \varphi_{\max}}, \tag{2.6}$$

lo cual se conoce como radio mínimo de giro. Sea v la velocidad lineal del punto R, la restricción anterior puede escribirse como

$$|\dot{\theta}| \leq \frac{|v|}{\rho_{\min}}, \tag{2.7}$$

o de otra manera, como

$$\dot{x}^2 + \dot{y}^2 - \rho_{\min}^2 \dot{\theta}^2 \geq 0. \tag{2.8}$$

A pesar de estas restricciones el robot es totalmente controlable, lo que significa que mediante una serie de maniobras el robot puede tomar cualquier configuración posible en el plano. Dicha demostración puede encontrarse en [10]. A este tipo de vehículo se lo conoce como STLC, por *Small Time Local Controllable*. Formalmente indica que el robot puede alcanzar cualquier configuración final dentro de un contorno local a la ubicación inicial del robot sin salirse de ese contorno, y esto se cumple sin importar cuanto se reduzca dicho contorno. Como conclusión, un robot *car-like* posee el mismo alcance que un robot holonómico pero con la desventaja de que en ocasiones lo hará de forma más lenta ya que debe intercalar movimientos en dirección hacia adelante y atrás.

La propiedad de STLC implica la siguiente consecuencia importante: si existe un camino libre desde q_{start} hasta el objetivo q_{goal} para un vehículo sin restricciones no holonómicas y además, existe un espacio libre alrededor de cada configuración q en el camino sin importar cuan pequeño sea, entonces existirá un camino para el vehículo *car-like* que tenga en cuenta sus restricciones de movimiento. Formalmente, si $\mathcal{Q}_{\text{free}}$ está completamente conectado de modo que cada $q \in \mathcal{Q}_{\text{free}}$ posee un contorno local de espacio libre, entonces existirá un camino para el robot *car-like* desde cualquier $q_{\text{start}} \in \mathcal{Q}_{\text{free}}$ hacia cualquier $q_{\text{goal}} \in \mathcal{Q}_{\text{free}}$. Llevado a un ejemplo de la vida cotidiana, esto implica que se puede estacionar un auto entre otros dos siempre que exista un espacio entre ellos $\epsilon > 0$ mayor a la longitud del auto a estacionar. Sin embargo, cabe destacar que el número de maniobras necesarias crece proporcionalmente a $1/\epsilon^2$, por lo que llevará mucho tiempo hacerlo si ϵ es demasiado chico [11].

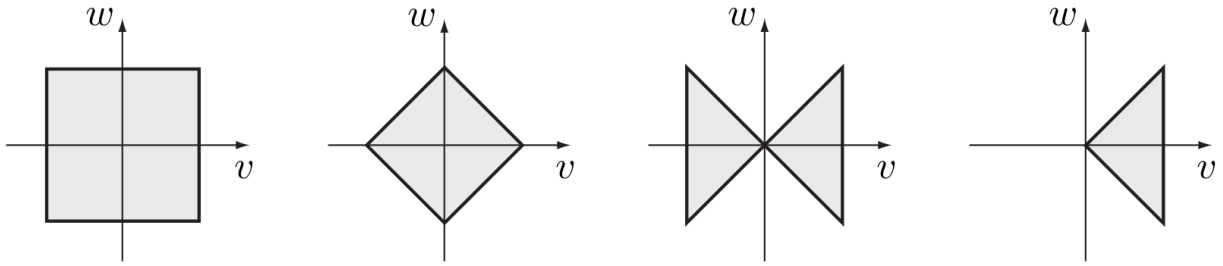
2.1.5. Modelo de control

Como vimos anteriormente, el espacio de configuraciones del robot *car-like* consiste en la ubicación del punto medio en su eje trasero (x, y), junto con la orientación del robot θ hacia donde éste apunta. Supongamos ahora que los controles del robot consisten en la velocidad lineal v y la tasa de cambio en la orientación del robot w. Luego, el modelo cinemático resulta en lo siguiente:

$$\begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \end{bmatrix} = \begin{bmatrix} \cos\theta & 0 \\ \sin\theta & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} v \\ w \end{bmatrix}. \tag{2.9}$$

Por lo general, el parámetro de control en este tipo de vehículos es el ángulo de dirección φ de la rueda imaginaria ubicada en el centro del eje delantero, en lugar de la velocidad angular del cuerpo del robot w. Sin embargo, podemos hacer dicha simplificación ya que con la siguiente fórmula se puede obtener este ángulo a partir de los controles (v, w) y el *wheelbase* L del robot:

$$\varphi = \arctan\left(\frac{Lw}{v}\right). \tag{2.10}$$



(a) Monociclo. **(b)** Manejo diferencial (*differential drive*). **(c)** *Car-like* con reversa. **(d)** *Car-like* sin reversa.

Figura 2.6: Diferentes combinaciones posibles para los controles (v, w) respecto del tipo de robot. Para un monociclo **(a)**, cualquier combinación de velocidad lineal y angular es posible, ya que ambos controles son independientes. Para un vehículo con manejo diferencial **(b)** vemos una dependencia en los controles. Por un lado, la máxima velocidad lineal se da cuando la velocidad angular es cero (ambas ruedas girando a máxima rpm en igual sentido). Por otro lado, para obtener la velocidad angular máxima la velocidad lineal será cero (ambas ruedas girando a máxima rpm pero en sentidos opuestos). Para el caso del robot *car-like* con reversa **(c)**, se aprecia que el mismo es incapaz de girar en el lugar, es decir, necesitará tener velocidad lineal para conseguir velocidad angular. El ángulo de la pendiente que se forma en esta figura de moño queda determinado por el radio mínimo de giro. Si además el robot no posee reversa **(d)**, solo una mitad del moño estará disponible en los controles. Imagen obtenida de [14].

Al modelo cinemático de la Ecuación (2.9) se lo conoce como *modelo no holonómico canónico* porque también modela al robot monociclo, triciclo y al de manejo diferencial, para los cuales también existen transformaciones que convierten a las variables de control (v, w) en los controles reales de cada robot. La única diferencia en el control de estos vehículos recae en los límites de los valores que pueden tomar los controles, como se puede ver en la Figura 2.6.

Para resumir, el modelo cinemático canónico nos indica que tenemos dos variables de control de velocidades para los tres grados de libertad del robot. Con un poco de manipulación de dicha ecuación podemos obtener la siguiente restricción:

$$\dot{x} \sin\theta - \dot{y} \cos\theta = 0. \tag{2.11}$$

Sea $A(q) = \begin{bmatrix} \sin\theta & -\cos\theta & 0 \end{bmatrix}$, la ecuación puede reescribirse de la forma $A(q)\,\dot{q} = 0$, lo cual se conoce como restricción Pfaffian. Dado que esta restricción es no integrable, resulta en una restricción no holonómica. La demostración puede encontrarse en [10]. La presencia de esta restricción es el motivo por el cual llamamos a dicho robot no holonómico.

2.2. Planificación de trayectorias para un robot *car-like*

La característica principal de los robots móviles es la posibilidad de navegar de forma autónoma por su espacio de trabajo mientras realiza la tarea encomendada. La navegación autónoma abarca problemas de localización, construcción de mapas y planificación de caminos. El problema de la localización consiste en dotar al robot de la habilidad de inferir su propia ubicación dentro de un entorno conocido. La tarea de construcción de mapas, llamada también mapeo, consiste en la generación de un modelo de representación detallado del entorno y debe ser abordada a partir de una correcta localización del robot en dicho entorno. Por último, conociendo su localización y contando con un modelo o mapa del entorno, el robot debe ser capaz de planificar una trayectoria que le permita moverse de manera segura mientras realiza sus tareas de forma autónoma.

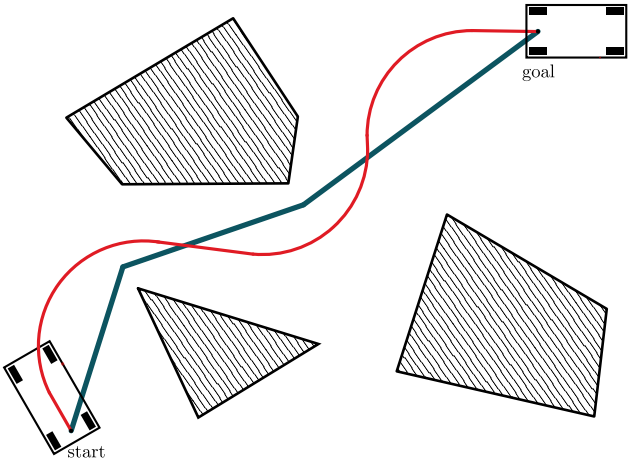
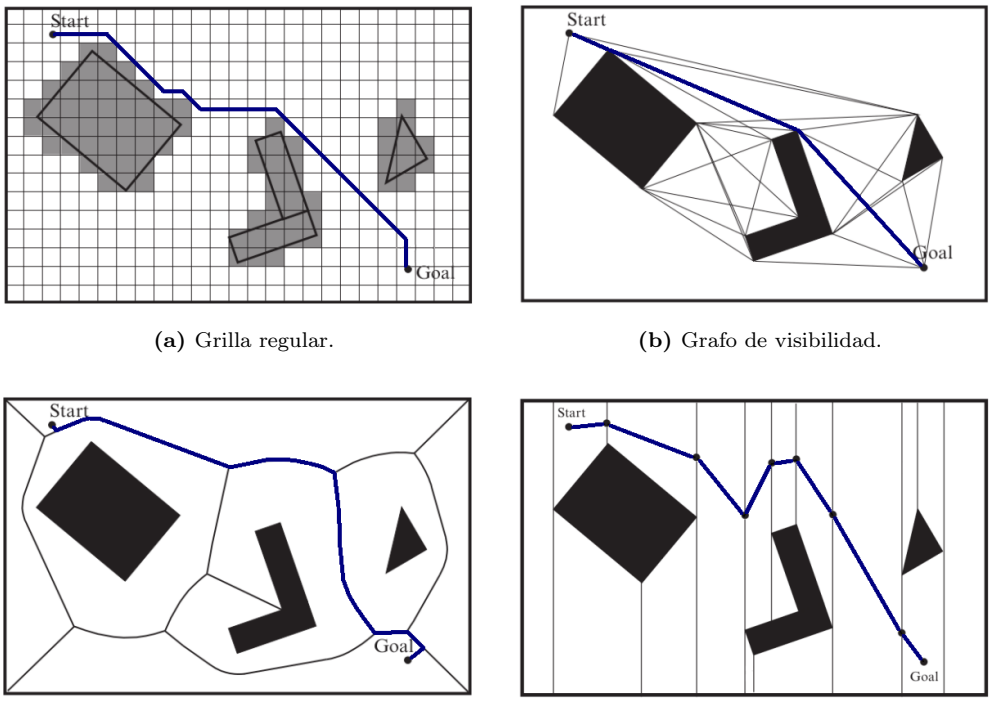


Figura 2.7: Plan global (azul) vs. plan local (rojo). El planificador de caminos global tiene como objetivo encontrar una forma de llegar desde el punto inicial al punto meta evadiendo los obstáculos conocidos pero sin considerar las restricciones físicas del robot. Luego, el planificador local será el encargado de ir refinando este plan para que pueda ser ejecutado por el robot teniendo en cuenta sus capacidades de maniobrabilidad.

El problema de la planificación de caminos consiste en determinar una ruta en el espacio libre de configuraciones desde la configuración inicial del robot hasta una configuración final de tal forma que el robot no colisione con ningún obstáculo del entorno y que el plan generado sea consistente con las restricciones cinemáticas del vehículo. Este problema puede extenderse buscando la optimización de alguna característica como puede ser la de longitud mínima, la de menor tiempo de ejecución, la que pase lo más lejos posible de los obstáculos o la más suave, por nombrar algunas, o bien se puede buscar una combinación ponderada de las anteriores. Cabe destacar que un camino óptimo en tiempo de ejecución generalmente no coincide con el camino de longitud mínima, puesto que es común que la velocidad máxima del vehículo disminuya con la curvatura del camino [29]. La descripción del camino planificado se puede enriquecer con información temporal que le indique al robot la manera en la que deberá ser llevado a cabo, a lo cual se le da el nombre de trayectoria.

Este problema de la planificación de caminos y trayectorias puede dividirse en planificación global y planificación local de evasión de obstáculos [30]. El planificador global tiene como principal objetivo la búsqueda de un camino libre de los obstáculos previamente conocidos que vaya desde un punto inicial a otro punto final generalmente alejado, considerando como espacio de trabajo la totalidad del entorno de navegación pero sin tener en cuenta las restricciones de maniobrabilidad que el robot pueda tener. Una vez obtenido este camino general entre ambos puntos distantes, el planificador local se encargará de refinar una porción pequeña del mismo en las cercanías del robot para que sea viable su ejecución, teniendo en cuenta tanto la cinemática del robot como también los nuevos obstáculos que puedan ir apareciendo, además de aumentarlo con información temporal para convertirlo en una trayectoria. El ajuste del camino global es realizado por el planificador local de forma incremental a medida que el robot va recorriendo el mismo. El planificador local utiliza no solo la información de la posición del robot sino también su orientación para el ajuste de las trayectorias y es su responsabilidad que el robot quede apuntando en la misma dirección que la configuración final. En la Figura 2.7 se muestra un ejemplo donde se pueden apreciar las diferencias entre ambos planificadores. En dicho ejemplo, la última curva del plan local se realiza solo con la finalidad de que el robot quede apuntando hacia la derecha y así coincida con la configuración objetivo.

En términos de mapeo, la planificación global requiere un modelo conocido de antemano de la totalidad del entorno en el que navegará el robot. Este modelo consistirá en una descripción simplificada del mundo real y podría estar desactualizado frente a cambios recientes en el entorno de navegación. El sistema de navegación local seguirá los pasos determinados por el plan global, manteniendo al robot cerca de dicho camino al mismo tiempo que evita su colisión con los obstáculos nuevos que puedan aparecer. Mientras que la planificación global requiere un mapa conocido de antemano del entorno, el planificador local requerirá



(a) Grilla regular.

(b) Grafo de visibilidad.

(c) Diagrama de Voronoi.

(d) Descomposición en celdas.

Figura 2.8: Diferentes formas para el modelado del entorno y sus caminos resultantes. El método de grilla regular **(a)** divide el espacio en múltiples celdas todas iguales generalmente cuadradas. Con el método de grafo de visibilidad **(b)** los caminos se construyen lo más cerca posible de los obstáculos, ideal para la búsqueda de una solución de longitud mínima. Por el contrario, el método de diagrama de Voronoi **(c)** busca generar caminos que pasen lo más alejado posible a cualquier obstáculo. Por último, la descomposición en celdas **(d)** se basa en la detección de puntos críticos del entorno.

un modelo que refleje el estado más actual del mismo, incluyendo los cambios que se produzcan a medida que el robot ejecuta la trayectoria, pero solo en las cercanías del robot. El modelo para el planificador global puede obtenerse mediante una fase previa de exploración, en la cual el robot recorre el entorno generando el mapa con la información obtenida de sus sensores, o bien puede haberse obtenido por otro medio o fuente de conocimiento, como ser una imagen satelital. El planificador local dependerá principalmente de la información en tiempo real obtenida a través de los distintos sensores que posea el robot. A medida que la trayectoria es ejecutada por el robot se podrá actualizar el modelo global con la información obtenida desde el modelo local.

2.2.1. Planificador Global

El planificador global se encargará de seleccionar un camino, *grosso modo*, para que el robot alcance su meta, sin considerar las restricciones no holonómicas del mismo. Mientras que el planificador local se encargará de los detalles de cómo seguir ese camino generando una trayectoria que tenga en consideración las restricciones dinámicas y de maniobrabilidad del robot, además de evitar colisionar con obstáculos imprevistos.

El modelado del entorno es un paso inevitable para ejecutar los diferentes tipos de planificadores globales. Dividir el espacio de navegación con una grilla regular (*regular grid*), como se ve en la Figura 2.8a, es uno de los métodos más comunes de modelado usado en robótica. La principal ventaja de las grillas regulares

consiste en su simplicidad de implementación y actualización. Esto se debe a que las grillas regulares poseen la misma cantidad de nodos y aristas sin importar el número de obstáculos que existan. Sin embargo, a pesar de ser el más utilizado posee dos desventajas a tener en cuenta. La primer desventaja consiste en que se obtiene una precisión limitada del modelado de los obstáculos, dependiente de la resolución de la grilla. La segunda desventaja radica en el uso elevado de memoria que se incrementa con el aumento de la resolución de la grilla.

Existen también métodos de planificación de caminos basados en otras formas de modelado del entorno de navegación que no involucran grillas regulares, de los cuales se destacan los grafos de visibilidad [31], los grafos tangentes [32], los diagramas de Voronoi [33], y los algoritmos de descomposición en celdas [34]. A estos métodos se los suele agrupar bajo el nombre de métodos de *roadmap*. Un *roadmap* consiste un mapa de caminos formado por líneas, curvas y sus puntos de intersección, que proporciona las posibles conexiones entre los diferentes sectores del espacio libre de navegación. La tarea de *path planning* se reduce entonces a conectar el punto inicial con el final por medio de una secuencia de estos caminos que conforman el *roadmap*.

El grafo de visibilidad de un espacio de configuraciones poligonal consiste en los caminos que unen todos los pares de vértices que pueden verse entre sí, como se muestra en la Figura 2.8b. Con este modelado se puede encontrar el camino de menor distancia entre el punto inicial y el final, pero tiene la desventaja de que dicho camino llevará al robot a acercarse mucho a los obstáculos, lo cual en ocasiones no es deseable. Por el contrario, el método de diagrama de Voronoi maximiza la distancia entre el robot y los obstáculos, como se puede apreciar en la Figura 2.8c. La principal desventaja de este método consiste en que al conducir al robot lejos de los obstáculos los sensores de corto alcance del robot podrían perder información de sus alrededores y perjudicar al sistema de localización. Por último, la descomposición en celdas se basa en el uso de funciones de Morse [35] y sus puntos críticos, donde se ubican los límites de las celdas. En su forma más simple, la descomposición es generada mediante una línea vertical barriendo el espacio de izquierda a derecha y estableciendo los límites de las celdas cuando la línea choca contra un obstáculo como puede verse en la Figura 2.8d. Esto genera una serie de celdas y un grafo de adyacencia entre las mismas. En comparación con la descomposición de grillas regulares, este método resulta eficiente en términos de almacenamiento para entornos grandes con pocos obstáculos dispersos, ya que se generará una cantidad pequeña de celdas.

Los diferentes métodos de modelado del entorno vistos permiten la representación del espacio de navegación como un grafo y la utilización de algoritmos de búsqueda sobre grafos para encontrar un camino óptimo entre dos nodos del mismo. Los nodos del grafo representan una porción del espacio libre, por ejemplo una celdas si se realiza la descomposición en grillas regulares, y las aristas del grafo representan las conexiones o caminos que existen entre dichas ubicaciones. Generalmente, estas aristas tendrán un peso que determina un costo de viajar desde una ubicación a otra, comúnmente determinado por la distancia entre las mismas. El principal algoritmo de búsqueda en grafos en este sentido es el propuesto por Edsger Dijkstra en 1959 [36]. Este algoritmo proporciona una manera de encontrar un camino óptimo de longitud mínima entre dos nodos de un grafo ponderado.

El algoritmo de Dijkstra se lo conoce como búsqueda ciega, debido a que simplemente enumera de forma exhaustiva los posibles caminos en el espacio de búsqueda hasta que no exista camino por analizar o bien se consiga un camino hacia la meta. Tomando en cuenta información adicional del dominio, es posible mejorar la búsqueda estableciendo en cada paso un costo estimado del resto del camino hacia la meta. La forma habitual de representar esta información es mediante la definición de una función $f(n)$ asociada con los nodos por visitar. Por convención, valores menores de $f(n)$ indican que el nodo n es más probable que se encuentre en el camino óptimo. La función $f(n)$ es generalmente expresada como la suma de dos componentes:

$$f(n) = g(n) + h(n), \tag{2.12}$$

donde $g(n)$ es el costo del mejor camino desde el nodo inicial hasta el nodo n y $h(n)$ es una estimación heurística del costo del mejor camino desde el nodo n hasta el nodo meta. La función $g(n)$ se puede calcular durante la ejecución del mismo algoritmo de búsqueda como $g(n) = g(n') + c(n', n)$, donde n' es el nodo padre de n, es decir, el que se visitó antes que n, y $c(n', n)$ consiste en el costo asociado a viajar desde el nodo n' al nodo n.

El algoritmo de Dijkstra expande los nodos comenzando en el nodo inicial de forma similar al algoritmo

de búsqueda en anchura (*breadth-first search*) excepto que primero expande los nodos con menor valor de $f(n)$. El proceso de expansión se realiza hasta que el nodo meta es alcanzado o no existen más nodos por visitar. Luego, se reconstruye el camino desde el nodo final al inicial con la información recolectada.

El algoritmo A* [37] es similar al de Dijkstra con la inclusión de la heurística $h(n)$, que proporciona información adicional sobre el entorno representado, con el fin de mejorar la eficiencia del algoritmo. Para garantizar que la solución sea óptima, la heurística debe ser una cota inferior, es decir subestimar el costo del resto del camino. El algoritmo A* se utiliza ampliamente con el modelado de grilla regular donde se toma como heurística la distancia entre cada celda y el objetivo sin tener en cuenta los obstáculos. Generalmente, con este algoritmo se reduce dramáticamente el número de nodos expandidos comparado con el algoritmo original de Dijkstra.

La principal desventaja de estos algoritmos de búsqueda radica en el tiempo de ejecución si el grafo está compuesto por una cantidad relativamente grande de nodos y no se puede obtener una heurística apropiada para reducir la dimensión del problema. Dentro de los métodos de planificación apropiados para estos casos podemos destacar el método RRT (*Rapidly-exploring Random Tree*) [38] y el algoritmo de campos potenciales (*potential fields*) [39].

El método RRT construye un grafo de configuraciones de manera *online* durante el proceso de exploración, por lo que solo se necesita tener un mapa de los obstáculos pero no un modelo previo del entorno. El algoritmo comienza con un árbol de exploración que solo contiene como nodo raíz la configuración inicial q_{start} del robot. En cada paso del proceso, se selecciona una configuración q_{rand} del espacio libre de forma aleatoria y se busca su nodo más cercano q_{near} en el árbol actual. Luego, se genera la configuración q_{new} mediante una arista de tamaño preestablecido desde dicho nodo en la dirección de q_{rand}. Si en este camino no existen obstáculos se añade al árbol el nuevo nodo con su arista, caso contrario se descarta. Finalmente, se repite el proceso hasta que se alcanza la configuración destino q_{end}. Si bien este algoritmo no garantiza completitud, se puede demostrar que es probabilísticamente completo, lo que significa que eventualmente se encontrará un camino si éste existe, añadiendo más y más nodos al árbol.

Por otro lado, el método de campos potenciales está basado en una analogía en la que el robot es tratado como una partícula actuando bajo la influencia de un campo potencial que representa el mapa del entorno. Los obstáculos en el mapa actúan como fuerza de repulsión para dicha partícula y la meta actúa como fuerza de atracción. Una de las desventajas de este método es la existencia de mínimos locales que dependerá de la forma y tamaño de los obstáculos. Otro problema puede aparecer ante la presencia de objetos cóncavos que pueden conducir a una oscilación en la búsqueda del camino hacia la meta.

Debido a que el *navigation stack* de ROS está diseñado para ser utilizado con una representación del entorno en formato de grilla regular, en esta tesina se utilizará dicho modelado junto con los algoritmos de búsqueda de Dijkstra y A*.

2.2.2. Planificador Local

Con el objetivo de transformar el plan global en una trayectoria acorde para el vehículo en cuestión, el planificador local crea nuevos puntos de referencia (*waypoints*) tomando en consideración los obstáculos desconocidos en el mapa global, las restricciones cinemáticas y dimensiones del robot. Para calcular este nuevo camino se toma una porción del mapa en las cercanías del robot y se actualiza a medida que el robot se mueve. No es posible utilizar el mapa completo ya que los sensores son incapaces de actualizar todas las regiones del mapa por su alcance limitado y además, un gran número de celdas conllevaría un alto costo computacional. Dicho de otra forma, con un mapa local actualizado por los sensores del robot y el plan global, el planificador local va generando pequeñas trayectorias adecuadas para el robot a su vez que intenta mantenerse lo más cerca posible del plan global. De los métodos desarrollados para la planificación local de trayectorias podemos destacar los algoritmos de DWA (*Dynamic Window Approach*) [40], EBand (*Elastic Band*) [41] y TEB (*Timed Elastic Band*) [42, 43, 44].

El planificador global solo genera *waypoints* con la posición del robot en cada paso del camino ya que por lo general considera al robot como un punto en el plano sin tener en cuenta las dimensiones o restricciones de movimiento del mismo. Por el contrario, el planificador local agrega la variable de orientación a los *waypoints*

completando la especificación de la configuración que debe tomar el robot en cada paso, generando además muchos más pasos intermedios afinando la resolución de la trayectoria. Dicho de otra manera, el planificador local crea una serie de poses intermedias de la forma $p_i = (x_i, y_i, \theta_i)$ que el robot deberá seguir en su recorrido hacia la meta. Adicionalmente, estos planificadores locales pueden ser capaces de generar también los comandos de velocidad lineal v y velocidad angular w vistos anteriormente, los cuales son enviados directamente a los *drivers* del robot para alcanzar estas poses. En esta construcción se tienen en cuenta, además de las dimensiones y restricciones cinemáticas del robot, sus límites de velocidad y aceleración como también sus características dinámicas.

Los métodos DWA y EBand no soportan la cinemática de un robot del tipo *car-like* con radio de giro mínimo limitado, por lo cual no se abordarán en detalle en este trabajo. De todos modos, a continuación se hace una breve descripción de los mismos. El método DWA realiza una búsqueda de los comandos para controlar el robot directamente sobre el espacio bidimensional de velocidades (v, w). El espacio de búsqueda se reduce para considerar solo las velocidades que resulten seguras con respecto a evitar la colisión con obstáculos. Un par (v, w) solo será admisible si el robot es capaz de frenar antes de acercarse demasiado a un obstáculo. Adicionalmente, solo se considera una ventana limitada de velocidades restringida por la dinámica del robot. Es decir, solo se tomarán en consideración las velocidades que puedan alcanzarse en un corto intervalo de tiempo dados los límites de aceleración del robot. Esta última característica es lo que le da el nombre al método de *dynamic window*. Con este espacio de búsqueda definido se intenta optimizar una función objetivo que favorece a la conducción del robot en dirección hacia el objetivo, el distanciamiento con los obstáculos y el aumento de la velocidad.

El método de EBand toma inicialmente una porción del camino generado por el planificador global y lo va ajustando con la aplicación de fuerzas artificiales para transformarlo en uno más suave que se mantenga libre de obstáculos. El nombre de *banda elástica* se origina debido a estas deformaciones en tiempo real que se realizan sobre la trayectoria para permitir que el robot se adecue a las incertidumbres previas del entorno y a la aparición de nuevos obstáculos. En pocas palabras, para mejorar la forma de una trayectoria, se le aplican dos fuerzas como si se tratase de una banda elástica: una fuerza interna de contracción y una fuerza externa de repulsión. La fuerza de contracción simula la tensión que recae sobre una banda elástica cuando es estirada, lo cual suaviza la trayectoria. Al mismo tiempo, para contrarrestar con esta fuerza de contracción y mantener al robot en el espacio libre, una fuerza de repulsión se aplica a la trayectoria en dirección opuesta a los obstáculos. Ambas fuerzas deforman la banda hasta alcanzar el estado deseado de equilibrio.

Es importante señalar que existen otros métodos para la planificación de caminos adecuados para vehículos del tipo car-like, como por ejemplo las curvas de Dubins [45]. Este método asegura la obtención del camino más corto entre dos puntos para un vehículo con radio mínimo de giro limitado pero no tiene en cuenta la evasión de los obstáculos. A pesar de ello, se puede combinar las curvas de Dubins con el algoritmo RRT para obtener un método que evite la colisión con obstáculos, como se puede ver en el trabajo [46]. El método de curvas de Dubins solo tiene en cuenta la conducción hacia adelante. Una extensión de éste, que considera el manejo tanto hacia adelante como en reversa, fue desarrollado posteriormente por Reeds y Shepp [47]. Sin embargo, el método TEB que se verá en la Sección 2.2.2.1, es el único algoritmo con implementación en las librerías oficiales de ROS que a hoy en día tiene soporte para robots del tipo *car-like*, y es el utilizado en la simulación presentada en este trabajo. Cabe destacar que el enfoque del algoritmo TEB, en lugar de buscar generar una trayectoria de longitud mínima, intenta minimizar también el tiempo que le tomará al robot recorrer el camino generado.

2.2.2.1. Timed Elastic Band

El planificador local conocido como *Timed Elastic Band* (TEB) [44] consiste en un optimizador *online* de trayectorias que tiene en consideración las restricciones cinemáticas del robot al mismo tiempo que incorpora información temporal con el propósito de generar un plan que alcance el objetivo en tiempo mínimo. Dicho planificador tiene soporte en ROS, implementado como un paquete *open source* bajo el nombre de **`teb_local_planner`**[1] y es compatible con el *navigation stack* de ROS. A continuación se describe brevemente su enfoque.

[1] TEB local planner, http://wiki.ros.org/teb_local_planner.

Se representa a una trayectoria discreta como una secuencia de *poses* acompañados de su *timestamp*.

$$b = [s_1, \Delta T_1, s_2, \Delta T_2, \ldots, \Delta T_{N-1}, s_N], \tag{2.13}$$

donde $s_i = (x_i, y_i, \theta_i)$ con $i = 1, \ldots, N$ representa una pose del robot y ΔT_i con $i = 1, \ldots, N-1$ representa el intervalo de tiempo necesario para la transición entre dos poses consecutivas s_i y s_{i+1}.

El objetivo del método TEB consiste en encontrar una trayectoria b^* que minimice una función de costo configurable. Las funciones de costo pueden capturar criterios de tiempo total del recorrido, consumo de energía, distancia del recorrido, y combinaciones ponderadas de éstas. Las soluciones admisibles consisten en las trayectorias que no se intercepten con ningún obstáculo y cumplan con las restricciones cinemáticas del robot en cuestión. Para alcanzar este objetivo, el planificador TEB convierte el problema de optimización con restricciones en una secuencia de subproblemas de optimización sin restricciones más fácil de resolver utilizando métodos de penalización [48]. De esta manera, se puede obtener con un proceso iterativo una solución aproximada al problema original. Para la solución de los problemas de optimización sin restricciones existen múltiples técnicas de métodos numéricos muy maduras.

Por ejemplo, un problema general de minimización

$$\begin{cases} \min & f(x) \\ \text{subject to} & g_i(x) \leq 0, \ \forall i \in I. \end{cases} \tag{2.14}$$

Puede ser resuelto con una serie de iteraciones de problemas de minimización sin restricciones del tipo

$$\min \Phi_k(x) = f(x) + \sigma_k \sum_{i \in I} \max(0, g_i(x))^2, \tag{2.15}$$

donde σ_k corresponden a los coeficientes de penalización. En cada iteración k, se incrementan los coeficientes de penalización σ_k (por ejemplo, en un factor de 10), se resuelve el problema y el resultado se utiliza como solución aproximada inicial para la siguiente iteración.

El problema de optimización definido por el método TEB consiste en buscar el b^* de la siguiente forma:

$$b^* = \underset{b \setminus \{s_1, s_N\}}{\arg\min} \sum_i \sigma_i f_i^2(b), \ i \in \{\mathcal{J}, \mathcal{P}\}, \tag{2.16}$$

donde f_i corresponden a las funciones de los objetivos para $i \in \mathcal{J}$ y las funciones de penalidad para $i \in \mathcal{P}$. El peso de cada término es ajustado por los coeficientes σ_i. La notación $b \setminus \{s_1, s_N\}$ representa todas las poses de la trayectoria menos la inicial y final, que al ser fijas no podrán estar sujetas a optimización. El problema es representado como un grafo y resuelto mediante el *framework* de optimización de grafos *g2o*[2] [49].

Las restricciones de desigualdad originales de la forma $g_i(b) \geq 0$ son aproximadas por una función de penalización $p_i(b) = \max(0, -g_i(b) + \epsilon)$. El parámetro ϵ añade un margen a la desigualdad que puede ser configurado por un parámetro llamado **`penalty_epsilon`**. Luego, tenemos que $f_i(b) = p_i(b), \ \forall i \in \mathcal{P}$. El resto de las funciones $f_i(b), \ \forall i \in \mathcal{J}$ corresponden a las funciones que representan los objetivos del planificador. Ejemplos de estas funciones f_i de objetivos y penalidad tenemos a la evasión de obstáculos, las restricciones de velocidad y aceleración, el cumplimiento con la cinemática no holonómica y el alcance de la configuración final en tiempo mínimo, entre otras. Para cada una de estas funciones sus coeficientes σ_i se convierten en parámetros de la configuración del planificador TEB.

En la Sección 4.2.2.1 se describen los parámetros más relevantes y los valores de configuración utilizados para la simulación del robot desmalezador.

2.3. Entorno de simulación

Uno de los principales propósitos de la navegación en entornos agrícolas consiste en que el robot recorra el campo de cultivos siguiendo las hileras de sembrado, para que de esta manera evite pisar los cultivos con sus

[2] G2o graph optimization, *https://openslam-org.github.io/g2o.html*.

ruedas. Solo en las cabeceras del campo el robot podrá realizar los giros para posicionarse en la siguiente hilera que se proponga a recorrer. El hecho de que el robot tenga que navegar muy cerca de las plantas, consideradas como obstáculos en el espacio de trabajo, y que además tenga que hacerlo entre líneas de plantaciones en un espacio muy estrecho de forma precisa propone un desafío interesante y complejo, tanto para la planificación y recálculo de la trayectoria como también para la ejecución de la misma. Asimismo, los vehículos utilizados en agricultura suelen ser del tipo no holonómicos con restricciones en sus movimientos, como ser un radio de giro mínimo limitado, lo cual aumenta la complejidad del proceso. Por todo lo anterior, la posibilidad de contar con un ambiente de simulación que refleje las características intrínsecas del robot y del entorno agrícola ayudará al desarrollo adecuado de los sistemas de navegación y planificación de trayectorias, y permitirá la rápida validación de los mismos.

A medida que la complejidad del sistema bajo investigación aumenta, el rol de la simulación se vuelve cada vez más importante. La prueba de los distintos métodos y algoritmos de navegación en entornos reales conlleva elevados costos tanto en logística como en ejecución. En la actualidad, la posibilidad de acceder a computadoras con alto poder de cómputo y GPUs cada vez más veloces, permitió un gran avance en el desarrollo de simuladores gráficos para la validación de diferentes soluciones, sin la necesidad siquiera de poseer el robot físicamente, reduciendo los tiempos y costos de la realización de dichas pruebas [50]. Adicionalmente, las simulaciones ofrecen una forma sencilla de ejecutar múltiples pruebas con diferentes entornos y configuraciones, y a su vez permiten el escalado a sistemas multirobot evitando los costos de fabricación.

La principal desventaja de las simulaciones radica en que el mundo real puede presentar situaciones complejas como perturbaciones inesperadas en los actuadores o ruido impredecible en los sensores resultado de condiciones naturales del entorno. A pesar de que las plataformas de simulación profesionales ofrecen herramientas avanzadas para crear escenas realistas, es prácticamente imposible cubrir completamente cada detalle del mundo real dentro de una simulación. Sin embargo, esto no implica un problema significativo, ya que la simulación se crea con el objetivo de evaluar ideas y pruebas de concepto, sin pretender reemplazar completamente las pruebas de campo con el dispositivo real.

2.3.1. Comparación de simuladores

Un simulador de propósito general en robótica consiste en una plataforma que permita modelar virtualmente los agentes robóticos junto con el resto de los objetos del entorno y que disponga de un motor físico con el cual emular (imitar) las ideas y procesos propuestos para el robot y su interacción con en el mundo real. Como primer paso en el desarrollo de una simulación para el robot desmalezador, debemos seleccionar una plataforma de simulación adecuada para nuestros objetivos. En la actualidad, no son muchos los simuladores robóticos maduros de propósito general disponibles en código abierto y utilizados masivamente. Dentro de las plataformas de este estilo podemos destacar ARGoS[3], Player/Stage[4], Webots[5], V-REP[6] y Gazebo[7], siendo los dos últimos los de mayor trascendencia.

El simulador ARGoS [51, 52] se utiliza casi exclusivamente para la simulación de enjambres de robots. Su principal objetivo consiste en afrontar y mejorar el degradado de eficiencia que sufren el resto de las plataformas a la hora de simular aplicaciones multirobots, aprovechando el paralelismo y la programación multihilo. El trabajo de esta tesina se enfoca en la simulación de un único agente robótico, para lo cual el resto de las plataformas son más adecuadas. Debido a esto, se descartó la alternativa de ARGoS. Por otro lado, Player/Stage [53] es simplemente el precursor de Gazebo, iniciando como un simulador 2D que todavía en la actualidad se sigue usando especialmente en ambientes educativos debido a su simplicidad. Puesto que tiene una gama muy limitada de tipos de sensores y robots que se pueden modelar y debido a la falta de simulación 3D tampoco se tuvo en cuenta para este trabajo. El resto de las tres plataformas mencionadas (Webots [54], V-REP [55] y Gazebo [56]) resultan muy similares en cuanto a sus características y funcionalidades que

[3] ARGoS, https://www.argos-sim.info.
[4] Player, http://playerstage.sourceforge.net.
[5] Webots, https://cyberbotics.com.
[6] CoppeliaSim (ex V-REP), https://www.coppeliarobotics.com.
[7] Gazebo Simulator, http://gazebosim.org.

ofrecen. Todos se integran de forma correcta y fácil con ROS, poseen diferentes formatos de modelado CAD y presentan una comunidad actualmente activa.

La simulación del robot desmalezador, objetivo de este trabajo, pudo haberse realizado sin mayores inconvenientes con cualquiera de estas plataformas. Sin embargo, se optó por el uso de Gazebo por los siguientes motivos. Primero, se encontró una mayor comunidad de usuarios activos y mayor soporte. Existe más documentación tanto en la web como en libros y publicaciones. Adicionalmente, debido a sus orígenes junto con ROS, notamos que se integra de una forma mucho más natural con este *framework*. Por último, el uso de Gazebo también tiene la ventaja de poder definir la escena y el modelo del robot en formato XML, característica que aprovechamos para generar los campos de cultivos de forma programática. Esto fue fundamental, ya que fácilmente se pudieron generar escenarios de campos de distintas formas y dimensiones de manera muy rápida. V-REP, por su parte, no ofrece la posibilidad de la descripción de escenas en XML, sino que utiliza formatos puramente binarios con el fin de realizar una carga más rápida. La única aparente desventaja que se encontró en Gazebo con respecto al resto consiste en que no permite la edición de las mallas de los modelos directamente desde la misma plataforma. Sin embargo, esto no resulta un inconveniente ya que puede realizarse fácilmente con otro software especializado en las cuestiones gráficas como Blender[8], el cual también es de código abierto.

2.3.2. ROS y Gazebo

Como se mencionó anteriormente, para este trabajo se utiliza la plataforma de simulación Gazebo junto con el *framework* de robótica ROS. Ambos poseen una excelente integración ya que fueron originados por la misma compañía, Willow Garage. La plataforma de Gazebo consta de un simulador multirobot tanto para ambientes *indoor* como *outdoor*, que emplea varias librerías de terceros, como el motor físico ODE y el motor gráfico Ogre [56]. Gazebo es capaz de simular una población de robots, una gran variedad de sensores, la colisión de objetos y la dinámica en general. Permite un renderizado 3D del ensayo junto con la simulación del comportamiento físico del robot interactuando con su entorno. Para la descripción de los ambientes, robots, sensores, fuentes de luz, y demás objetos de la simulación se emplean archivos en formato XML.

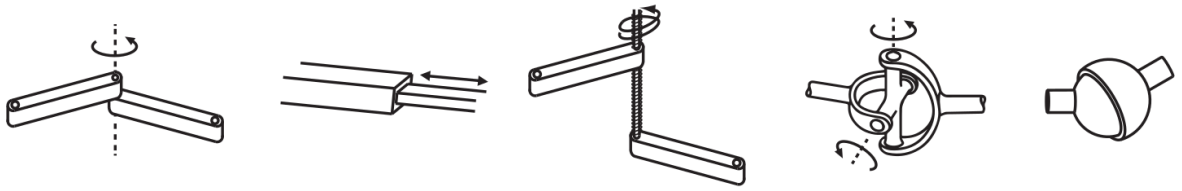
Por su parte, para el desarrollo de los sistemas de control y navegación del robot se utilizará el *framework* ROS (*Robot Operating System*), que es considerado un estándar para el desarrollo de software en robótica de código abierto. ROS no es un sistema operativo en el sentido tradicional, sino que se trata principalmente de un *framework* que proporciona una capa de abstracción para simplificar la comunicación entre diferentes sistemas heterogéneos. Está diseñado para crear sistemas modulares complejos, compuestos por múltiples nodos, para lo cual ofrece una gran variedad de formas diferentes de intercambio de mensajes tipados entre los nodos, como ser el *broadcast* en un canal (*topic*) o el modelo síncrono de *request-response*. Por otra parte, también proporciona servicios de abstracción de hardware, controladores de dispositivos de bajo nivel y un manejador de paquetes, entre otras herramientas [57].

La elección de ROS nos facilita el uso de una gran cantidad de paquetes aportados por la comunidad (organizados en conjuntos llamados *stacks*) que implementan soluciones a problemas muy estudiados tales como localización y mapeo simultáneo, planificación, percepción, visualización, control, entre otras. Esto hace posible el desarrollo de un comportamiento robótico complejo y robusto de forma relativamente sencilla. Por otro lado, este trabajo se realiza teniendo como objetivo primordial que la mayor cantidad de código desarrollado para la simulación pueda llevarse al robot real con la menor cantidad de modificaciones posibles. La elección conjunta de ROS y Gazebo facilitan lograr esto [58].

2.3.3. Modelo del robot

Un robot está compuesto mecánicamente por una serie de cuerpos, generalmente rígidos, llamados *links* conectados unos a otros mediante diferentes tipos de articulaciones o *joints*. Un *joint* puede decirse que dota de movilidad a un *link* acoplándolo a otro. Los actuadores, como por ejemplo los motores eléctricos, distribuyen fuerzas y torques que causan el movimiento de estos *links* sobre los *joints* dependiendo de los

[8]Blender, https://www.blender.org.



(a) Rotacional (*revolute*). **(b)** Prismático (*prismatic*). **(c)** Helicoidal (*helical*, *screw*). **(d)** Universal. **(e)** Esférico (*spherical*, *ball-socket*).

Figura 2.9: Diferentes tipos de articulaciones (*joints*) para el modelado de un robot. El formato URDF solo soporta los tipos rotacional y prismático, mientras que el formato SDF tiene soporte para todos.

grados de libertad que éste proporcione. Los diferentes tipos de articulaciones se clasifican de acuerdo al movimiento que permiten entre los *links*. La Figura 2.9 ilustra los tipos de *joints* más comunes.

Para describir la estructura del robot se utilizan los formatos URDF (*Unified Robot Description Format*) y SDF (*Simulation Description Format*). Ambos son archivos XML con los cuales se pueden definir los *links* y *joints* que conforman al robot, al mismo tiempo que sus propiedades físicas como masa, momento de inercia y coeficiente de fricción, como así también las propiedades visuales como formas, colores, texturas y mallas. URDF[9] fue creado para ser usado en combinación con ROS y está compuesto por varios paquetes de utilidades. Solo soporta los *joints* de tipo rotacional y prismático, además de que no admite ciclos en la definición de los *links*. En Gazebo se creó un nuevo formato llamado SDF[10] para salvar estas limitaciones, y que además agrega otras funcionalidades como la posibilidad de describir propiedades físicas y visuales del entorno en lugar de solo describir el robot. De todos modos, es posible utilizar una descripción del robot en URDF dentro del simulador Gazebo ya que es posible realizar una conversión a SDF.

Para modelar un robot de tipo car-like que utilice un mecanismo de dirección de Ackermann se necesita poder armar una cadena con cuatro *links* formando un ciclo. Más aún, si se quiere representar de forma exacta la construcción del mecanismo utilizado en el robot desmalezador sería necesario también contar con la posibilidad de utilizar *joints* del tipo esférico. Como se mencionó anteriormente, debido a sus limitaciones esta descripción no puede hacerse directamente en URDF. Por otro lado, si bien el formato SDF posee estas características, no está soportado por el *framework* de ROS. Como solución a este inconveniente, se decidió utilizar URDF para que sea compatible con ROS y simular el mecanismo de dirección de Ackermann por medio de un *plugin* en Gazebo como se explicará más adelante en la sección 3.3.

Cada *link* que forma parte de la descripción del robot establece un marco de coordenadas en su origen. Los diferentes sensores y actuadores del robot estarán definidos en los marcos de coordenadas donde se instalen. Para que la tarea de llevar registro de cada marco de coordenadas no se convierta en algo tedioso, la comunidad de ROS desarrolló el paquete llamado `tf`[11] (por *Transform Library*). Esta librería, descrita en [59] y utilizada ampliamente en esta tesina, permite realizar transformaciones entre los diferentes marcos de coordenadas de forma rápida y sencilla.

La librería `tf` se diseñó para ofrecer una forma estándar de llevar registro de todos los marcos de coordenadas existentes en el sistema y realizar transformación de datos entre estos, de modo que cada subcomponente pueda tener la seguridad y confianza de recibir la información en el marco de coordenadas esperado, sin necesidad de conocer todos los marcos de coordenadas del sistema. Esto permite que los subcomponentes puedan enfocarse de forma más precisa en la tarea que les concierne utilizando solo los marcos de coordenadas que necesiten. Esto resulta de gran relevancia en los sistemas que reciben y fusionan información de múltiples sensores, cada uno ubicado en un lugar distinto del robot y por lo tanto, proporcionando datos en diferentes sistema de coordenadas.

[9] URDF, http://wiki.ros.org/urdf.
[10] SDF, http://sdformat.org/spec.
[11] Transform Library, http://wiki.ros.org/tf.

2.3.4. Framework de control

Existe en el entorno de ROS un *framework* llamado **ros_control** que ofrece diferentes controladores comunes combinables y, en caso de no encontrarse el apropiado, también provee una serie de herramientas para poder desarrollar uno particular de forma más rápida [60]. La columna vertebral de este *framework* es la capa de abstracción de hardware, que sirve de intermediario para diferentes robots simulados y reales. Esta abstracción es provista por la clase **hardware_interface::RobotHW**, y los robots específicos deben heredar de esta clase. Esto permite intercambiar de forma trasparente para las capas superiores entre el robot real y el simulado. La idea principal de esta librería, es que no se exponga al controlador el acceso directo al hardware y exista un desacople entre controlador y robot. Adicionalmente, provee una serie de interfaces para controlar la posición, velocidad, fuerza o torque de los distintos *joints*, o bien obtener su estado actual. Permite definiciones de transmisiones directamente en los archivos URDF, como así también límites en los *joints* y otras herramientas como controladores PID. El código de **ros_control** es open source[12].

La comunidad de Gazebo, por su parte, desarrolló un paquete llamado **gazebo_ros_control** que consiste en un *plugin* de Gazebo para utilizar con **ros_control**. Este *plugin* por defecto publica interfaces leyendo directamente la descripción del robot en URDF, pero se pueden construir otros *plugins* a partir de éste, para lo cual ofrece varias herramientas que facilitan la tarea. También, puede leer directamente de los archivos URDF transmisiones y límites para los *joints*. Los límites resultan una especie de última línea de defensa para no enviar valores inapropiados al hardware simulado. Por otro lado, las transmisiones pueden ser, por ejemplo, reductores y diferenciales controlados por software. En el Capítulo 3 se verá el uso de esta librería en la simulación del robot desmalezador.

2.4. Estado del arte

A pesar de los grandes avances en los últimos años en robótica móvil, su explotación en entornos agrícolas continua siendo un área de investigación actualmente activa, y la tarea de planificación de trayectorias constituye uno de sus principales problemas de estudio [61]. Los robots agrícolas deben operar al aire libre, en terrenos irregulares poco estructurados y con condiciones atmosféricas como luz, viento y polvo constantemente cambiantes. La complejidad aumenta si consideramos que deben realizar su trabajo rodeados de objetos naturales, tales como árboles o cultivos, que cambian de forma, color y tamaño a lo largo de todo el año [62]. Las aplicaciones encontradas en la literatura son en su gran mayoría experimentales y no se encuentran todavía disponibles comercialmente a gran escala. Los robots agrícolas deben ser capaces de recorrer los campos entre las filas de árboles o cultivos para así evitar dañarlos o dañarse a sí mismos. Para lograr esto, suelen utilizar una combinación de varios sensores diferentes como ser GPS, IMU, cámara estéreo y láser LIDAR. En esta sección se describen algunos de los trabajos recientes más destacados.

Muchos de los robots agrícolas estudiados consisten en vehículos con cuatro ruedas direccionables, lo cual le otorga un amplio grado de maniobrabilidad y la posibilidad de girar sobre sí mismos. Un ejemplo de estos es [63], una plataforma de bajo costo diseñada para la inspección visual y recolección de datos de los suelos en campos de maíz. Este prototipo utiliza el planificador global estándar de ROS y un planificador local muy simple basado en una máquina de estados. Otro vehículo con configuración de cuatro ruedas direccionables es el llamado BoniRob [64] que se utiliza para el fenotipado de plantas. Este vehículo tiene la posibilidad de cambiar de configuración para hacerse más angosto o ancho y adaptarse a la distancia entre las hileras de cultivos. Utiliza sensores GPS y LIDAR, con un planificador basado en modelos probabilísticos. En dicho trabajo se hace uso del simulador Gazebo para la validación de los algoritmos. En el trabajo [65] se presenta un robot para el desmalezado de campos de remolacha azucarera que utiliza en combinación un sensor GPS y una cámara estéreo para la detección de las hileras de cultivos. El planificador utilizado consiste en un simple controlador PID para la navegación en línea recta entre los cultivos y no tiene en consideración la evasión de obstáculos. Al finalizar el recorrido de una hilera, utiliza un algoritmo que genera líneas suaves para realizar el giro en la cabecera empleando funciones *B-spline*.

Por otro lado, podemos encontrar trabajos desarrollados sobre vehículos con configuración de tipo *skid-*

[12] ROS control, https://github.com/ros-controls/ros_control.

steer. Por ejemplo, en [66] se presenta un robot que se utiliza para tareas de fenotipado en campos de algodón. El trabajo utiliza ROS y Gazebo para las simulaciones, empleando para la navegación global un método de definición de *waypoints* mediante el uso de GPS y para el planificador local el algoritmo *pure pursuit*. Este algoritmo simplemente calcula la curvatura necesaria para mover el vehículo desde la posición actual hacia un punto unos pasos adelante en la trayectoria global, de ahí su nombre ya que se lo puede ver como persiguiendo ese punto móvil en la trayectoria global. Sin embargo, en dicho trabajo se advierte de la desventaja de que el prototipo no es capaz de evadir obstáculos. Por otro lado, [67] expone la implementación de un robot de bajo costo equipado solo con una cámara RGB-D que se utiliza en viñedos. Este sensor junto con la unidad de cómputo están montados sobre un vehículo *skid-steer* de uso comercial y utiliza un método de aprendizaje profundo para mantener al robot entre medio de las vides, pero ante la detección de obstáculos simplemente detiene su ejecución. El robot en [68], utilizado en invernaderos, emplea también un único sensor pero en este caso de tipo LIDAR. Los algoritmos implementados en este trabajo buscan mantener la posición del robot equidistante a las hileras de cultivos mediante un simple controlador PID. En dicho trabajo no se especifica el método de realización de giros en las cabeceras.

Adicionalmente, podemos encontrar trabajos basados en tractores o maquinarias agrícolas ordinarias adaptándolas para la navegación autónoma. Entre estos podemos destacar [69, 70] que consiste en un transplantador de arroz comercial, modificado para poder comandar electrónicamente la dirección y velocidad de las ruedas. Este vehículo funciona en dos modos: primero siguiendo una línea recta y luego girando en las cabeceras mediante un patrón preestablecido. En este caso no hay líneas de cultivos preexistentes para seguir sino que el dispositivo las va definiendo a medida que realiza el plantado. Otro trabajo similar es [71], que consiste en un tractor al que se le añade un sensor LIDAR. En este caso se utiliza un planificador que emplea métodos de programación lineal para optimizar una función objetivo que busca repeler al robot de los obstáculos. Dicho trabajo fue validado utilizando simulaciones en la plataforma Gazebo.

Existen también algunos pocos trabajos que utilizan robots con otras configuraciones de ruedas más similares a las del robot desmalezador. En [72] se presenta un robot con configuración de triciclo para el recorrido de maizales. En este trabajo se utiliza un sensor LIDAR 2D de bajo costo y el método de filtro de partículas para detectar la ubicación precisa del robot y así mantenerlo centrado entre las filas de cultivos. Destaca que puede funcionar correctamente incluso si las líneas de cultivos están curvadas. Debido a que el robot debe circular por entre medio de dos hileras de plantas de maíz y en ocasiones es tapado por las mismas, el sensor es ubicado en un soporte vertical que lo eleva por encima de las plantas. El trabajo [73], es uno de los pocos que utiliza un robot del tipo *car-like* en un entorno agrícola. El vehículo se construye para la navegación en huertos de árboles frutales. Desarrollan un planificador basado en el conocido algoritmo A* combinándolo con una apropiada definición del espacio de configuración [10] que involucra las restricciones de maniobrabilidad del robot. Utilizando varios parámetros de configuración y diferentes pesos que ajustan la función de penalidad a minimizar, permiten que el usuario pueda determinar características como el ángulo de giro permitido, la preferencia de conducción hacia adelante y la prevención de cambios de dirección muy frecuentes. Se muestran los experimentos realizados sobre un robot simulado en donde se pueden apreciar diferentes maniobras para la transición entre dos filas de árboles, las cuales dependen de las características del robot y de la configuración del planificador utilizada.

Por último, en [74] se presenta un robot de tipo *skid-steer* llamado Robotanist, que se utiliza para el fenotipado de plantaciones de sorgo. En dicho trabajo se utiliza un planificador del tipo *pure pursuit*. Posteriormente en [75], el mismo robot se prueba en el campus de la Universidad de Carnegie Mellon utilizando el planificador local TEB. Dicho robot consiste en una plataforma de bajo costo equipado con sensores GPS, LIDAR 2D y cámara estéreo. En el último de los trabajos, se menciona que también se realizaron pruebas con el planificador DWA, pero el TEB resultó ser más rápido en la generación de trayectorias y las mismas resultaron ser más suaves.

Por otra parte, existen también trabajos que estudian los planificadores disponibles en las librerías de ROS pero en otros ámbitos no agrícolas. El trabajo [76] ofrece una comparación de los planificadores locales más utilizados en ROS: DWA, EBand y TEB. Las pruebas son realizadas utilizando un robot con manejo diferencial en el simulador Gazebo. Se concluye que el planificador TEB genera trayectorias de mayor suavidad en la evasión de obstáculos y con menores tiempos de ejecución. El trabajo [77] realiza un exhaustivo análisis de los planificadores disponibles en ROS tanto globales como locales en ambientes bidimensionales estáticos.

Para las pruebas de los planificadores globales, los resultados indicaron que los algoritmos A* y Dijkstra generan los caminos con menor longitud aunque no los más suaves. De todos modos, como éstos luego serán ajustadas por el planificador local en este modelo de dos planificadores, se concluye que lo último no debería ser un problema. Por el lado de los planificadores locales, destacan la efectividad y robustez del planificador TEB, generando los recorridos que fueron ejecutados por el robot en el menor tiempo. Las pruebas fueron llevadas a cabo tanto en entornos simulados con Gazebo, como en entornos reales. Por su parte, en [78] se estudia el uso del planificador global de Dijkstra junto con el planificador local TEB en un vehículo *car-like* llamado iCab, que consiste en un carrito de golf modificado con el agregado de sensores, navegando alrededor del campus de una universidad. También se destaca la robustez del planificador TEB pudiendo evadir sin problemas los obstáculos presentados.

El estado del arte demuestra que existen varios prototipos de robots agrícolas desarrollados actualmente, de los cuales la gran mayoría utiliza planificadores hechos a medida que dividen el problema en una etapa encargada de mantener al robot entre medio de las hileras de cultivos mientras realiza un recorrido en línea recta y otra etapa que se encarga de los giros en las cabeceras del campo. Sin embargo, estos métodos desarrollados por lo general no son capaces de evadir obstáculos inesperados. Por otro lado, también existen varias pruebas de robots en ambientes abiertos como campus de universidades utilizando los planificadores disponibles en ROS, pero vemos una carencia del uso de estos en entornos agrícolas. En este trabajo se desarrolla un modelo de navegación para el robot desmalezador que busca aprovechar los beneficios del código maduro y ampliamente probado del *navigation stack* de ROS utilizando los planificadores globales A* y Dijkstra junto con el planificador local TEB en ambientes agrícolas. Para poder lograr su correcto funcionamiento se deben hacer ciertas adaptaciones entre otras cosas al modelado de los mapas como se verá más adelante. El uso del planificador genérico TEB trae consigo también la ventaja de permitir la evasión de obstáculos imprevistos. En los siguientes capítulos se presenta en forma detallada el desarrollo del entorno de simulación para el robot desmalezador utilizando Gazebo, junto con los algoritmos de planificación global y local usados, su configuración y puesta en marcha.

Capítulo 3

Simulación del robot desmalezador

En este capítulo se describe todo lo relacionado con el desarrollo de la simulación y control del robot desmalezador. Se detalla el modelo de representación del robot y de su entorno virtual de navegación. Por otro lado, se describe el software de control empleado para el manejo de las ruedas y dirección del robot. En este capítulo se utilizará como referencia el diagrama presentado en la Figura 3.1. Dicho diagrama se extiende con la parte de navegación del siguiente capítulo y su versión completa puede verse en el apéndice A.

Para realizar la simulación del robot desmalezador y su entorno de navegación se utiliza el simulador Gazebo y el *framework* ROS, los cuales se acoplan de manera simple pero eficaz. Todo el software para la navegación y control del robot se desarrolla en ROS, y el mismo podrá ser migrado al robot real sin mayores cambios. Ambos programas deben correr en Ubuntu y para obtener la mayor fluidez y compatibilidad se deben utilizar versiones específicas[1][2]. Se utilizó Ubuntu 18.04 Bionic, ROS Melodic Morenia, Gazebo 9. Se seleccionaron versiones con soporte de largo plazo (LTS). Adicionalmente, cabe destacar que todo el software previamente mencionado es de código abierto.

3.1. Descripción URDF

Para utilizar el robot desmalezador en la simulación es necesario describir su estructura de *links* y *joints*, así como también sus propiedades cinemáticas y dinámicas. Esta información será utilizada por Gazebo para simular el robot, por RViz para visualizar el estado de cada una de sus partes y por el *navigation stack* de ROS para realizar la navegación por el entorno. Como se indicó en la sección 2.3.3, la descripción del modelo se realiza en un formato XML llamado URDF. Los dos elementos principales de este formato son los *links* y *joint*s. Un *link* representa una parte rígida del robot y un *joint* representa una articulación que une dos o más de estas partes. En el formato URDF, el elemento *link* poseerá un atributo *name* con el nombre de dicha parte y varios sub-elementos para describir las propiedades inerciales, visuales y de colisión. A continuación, se listan las partes en las que se dividió el robot desmalezador. Se utilizaron las letras **`f`** y **`r`** para nombrar las partes delanteras (*front*) y traseras (*rear*) respectivamente, y las letras **`l`** y **`r`** para nombrar las partes de la izquierda (*left*) y derecha (*right*) del robot.

- chasis: **`chassis_link`**
- conjunto de paneles solares: **`panels_link`**
- conjunto de rociadores: **`sprayers_link`**
- 4 horquillas: **`fork_fl_link`**, **`fork_fr_link`**, **`fork_rl_link`**, **`fork_rr_link`**
- 4 ruedas: **`wheel_fl_link`**, **`wheel_fr_link`**, **`wheel_rl_link`**, **`wheel_rr_link`**

[1] Versiones de ROS, http://wiki.ros.org/Distributions.
[2] Versiones de Gazebo, http://gazebosim.org/tutorials/?tut=ros_wrapper_versions.

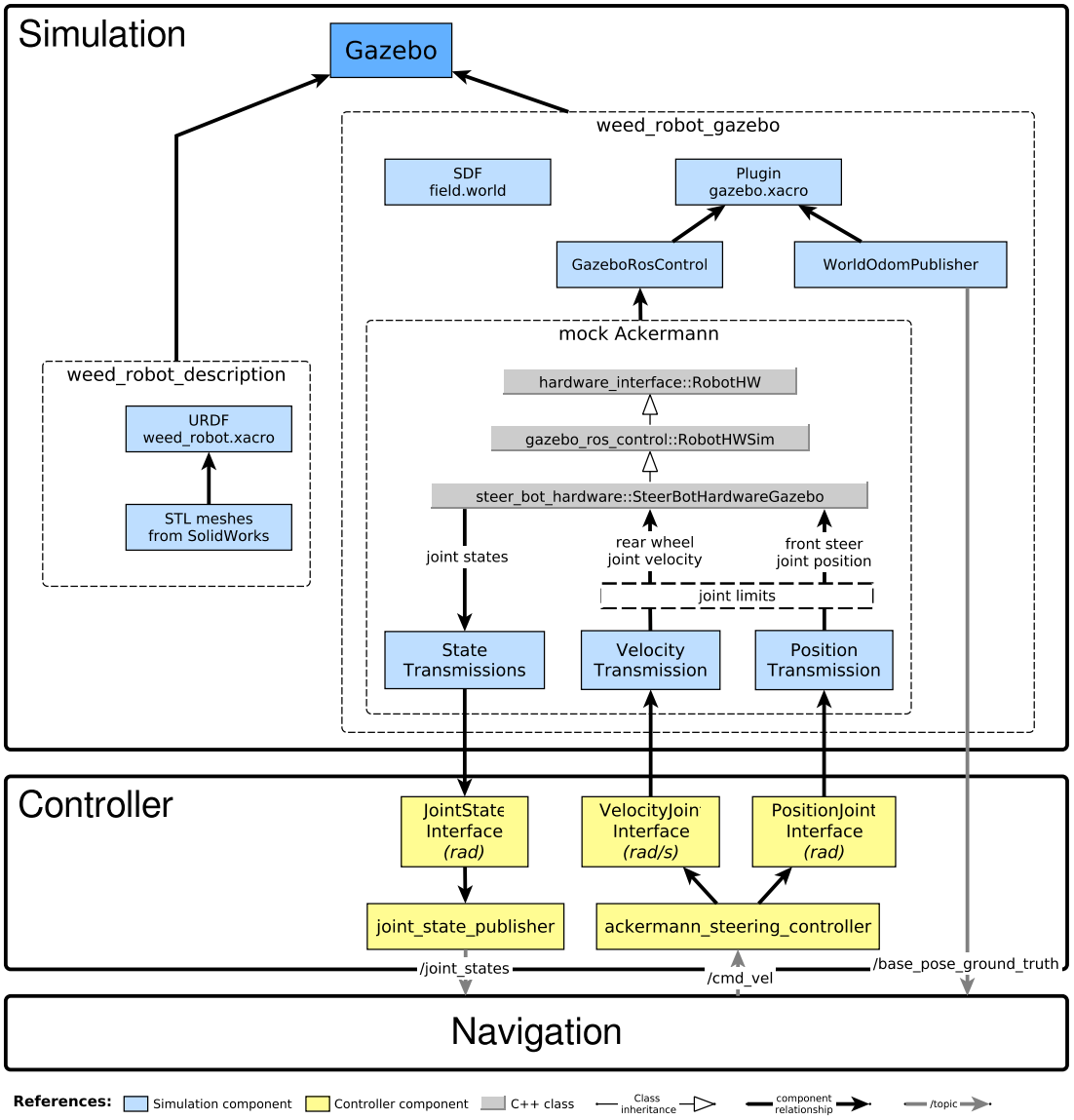


Figura 3.1: Diagrama de simulación y control del robot desmalezador. En la parte inferior, el controlador recibe los comandos del módulo de navegación a través del *topic* **`/cmd_vel`**. El controlador convierte estos comandos y envía una señal de velocidad angular para una única rueda trasera imaginaria y una señal de posición para la dirección de una única rueda delantera también imaginaria, ambas ubicadas en el centro del chasis como si se tratase de una bicicleta. Luego, el submódulo *mock Ackermann* se encarga de enviar la velocidad angular a ambas ruedas traseras y transformar la señal de dirección a las posiciones de ambas ruedas delanteras imitando la relación de Ackermann. Por otro lado, un archivo URDF describe tanto las características físicas del robot desmalezador como las visuales mediante el uso de mallas STL y un archivo SDF describe el entorno del campo de cultivos.

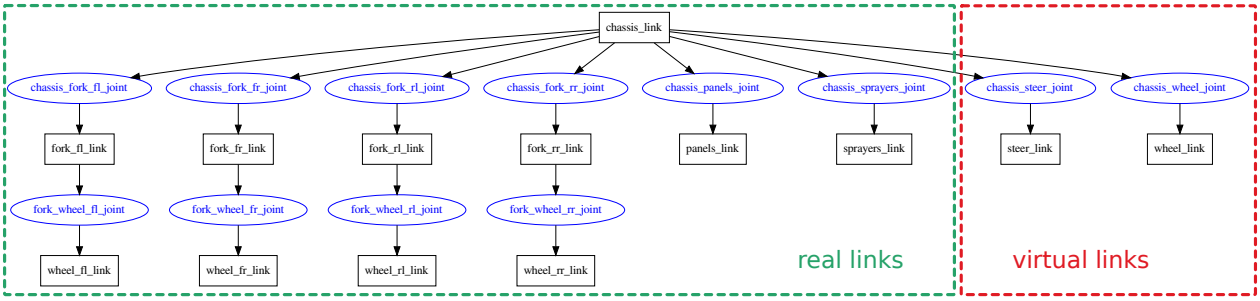


Figura 3.2: Modelo de *links* y *joints* del robot desmalezador. Los *links* se representan como cajas negras y los *joints* como óvalos azules. Este árbol muestra la estructura real del robot demarcada con un recuadro verde y su extensión con *links* virtuales demarcada con un recuadro rojo.

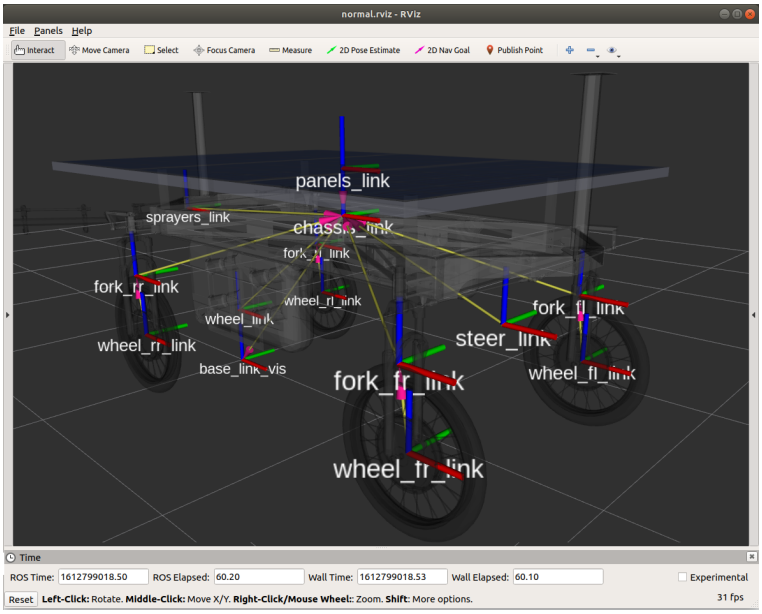


Figura 3.3: Visualización en RViz del modelo del robot desmalezador con las partes reales y virtuales.

Los tipos de *joints* utilizados en la descripción del robot desmalezador resultaron ser solo tres: *revolute*, *continuous* y *fixed*. Los tipos *revolute* y *continuous* se utilizan para definir relaciones rotacionales sobre un único eje con o sin un límite de rango angular respectivamente. El tipo *continuous* se utiliza para conectar las ruedas con las horquillas y el tipo *revolute* para la conexión de las horquillas delanteras con el chasis, lo que permitirá el manejo de la dirección al robot. Por otro lado, el tipo llamado *fixed,* no es realmente una articulación ya que restringe todos los grados de libertad del *link* que conecta, dejándolo en una posición fija. Sin embargo, resulta de utilidad para subdividir al robot en partes más simples, evitando la necesidad de generar *links* con formas más complejas. En el robot desmalezador se utiliza el tipo *fixed* para conectar el chasis con los paneles solares, rociadores y horquillas traseras.

Cada *link* genera un marco de coordenadas que luego será publicado en el *topic* **`/tf`**. En los *joints* se describen propiedades dinámicas como la fricción y amortiguación (*damping*), así como también su ubicación con respecto al *link* padre. Adicionalmente, se definen límites de velocidad y fuerza (o torque). Para la asignación de los nombres de los *links* y *joints*, así como para las unidades y marcos de coordenadas utilizados se siguieron los estándares establecidos en los ROS Enhancement Proposals[3][4] (REP).

[3]Standard Units of Measure and Coordinate Conventions (REP 103), https://www.ros.org/reps/rep-0103.html.
[4]Coordinate Frames for Mobile Platforms (REP 105), https://ros.org/reps/rep-0105.html.

Se definieron dos *links* adicionales que no representan partes reales del robot sino que fueron agregados únicamente porque son necesarios para poder utilizar el controlador **ackermann_steering_controller** como se explica más adelante en la sección 3.3.1. Llamaremos a estos componentes *virtuales*, en contraposición de los anteriores, los cuales diremos que son *reales*:

- única dirección frontal virtual: **steer_link**
- única rueda trasera virtual: **wheel_link**

En la Figura 3.2 se muestra el grafo completo de todos los *links* y *joints* que forman parte del modelo del robot desmalezador. Dicho grafo fue generado a partir del formato DOT utilizando la herramienta **urdf_to_graphiz**. La Figura 3.3 muestra una captura de RViz donde se puede apreciar estos *links* y su ubicación en el robot desmalezador con todos los marcos de coordenadas generados. Notar que las partes virtuales no poseen propiedades visuales, sino que simplemente son agregados para que cuando se cargue el archivo URDF se creen sus interfaces y se pueda hacer uso del controlador mencionado.

3.1.1. Modelo visual y de colisión

Los *links* podrán describirse con mallas CAD 3D de diferentes formatos (STL, DAE, etc.) o mediante figuras simples: *box*, *cylinder*, o *sphere*. Se deben especificar dos modelos del robot a simular: uno visual y otro de colisión. El modelo visual suele estar construido por mallas complejas y será el que le proporciona la estética a la simulación. El modelo de colisión, en cambio, es el que se utilizará para los cálculos de la interacción del robot con el entorno y se suelen utilizar figuras simples o mallas simplificadas para reducir el tiempo de cómputo. Las definiciones en XML junto con las mallas STL del modelo visual y archivos adicionales de visualización y configuración se agrupan en el paquete **weed_robot_description**.

En los diferentes *links* se describen tanto las dimensiones del cuerpo como las diferentes propiedades físicas del mismo: masa, centro de masa, momento de inercia, etc. Se utilizó un modelo del robot desmalezador desarrollado en SolidWorks para la generación de las mallas y la obtención de una aproximación de las propiedades físicas de las diferentes partes. En la Figura 3.4 se puede ver el modelo del robot desmalezador en SolidWorks. Para la exportación de las mallas se utiliza el *plugin* **sw_urdf_exporter**[5]. En la Figura 3.5 se muestra la comparación de los modelos visual y de colisión del robot desmalezador en el simulador Gazebo. Para la generación del modelo de colisión se utilizaron figuras simples del tipo *cylinder* para las ruedas y *box* para el resto de las partes.

3.1.2. Limitaciones de URDF

Actualmente, el formato URDF posee ciertas limitación que impiden la descripción de algunas configuraciones de robots. Una de ellas consiste en la imposibilidad de definir cadenas cerradas de *links*, es decir, la relación entre las partes del robot solo podrá tener una estructura de árbol (sin ciclos). Otra limitación consiste en la falta de ciertos tipos de articulaciones, dentro de los cuales se encuentra el tipo esférico (también conocido como *ball-socket*). Debido a esto, no es posible modelar la dirección de Ackermann que posee el robot desmalezador, por lo que la relación entre ambos ángulos de dirección de las ruedas delanteras deberá ser emulada en el controlador como se verá en la sección 3.3.

Como una alternativa al formato URDF, la comunidad de Gazebo desarrolló un nuevo formato para la descripción de robots llamado SDF, el cual no posee las limitaciones anteriormente mencionadas, y a su vez agrega varias características nuevas. El formato URDF solo puede especificar las propiedades de un único robot de forma aislada, en cambio con SDF se puede especificar también el entorno de navegación del robot (conocido como *world*), además de fuentes de luz, viento, gravedad, cámaras, entre otras. No obstante a esto, se decidió seguir utilizando el formato URDF para la tarea específica de la definición del modelo del robot por compatibilidad con ROS. Este modelo en URDF es convertido automáticamente por Gazebo al formato SDF al ejecutarse la simulación. De esta manera evitamos tener dos especificaciones diferentes del modelo

[5]SolidWorks to URDF Exporter, http://wiki.ros.org/sw_urdf_exporter.

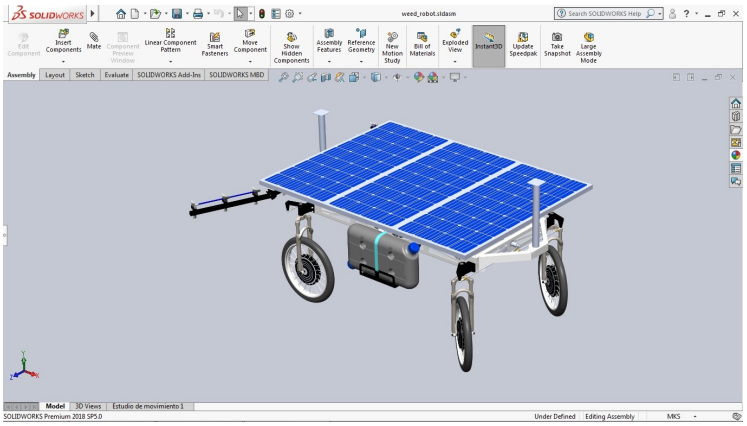
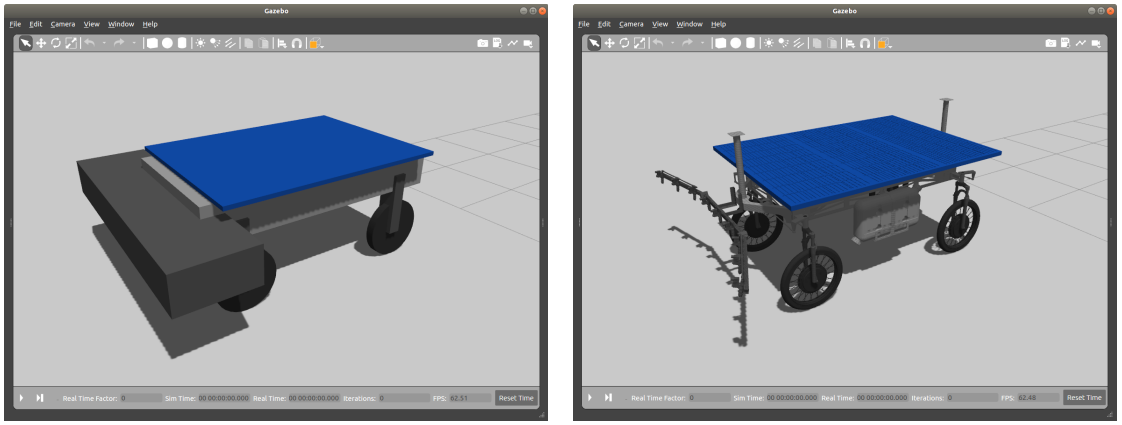


Figura 3.4: Modelo del robot desmalezador desarrollado en SolidWorks. Este modelo se utilizó como base para exportar las mallas de las diferentes partes del robot que luego son utilizadas en la simulación.



(a) Modelo de colisión. **(b)** Modelo visual.

Figura 3.5: Comparación entre **(a)** modelo de colisión y **(b)** modelo visual del robot desmalezador. El modelo de colisión es utilizado por el motor físico para los cálculos de las interacciones del robot con el entorno, mientras que el modelo visual simplemente le proporciona una estética realista a la simulación.

en formato URDF y SDF. Algunas definiciones específicas de SDF pueden ser añadidas al archivo URDF usando la etiquera **<gazebo>**, las cuales serán ignoradas por ROS pero utilizadas en la simulación de Gazebo. Esto resulta de gran utilidad para especificar la utilización de diferentes *plugins* de Gazebo como veremos más adelante. Existirá también un archivo en formato SDF al cual nombramos como **field.world**, pero solo tendrá descripciones del entorno de navegación y no del modelo del robot. Dicho archivo se describe en la sección 3.4.

3.1.3. Parametrización con Xacro

La herramienta Xacro[6] consiste en un sistema de macros para documentos XML que simplifica la definición del modelo del robot en formato URDF, aumentando su legibilidad, evitando la duplicación de código y facilitando su mantenimiento. El programa de consola **xacro** recibe como entrada un archivo en este formato y expande las macros para generar el archivo URDF. Un ejemplo del uso de este comando para generar el URDF del robot desmalezador se muestra a continuación:

[6]Xacro (XML Macros), http://wiki.ros.org/xacro.

```
$ xacro urdf/weed_robot.xacro meshes:=true > weed_robot.urdf
```

Dentro de las características que ofrece Xacro se encuentra la posibilidad de recibir argumentos desde la línea de comandos que parametricen la ejecución de las macros, como se puede ver en el ejemplo anterior con el parámetro **meshes**. Esta propiedad se definió para indicar si utilizar mallas STL o figuras geométricas simples para el modelo visual del robot. Esto resulta de mucha utilidad ya que para las pruebas de concepto se puede utilizar el modelo sin mallas que requiere un menor procesamiento de la CPU y principalmente de la tarjeta gráfica. Si bien para el modelo de colisión siempre se utilizan las figuras geométricas simples, con esta propiedad en *false* lo que se consigue es también utilizar el mismo modelo de colisión simple para el modelo visual. Para generar las imágenes de la Figura 3.5 se hizo uso de esta propiedad con el objetivo de evidenciar la diferencia entre ambos modelos. Esto también permitió una rápida validación preliminar del modelo de colisión.

Por otro lado, el uso de Xacro también ofrece la posibilidad de una parametrización mediante archivos en formato YAML. Los valores de las dimensiones y masas de todas las partes del robot, como así también la fricción de las ruedas, se configuraron en un archivo **weed_robot.yaml**, de manera que se desacopla del XML y se facilita su ajuste y mantenimiento. Otras características que se aprovecharon de Xacro fueron el uso de definiciones condicionales y expresiones matemáticas, y la división de la descripción del robot en varios archivos.

3.1.4. Propiedades físicas

Es fundamental definir correctamente las propiedades físicas y cinemáticas en la descripción del modelo del robot para obtener una simulación relativamente buena que se asemeje a la realidad. Se comprobó empíricamente que utilizando valores muy alejados a los reales el comportamiento del robot se torna demasiado errático, presentándose situaciones de sobreaceleraciones bruscas, deslizamiento excesivo de las ruedas o hundimiento del robot en el suelo. De todos modos, se observó que tampoco es necesario que dichas propiedades sean extremadamente precisas, con una simple aproximación resulta suficiente para obtener una simulación realista.

Para estimar las propiedades de masa y momento de inercia de las diferentes partes del robot se hizo uso de las herramientas de medición que ofrece SolidWorks, utilizando el modelo del robot desmalezador que se tenía definido de antemano en dicho software (ver Figura 3.4). En cuanto a la definición de las propiedades de fricción de las ruedas, primero se buscaron los valores normales para los materiales utilizados y luego se realizó un ajuste más fino realizando pruebas en la simulación hasta obtener un comportamiento aceptable.

3.2. Dirección de Ackermann

Supongamos el caso de un vehículo de cuatro ruedas del tipo *car-like* con dirección delantera realizando un giro como el de la Figura 3.6. Veremos que la rueda interna dibujará una circunferencia de radio menor a la rueda externa. Para evitar el deslizamiento lateral de las ruedas, la rueda interna deberá tener un ángulo de dirección más pronunciado que el de la rueda externa. Esta condición se consigue con lo que se conoce como geometría de Ackermann. Como se ilustra en la Figura 3.6, esta diferencia en los ángulos de dirección de las ruedas frontales produce que las líneas perpendiculares a dichas ruedas se intersequen en el mismo punto al cruzar el eje trasero. Dicho punto se lo conoce como centro de rotación (CoR, por sus siglas en inglés *Center of Rotation*). El radio de giro del vehículo quedará determinado por la distancia entre el CoR y el centro del eje trasero y se denota con la letra ρ. Cuando el robot se mueve en línea recta hacia adelante o atrás el radio de giro se considera infinito y las líneas perpendiculares a ambas ruedas de dirección resultan paralelas entre sí.

Para determinar los ángulos de las ruedas delanteras derecha e izquierda en una configuración de Ackermann, se considera en primer lugar una rueda imaginaria ubicada en el centro del eje delantero. En dicha rueda imaginaria se medirá el ángulo de dirección del vehículo que se lo notará con φ. Los ángulos de las

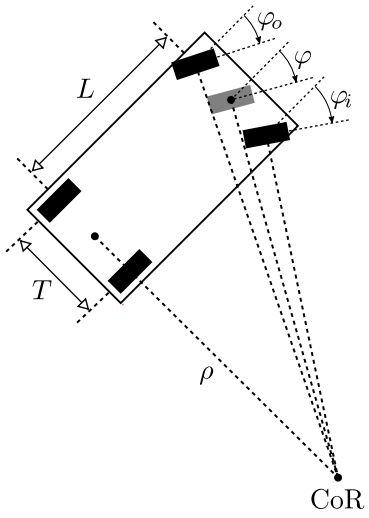


Figura 3.6: Dirección de Ackermann. Para evitar el deslizamiento lateral, la rueda interna deberá tener un ángulo de dirección más pronunciado que la rueda externa. Esta diferencia produce que las líneas perpendiculares a las ruedas frontales se intersequen en el mismo punto al cruzar la continuación del eje trasero. Dicho punto se lo conoce como CoR (centro de rotación).

ruedas internas y externas se notan con φ_i y φ_o respectivamente (i por *inner*, o por *outer*). Será necesario tener en cuenta también el *wheelbase* y *track width* del vehículo. Se denomina *wheelbase* (batalla) a la distancia entre los ejes delanteros y traseros del vehículo y se lo denota con la letra L. Por otro lado, se denomina *track width* (vía) a la distancia entre las ruedas del mismo eje y se lo denota con la letra T. Por trigonometría obtenemos las siguientes ecuaciones:

$$\tan(\varphi) = \frac{L}{\rho}, \qquad \tan(\varphi_i) = \frac{L}{\rho - \frac{T}{2}}, \qquad \tan(\varphi_o) = \frac{L}{\rho + \frac{T}{2}}. \tag{3.1}$$

Despejando de estas ecuaciones podemos obtener los ángulos de dirección para las dos ruedas reales dado el ángulo de dirección φ deseado para el vehículo:

$$\varphi_i = \arctan\left(\frac{L}{\rho - \frac{T}{2}}\right), \qquad \varphi_o = \arctan\left(\frac{L}{\rho + \frac{T}{2}}\right), \tag{3.2}$$

donde $\rho = \frac{L}{\tan(\varphi)}$. Para lograr dicha relación entre los ángulos de la rueda interna y externa, la dirección de Ackermann utiliza un mecanismo articulado de cuatro barras. Como se mencionó anteriormente, las limitaciones del formato URDF imposibilitan la descripción de este mecanismo en el modelo del robot, particularmente debido a no poder definir disposiciones de *links* en ciclos y por la falta de un tipo de *joint* esférico. Debido a esto, la relación de dirección de Ackermann que posee el robot desmalezador será emulada por código en el controlador **`SteerBotHardwareGazebo`** (ver diagrama de la Figura 3.1).

El ángulo de dirección posee valores mínimos y máximos que provocan que el robot tenga radios mínimo de giro limitado en ambos sentidos. Para el robot desmalezador los ángulos mínimo y máximo de giro son de 33°, lo cual da un radio mínimo de giro de aproximadamente 2.4 m. Esto desencadena en el hecho de que el robot será incapaz de girar en el lugar, lo cual lo diferencia de los robots con manejo diferencial. Para que un robot con dirección de Ackermann tenga compatibilidad de comando con los de manejo diferencial, una conversión en el mensaje utilizado es requerido como se verá en la siguiente sección.

3.3. Control del robot desmalezador

Para realizar el control del robot lo primero que debe hacerse es definir una interfaz de control. En el entorno de ROS existen diferentes tipos de mensajes estándares para el control de robots móviles, dentro de los cuales el más utilizado es el **geometry_msg/Twist**. Este mensaje está compuesto por seis parámetros: tres velocidades lineales y tres velocidades angulares. Con dicha interfaz se puede realizar el control de cualquier robot en el espacio tridimensional. Sin embargo, para el control de vehículos terrestres solo se utilizan los componentes **linear.x** y **angular.z**, que representan su velocidad lineal hacia adelante en m/s y su velocidad angular de guiñada (*yaw*) en rad/s.

```
geometry_msgs/Twist
      geometry_msgs/Vector3 linear
            float64 x
            float64 y
            float64 z
      geometry_msgs/Vector3 angular
            float64 x
            float64 y
            float64 z
```

Esta interfaz es muy utilizada para los robots de manejo diferencial, que pueden moverse hacia adelante y atrás, y de forma independiente también rotar sobre sí mismo. Para el caso de los robots de tipo *car-like* con dirección de Ackermann este tipo de mensaje en principio podría no ser el más apropiado ya que dicho vehículo no puede rotar sobre sí mismo de forma instantánea sin moverse hacia adelante. Por ejemplo, si en el mensaje de control se indica un valor en cero de **linear.x** y un valor distinto de cero en **angular.z** el robot *car-like* no podrá cumplirlo, a diferencia de uno de manejo diferencial. Debido a esto, la comunidad de ROS desarrolló el tipo **ackermann_msgs/AckermanDrive**. La principal diferencia de esta interfaz con la anterior consiste en que no se controla la velocidad angular del robot sino la posición de su dirección. Concretamente, este tipo posee, por un lado, parámetros para controlar la posición y velocidad del ángulo de dirección, y por el otro, parámetros para controlar la velocidad, aceleración y sobreaceleración lineal del robot.

```
ackermann_msgs/AckermannDrive
      float32 steering_angle
      float32 steering_angle_velocity
      float32 speed
      float32 acceleration
      float32 jerk
```

El ángulo de dirección controlado con este tipo de mensaje corresponde a una rueda imaginaria ubicada en el centro del eje delantero, como si fuera un triciclo o una bicicleta. Posteriormente, se deberá utilizar la relación de Ackermann para determinar el ángulo de dirección de cada una de las ruedas delanteras, si el vehículo es de tipo *car-like*.

A pesar de la existencia de esta nueva interfaz para vehículos con dirección de Ackermann, en este trabajo se utiliza la interfaz tradicional **geometry_msg/Twist**, ya que la misma está soportada por el *navigation stack* de ROS, que se verá en detalle en el capítulo 4, y así aprovechar la gran cantidad de código ya desarrollado compatible con éste. Para poder utilizar esta interfaz, deberá realizarse una adaptación de los controles recibidos para poder manejar adecuadamente las ruedas del robot desmalezador. En la Figura 3.7 se muestran los pasos para transformar el mensaje desde la etapa de navegación hasta Gazebo, los cuales se describen en detalle a continuación.

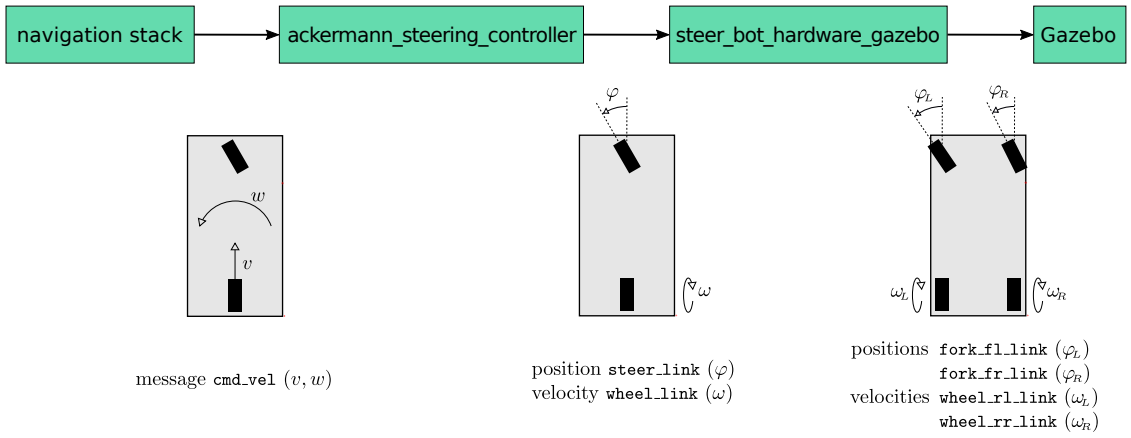


Figura 3.7: Transformación del mensaje desde el *navigation stack* hasta el simulador Gazebo. Desde el componente de navegación se reciben los controles de velocidad lineal y angular del robot (v, w). En primera instancia, se convierten dichos controles a una velocidad angular ω para la rueda trasera del robot y una posición angular φ de la dirección delantera, como si se tratase de una bicicleta. Luego, se realiza una conversión utilizando la relación de Ackermann para obtener los ángulos de las ruedas izquierda y derecha (φ_L, φ_R). Las velocidades de ambas ruedas traseras (ω_L, ω_R) se dejan en principio con el mismo valor.

3.3.1. Controlador de Ackermann

El primer paso de la conversión se hace mediante el empleo del controlador **`ackermann_steering_controller`**[7]. Este controlador se diseñó para manejar solo dos *joints* como si el vehículo fuera una bicicleta. Recibirá por medio del *topic* **`/cmd_vel`** los comandos de velocidad lineal v y velocidad angular w, y los transformará en una velocidad angular ω para la rueda trasera y en un ángulo de dirección φ para la horquilla delantera. Se comunicará con el *driver* del robot mediante dos interfaces de tipo **`VelocityJointInterface`** y **`PositionJointInterface`** para enviar los valores de ω y φ respectivamente como se puede ver en el diagrama de la Figura 3.1. Posteriormente el *driver* deberá realizar una segunda transformación de estos controles a los *joints* reales del robot *car-like*. Esto se realiza mediante el *plugin* **`SteerBotHardwareGazebo`**. Este *plugin* transforma la velocidad angular ω de una única rueda trasera virtual en una velocidad angular ω_L y ω_R, para cada una de las ruedas traseras reales mediante dos controladores PID configurados de forma independiente. Si bien se busca establecer la misma velocidad en ambas ruedas podría agregarse una relación diferencial para que, cuando el vehículo esté realizando un giro, la velocidad de la rueda interna sea menor que la de la rueda externa. Sin embargo, esto quedará como una mejora a futuro. Por otro lado, el *plugin* transforma el ángulo de dirección de la rueda frontal imaginaria φ en dos ángulos de dirección φ_L y φ_R para las direcciones de las ruedas izquierda y derecha respectivamente, siguiendo la relación de Ackermann.

Para transformar la velocidad lineal v en una velocidad angular para aplicar a la rueda virtual trasera solo será necesario conocer el radio de la rueda que lo notaremos con r y aplicar la siguiente ecuación:

$$\omega = \frac{v}{r}. \tag{3.3}$$

Para obtener la posición de la dirección de la rueda delantera primero necesitamos conocer el radio de giro del robot ρ que se puede obtener de la siguiente manera:

$$\rho = \frac{v}{w}. \tag{3.4}$$

Luego, conociendo el *wheelbase* L del vehículo y usando trigonometría se puede deducir:

[7] Ackermann steering controller, http://wiki.ros.org/ackermann_steering_controller.

$$\varphi = \arctan\left(\frac{L}{\rho}\right). \tag{3.5}$$

Además de la conversión descrita, el controlador **ackermann_steering_controller** ofrece dos características adicionales. En primer lugar, la posibilidad de fijar límites en la velocidad, aceleración y sobreaceleración del componente lineal y angular del robot, y por otro lado, el cálculo de la odometría mediante el *feedback* recibido desde el hardware.

El conjunto del controlador y *driver* funciona como un nodo de ROS y al mismo tiempo como un *plugin* de Gazebo, que recibe los comandos de control en el *topic* **/cmd_vel**, realiza las conversiones oportunas y finalmente acciona las ruedas del robot simulado. En resumen, en este conjunto se realizan los cálculos para transformar la velocidad lineal y angular de control, en velocidades angulares para las ruedas traseras y ángulos de dirección para las horquillas delanteras siguiendo la relación de Ackermann. Cabe destacar que como simplificación solo las ruedas traseras son accionadas, a pesar de que el robot desmalezador real posee motores en las cuatro ruedas. Queda pendiente como una mejora a futuro el accionado de todas las ruedas del robot.

Para poder visualizar el estado del robot en todo momento mientras se realiza la navegación, se hace uso de otro controlador llamado **joint_state_publisher** que publica en el *topic* **/joint_states** el estado de todos los *joints* del robot, información que es obtenida a través del *hardware* simulado del robot.

3.3.2. Test comandado por teclado

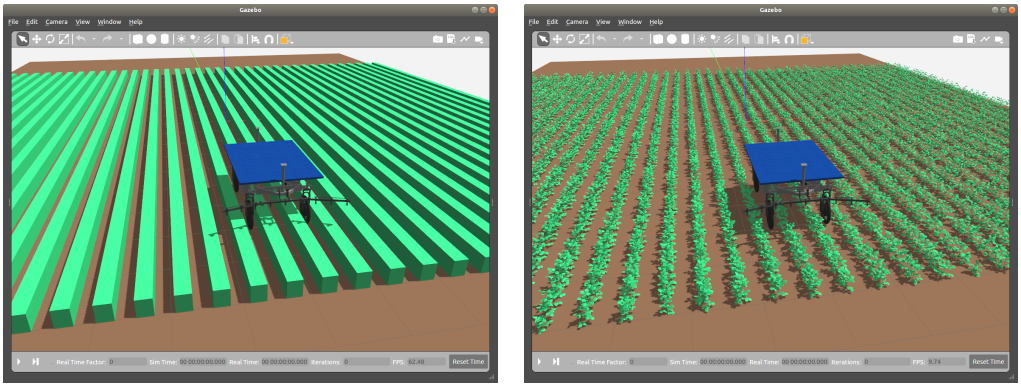
Para realizar las primeras pruebas de navegación del robot desmalezador y la validación de todo lo expuesto en cuanto al modelo simulado y control se utilizó el comando por teclado ofrecido por el paquete ROS **teleop_twist_keyboard**[8]. Este paquete lee comandos desde el teclado y los publica en el *topic* **/cmd_vel** con el tipo de mensaje **geometry_msgs/Twist**. Se verificó que el robot pueda cumplir con un margen de error razonable el mínimo radio de giro al llevar la dirección del mismo al máximo ángulo posible. Esta prueba nos permite detectar si existe mucho deslizamiento lateral y así poder ajustar correctamente el parámetro de fricción de la ruedas, entre otros. El radio mínimo de giro obtenido experimentalmente resultó ser de 2.5 m aproximadamente, lo cual es suficientemente cercano a los 2.4 m esperados. Como se verá luego, el planificador de trayectorias se configura con un valor superior a éste para asegurarnos de que el robot pueda llevar a cabo las trayectorias generadas.

3.4. Descripción del entorno

Un entorno de simulación en Gazebo se lo conoce por el término *world* y para su definición se utiliza el formato SDF. Este es un formato XML mediante el cual se pueden describir, por un lado, las propiedades físicas del entorno en general como la gravedad, viento, campo magnético, temperatura, presión, etc., y por otro lado, agregar objetos estáticos y dinámicos que formarán parte del escenario de simulación. La interfaz gráfica de Gazebo permite la creación en forma interactiva de un *world* agregando figuras geométricas simples o modelos de objetos que se pueden obtener desde la base de datos de Gazebo *online* o modelos que tengamos en nuestro sistema de archivos local. Los modelos pueden estar compuestos por figuras geométricas simples para los cuales existen las etiquetas *box*, *cylinder* y *sphere*, o bien pueden incluir referencias a mallas complejas con o sin texturas en formatos como STL o DAE. Luego de crear el entorno deseado, el mismo se puede guardar para su posterior uso bajo la extensión *.world*. Por otro lado, al tratarse de un simple archivo de texto, éste puede ser fácilmente creado de forma programática.

El entorno de navegación del robot desmalezador consiste en un campo con hileras de cultivos de soja. Las dimensiones del campo, cantidad de hileras, alto y ancho de las plantas, y varias características más pueden variar de un campo a otro y resulta útil poder generar distintos entornos para la realización de diferentes pruebas. Para lograr esto, se creó el programa *world_generator.py* que recibe una serie de parámetros

[8]Teleoperation twist keyboard, http://wiki.ros.org/teleop_twist_keyboard.



(a) World de colisión. **(b)** World visual.

Figura 3.8: Entorno de simulación de un campo agrícola con cultivos de soja modelando las hileras con cajas simples **(a)** y con mallas STL de plantas **(b)**. El modelo de mallas le da un aspecto más realista a la simulación.

relacionados con las dimensiones y disposiciones del campo y las plantas que forman las hileras, y genera un archivo *world* en XML. Para las propiedades generales del entorno se utiliza una plantilla previamente generada a la cual se le ajustan ciertos parámetros como, por ejemplo, las dimensiones del campo. Luego, el programa va añadiendo distintos modelos de objetos para construir el entorno deseado.

En una primera instancia, el campo se construyó con una serie de cajas alargadas que corresponden a toda una hilera de plantas. Posteriormente, para darle un aspecto más estético, se crearon diferentes modelos de plantas y se conformaron las hileras con estas. En la Figura 3.8 se pueden ver capturas de pantallas de los entornos con plantas y con cajas simples.

Para la creación de las plantas se utilizó SolidWorks. Se hizo una pequeña búsqueda en la web de las formas y dimensiones de las plantas de soja y se crearon tres modelos de plantas como se puede ver en la Figura 3.9. Luego se exportaron estos modelos para crear mallas STL de la misma forma que se lo hizo con las partes del robot desmalezador. Por último, se ubicaron dichas plantas en los lugares correspondientes a las hileras dentro del campo simulado. Para darle un aspecto más realista, el programa de generación del *world* toma en cada caso uno de los tres modelos de plantas de forma aleatoria y lo ubica dentro de un pequeño rango de la posición en la que debería estar la planta y le da un ángulo de *yaw* también tomado de forma aleatoria. Si bien no es un problema abordado en este trabajo, la creación de las mallas de plantas no es solo una cuestión estética sino que puede ser de utilidad en un futuro para el sistema encargado de la detección de los surcos y la localización del robot en su entorno.

Cabe destacar que el modelo que emplea las mallas de plantas exige mayor procesamiento de GPU. Debido a esto, la mayoría de las pruebas se realizaron con el modelo de cajas simples. Otra alternativa que resultó exitosa para reducir los costos de la simulación, fue la de simplificar la resolución de las mallas con el programa de modelado y animación tridimensional Blender. Dicho software se utilizó también para otras tareas simples como el ajuste de los ejes de coordenadas y como herramienta para la verificación de las mallas exportadas.

3.5. Odometría

Como mencionamos anteriormente, el controlador **`ackermann_steering_controller`** que se utiliza publica en el *topic* **`/odom`** la odometría estimada del robot, calculada a partir del *feedback* recibido desde el hardware simulado. Adicionalmente a esto, se creó un nuevo componente llamado **`world_odom_publisher`** que se encargará de informar la odometría real sin errores del robot, lo cual se conoce como *ground-truth*. Dicho componente se ejecutará al mismo tiempo como un *plugin* de Gazebo y como un nodo de ROS. Al ser un *plugin* de Gazebo puede obtener a través de su API C++ la posición y velocidad real del robot simulado, luego realiza ciertas

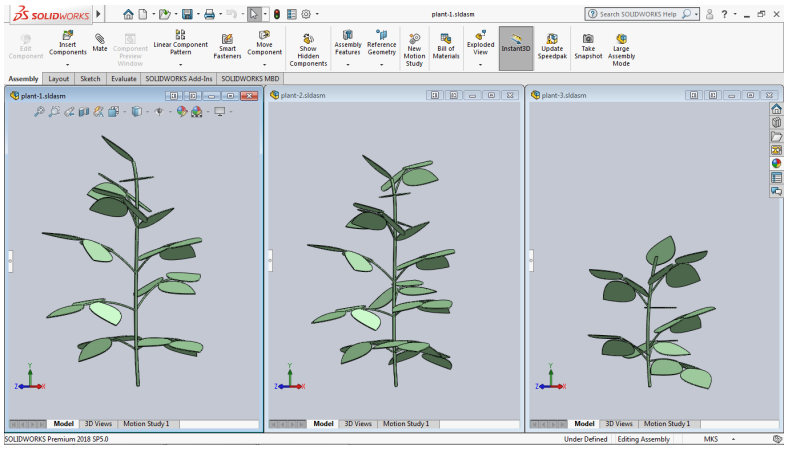


Figura 3.9: Modelos de plantas desarrollados en SolidWorks. Se crearon tres modelos con diferentes alturas y cantidades de hojas para imitar plantas reales de soja. Estos modelos se utilizan para la generación del entorno de simulación en Gazebo.

transformaciones para adaptar dicha información al tipo de mensaje estándar de odometría, y finalmente, al ejecutarse también como un nodo ROS puede publicar este mensaje en el *topic* **`/odom`**. Para que no exista conflicto entre ambos mensajes de odometría estimada y real, se agrega un *namespace* al *topic* utilizado por cada uno quedando **`/ackermann_steering_controller/odom`** y **`/world_odom_publisher/odom`** respectivamente. El tipo de mensaje utilizado se muestra a continuación:

```
nav_msgs/Odometry
      std_msgs/Header header
            uint32 seq
            time stamp
            string frame_id
      string child_frame_id
      geometry_msgs/PoseWithCovariance pose
            geometry_msgs/Pose pose
                  geometry_msgs/Point position
                        float64 x
                        float64 y
                        float64 z
                  geometry_msgs/Quaternion orientation
                        float64 x
                        float64 y
                        float64 z
                        float64 w
            float64[36] covariance
      geometry_msgs/TwistWithCovariance twist
            geometry_msgs/Twist twist
            float64[36] covariance
```

Dicho mensaje resulta un estándar para la odometría en ROS. Contiene en primer lugar, un número secuencial y un *timestamp* para ubicar el valor de odometría en tiempo. Luego, el mensaje muestra el valor de la *pose* (posición y orientación) del robot con respecto al marco de coordenadas especificado en **`header.frame_id`**, que en nuestro caso será un marco fijo en el entorno llamado **`odom`**, y el valor de *twist* (velocidad lineal y angular) del robot con respecto al marco de coordenadas especificado en **`child_frame_id`**, que en nuestro caso será el marco ubicado en el centro del eje trasero del robot llamado **`base_link_vis`**. Este mensaje es una representación genérica de la odometría para tres dimensiones. Por supuesto, al utilizar un robot que navega en el plano bidimensional, ciertos parámetros del mensaje tendrán siempre un valor nulo. Adicionalmente a

lo descrito, el mensaje posee una matriz de covarianza para indicar incertidumbre pero no será utilizada, por lo que también llevará valores nulos.

Con el **`world_odom_publisher`** se pretende, al eliminar los errores de odometría, poder abstraer del problema de navegación lo relacionado con la localización del robot. Esto se logrará en conjunto con la utilización del componente **`fake_localization`** como se explicará más adelante. De esta manera podremos enfocarnos únicamente en los problemas de planificación y ejecución de las trayectorias. Por otro lado, se podría también utilizar el **`world_odom_publisher`** para calcular el error de la estimación de odometría. Esto sería similar, en el escenario real, a comparar la información obtenida mediante el GPS RTK con el cálculo de odometría estimada obtenida de la combinación del resto de los sensores del robot. Sin embargo, este paso queda fuera del alcance de este trabajo.

Capítulo 4

Navegación y planificación de trayectorias

Como resultado de lo descrito en el capítulo anterior, se obtiene un modelo virtual funcional del robot desmalezador junto con su interfaz de control y diferentes escenarios de campos de cultivos en el simulador Gazebo. Este capítulo corresponde a la definición de los algoritmos para planificar las trayectorias que deberá seguir el robot y a la generación de los comandos necesarios para ejecutarlas, los cuales serán enviados al controlador para completar la tarea de navegación autónoma. Para lograr esto, el *framework* de ROS provee una serie de herramientas agrupadas en un conjunto de paquetes conocido como *navigation stack*. El *navigation stack* de ROS está compuesto por varios módulos diseñados para realizar las tareas de localización, mapeo y planificación de caminos tanto global como local. A continuación se describirán los paquetes que fueron utilizados y los nuevos desarrollos que se realizaron. En este capítulo se utilizará como referencia el diagrama de la Figura 4.1. Dicho diagrama se extiende con la parte de simulación del capítulo anterior y su versión completa puede verse en el apéndice A.

En resumen, el proceso de navegación implementado para el robot desmalezador se puede descomponer en la ejecución de los siguientes pasos:

1. En primer lugar, el nodo **`waypoint_server`** genera una serie de puntos intermedios ordenados, llamados *waypoints*, que el robot debe seguir durante su navegación para cubrir por completo el campo. Estos *waypoints* determinan a muy grandes rasgos por dónde circulará el robot, estableciendo básicamente el orden en que se deben recorrer las hileras de cultivos. Por un lado, este nodo publica todos los *waypoints* generados sobre un *topic* para su visualización en RViz y, por otro lado, ofrece un servicio para que puedan ser obtenidos por el nodo que se describe a continuación.

2. El nodo **`goal_publisher`** recibe los *waypoints*, cada uno compuesto por una posición y orientación deseada, y publica mensajes del tipo **`move_base_msgs/MoveBaseGoal`** de a uno por vez a través del protocolo del acciones de ROS. Una vez que el robot alcance un *waypoint* dentro de una cercanía configurable, se publica el siguiente. Cada *waypoint* corresponde al objetivo o meta del planificador global en el siguiente paso.

3. El planificador global, implementado en el nodo **`global_planner`**, genera un camino hacia el objetivo que no pase por los obstáculos utilizando el algoritmo de Dijkstra o el algoritmo A*, según esté configurado, pero sin tener en cuenta las dimensiones ni las características cinemáticas del robot, es decir, considerando al robot como un punto en el plano. Para realizar esta tarea, este nodo utiliza la información obtenida del mapa estático conocido de antemano que representa al campo con sus hileras de cultivos. Dicho mapa es proporcionado por el nodo **`map_server`**. Adicionalmente, recibe la ubicación actual del robot emitida por el nodo **`fake_localization`**.

4. Posteriormente, se pasa el camino generado por el planificador global al planificador local, implementado en el nodo **`local_planner`**, el cual hace los ajustes necesarios para adaptarla a las características y

dimensiones del robot desmalezador. Dicho nodo genera una trayectoria solamente para las cercanías del robot y no hasta el objetivo final. La trayectoria luego se irá recalculando a medida que el robot avance por la misma. A diferencia del anterior, este planificador no se basa en el mapa estático del campo sino que va construyendo un mapa más actualizado con la información obtenida a partir de los sensores que posea el robot. Dicha información es provista por el nodo **`point_cloud_obstacle`**. Al igual que el planificador global, también recibe del nodo **`fake_localization`** la ubicación actual del robot.

5. Finalmente, la trayectoria local generada es enviada por medio de comandos de velocidad lineal y angular al controlador del robot utilizando el *topic* **`/cmd_vel`**.

El resto de este capítulo detalla los pasos previamente indicados y los nodos desarrollados para el sistema de navegación autónoma del robot desmalezador.

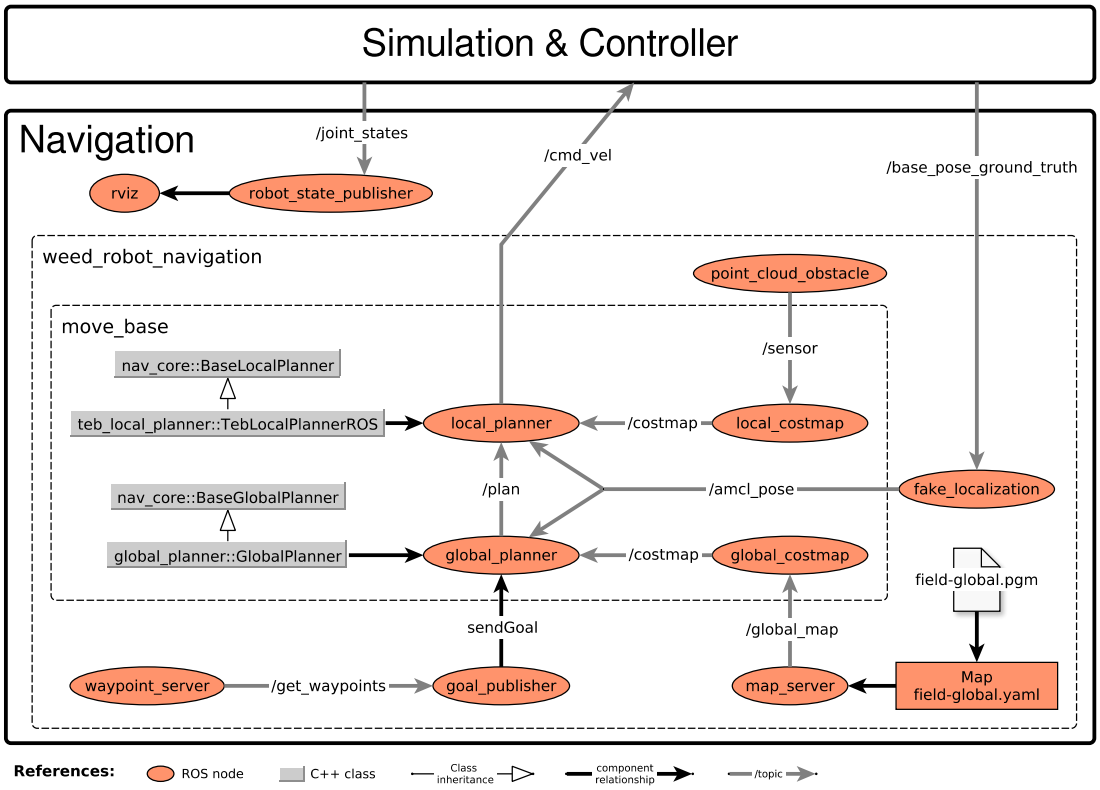


Figura 4.1: Diagrama de navegación en ROS para el robot desmalezador. En la parte inferior, el nodo **`waypoint_server`** genera los puntos intermedios que el robot deberá seguir para cubrir completamente el campo de cultivos. Dichos puntos se publican de a uno a la vez como objetivos para el conjunto **`move_base`**, compuesto por el planificador global y local, para finalmente enviar los comandos de control al robot simulado. Al mismo tiempo, se recibe *feedback* desde el simulador con la *pose* del robot para la localización y la ubicación de cada una de sus partes para la visualización en RViz.

4.1. Localización y mapeo

Antes de poder generar un plan de trayectorias para que ejecute el robot es necesario un modelo que represente el entorno donde realizará su navegación. A este modelo se lo conoce como mapa del entorno. Por otro lado, se requiere conocer la ubicación actual del robot en ese mapa y la ubicación del objetivo. Cuando hablamos de ubicación nos referimos tanto a la posición como a la orientación, lo cual se conoce como *pose*.

Existen diferentes formas de obtener estos elementos, por ejemplo, el mapa puede ser conocido de antemano y proporcionado por una fuente externa al robot o puede ser generado de forma incremental a medida que el robot recorre el entorno. Lo mismo sucede con la *pose* del robot, la cual puede ser proporcionada por un GPS de alta precisión o puede ser estimada a partir de los diferentes sensores como la cámara estéreo a medida que el robot navega el entorno.

4.1.1. Localización real desde el simulador

Los elementos utilizados comúnmente para obtener una primera estimación de la *pose* del robot son la odometría generada a partir de la información de las ruedas y dirección *(encoders)* y la unidad de medición inercial (IMU). Esta estimación inicial puede tener un error considerable, por lo cual se suele utilizar la información de los alrededores del robot obtenida por medio de los sensores como una cámara estéreo para corregir dicho error. En una configuración estándar del *navigation stack* de ROS, el método más utilizado para resolver la localización del robot es el llamado Adaptive Monte Carlo Localization [79], implementado en el paquete **`amcl`**[1]. Este método utiliza un filtro de partículas (*particle filter*), que consiste básicamente en lo siguiente. La *pose* desconocida del robot es representada por un conjunto de ejemplos llamados partículas, inicialmente distribuidas uniformemente por todo el entorno, si no se tiene información de la configuración inicial del robot. Luego, a medida que el robot se mueve por el entorno, las partículas son desplazadas consistentemente con movimiento del robot y se calcula la probabilidad para cada una de ellas de representar la *pose* real del robot basándose en el conocimiento previo del entorno y la información obtenida de los sensores. Finalmente, al repetir estos pasos varias veces, se espera que las partículas converjan a la *pose* real del robot y se reduzca el error de la *pose* estimada.

A pesar de ser una técnica muy utilizada, inevitablemente el empleo de cualquier sistema de localización que haga uso de mediciones que contengan error, introducirá un error en la localización del robot que afectará el resto de los componentes de navegación. Como este trabajo se enfoca únicamente en la planificación de trayectorias, se decidió utilizar la pose real del robot en lugar de una estimación. Esto elimina cualquier tipo de error que pueda darse en la generación de la trayectoria producto de una localización deficiente. Para lograr esto, se hizo uso del paquete provisto por ROS llamado **`fake_localization`**[2]. Dicho componente tiene como entrada la pose inicial del robot y se suscribe al *topic* que publica la odometría que se genera con información del estado actual del robot en el simulador, evitando todo tipo de error en la ubicación. Luego, publica la pose real del robot en el *topic* **`/amcl_pose`**, que es donde espera recibir dicha información el planificador global y local.

4.1.2. Construcción de mapas

En el *navigation stack* de ROS el modelo del entorno se representa con un mapa de dos dimensiones formado por una grilla regular. Para el planificador global, la grilla regular es generada a partir de la información de píxeles obtenida de una imagen de mapa de bits en escala de grises que representa la totalidad del espacio de navegación. En cambio, para el planificador local, la grilla es generada de manera *online* a partir de la información de los sensores del robot. Para completar la definición del espacio de configuraciones, se necesitará definir adicionalmente lo que se denomina como *footprint* o huella del robot. Con estos dos elementos, el mapa bidimensional y el *footprint* del robot, los planificadores podrán conocer cuales configuraciones estarán en colisión con los obstáculos y cuales formaran parte del espacio libre de configuraciones.

Se definirá un mapa y un *footprint* diferente para cada uno de los dos planificadores, los cuales estarán construidos por medio del paquete **`costmap_2d`**[3] a partir de diferentes fuentes de información. Los mapas estarán formados por una grilla de ocupación (*occupancy grid*). Cada celda de la grilla representará una porción puntual del entorno que quedará determinada por la resolución de la misma. A su vez, cada celda tendrá un valor que indicará la probabilidad de que la misma esté ocupada, lo cual es de utilidad a la hora de combinar información inexacta recibida desde diferentes sensores.

[1] Adaptive Monte Carlo Localization, http://wiki.ros.org/amcl.
[2] Fake localization, http://wiki.ros.org/fake_localization.
[3] 2D costmap, http://wiki.ros.org/costmap_2d.

El proceso de construcción del mapa consiste en lo siguiente. Cada celda del mapa puede tener un costo que va desde 0 a 255, donde 0 indica espacio libre y 255 indica colisión segura con un obstáculo. En primera instancia, se recibe la información de los sensores o de la imagen que representa el entorno, y se la convierte a una grilla que especifica para cada celda un valor de libre u ocupado. Luego, a cada celda con valor ocupado se le asigna el mayor costo 255 y se realiza un proceso de inflado (*inflation*) del obstáculo, el cual consiste en propagar dicho costo a las celdas cercanas con radios de inflado parametrizables como se describen a continuación. Primero, se define el espacio *inscribed* que consiste en las celdas a una distancia menor que el radio inscrito del robot. Por lo tanto, el robot estará en colisión segura si su centro se encuentra en alguna de las celdas del espacio *inscribed*, sin importar la orientación del mismo. En segundo lugar, se define el espacio *circunscribed* que consiste en las celdas a una distancia de obstáculo mayor a la anterior pero menor al radio circunscripto del robot. En este caso, si el centro del robot se encuentra en el espacio *circunscribed* dependerá de la orientación del mismo el estar o no en colisión con el obstáculo. Por último, se le asigna al resto de las celdas un peso decreciente a medida que se aleja del área de obstáculos hasta llegar a un peso de 0. El grado de disminución de este peso también es configurable mediante diferentes parámetros. Esto último, permite penalizar caminos que pasen cerca de los obstáculos sin impedir que sean considerados como factibles.

Para el caso del robot desmalezador, las hileras de cultivo de soja serán consideradas como los obstáculos el campo, puesto que no queremos que las mismas sean pisadas por las ruedas del robot. Sin embargo, el robot estará transitando la mayor parte del tiempo con dos hileras de cultivos pasando por debajo de su chasis. Debido a esto, no se puede describir el mapa y el *footprint* del robot de manera tradicional ya que esto impediría que el robot ingrese a una hilera para realizar su recorrido. En el resto de la sección, se describen las soluciones adoptadas para este problema en cada uno de los planificadores global y local.

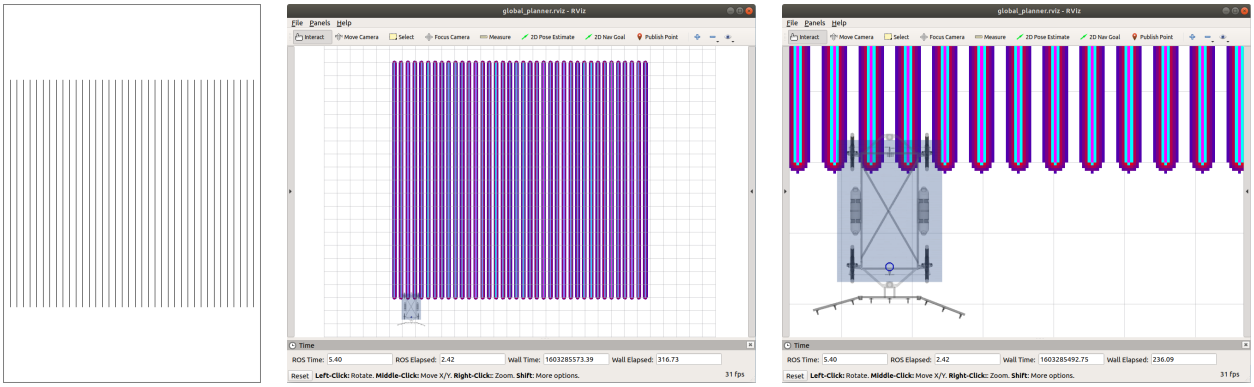
4.1.3. Mapa estático global

El mapa global se construye a partir de la información obtenida de una imagen de mapa de bits en escala de grises que representa la ubicación de las hileras de cultivos en el campo. Dicho mapa será estático, es decir, será proporcionado al conjunto **`move_base`** una única vez al iniciar el proceso de navegación y no será actualizado. Este mapa representa el conocimiento previo que se tiene del entorno que consiste en el campo de cultivos completo por donde navegará el robot. En la Figura 4.2a se puede ver el archivo de la imagen y en la Figura 4.2b el mapa generado. Para la generación de la imagen se desarrolló un pequeño programa que recibe los mismos parámetros que el programa de generación del escenario de la simulación, pero en este caso se crea una imagen de mapa de bits en formato PGM en lugar de un *world* en formato SDF. El mapa generado es publicado por el nodo **`map_server`** sobre el *topic* **`/global_map`**.

El planificador global no deberá preocuparse por las propiedades cinemáticas del robot y su plan generado será luego refinado por el planificador local. Debido a esto, se decidió describir el *footprint* del robot simplemente como un pequeño circulo ubicado en el centro de giro del robot. De esta manera, permitimos que se generen caminos que pasen por entre medio de cualquier par de hileras. Los caminos generados ubicarán al robot siempre en el centro de uno de los surcos, lo que permitirá que dos hileras de cultivos pasen por debajo del robot y ninguna de sus ruedas quede pisando las plantas. El *footprint* del robot puede visualizarse en la Figura 4.2c como un pequeño circulo azul en el centro del eje trasero del robot.

4.1.4. Mapa dinámico local

El mapa para el planificador local consistirá en una representación de las cercanías de la posición actual del robot y no de la totalidad del entorno de navegación. Esto permite el uso de una mayor resolución que en el mapa global sin desencadenar en un exceso de uso de memoria para su almacenamiento. A este tipo de mapa se lo conoce como *rolling window* ya que se lo puede ver como una ventana en las cercanías del robot que se va desplazando y actualizando a medida que el robot se mueve por el entorno de navegación. Para la construcción de este mapa se utiliza la información obtenida de los sensores del robot, en nuestro caso la cámara estéreo. La cámara estéreo deberá generar una nube de puntos la cual permitirá identificar las hileras de cultivos. Como una primera aproximación a este enfoque, se creó un nodo llamado **`point_cloud_generator`** el cual simula los mensajes de nube de puntos generados por la cámara. Queda como trabajo a futuro, el



(a) Imagen PGM. **(b)** Mapa global generado. **(c)** Acercamiento mapa global.

Figura 4.2: Mapa para el planificador global. El mapa es generado a partir de una imagen PGM como el que se muestra en **(a)**, que representa las ubicaciones de las hileras de cultivos en el campo. Las líneas negras representan los cultivos y los espacios en blanco entre ellas representan los surcos. En **(b)** se puede ver el mapa del tipo *grilla de ocupación* generado a partir de dicha imagen con los diferentes pesos en las celdas. En **(c)** se muestra un acercamiento de la anterior donde se puede distinguir mejor el desvanecido de los pesos desde el obstáculo hasta el espacio libre. Se puede ver también que el robot desmalezador pasará en gran medida por encima de los obstáculos pero sus ruedas deberán evitar colisionar con éstos. El *footprint* del robot se muestra como un pequeño circulo azul en el centro del eje trasero del mismo.

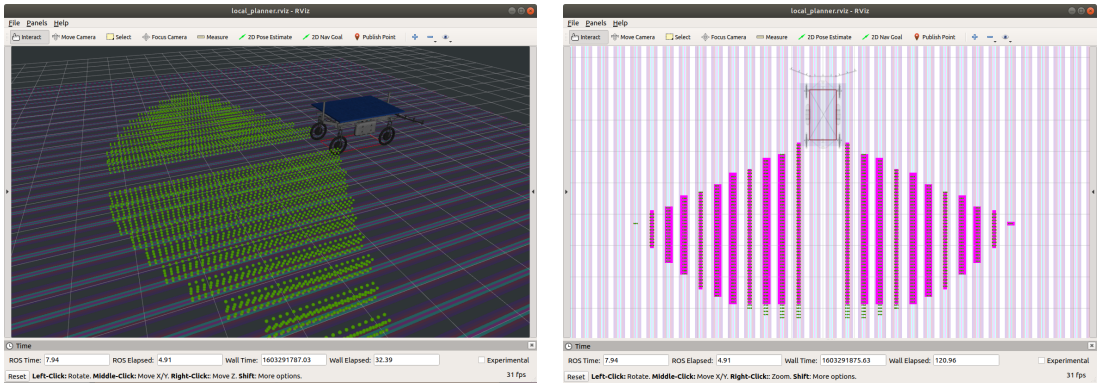
agregado de una cámara en el simulador Gazebo y la obtención de la nube de puntos a partir de ésta.

El *footprint* definido para este caso consistirá en el rectángulo formado por los puntos de apoyo en el suelo de las cuatro ruedas del robot desmalezador. Debido a que dos hileras de cultivos deberán pasar por el medio de este *footprint* se ideó el agregado de un filtro en la nube de puntos que elimina todos los puntos que puedan pasar por debajo del robot, para no considerarlos como obstáculos. Los puntos eliminados serán los que se ubican por debajo de una altura determinada y dentro del ancho de ruedas del robot, tomando como punto central el surco por el cual pasa camino del plan global. Cabe destacar, que ambas tareas de generación de nube de puntos y filtrado de los mismos se realiza en el mismo nodo **`point_cloud_generator`**. En la Figura 4.3a se puede ver la nube de puntos generada en verde, con el filtro de las dos hileras centrales ya aplicado. En la Figura 4.3b se muestra el mapa resultante con los obstáculos en color magenta y el *footprint* del robot como un rectángulo de color rojo.

4.2. Planificador de trayectorias

Una vez obtenido el modelo del entorno de navegación y la localización del robot en el mismo, podemos pasar a la etapa de la planificación de trayectorias. Como se mencionó anteriormente, esta tarea se divide en dos partes: un planificador global que no tendrá en cuenta las dimensiones del robot ni sus restricciones dinámicas y cinemáticas, y un planificador local que irá ajustando el plan global en las cercanías del robot teniendo en consideración dichas particularidades. El planificador local podrá adaptarse a la aparición de obstáculos desconocidos previamente que sean detectados por la cámara del robot a medida que realiza su navegación. El *navigation stack* de ROS define dos interfaces para los planificadores global y local llamadas **`nav_core::BaseGlobalPlanner`** y **`nav_core::BaseLocalPlanner`**, respectivamente. Cualquier implementación de planificador que cumpla con dichas interfaces puede ser acoplado al resto de los componentes de la navegación sin mayores dificultades.

Actualmente, existen diferentes implementaciones de planificador local para usar con el *navigation stack* de ROS, entre las cuales podemos nombrar a *Dynamic Window Approach* [40], *Elastic Band* [41] y *Timed Elastic Band* [42, 43] como las más populares. Estos planificadores utilizan distintas técnicas para la generación



(a) Nube de puntos. **(b)** Mapa local generado.

Figura 4.3: Mapa para el planificador local. En la imagen **(a)** se muestra la nube de puntos en verde obtenida por el sensor de cámara simulado. Esta nube de puntos representa los lugares en donde se encuentran las plantas de soja. Dos hileras de cultivos en el centro del robot son filtradas para que el planificador no los considere como obstáculos. En **(b)** se ve el mapa local generado con los obstáculos en rosa y el *footprint* del robot como un rectángulo rojo.

de trayectorias y algunos son más adecuados para utilizar con un tipo de robot específico que con otro. En este trabajo se utiliza el *Timed Elastic Band* (TEB) debido, principalmente, a que soporta de origen vehículos del tipo *car-like* con radio de giro limitado. Por el contrario, para la planificación global existe una implementación estándar utilizada muy ampliamente. Esta implementación se encuentra en el paquete llamado **`global_planner`**[4] y será el utilizado en este trabajo. A continuación de describen los detalles de ambos planificadores.

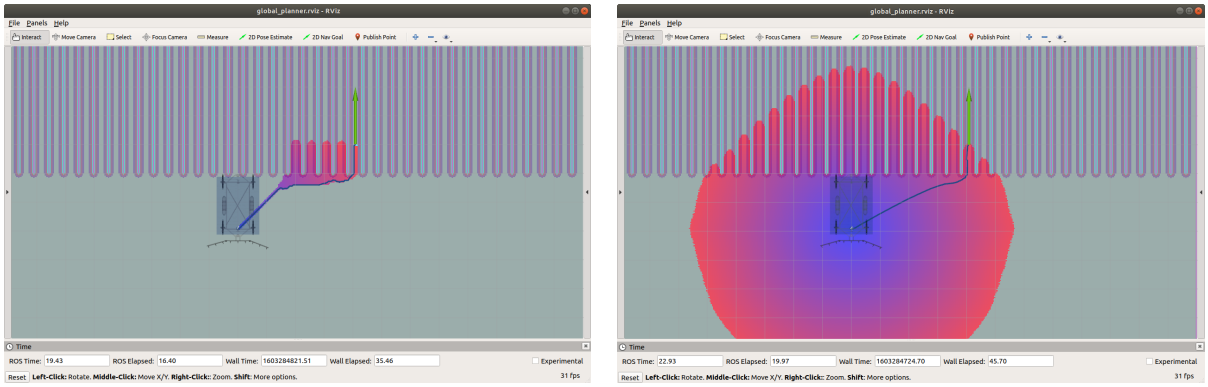
4.2.1. Planificador global

Para el planificador global se utiliza el paquete **`global_planner`**. Esta implementación permite elegir entre el uso de los algoritmos A* y Dijkstra mediante un parámetro de configuración. Se realizaron pruebas con ambos algoritmos y no se obtuvieron diferencias significativas en la práctica. El método Dijkstra explora más celdas que el A*, como puede apreciarse en la Figura 4.4. Sin embargo, esto tampoco produce una diferencia notable en el rendimiento final, ya que el mayor costo computacional de todo el proceso de navegación está determinado por el planificador local. Por otro lado, los caminos generados por el método de Dijkstra también suelen ser apenas un poco más cortas que las generadas por el algoritmo A*, lo cual tampoco tiene relevancia ya que ambas son ajustadas por el planificador local produciendo el mismo resultado de trayectoria final. Como se mencionó anteriormente, este planificador utiliza el mapa global provisto por el **`global_costmap`** construido a partir de la imagen PGM. Una vez que el camino fue generado, se la publica en el *topic* **`/plan`** para que lo reciba el planificador local.

4.2.2. Planificador local

El planificador local será el encargado de construir un plan de navegación para las cercanías del robot que tenga en cuenta sus características particulares y las del entorno, para posteriormente generar los comandos de velocidades que serán enviados directamente al controlador del robot. En este trabajo, se utiliza como planificador local el método Timed Elastic Band (TEB), cuya implementación se encuentra en el paquete **`teb_local_planner`**. Dicho planificador realiza una optimización de una porción del camino obtenido desde el planificador global teniendo en consideración restricciones dinámicas y cinemáticas del robot, la evasión de obstáculos y la minimización del tiempo y distancia del recorrido para alcanzar el objetivo.

[4]Global planner, http://wiki.ros.org/global_planner.



(a) Algoritmo A*. **(b)** Algoritmo de Dijkstra.

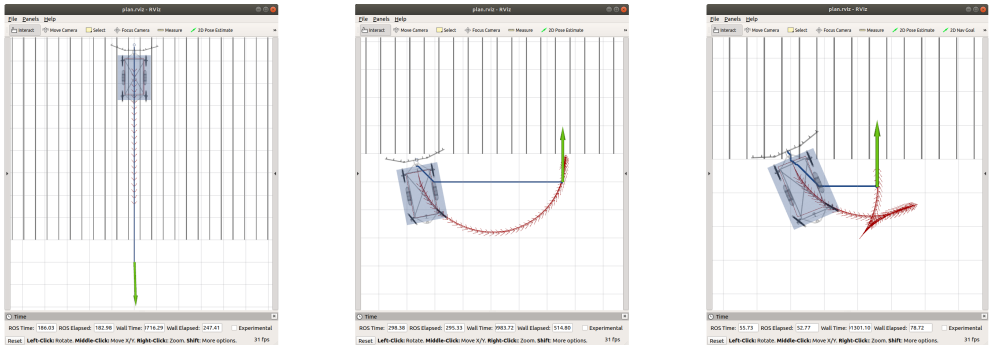
Figura 4.4: Planificador global. El plan global se puede ver como una línea azul que va desde el centro del eje trasero del robot hasta el objetivo marcado con una flecha verde. El área coloreada del mapa indica las celdas que fueron analizadas por el planificador. El degradado de colores representa el campo potencial del entorno que se torna más azul a medida que se aleja de los obstáculos y se acerca al punto de partida. Se ve claramente que el planificador A* en **(a)** analiza menos celdas que el de Dijkstra en **(b)**, sacrificando algo de calidad en el camino generado.

En la Figura 4.5, se pueden ver algunas de las trayectorias generadas por este planificador local para el robot desmalezador. La configuración objetivo, formada por la posición y orientación final deseadas, se muestra con una flecha verde. El camino correspondiente al planificador global se muestran con una línea azul y el correspondiente al planificador local como una serie de flechas rojas. Podemos notar que los caminos global y local se diferencian en varios aspectos. En primer lugar, el camino global no indica la orientación en la que el robot debe seguirlo y solo busca la forma más directa de alcanzar el objetivo considerando al vehículo simplemente como un punto en el plano. Por el contrario, la trayectoria local tiene en consideración la dinámica y cinemática del robot por lo que además de la posición indica la orientación que tendrá el vehículo en cada paso de la trayectoria. Por momentos esto hace que el vehículo se aleje del objetivo para después poder alcanzarlo respetando su radio mínimo de giro. En la captura de la Figura 4.5c podemos ver también la utilización de reversa en el plan local resultante. Por otro lado, en la captura de la Figura 4.5a podemos ver que el plan local, como su nombre lo indica, solo se genera para las cercanías del robot y no llega hasta el objetivo final. Por el contrario, el plan global siempre va desde la posición actual del robot hasta el objetivo final.

Como se explicó en la Sección 2.2.2.1, el planificador TEB utiliza una técnica de métodos de penalización para la optimización de la función objetivo, lo que produce una serie de parámetros a configurar relacionados principalmente con los pesos que tendrá cada término de la función a optimizar. A continuación, se describen los parámetros más relevantes y los valores asignados para la simulación del robot desmalezador.

4.2.2.1. Parametrización

Ajuste de la función a optimizar (penalización) El parámetro **`penalty_epsilon`** agrega un pequeño margen de seguridad a todas las restricciones de desigualdad, como puede verse en las ecuaciones de la Sección 2.2.2.1. Los parámetros **`weight_max_vel_x`**, **`weight_max_vel_theta`**, **`weight_acc_lim_x`** y **`weight_acc_lim_theta`** ajustan el peso que tendrán los límites de velocidad y aceleración tanto lineal como angular. El parámetro **`weight_kinematics_nh`** determinará el peso que se le da a las restricciones no holonómicas del robot. Ya que el robot desmalezador es un vehículo no holonómico se deberá dar un valor grande a este peso para que los comandos enviados cumplan con las cinemática del robot. Por otro lado, el parámetro **`weight_kinematics_turning_radius`** indica la importancia que se le dará al cumplimiento del radio mínimo de giro. Cabe destacar que si la trayectoria generada no cumple estrictamente con dicho radio no habrá



(a) Trayectoria en línea recta. **(b)** Giro en cabecera sin cruce. **(c)** Giro en cabecera con cruce.

Figura 4.5: Diferentes planes de trayectorias creados por el planificador local TEB. El objetivo final se muestra con una flecha verde, el camino global con una línea azul y la trayectoria local con una serie de pequeñas flechas rojas. En **(a)** se aprecia que el plan local solo se genera para las cercanías del robot y no hasta el objetivo final. En el resto de las figuras se muestran las trayectorias que se generan cuando el robot termina de recorrer ciertas hileras de cultivos del campo y se dispone a realizar las maniobras necesarias para posicionarse a recorrer otras. Para realizar el giro mostrado en **(b)** el robot no necesitará utilizar reversa. Por el contrario, para poder ejecutar la trayectoria mostrada en **(c)** sí necesitará reversa.

mayores inconvenientes ya que se va recalculando todo el tiempo a medida que el robot realiza el recorrido. El parámetro **weight_kinematics_forward_drive** se utiliza para indicar al robot que priorice trayectorias en dirección hacia adelante. En el caso del robot desmalezador la conducción hacia adelante es preferible especialmente durante el recorrido de las hileras pero se debe permitir reversa para la realización de los giros en las cabeceras del campo. Los parámetros **weight_shortest_path** y **weight_optimaltime** ajustan el objetivo entre obtener la trayectoria de menor longitud o la que lleve menor tiempo de ejecución, respectivamente. Notar que no siempre la trayectoria de menor longitud puede realizarse en el menor tiempo posible ya que si la misma posee muchos giros o alterna entre manejo hacia adelante y atrás, el robot deberá disminuir su velocidad para ejecutarla. Por último, los parámetros **weight_obstacle** y **weight_inflation** indican la importancia que se le dará a la evasión de obstáculos. En nuestro caso, un valor alto en el primer parámetro permite que el robot circule en línea recta al recorrer las hileras de cultivos, pero deberá darse un valor pequeño al segundo debido a que el corredor entre dichos obstáculos resulta muy angosto y el robot tendrá que circular muy cerca de los obstáculos inflados. Valores altos en el último parámetro pueden impedir que el robot entre en los corredores entre las hileras de cultivos. A continuación se muestran los valores configurados:

```
penalty_epsilon: 0.06
weight_max_vel_x: 1
weight_max_vel_theta: 1
weight_acc_lim_x: 1
weight_acc_lim_theta: 1
weight_kinematics_nh: 100
weight_kinematics_forward_drive: 1
weight_kinematics_turning_radius: 1.5
weight_optimaltime: 1
weight_shortest_path: 0
weight_obstacle: 1
weight_inflation: 0.1
```

Grado de optimización Con los siguientes parámetros se configura la precisión del proceso de optimización pero hay que tener en cuenta que a mayor precisión más costo computacional. Por un lado, el parámetro **dt_ref** se utilizan para determinar la resolución temporal de la trayectoria en segundos, la cual podrá tener

un rango de ajuste de más-menos **dt_hysteresis**. Por otro lado, el parámetro **no_outer_iterations** indica la cantidad veces que se resolverá el problema de optimización de la ecuación 2.16. En la primer iteración se tomará como aproximación inicial la trayectoria obtenida del planificador global y en las subsiguientes iteraciones la trayectoria de la iteración anterior. Luego el parámetro **no_inner_iterations** indica el número de iteraciones del algoritmo de optimización interno que resuelve dicho problema.

```
dt_ref: 0.45
dt_hysteresis: 0.1
no_outer_iterations: 8
no_inner_iterations: 4
```

Tolerancia al objetivo Con los siguientes parámetros se especifica la tolerancia absoluta para considerar que el robot ha llegado a su objetivo definida en metros y radianes. Adicionalmente, se configura si el robot debe alcanzar el objetivo deteniéndose a velocidad cero o puede hacerlo en conducción con velocidad distinta de cero.

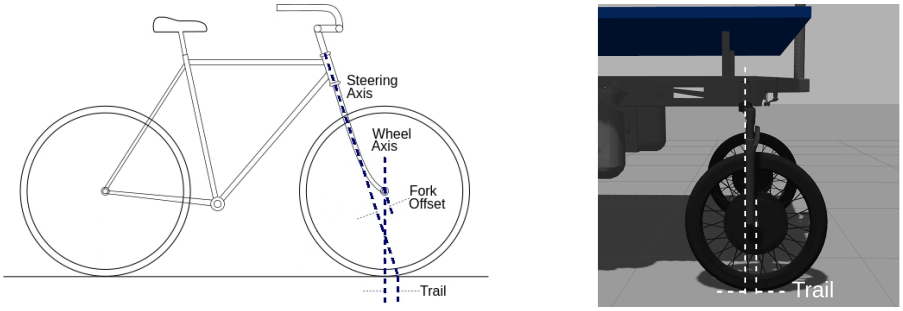
```
xy_goal_tolerance: 0.25
yaw_goal_tolerance: 0.15
free_goal_vel: true
```

Propiedades del robot Los siguientes parámetros describen las propiedades del robot como el radio mínimo de giro, el *wheelbase* o batalla del robot, y los límites de velocidad y aceleración tanto lineal como angular. Notar que se configura una velocidad nula en dirección del eje y puesto que el robot desmalezador no puede moverse lateralmente. Se configura una velocidad en reversa algo menor que la velocidad hacia adelante para que el método priorice generar trayectorias que hagan que el robot circule hacia adelante. Las unidades son: m para las distancias, m/s y rad/s para las velocidades, y m^2/s y rad^2/s para las aceleraciones. Notar que se configura un radio mínimo de giro levemente mayor al que puede realizar el robot para aumentar la posibilidad de que pueda ejecutar las trayectorias generadas con la menor cantidad de recálculos de trayectoria posibles. En cuanto a la velocidad lineal máxima, el robot está diseñado para alcanzar hasta 4km/h, lo cual daría alrededor de $1,11$m/s. Sin embargo, en esta configuración se tuvo que definir un valor levemente inferior debido a que con velocidades mayores no se lograban buenos resultados debido principalmente al tiempo que le lleva al algoritmo recalcular las trayectoria.

```
min_turning_radius: 2.5
wheelbase: 1.582
max_vel_x: 0.8
max_vel_x_backwards: 0.4
max_vel_y: 0.0
max_vel_theta: 0.3
acc_lim_x: 0.08
acc_lim_theta: 0.1
```

Otros parámetros El parámetro **max_global_plan_lookahead_dist** indica la distancia en metros máxima del camino global que se optimizará. Si el objetivo está a menor distancia se optimizará todo el camino hasta el objetivo. Luego de realizada la optimización y construido el plan local, se toma un cierto número de *poses* de la trayectoria generada, indicado por el parámetro **feasibility_check_no_poses**, y se verifica que dichas *poses* no entren en colisión con ningún obstáculo. En caso de que se detecte colisión, se ajustan automáticamente ciertos parámetros como el rango temporal y se vuelve a recalcular.

```
max_global_plan_lookahead_dist: 7
feasibility_check_no_poses: 4
```



(a) Trail positivo en una bicicleta. **(b)** Trail negativo en el robot desmalezador.

Figura 4.6: Disposición de horquilla y rueda en diferentes vehículos. Se conoce como *trail* a la distancia entre los puntos de intersección con el suelo del eje de la rueda y del eje de dirección. Un *trail* será positivo si la intersección del eje de dirección está por delante de la del eje de la rueda **(a)**. Esto le proporciona mayor estabilidad al vehículo produciendo una fuerza que tiende a centrar la rueda por si sola durante la conducción hacia adelante. Por el contrario, un *trail* negativo como el del robot desmalezador **(b)**, donde el eje de dirección se encuentra por detrás del eje de la rueda, hace que la conducción hacia adelante sea menos estable.

4.2.2.2. Conducción hacia adelante

Como se mencionó previamente, se da prioridad a la conducción hacia adelante ya que el robot necesita recorrer las hileras de cultivos en ese sentido debido a que posee sus sensores y rociadores configurados para trabajar de esta manera. Es común ver aplicaciones en las que el planificador de trayectorias se configura para preferir la conducción hacia adelante también debido a que el robot suele tener mayor estabilidad en dicho sentido de circulación. Por el contrario, esto es al revés para el robot desmalezador producto de la disposición de las horquillas y las ruedas. Como puede verse en la Figura 4.6b el eje de rotación de la horquilla, que corresponde con el eje de dirección, se encuentra por detrás del eje de rotación de la rueda cuando el robot conduce hacia adelante, por lo que el robot tendría menor estabilidad en ese sentido que cuando conduce en reversa. De todos modos, esta distancia conocida como *trail* es relativamente pequeña y los resultados de las pruebas indican que es lo suficientemente estable para la conducción en ambas direcciones.

En una configuración normal de un vehículo de dos ruedas como la bicicleta, al estar la horquilla en un ángulo inclinado hacia atrás se produce un *trail* positivo como se ve en la Figura 4.6a. Esto es deseable ya que produce una fuerza de estabilización que hace que la rueda tienda a centrarse por si sola durante la conducción, el mismo efecto que se produce con las ruedas de los carritos de supermercados o las sillas de escritorio. Sin embargo, un *trail* muy grande tampoco es deseable. Habitualmente, las horquillas añaden un *offset* con el que adelantan el eje de la rueda para disminuir un poco dicho *trail*. En el robot desmalezador las horquillas están ubicadas de forma vertical por lo que este *offset* hace que el *trail* sea negativo como se puede ver en la Figura 4.6b, es decir, que la intersección del eje de la rueda con el suelo quede por delante de la del eje de dirección. Las horquillas fueron colocadas en este sentido en el robot desmalezador por cuestiones de diseño.

4.3. Algoritmo de cobertura

Para realizar su tarea de desmalezado, el robot debe recorrer el campo de forma completa. Dicho campo estará formado por hileras de cultivos de soja intercaladas por surcos. Se desea que las ruedas del robot estén siempre sobre los surcos para que no pisen el cultivo. El robot realizará los recorridos en la dirección de los surcos desde un extremo al otro del campo, para luego realizar un giro en la cabecera del mismo y posicionarse con las ruedas en otro par de surcos para recorrer a continuación. En cada recorrido el robot cubrirá con sus rociadores cinco surcos y cuatro hileras de cultivos, como se puede apreciar en la Figura 4.7. El centro

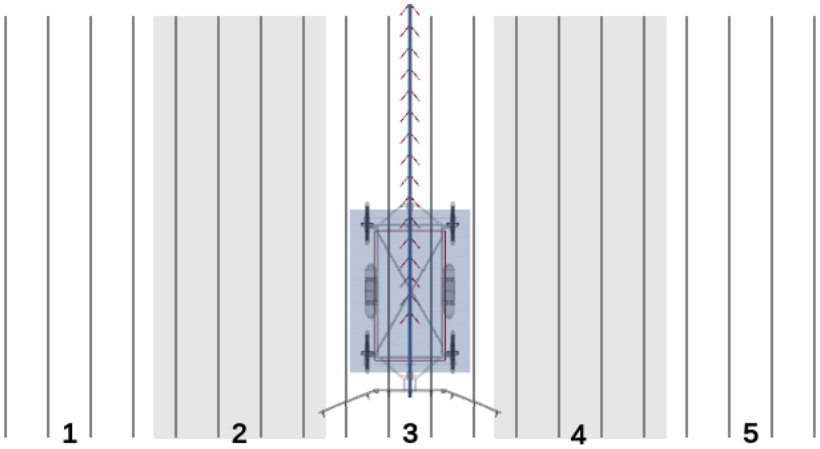


Figura 4.7: Subdivisión del campo de cultivos en franjas. Una franja abarca el espacio que el robot puede cubrir en un recorrido en línea recta desde el extremo inferior al extremo superior del campo. Cada franja está compuesta por 4 hileras de cultivos.

del robot deberá estar siempre alineado con el centro de un surco para que las ruedas se ubiquen en los dos surcos de los lados y no se pisen los cultivos. De esta manera, dos hileras de cultivos pasarán por debajo del robot y podrá cubrir con sus rociadores una más a cada uno de sus lados. Debido a que la detección de las malezas es menos precisa en los sitios más alejados al centro del robot donde se ubica la cámara, se decidió que los surcos de los extremos sean barridos dos veces por los rociadores. Salvo por esa excepción, se busca que ninguna otra área sea barrida más de una vez por los rociadores del robot. Como una simplificación, podemos pensar que el robot cubre hasta la mitad de los surcos de los extremos, y que entonces su objetivo sea que no se solape ningún área cubierta. De esta manera, podemos numerar los sectores cubiertos por cada recorrido del robot como en la Figura 4.7. A cada una de estas porciones se las conoce en la literatura como franjas (*tracks*). Como los campos suelen ser relativamente extensos con muchos surcos y líneas de cultivos, consideraremos para el análisis siempre una cantidad entera de franjas.

Se desarrolló un nodo en ROS llamado **`waypoint_server`** que genera una serie de subobjetivos para el planificador global siguiendo diferentes secuencias de franjas. Como se verá en el Capítulo 5, se realizaron pruebas con diferentes secuencias. Inicialmente se probó una secuencia trivial de franjas consecutivas $\langle 1, 2, 3, 4, 5, 6, 7, 8, \dots \rangle$. Con dicha secuencia, el robot deberá realizar giros bastantes cerrados en las cabeceras del campo, menores al radio mínimo de giro, por lo que deberá usar reversa o *abrirse* para poder completar el giro y posicionarse en la siguiente franja. Para mejorar este escenario se probaron secuencias que tienen mayor distancia entre surcos como por ejemplo $\langle 1, 4, 7, 3, 6, 2, 5, 8, \dots \rangle$. Esta secuencia tiene saltos de no menos de tres franjas, por lo que los giros pueden realizarse todos, en principio, sin necesidad de maniobras de reversa. De esta forma el recorrido resulta más eficiente.

Los objetivos generados son transmitidos al planificador global por el nodo **`goal_publisher`** de a uno por vez con el uso de la librería de acciones de ROS. El mensaje utilizado para los objetivos es del tipo **`move_base_msgs/MoveBaseGoal`**. Dicho mensaje contiene la posición y orientación deseada con respecto al marco de coordenadas fijo del mapa del entorno.

```
move_base_msgs/MoveBaseGoal
      geometry_msgs/PoseStamped target_pose
            std_msgs/Header header
            geometry_msgs/Pose pose
```

El nodo genera un objetivo al inicio y al final de cada surco que se quiera recorrer. Una vez que el robot se encuentra dentro de un rango en las cercanías del objetivo anterior, el próximo objetivo es enviado al planificador global. El rango de cercanía se configura con una tolerancia de posición y orientación de forma

independiente. Adicionalmente, al nodo generador de objetivos se le debe especificar la secuencia de surcos a seguir, si iniciar la secuencia en la parte inferior o superior del campo, las dimensiones del campo, las dimensiones de las cabeceras, cantidad de hileras de cultivos y cantidad de surcos.

Capítulo 5

Experimentación y resultados

A partir del entorno de simulación desarrollado, se diseñaron diferentes experimentos que permiten evaluar la precisión, robustez y eficiencia de los métodos involucrados en la navegación del robot desmalezador. Todas las trayectorias generadas por el planificador local TEB son evaluadas de forma *offline* con la información recolectada en la ejecución de los experimentos. En todos los casos, el hardware utilizado para el procesamiento de los experimentos corresponde a una computadora laptop Intel(R) Core(TM) i7-1065G7 de 1.30GHz cuatro núcleos, 8GB de RAM y tarjeta gráfica Nvidia GeForce MX230.

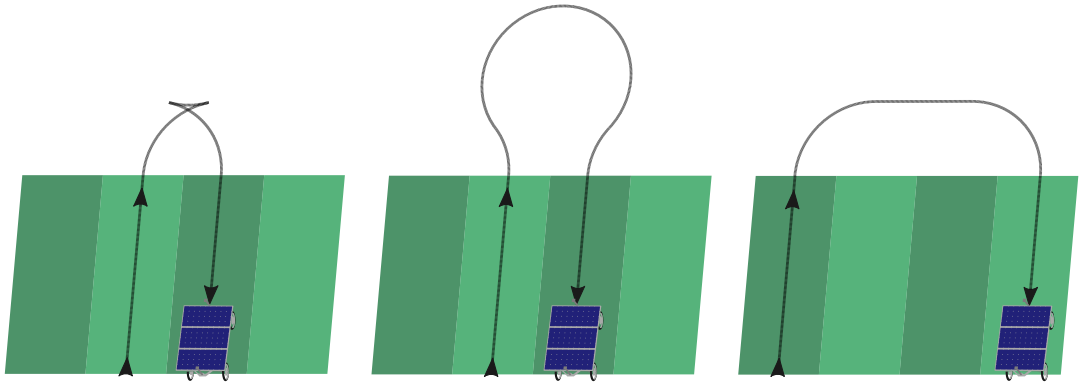
5.1. Elección de la secuencia de franjas

Básicamente, la navegación por el campo agrícola se realiza en dos etapas que se repiten con alternancia: una consiste en el recorrido en línea recta por una franja y la otra consiste en la realización de un giro en la cabecera del campo para posicionarse en la franja que se recorrerá a continuación. Como se explicó en la sección 4.3, el robot desmalezador cubre con sus rociadores una porción del campo compuesta por una cierta cantidad de surcos y líneas de cultivos simultáneamente en cada uno de sus recorridos entre los extremos del campo. A cada una de estas porciones de campo las llamaremos franjas y las numeraremos en forma consecutiva de izquierda a derecha, con el fin de poder identificarlas fácilmente.

En primera instancia, se podría pensar que la forma más eficiente de recorrer estas franjas sería con la secuencia $\langle 1, 2, 3, 4, 5, 6, 7, 8, \dots \rangle$, que consiste en elegir siempre la franja inmediata posterior como la siguiente a recorrer hasta llegar al final del campo. Dicha secuencia la denominaremos secuencia trivial. Sin embargo, debido a que el radio mínimo de giro del robot es mayor a la distancia entre dos franjas consecutivas, el robot no podrá realizar un giro directo hacia la siguiente franja y deberá utilizar reversa o *abrirse* para poder lograrlo. En esta sección se busca determinar cuál es el orden más eficiente en que se deben recorrer dichas franjas.

Para lograr esto, el primer paso consiste en determinar el costo de realizar los diferentes giros entre las franjas. En los trabajos [80, 81] se identifican diferentes tipos de giros, como se muestran en la Figura 5.1 y se presentan las fórmulas matemáticas para calcular la longitud de cada uno de ellos. En dichos trabajos, estas distancias se toman como costos para luego calcular la secuencia más eficiente. En nuestro caso, los costos serán determinados por los valores de duración de los recorridos obtenidos de los experimentos.

El giro Π (ver Figura 5.1c) solo puede realizarse si la distancia d entre las franjas inicial y final cumple con la restricción $d \geq 2\rho_{\min}$, donde $\rho_{\min}$ es el radio mínimo de giro del robot. Caso contrario, si $d < 2\rho_{\min}$, se tiene que utilizar alguno de los otros dos giros. Para poder realizar el giro T (ver Figura 5.1a) el robot debe poder conducir en reversa. Por otro lado, el giro Ω (ver Figura 5.1b) necesita mayor distancia en la cabecera del campo.



(a) Giro tipo T. **(b)** Giro tipo Ω. **(c)** Giro tipo Π.

Figura 5.1: Tipos de giros. Para ejecutar un giro T como se muestra en **(a)** el robot debe contar con reversa. Por el contrario, en los giros mostrados en **(b)** y **(c)**, conocidos como Ω y Π respectivamente, el robot siempre conduce en dirección hacia adelante. Sin embargo, para que el robot pueda realizar un giro Π la distancia entre los centros de las franjas inicial y final debe ser mayor o igual que dos veces su radio mínimo de giro.

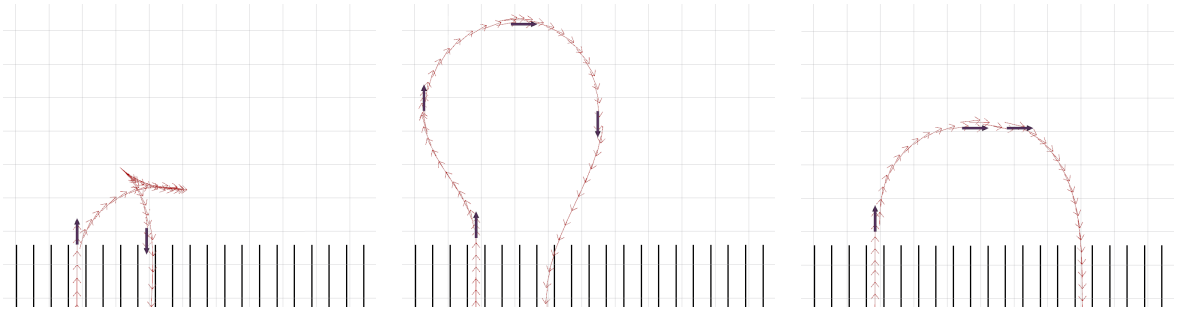
5.2. Evaluación de giros

La forma exacta en la que el robot realiza su giro para posicionarse sobre la siguiente franja a recorrer depende exclusivamente de la trayectoria generada por el planificador local TEB, la cual a su vez depende de los objetivos suministrados por el nodo **`waypoint_server`**. Para lograr los tipos de giro indicados previamente se necesitaron agregar varios subojetivos como se muestra en la Figura 5.2. Para el giro T, solo se necesitó un objetivo al final de la franja y otro al inicio de la siguiente (ver Figura 5.2a). Para lograr el giro Ω, se necesitaron agregar cuatro subobjetivos intermedios como se ve en la Figura 5.2b. Por último para el giro Π, se usaron tres objetivos: uno en la franja inicial y los otros dos a una distancia de $\rho_{\min}$ de la franja inicial y final (ver Figura 5.2c). De esta manera, se obtienen resultados relativamente similares en cuanto a las formas de las trayectorias generadas a lo largo de las distintas pruebas.

Como se mencionó anteriormente, una franja está compuesta por 4 hileras de cultivos y comprende el área que cubre el robot con sus rociadores en un recorrido. Cada hilera de cultivo, incluyendo el sector de plantación y medio surco a cada lado, tiene un ancho de 0.52 m. Esto da como resultado un ancho de franja de 2.08 m. Teniendo en cuenta que el radio de giro mínimo del robot desmalezador es de alrededor de 2.4 m, tenemos que un giro de tipo Π puede hacerse solo recién con un salto de 3 franjas. Para los giros con saltos de 1 y 2 franjas se deberán usar los tipos T u Ω. De aquí en adelante, vamos a notar a los giros con la letra correspondiente a su tipo seguida de un subíndice que indique el número de saltos de franjas. Dicho de otra manera, el subíndice marcará la distancia entre el centro de la franja inicial y la final, en términos de cantidad de franjas. De esta manera, un giro T con reversa que va desde una franja a la inmediata siguiente se notará con T_1, y uno sin el uso de reversa se notará con Ω_1. De la misma forma, un giro que va por ejemplo desde la franja 1 hasta la franja 4, se notará con Π_3.

Se analizaron 8 giros distintos: $\mathrm{T}_1, \mathrm{T}_2, \Omega_1, \Omega_2, \Pi_3, \Pi_4, \Pi_5, \Pi_6$. Por cada giro, se ejecutaron 30 pruebas con la simulación del robot desmalezador y se recolectaron los datos de duración y longitud del recorrido. En la Figura 5.3 se muestran los resultados obtenidos en términos de longitud, duración y velocidad de los recorridos para los diferentes tipos de giro utilizando gráficos de *boxplot*. La velocidad que se toma en cada prueba, corresponde a la velocidad promedio de la totalidad del giro.

Con respecto a la longitud del recorrido, podemos ver que los giros más cortos son el T_1 y el T_2, seguido por el giro Π_3 con muy poca diferencia. Como era de esperarse, los giros de tipo Π aumentan su longitud de recorrido de forma lineal a medida que se aumenta el número de saltos de franjas. El giro Ω_1 es el que posee mayor longitud en su recorrido. En todos los tipos de giro, la variabilidad de las longitudes de los recorridos es relativamente pequeña a lo largo de todas las pruebas, con un rango intercuartil de al rededor de 30 cm



(a) Sub-objetivos para giro T. **(b)** Sub-objetivos para giro Ω. **(c)** Sub-objetivos para giro Π.

Figura 5.2: Distribución de sub-objetivos para la obtención de los diferentes tipos de giros. Las imágenes se crearon a partir de la combinación de varias capturas de pantallas, una por cada sub-objetivo y una más para el objetivo final en el otro extremo del campo que no se ve en la imagen. Las flechas violetas muestran los sub-objetivos y la serie de flechas rojas forman la trayectoria generada por el planificador local para dichos objetivos. Las líneas verticales negras indican los sectores de las plantas de soja.

o menos. Esto se debe principalmente a la utilización de los sub-objetivos intermedios que ayudan a generar todas trayectorias similares para un mismo tipo de giro.

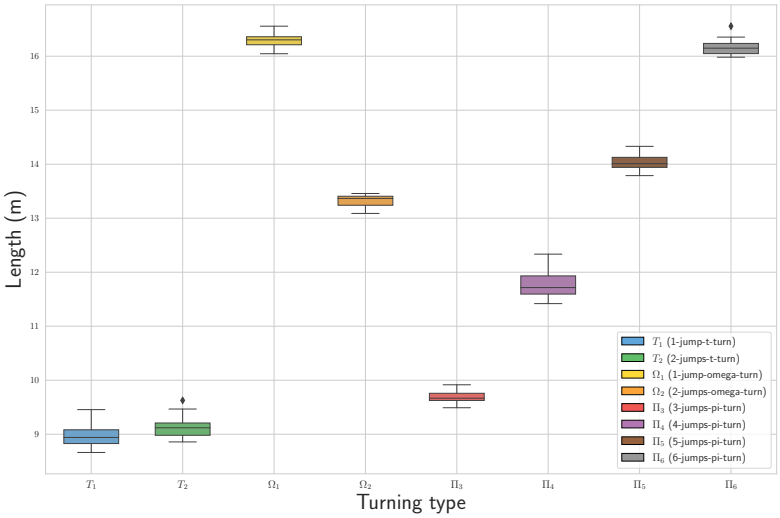
Si nos quedamos con los datos de longitud del recorrido, podríamos pensar que los giros de tipo T son los más eficientes, pero si analizamos la información de la duración de los recorridos vemos que los giros Π_3 y Π_4 tienen un desempeño considerablemente mejor que los anteriores. Recién el giro Π_4 posee un tiempo de duración similar a un giro T_2. Esto trae como resultado que las secuencias de franjas óptimas tendrán saltos de 3 y 4 franjas, en lugar de la secuencia trivial con todos saltos de a 1 franja, si tomamos como criterio de eficiencia el tiempo de duración del recorrido. Con respecto a la variabilidad de las duraciones de los recorridos, vemos que también es relativamente pequeña. Encontramos que las más variables son los giros de tipo T y Ω, con un rango intercuartil del orden de los 1.2 s. Los giros de tipo Π, en cambio, resultaron más estables en este sentido con un rango intercuartil de alrededor de 0.4 s.

En cuanto a las velocidades de los giros, podemos destacar que los giros de tipo T poseen menor velocidad promedio. Esto es debido a que el robot debe disminuir la velocidad hasta detenerse dos veces en cada giro para cambiar el sentido de conducción a reversa y de nuevo hacia adelante. Los giros de tipo Ω tienen velocidad levemente menor a los del tipo Π porque el recorrido es mayormente curvo y el robot debe disminuir un poco su velocidad mientras gira. Podemos apreciar también en los giros Π que a medida que mayor parte del recorrido es recto la velocidad promedio se acerca cada vez más a la máxima permitida que es de 0.8 m/s.

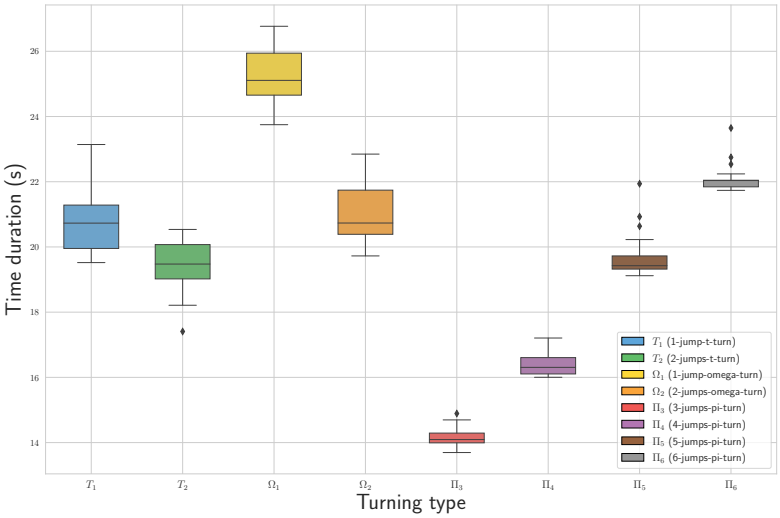
En la Tabla 5.1 se muestra a modo de resumen los valores de las medianas para los diferentes giros en términos de longitud, duración y velocidad. Por otro lado, la Figura 5.4 muestra las trayectorias realizadas por el robot desmalezador para cada uno de los giros. Para realizar este gráfico se tomó la ejecución que corresponde con la mediana de cada uno de los giros.

Tipo de giro	T_1	T_2	Ω_1	Ω_2	Π_3	Π_4	Π_5	Π_6
Longitud (m)	**8,941**	9,119	16,303	13,367	9,667	11,716	14,010	16,149
Duración (s)	20,730	19,475	25,108	20,733	**14,093**	16,307	19,425	22,036
Velocidad (m/s)	0,437	0,468	0,648	0,642	0,687	0,718	0,725	**0,735**

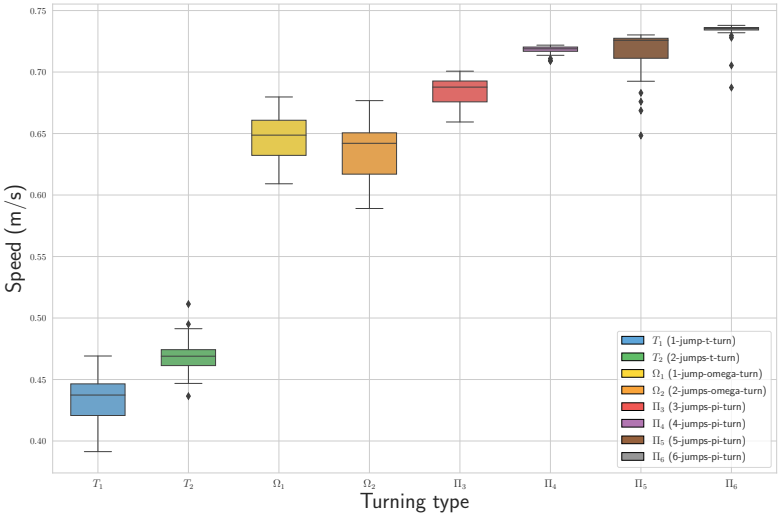
Tabla 5.1: Resultados de las medianas en longitud, duración y velocidad promedio de los recorridos para 30 ejecuciones en los 8 tipos de giros analizados.



(a) Comparación de recorridos en longitud.



(b) Comparación de recorridos en duración.



(c) Comparación de recorridos en velocidad promedio.

Figura 5.3: Gráficos generados a partir de 30 ejecuciones para cada tipo de giro.

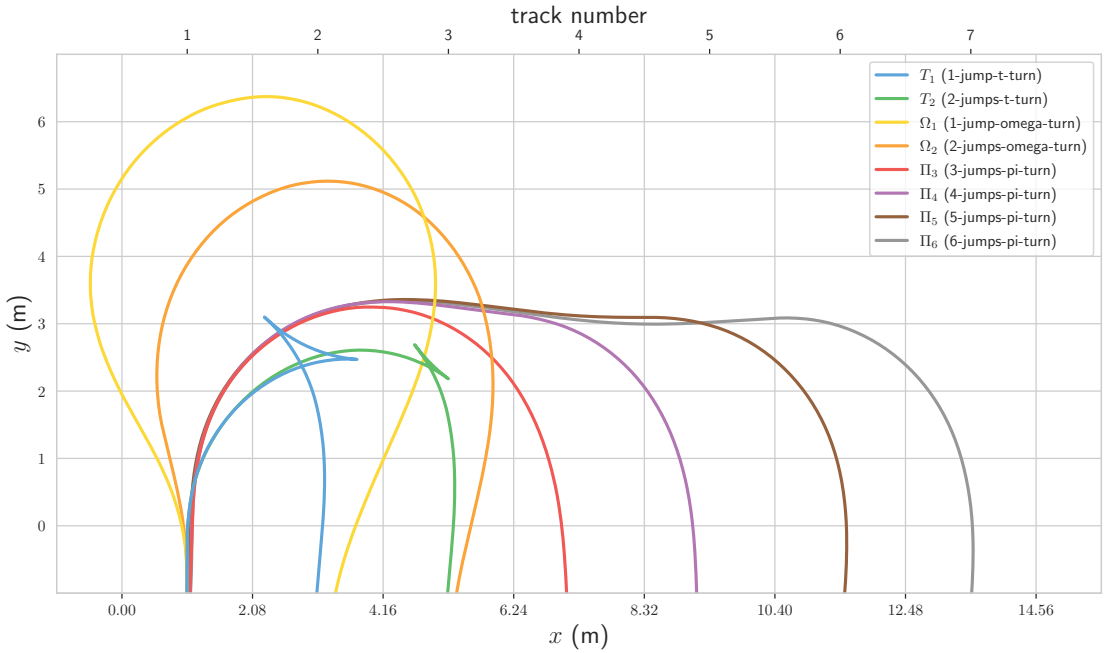


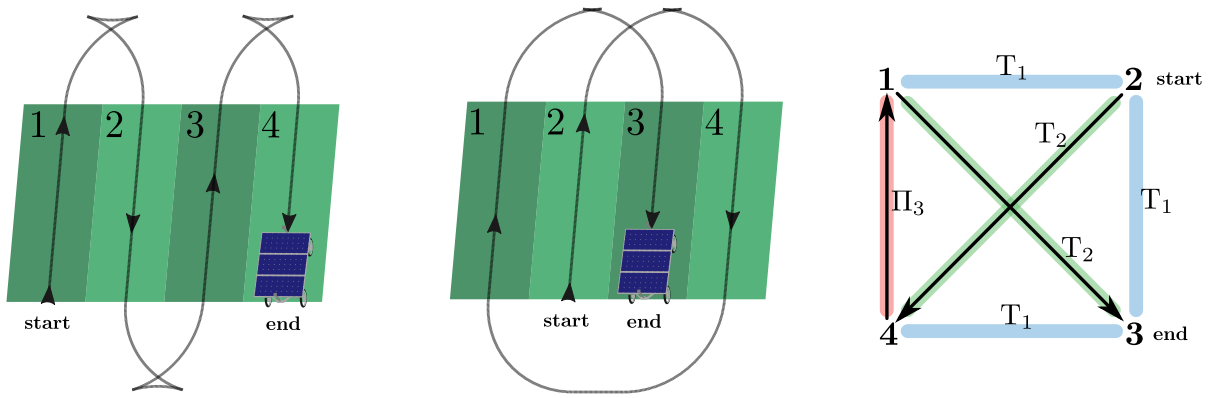
Figura 5.4: Trayectorias realizadas por el robot desmalezador para los diferentes tipos de giros. En la parte superior se muestra el número de franja. Notar que cada franja tiene un ancho de 2.08 m.

Con estos resultados podemos concluir que una secuencia de recorrido de franjas que realice varios saltos de a 3 franjas será más eficiente que una secuencia con todos saltos de a 1 sola franja. En la siguiente sección se verá de qué manera se puede obtener una secuencia de recorrido que cubra la totalidad del campo y aproveche los datos obtenidos de este análisis para minimizar el tiempo total empleado.

5.3. Búsqueda de trayectoria óptima

Para obtener la secuencia de recorrido de franjas que minimice el tiempo total necesario para que el robot cubra el campo completamente, podemos plantear el problema utilizando una representación en forma de grafo $G = (V, E)$ donde el conjunto de vértices o nodos V estará formado por las franjas del campo y el conjunto de aristas E representará los giros en las cabeceras para pasar de una franja a otra. Este grafo, tendrá algunas particularidades que detallamos a continuación. Primero, cabe destacar que será un grafo completo, es decir, cada par de vértices estará conectado por una arista. Esto se debe a que el robot podrá alcanzar por la cabecera del campo cualquier otra franja una vez que termine de recorrer una en particular. Por otro lado, será un grafo ponderado. A cada arista se le asignará como costo el tiempo que tarda el robot en pasar de una franja a otra. Tomaremos como costos los valores de las medianas que obtuvimos en el experimento de la sección anterior, correspondientes a la segunda fila de la Tabla 5.1. Por último, cabe destacar que se tratará de un grafo no dirigido puesto que resultará indistinto en términos de costos pasar, por ejemplo desde la franja 1 a la franja 3 que a la inversa (de la 3 a la 1).

La asignación de costos a las aristas quedará determinada por la distancia entre las franjas involucradas. Por ejemplo, el costo para ir desde la franja 1 a la franja 4 será el mismo que el de ir desde la franja 2 a la 5, y corresponderá con el costo del giro Π_3. Diremos que este giro realiza un salto de 3 franjas. Para saltos de 1 y 2 franjas tenemos dos tipos de giro: el T y el Ω. En nuestro análisis elegimos utilizar solo los giros de tipo T y descartamos los giros Ω, por ser los primeros considerablemente más eficientes y ya que el robot desmalezador, al poder funcionar en reversa, no tendrá problema en ejecutarlos. Por otro lado, debemos notar que el análisis de los tiempos de recorridos se realizó solo hasta saltos de 6 franjas, con un giro Π_6. De todos



(a) Trayectoria trivial. **(b)** Trayectoria óptima. **(c)** Grafo de representación.

Figura 5.5: Representación en forma de grafo y búsqueda de la secuencia óptima para un campo de 4 franjas. En **(a)** se muestra la secuencia trivial que consiste en comenzar en la franja de más a la izquierda y elegir siempre la inmediata posterior como la próxima a recorrer hasta terminar en la franja de más a la derecha. La figura **(b)** muestra la trayectoria óptima que se obtiene luego de resolver el problema de camino más corto representado por el grafo **(c)**.

modos, podemos asumir que un giro con un número mayor de saltos de franjas llevará más tiempo que el de 6 franjas. Como el objetivo es minimizar el costo total, asignaremos a los saltos mayores de 6 el mismo costo que el Π_6. Luego, si la secuencia óptima no tiene saltos de este tipo, sabremos que también resultará la óptima para los costos reales, que suponemos deben ser mayores a los asignados.

Una vez obtenido este grafo, el objetivo se reduce a encontrar el camino de menor costo posible que pase por todos los nodos del mismo, sin visitar más de una vez cada nodo. El requisito de no visitar más de una vez cada nodo, se desprende de evitar que el robot pase más de una vez por las mismas hileras de cultivos. Este problema es equivalente al famoso problema del viajante [82] (TSP por sus siglas en inglés, *Travelling Salesman Problem*). El TSP es un problema de tipo NP-difícil, lo que implica que el costo computacional requerido para resolverlo aumentará drásticamente con la dimensión del problema. Su nombre se origina de una interpretación práctica en la que un vendedor viajante necesita encontrar el camino más corto para visitar un determinado número de clientes o ciudades. El TSP es uno de los problemas de optimización más ampliamente estudiados y desafiantes a pesar de su simple definición.

En la Figura 5.5c se puede ver un ejemplo del grafo resultante para un campo de 4 franjas. Vemos que en este caso la solución óptima consiste en la secuencia $\langle 2, 4, 1, 3\rangle$ de la Figura 5.5b. Esta secuencia está compuesta por dos saltos tipo T_2 y un salto tipo Π_3. Dicha secuencia es más eficiente que la secuencia trivial $\langle 1, 2, 3, 4\rangle$ de la Figura 5.5a, que tiene todos saltos tipo T_1, ya que tanto el giro T_2 como el Π_3 tienen menor tiempo de ejecución que el giro T_1.

5.3.1. Resolución del problema TSP

Una forma de resolver el problema de TSP consiste en probar todas las secuencias posibles, calcular el costo de cada una y quedarnos con la de menor costo. Sea n el número de franjas en el campo, el conjunto de todas las secuencias posibles consiste en todas las permutaciones del conjunto $[1..n]$. La complejidad del algoritmo de fuerza bruta resulta entonces $\mathcal{O}(n!)$. En la práctica, solo resultó factible calcular la solución para un campo de hasta 12 franjas.

Un mejor enfoque para resolver este problema de forma exacta, consiste en utilizar técnicas de programación dinámica. Con estas técnicas se evita el desperdicio de tiempo recalculando soluciones óptimas a subproblemas que ya han sido resueltos, almacenando dichas soluciones para su posterior uso. Entonces, si

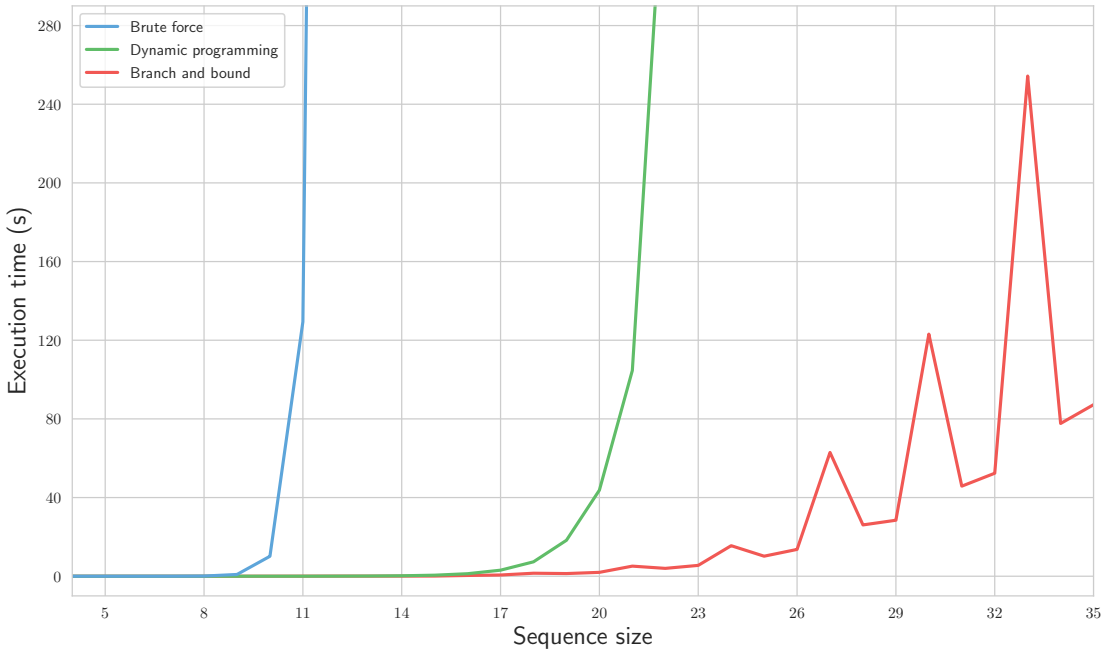


Figura 5.6: Comparación de los tiempos de ejecución de los algoritmos utilizados para encontrar la secuencia de franjas óptima.

necesitamos resolver el mismo problema más tarde, podemos obtener la solución de la lista de soluciones ya calculadas y reutilizarla. Esta técnica se conoce también como *memoización*. El algoritmo para resolver el problema de TSP utilizando este enfoque lleva el nombre de Bellman–Held–Karp por sus autores y tiene una complejidad computacional de $\mathcal{O}\left(n^2 2^n\right)$. Con dicho algoritmo se pudo encontrar las soluciones óptimas para campos de hasta 23 franjas. El límite en este caso fue el consumo de memoria del algoritmo.

Por último, tenemos el enfoque conocido como *branch and bound* o algoritmo de ramificación y poda. Este algoritmo se basa en la búsqueda de la solución óptima explorando el espacio de soluciones representado como un árbol. Cada rama del árbol llevará a una posible solución final. Todas las posibles soluciones corresponderán a las hojas de dicho árbol. La técnica de *branch and bound* se encarga de detectar anticipadamente qué ramas del árbol ya no podrán conducir a una solución óptima para podarla y, de esta manera, ahorrar recursos al no continuar la búsqueda por dicho camino. La complejidad del peor caso para este algoritmo continúa siendo $\mathcal{O}\left(n!\right)$ como el de fuerza bruta, ya que podría darse el caso de nunca encontrarse con la oportunidad de podar una rama. Sin embargo, para nuestro caso práctico resultó el más eficiente de los tres, pudiendo resolver campos de hasta 35 franjas. Esto se debe principalmente a las propiedades particulares del problema, como por ejemplo que tenemos solo 6 costos diferentes de aristas.

En el gráfico de la Figura 5.6 se pueden apreciar los tiempos de ejecución de los diferentes algoritmos mencionados anteriormente y la Tabla 5.2 muestra los resultados obtenidos hasta 35 franjas. Podemos ver que la duración promedio por giro disminuye hacia el valor correspondiente a la duración del giro Π_3 que es 14.093 s. Esto se debe a que las secuencias tienden a utilizar todos giros de tipo Π_3 excepto por dos. Hasta 15 franjas vemos que hay secuencias optimas que utilizan solo giros de tipo Π_3 y Π_4, con lo que el recorrido completo se podría realizar sin utilizar reversa. Para campos con un número de franjas mayor a 15, vemos que existen dos patrones de secuencias óptimas. Si el número de franjas es múltiplo de 3, la secuencia óptima utiliza un giro T_1, un giro T_2 y el resto de tipo Π_3. Por otro lado, si el número de franjas no es múltiplo de 3, la secuencia óptima utiliza dos giros de tipo T_2 y el resto de tipo Π_3. La duración promedio por giro resulta un poco mayor en los casos que el número de franjas es múltiplo de 3 puesto que el giro T_1 tiene mayor duración que el T_2. Esto produce los picos que se ven en el gráfico de la Figura 5.6 para el caso del *branch and bound*, debido a que al tener un óptimo mayor existen menos posibilidades de podas prematuras

de ramas en su proceso de búsqueda.

Franjas	Nro. de saltos				Duración	Promedio por	Tiempo de
	1	2	3	4	total (s)	franja (s)	resolución (min:s)
4		2	1		53,043	17,681	00:00,0001
5	1		2	1	65,223	16,305	00:00,0002
6			3	2	74,893	14,978	00:00,0002
7			4	2	88,986	14,831	00:00,0002
8			5	2	103,079	14,725	00:00,0004
9	1		6	1	121,595	15,199	00:00,0036
10		2	7		137,601	15,289	00:00,0151
11		2	8		151,694	15,169	00:00,0280
12	1		8	2	166,088	15,098	00:00,0579
13			8	4	177,972	14,831	00:00,0440
14			9	4	192,065	14,774	00:00,0566
15			10	4	206,158	14,725	00:00,0817
16		2	13		222,159	14,810	00:00,3828
17		2	14		236,252	14,765	00:00,6169
18	1	1	15		251,600	14,800	00:01,5238
19		2	16		264,438	14,691	00:01,3538
20		2	17		278,531	14,659	00:01,9666
21	1	1	18		293,879	14,693	00:05,1294
22		2	19		306,717	14,605	00:03,9899
23		2	20		320,810	14,582	00:05,5100
24	1	1	21		336,158	14,615	00:15,5260
25		2	22		348,996	14,541	00:10,1875
26		2	23		363,089	14,523	00:13,6265
27	1	1	24		378,437	14,555	01:02,9374
28		2	25		391,275	14,491	00:26,0864
29		2	26		405,368	14,477	00:28,4648
30	1	1	27		420,716	14,507	02:03,1033
31		2	28		433,554	14,451	00:45,7961
32		2	29		447,647	14,440	00:52,3620
33	1	1	30		462,995	14,468	04:14,3354
34		2	31		475,833	14,419	01:17,6765
35		2	32		489,926	14,409	01:27,2260

Tabla 5.2: Resultados de la búsqueda de secuencia óptima de recorrido de franjas. La primer columna indica la cantidad de franjas del campo. La columna compuesta nombrada como saltos, indica la cantidad que se utilizan de cada tipo de giro en la secuencia óptima resultante. Notar que en ninguna secuencia óptima se utilizaron giros con más de 4 saltos. La última columna muestra los tiempos de ejecución del algoritmo *branch and bound* para obtener dicha solución.

5.3.2. Patrón de recorrido utilizando giros Π_3

Del análisis anterior observamos que para campos con muchas franjas, más de 15, las secuencias óptimas están compuestas por todos giros de tipo Π_3, excepto por dos giros de tipo T. Podemos idear un patrón de recorrido de este estilo que sirva para cualquier número de franjas. La idea general consiste en lo siguiente. El robot comienza el recorrido del campo en el extremo izquierdo y ejecuta giros Π_3 hasta llegar al extremo derecho. Luego, realiza un giro de tipo T y vuelve hacia el extremo izquierdo nuevamente con giros Π_3. En este extremo también realiza un giro T y una vez más regresa con giros Π_3 al extremo derecho. En cada pasada a lo ancho del campo el robot estará recorriendo las franjas cuyo resto (o módulo) de dividir el número de

la franja en 3 resulte 0, 1, o 2. Exactamente en qué franja del extremo izquierdo comenzar el recorrido y qué tipo de saltos T se deben realizar en ambos extremos dependerá del número total de franjas del campo, como se muestra en la Figura 5.7. Por ejemplo, si el número de franjas del campo es múltiplo de 3, tenemos que primero se recorren las franjas cuyo módulo 3 dan 1 de izquierda a derecha $\langle 1, 4, 7, 10, \dots \rangle$, luego las que dan 0 de derecha a izquierda $\langle \dots, 12, 9, 6, 3 \rangle$ y finalmente las que dan 2 nuevamente de izquierda a derecha $\langle 2, 5, 8, 11, \dots \rangle$. En los extremos izquierdo y derecho del campo, para pasar de una sub-secuencia a la otra se deben realizar giros T con saltos de a 1 o 2 franjas según el caso. La secuencia para un campo de 12 franjas quedaría: $\langle 1, 4, 7, 10, 12, 9, 6, 3, 2, 5, 8, 11 \rangle$.

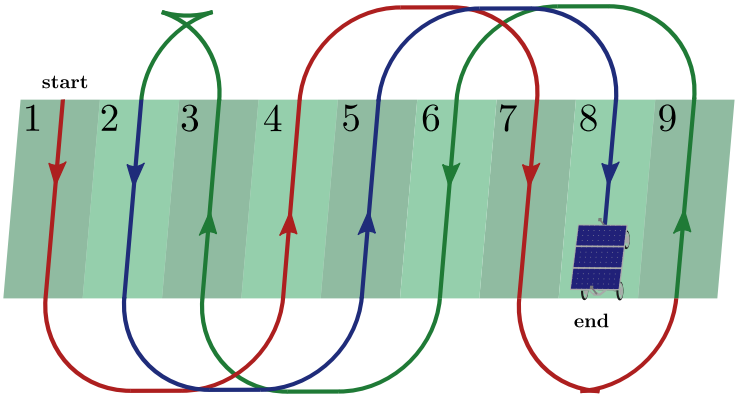
5.4. Caso de estudio: campo de 15 franjas

El patrón de recorrido del campo presentado más arriba que utiliza casi todos giros Π_3 y solo dos giros de tipo T, posee algunas características que podrían no ser las deseadas. En primer lugar, notamos que la cobertura total del campo se realiza en 3 etapas, cada una de las cuales cubre franjas repartidas por todo el ancho del campo. Dicho con otras palabras, el robot va y viene por todo lo ancho del campo varias veces hasta terminar de cubrirlo completamente. Esto puede ser problemático, por ejemplo, si tenemos otro vehículo con distinto ancho de cobertura que quiera recorrer el campo de forma simultánea con el robot desmalezador. Con este patrón, no habrá sectores del campo que queden completamente cubiertos por el robot desmalezador hasta que no se comience con la última de las 3 etapas. En segundo lugar, al utilizar dos giros con saltos de a 1 y 2 franjas, el patrón necesita tener reversa si se utilizan giros T, o bien el campo debe poseer un mayor tamaño de cabeceras si se decide utilizar giros Ω.

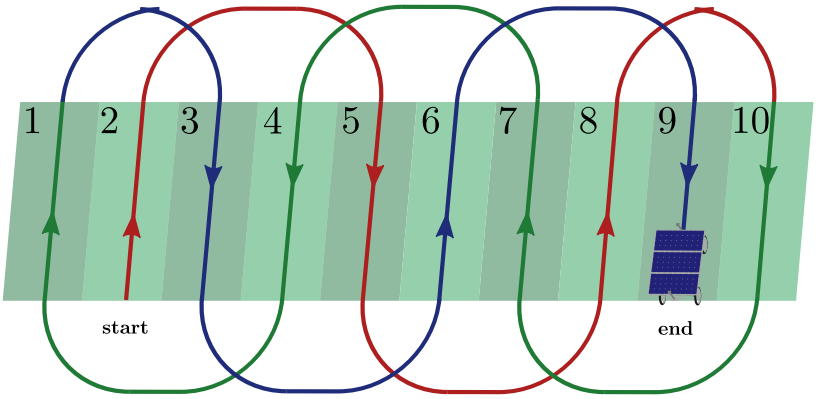
Otro patrón interesante, que no posee los inconvenientes antes mencionados, consiste en utilizar la secuencia óptima que se obtiene para 8 franjas y repetir dicha secuencia la cantidad de veces que sean necesarias hasta cubrir todo el ancho del campo, $\langle 1, 4, 7, 3, 6, 2, 5, 8 \rangle$. Dicha secuencia comienza en la franja de más a la izquierda y finaliza en la de más a la derecha, cubriendo todas las intermedias, por lo que fácilmente se puede repetir para cubrir campos más anchos si el número de franjas es múltiplo de 7 más 1, por ejemplo 8, 15, 22, 29, etc. Este patrón tiene las ventajas de que solo utiliza giros de tipo Π, evitando la necesidad de reversa, y recorre el campo por porciones de a 7 franjas de izquierda a derecha, por lo que otras máquinas podrían comenzar a trabajar en esos sectores a medida que son dejados por el robot desmalezador. Esta cobertura del campo de a tramos facilita también la pausa y reanudación en el recorrido total del campo si fuese necesario.

Para los experimentos que se muestran a continuación, tomamos un campo compuesto por 15 franjas y probamos la cobertura con el simulador utilizando 4 recorridos diferentes. Los recorridos de la Figura 5.8a y la Figura 5.8b muestran una secuencia trivial en la que se comienza de la franja de más a la izquierda y se elije siempre la franja inmediata de la derecha, usando giros de tipo Ω_1 y T_1 respectivamente. Por otro lado, la Figura 5.8c muestra la secuencia siguiendo el patrón visto en la sección anterior que utiliza casi todos giros Π_3 y solo dos giros de tipo T y, por último, la Figura 5.8d muestra la secuencia óptima para 15 franjas que corresponde al patrón que no utiliza reversa y recorre el campo por porciones de a 7 franjas.

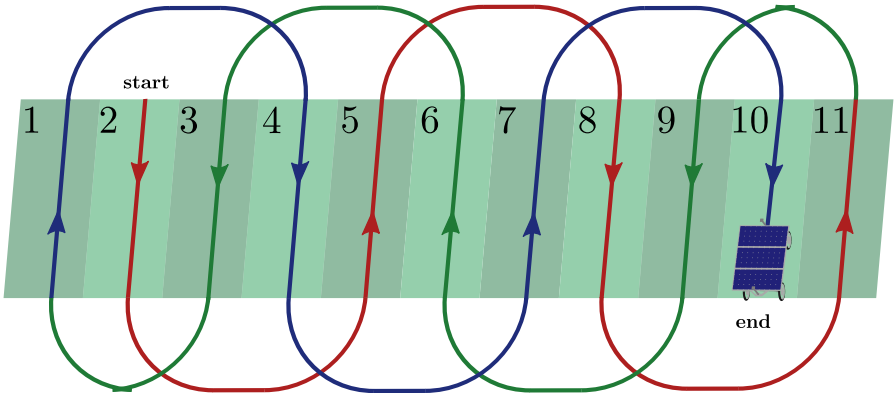
Para cada una de las pruebas, se tomaron las medidas de la longitud y duración total del recorrido. Dichos datos se muestran en la Tabla 5.3. Podemos observar que con el recorrido óptimo se obtiene un ahorro de casi el 20 % del tiempo total que con el recorrido trivial de giros Ω. Por otra parte, con el objetivo de comprobar la eficacia del planificador en términos de evitar el pisado de los cultivos, se realizaron mediciones acerca de cuán lejos pasa la trayectoria real del robot del centro de las franjas. Para tomar estas medidas, se emplearon métricas comúnmente usadas en SLAM [83, 84], llamadas ATE y RPE. En los problemas de SLAM, estas métricas comparan la trayectoria estimada por el método de SLAM contra la trayectoria real del robot, también conocido como *ground truth*. En nuestro caso, comparamos la trayectoria real hecha por el robot con la trayectoria ideal deseada, que será la que transita por el centro de la franja. Es decir, lo que tradicionalmente es la trayectoria estimada por el método de SLAM, en nuestro caso será la trayectoria real del robot, a la cual llamaremos trayectoria estudiada. Por otro lado, el *ground truth* contra el cual se realiza la comparación, en nuestro caso será la trayectoria ideal que va por el centro de las franjas, a la cual llamaremos trayectoria de referencia. Para realizar estas mediciones, ambas trayectorias a comparar deben estar sincronizadas en el tiempo.



(a) Recorrido para un campo con un número de franjas n tal que $n \bmod 3 = 0$.
Comenzando en la franja 1, recorre las franjas f tales que $f \bmod 3 = 1$ con giros Π_3 de menor a mayor (rojo). Realiza un giro T_2 desde la franja $n-2$ a la n. Recorre las franjas f tales que $f \bmod 3 = 0$ con giros Π_3 de mayor a menor (verde). Realiza un giro T_1 desde la franja 3 a la 2. Recorre las franjas f tales que $f \bmod 3 = 2$ con giros Π_3 de menor a mayor (azul).

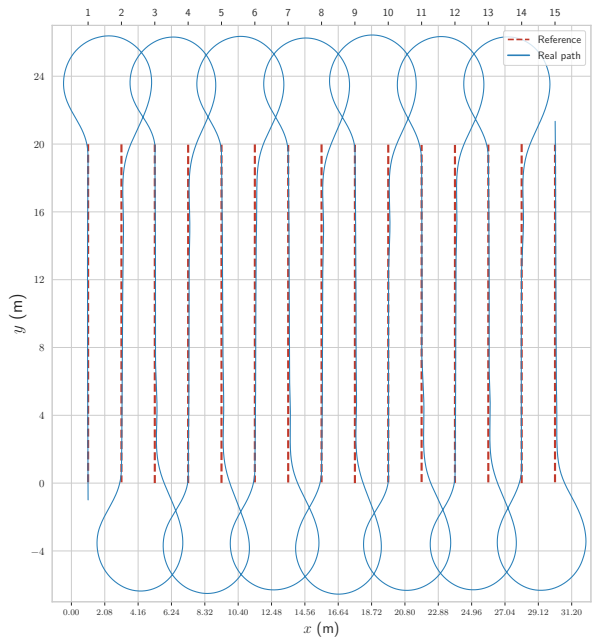


(b) Recorrido para un campo con un número de franjas n tal que $n \bmod 3 = 1$.
Comenzando en la franja 2, recorre las franjas f tales que $f \bmod 3 = 2$ con giros Π_3 de menor a mayor (rojo). Realiza un giro T_2 desde la franja $n-2$ a la n. Recorre las franjas f tales que $f \bmod 3 = 1$ con giros Π_3 de mayor a menor (verde). Realiza un giro T_2 desde la franja 1 a la 3. Recorre las franjas f tales que $f \bmod 3 = 0$ con giros Π_3 de menor a mayor (azul).

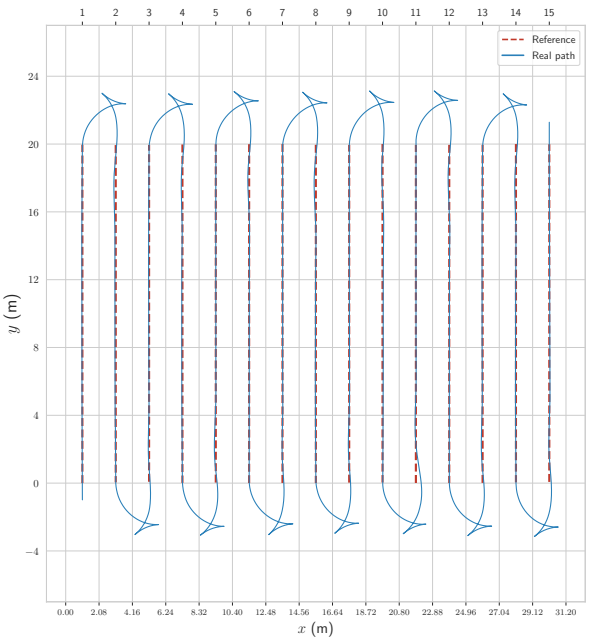


(c) Recorrido para un campo con un número de franjas n tal que $n \bmod 3 = 2$.
Comenzando en la franja 2, recorre las franjas f tales que $f \bmod 3 = 2$ con giros Π_3 de menor a mayor (rojo). Realiza un giro T_2 desde la franja n a la $n-2$. Recorre las franjas f tales que $f \bmod 3 = 0$ con giros Π_3 de mayor a menor (verde). Realiza un giro T_2 desde la franja 3 a la 1. Recorre las franjas f tales que $f \bmod 3 = 1$ con giros Π_3 de menor a mayor (azul).

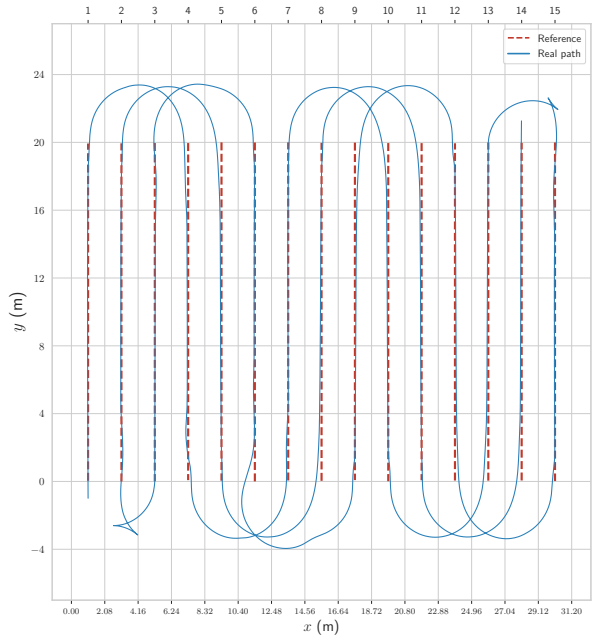
Figura 5.7: Patrones para recorrer el campo de forma completa utilizando casi todos saltos de tipo Π_3.



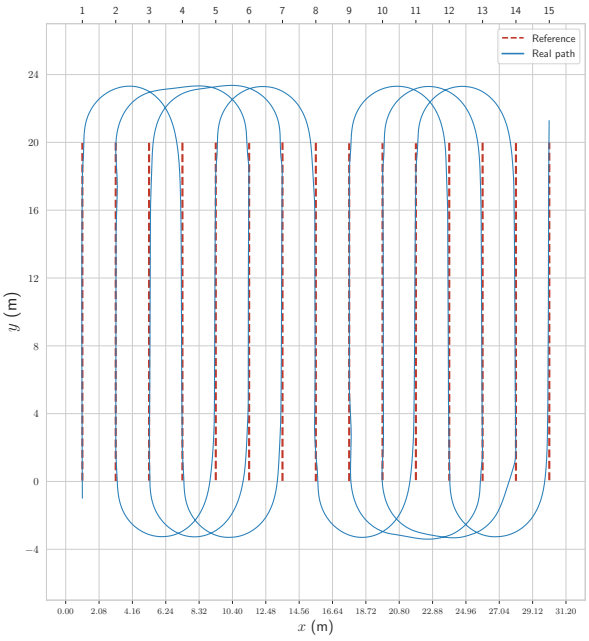
(a) Recorrido trivial utilizando giros $\Omega_1$para 15 franjas.



(b) Recorrido trivial utilizando giros T_1para 15 franjas.



(c) Patrón que utiliza casi todos giros Π_3 para 15 franjas.



(d) Patrón óptimo para 15 franjas que utiliza giros Π_3 y Π_4.

Figura 5.8: Diferentes patrones para la cobertura de un campo de 15 franjas. La línea continua azul corresponde al recorrido realizado por el robot desmalezador en la simulación. Las líneas discontinuas rojas indican el centro de cada una de las franjas. Se observa que en los sectores cercanos a las cabeceras del campo, donde se realizan los giros, el robot se aleja un poco del centro de la franja, pero en pocos metros se vuelve a alinear correctamente.

La métrica ATE (*Absolute Trayectory Error*) mide la distancia absoluta entre cada pose de la trayectoria estudiada contra la pose correspondiente de la trayectoria de referencia. Esto mide la consistencia global de la trayectoria estudiada. Desde una perspectiva práctica, el ATE se visualiza fácilmente como la distancia entre ambas trayectorias. Sin embargo, para algunos casos esta métrica no resulta la más adecuada, ya que un gran error al inicio desplazaría toda la trayectoria estudiada dando como resultando un gran error acumulado total. En cambio, si el mismo error se diera al final de la trayectoria se tendría un resultado de error total mucho menor. Por otro lado, debido a que por lo general las trayectorias a comparar están tomadas a partir de diferentes marcos de coordenadas, la métrica ATE requiere una etapa previa de alineamiento.

Debido a estos inconvenientes, se desarrolló la métrica llamada RPE (*Relative Pose Error*) que en lugar de medir la distancia entre las poses de la trayectoria estudiada con la de referencia, compara los desplazamientos entre dos o más poses de una trayectoria con los de la otra. Con esto se obtiene una medida de la precisión local de la trayectoria sobre un intervalo fijo de tiempo. Dicho con otras palabras, se toma el desplazamiento relativo entre un rango fijo de poses de la trayectoria estudiada y se lo compara con el desplazamiento relativo entre las poses correspondientes al mismo rango pero en la trayectoria de referencia. Al comparar desplazamientos relativos, la métrica RPE no se ve afectada por los marcos de coordenadas en los que se miden ambas trayectorias, por lo que no se requiere una etapa de alineamiento. Por otro lado, grandes errores en algunas pocas poses influyen en menor medida en el error final que con la métrica anterior, incluso si éstos se dan al inicio de la trayectoria. Para ambas métricas se pueden medir las diferencias en los componentes de traslación, rotación o ambos a la vez. Sin embargo, los errores rotacionales generalmente se manifiestan en traslaciones incorrectas por lo que son capturados indirectamente por el componente traslacional.

Como mencionamos anteriormente, para nuestro caso de estudio se toma como referencia la trayectoria que pasa por el medio de la franja que está transitando el robot. Para sincronizar la trayectoria estudiada con la de referencia, se toma para cada pose de la trayectoria real el punto más cercano en el centro de la franja para construir la trayectoria de referencia. Dicha trayectoria se genera en el mismo marco de coordenadas que la trayectoria estudiada, por lo que no requiere de una etapa de alineamiento. Debido a estas características, consideramos que la métrica ATE resulta una buena métrica ya que los problemas antes mencionados no se dan en nuestro caso y, además, tiene la ventaja de que compara en cada momento la distancia entre la pose del robot y el punto en el que desearíamos que esté en ese momento, que consiste en el centro de franja. Todos los errores se suelen acumular calculando el promedio, la mediana o bien el RMSE (*Root Mean Square Error*). En la Tabla 5.4 se muestran los resultados de las métricas ATE y RPE para los cuatro recorridos presentados. Podemos observar que en todos los casos los errores resultan ser relativamente pequeños con una distancia al centro de franja que en ningún caso supera los 61 cm o los 0.43 rad (25°) de desvío en el ángulo de dirección, y un promedio menor a los 10 cm de distancia y 2° de ángulo. Con dichos resultados podemos concluir que las trayectorias generadas por los planificadores elegidos y ejecutadas por la simulación del robot son lo suficientemente buenas para realizar la tarea deseada y evitar el pisado de cultivos.

En la Figura 5.9 se puede ver el detalle de los errores ATE traslacional y rotacional para la trayectoria óptima de 15 franjas. El error mostrado en la Figura 5.9a se puede interpretar como la distancia en metros con respecto al centro de la franja que se está recorriendo en cada momento. Por su parte, el error mostrado en la Figura 5.9b se puede interpretar como la diferencia de ángulo que tiene el robot en cada momento comparado con 90° en caso que el mismo esté recorriendo la franja desde el extremo inferior al superior, o bien comparado con −90° en el caso contrario. Se pueden apreciar una serie de picos en ambas gráficas distribuidos equitativamente a lo largo de la duración total del recorrido. Dichos picos corresponden a los sectores del recorrido de las franjas próximos a los momentos en los que el robot realiza los giros. Vemos que el número de picos coincide con el número de giros del recorrido, que son 14. En la Figura 5.8d podemos ver que en los sectores cercanos a las cabeceras del campo el robot se aleja del centro de la franja para prepararse a realizar el giro y al reingresar en la próxima franja también demora cierto tiempo en alinearse con la misma. Por otro lado, podemos ver en las gráficas que los errores se vuelven prácticamente nulos, lo que indica que luego el robot logra alinearse perfectamente con el centro de la franja que está recorriendo. Se mostraron los gráficos para la secuencia óptima porque es la que consideramos de mayor interés, pero cabe destacar que el comportamiento para el resto de los recorridos resultó muy similar.

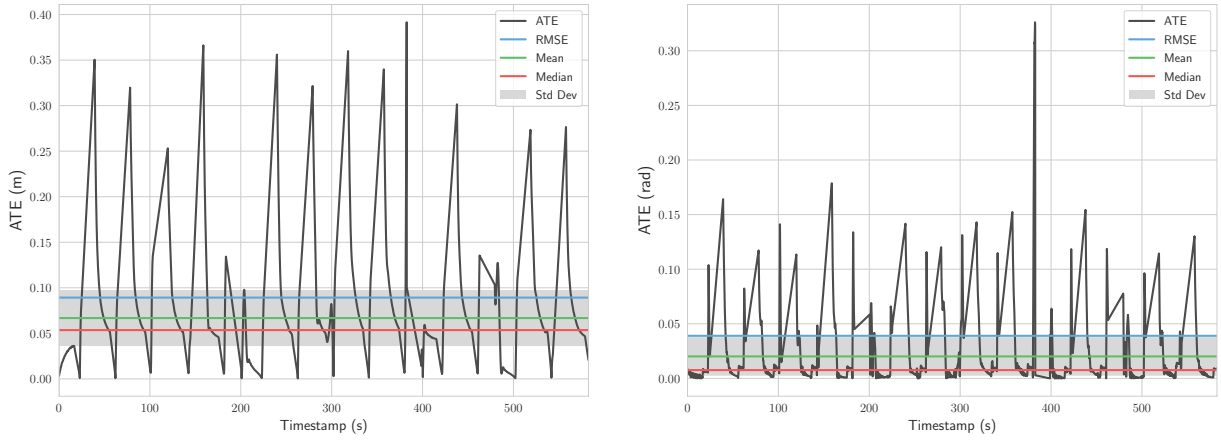
Recorrido	Trivial con Ω_1	Trivial con T_1	Patrón Π_3	Patrón óptimo
Longitud (m)	532,018	**428,508**	441,866	447,410
Duración (s)	741,358	680,171	611,812	**598,505**
(min:s)	(12:21,358)	(11:20,171)	(10:11,812)	**(09:58,505)**

Tabla 5.3: Resultados de longitud y duración de diferentes recorridos para un campo de 15 franjas. La longitud de los recorridos se muestra en metros. La duración de los recorridos se muestra en segundos y, entre paréntesis, en formato minutos y segundos.

Recorrido	ATE							
	Traslacional (m)				Rotacional (rad)			
	Máximo	RMSE	Media	SD	Máximo	RMSE	Media	SD
Trivial con Ω_1	0,588	0,113	0,075	0,084	0,426	0,082	0,035	0,074
Trivial con T_1	**0,331**	**0,036**	**0,025**	**0,026**	**0,165**	**0,030**	**0,014**	**0,026**
Patrón Π_3	0,609	0,089	0,065	0,061	0,317	0,041	0,020	0,035
Patrón óptimo	0,391	0,089	0,066	0,059	0,326	0,039	0,020	0,033

Recorrido	RPE							
	Traslacional (m)				Rotacional (rad)			
	Máximo	RMSE	Media	SD	Máximo	RMSE	Media	SD
Trivial con Ω_1	1,023	0,048	0,003	0,048	0,731	0,037	0,005	0,036
Trivial con T_1	**0,331**	**0,011**	**0,001**	**0,011**	**0,090**	**0,003**	**0,001**	**0,003**
Patrón Π_3	0,569	0,026	0,001	0,026	0,264	0,010	0,002	0,009
Patrón óptimo	0,531	0,028	0,002	0,028	0,320	0,012	0,002	0,011

Tabla 5.4: Resultados de los errores ATE y RPE para diferentes recorridos de un campo de 15 franjas. Se compara la trayectoria realizada por el robot en la simulación con respecto a la trayectoria ideal que pasa por el centro de cada franjas. Solo se toman los errores entre los extremos superior e inferior de cada franja, sin incluir los giros en las cabeceras donde no hay referencia contra la cual comparar.



(a) Error ATE traslacional para el recorrido óptimo.

(b) Error ATE rotacional para el recorrido óptimo.

Figura 5.9: Errores ATE para la secuencia óptima en un campo de 15 franjas comparando la trayectoria real realizada por el robot contra la trayectoria ideal que pasa por el centro de cada franja. El error traslacional en 5.9a indica la distancia al centro de franja en metros y el error rotacional en 5.9b indica la diferencia de ángulo con respecto a $\pm 90°$ según la dirección del robot.

Capítulo 6

Conclusiones

En este trabajo se presentó la implementación de un ambiente de simulación para el robot desmalezador de cultivos de soja desarrollado por el CIFASIS. Una de las partes principales de dicho ambiente consistió en la definición del modelo virtual del robot, con el cual se describen tanto sus propiedades físicas como sus características visuales con el objetivo de generar una simulación realista. Por otro lado, se desarrolló un sistema de navegación autónomo haciendo uso del *navigation stack* de ROS, enfocándonos principalmente en la etapa de planificación de trayectorias. Se analizaron y evaluaron varios planificadores locales, y se concluyó que el planificador TEB (*Timed Elastic Band*) es el más adecuado para utilizar en el robot desmalezador. Fue necesario configurar el planificador local teniendo en cuenta las restricciones actuales del robot. Se requirió del desarrollo de varios nodos auxiliares que conectan los diferentes módulos utilizados con el planificador. Dentro de los nodos desarrollados, se encuentra el generador de *waypoints* el cual permite que el robot pueda realizar una navegación autónoma completa cubriendo la totalidad del campo. Para esto, se evaluaron diferentes tipos de giros y maniobras que puede realizar el robot, y los diferentes patrones de cobertura del campo. Finalmente, se presentó un extenso análisis de la simulación generada y de la robustez del planificador para la navegación entre hileras de cultivos, junto con el análisis de las distintas secuencias de cobertura.

Se observó, producto del análisis de las características cinemáticas del robot desmalezador, cómo se establece su espacio de configuraciones en el cual realizar la navegación. Dicho espacio queda determinado por las restricciones holonómicas, que confinan al robot al plano bidimensional. Como consecuencia, una configuración del robot queda definida por tres componentes que forman lo que se conoce como *pose* del robot, que especifica su posición (x, y) en el plano junto con su ángulo θ de orientación. Por otro lado, vimos que las restricciones no holonómicas reducen el espacio de las posibles velocidades del robot, sin alterar su espacio de configuraciones. Una restricción de este tipo evita que el robot pueda moverse de forma instantánea lateralmente y restringe el radio de giro mínimo del robot, característica que debió ser tenida en cuenta en la generación de las trayectorias. Adicionalmente, analizamos el sistema de direccionamiento del robot desmalezador, el cual es conocido como mecanismo de Ackermann. Dicho sistema evita el deslizamiento de las ruedas delanteras al distribuir ángulos de dirección diferentes a cada una de ellas en la realización de los giros. Vimos que los robots de este estilo son conocidos como vehículos *car-like* y, debido a limitaciones en el lenguaje de modelado, este tipo de dirección tuvo que emularse mediante un *plugin* en la simulación.

Como primera conclusión de este trabajo, podemos determinar que el uso del *navigation stack* de ROS junto con el planificador local TEB resulta una opción viable para emplear en la navegación del robot desmalezador a través de los campos de cultivos. Se obtuvieron buenos resultados en términos de evitar el pisado de los cultivos, incluso con una aproximación del modelo cinemático del vehículo. Los errores en los ajustes de estas propiedades provocan que en ocasiones el robot se desvíe muy levemente del plan inicial pero debido a la continua actualización de la trayectoria no resulta un mayor problema. Por otro lado, dicho análisis fue posible gracias al desarrollo del modelo simulado del robot. La simulación permitió la validación de diferentes soluciones, sin la necesidad siquiera de emplear el prototipo real del robot, reduciendo los tiempos y costos de la realización de dichas pruebas. Adicionalmente, cabe destacar que gracias a la simulación se pudieron ejecutar múltiples pruebas con diferentes configuraciones y entornos, como por ejemplo, el análisis

de los diferentes tipos de giros en las cabeceras del campo.

Como última etapa, se hizo un análisis de las diferentes formas en las que el robot puede realizar una cobertura completa del campo, lo que desencadenó en el estudio de la elección de diferentes secuencias de franjas. Se llegó a la conclusión de que el uso de una secuencia optimizada mejoraría de forma no despreciable los tiempos de cobertura total del campo, además de permitir cabeceras más pequeñas y mayor aprovechamiento del espacio para los cultivos. Se presentaron dos secuencias que resultaron de especial interés. Por un lado, un patrón que utiliza casi todos giros con saltos de a 3 franjas y que puede ser extendido para cualquier cantidad de franjas, el cual resulta el óptimo para campos de más de 15 franjas. Por otro lado, un patrón que utiliza giros de saltos de a 3 y 4 franjas que resulta el óptimo para 8 y 15 franjas. Dicho patrón puede extenderse solo para un número de franjas múltiplo de 7 más 1, pero posee la ventaja de ir recorriendo el campo de forma incremental desde un extremo al otro por todo lo ancho del campo. Debido a que los campos son relativamente extensos con muchas hileras de cultivos, si quedaran franjas sin cubrir al final del campo se podría utilizar otra secuencia solo para estas sin alterar la eficiencia general del patrón.

6.1. Trabajo futuro

A pesar de obtener buenos resultados con el planificador TEB se necesitaron crear varios *waypoints* intermedios para la realización de los giros deseados. Por otro lado, la configuración de los parámetros de optimización del planificador para lograr una estabilidad razonable constituyó una tarea compleja y notamos que el resultado final es muy sensible a estos parámetros. Como trabajo futuro, resulta de interés desarrollar y probar otros planificadores diferentes para comparar los resultados con los obtenidos por el TEB. Un método de planificación de trayectorias ideado para vehículos con radio de giro mínimo limitado también muy usado en robótica es el método de las curvas de Dubins o Reeds-Shepp. El primero se aplica a vehículos que solo pueden circular hacia adelante y el segundo trata los vehículos que también poseen reversa. Ambos métodos aseguran la obtención del camino más corto entre dos puntos pero no tienen en cuenta la evasión de los obstáculos. A pesar de ello, se pueden combinar con algoritmos de tipo RRT (*Rapidly-exploring Random Tree*) para obtener un método que evite la colisión con obstáculos. A diferencia de éstos, el método utilizado en este trabajo tiene como uno de sus objetivos, dentro de su función a optimizar, la generación de una trayectoria que pueda ejecutarse en el menor tiempo posible. Los métodos de Dubins y Reeds-Shepp mencionados, buscarán únicamente generar la trayectoria de menor longitud, la cual en ocasiones no resultará óptima en tiempos de ejecución. Solamente si supusiéramos un escenario no realista en el que no existieran límites en las aceleración del robot, la trayectoria de longitud mínima correspondería con la de menor tiempo de ejecución.

El procedimiento que se desarrolló de filtrar las líneas de cultivos que pasan por debajo del robot para que no interfieran con la generación de la trayectoria sirve para mantener al robot en línea recta con las ruedas dentro de los surcos. Sin embargo, si aparece un obstáculo en el medio del camino, el robot debería pasar a otro modo en el cual no considere a ninguna fila de cultivo como obstáculo y así pueda pasar por sobre las mismas para esquivar el obstáculo real. El planificador TEB utilizado resulta conveniente para la evasión de obstáculos de este tipo pero es requerida dicha modificación en la información recibida por los sensores para poder concretar dicha tarea. Por otro lado, los datos obtenidos del sensor de la cámara fueron generados programáticamente y no se incorporaron errores en los mismos. Una primer mejora consiste en simular ruido a dichos datos. Posteriormente, se podría introducir al modelo de simulación directamente un sensor de cámara, para los cuales ya existen algunas implementaciones, y obtener la nube de puntos directamente desde la cámara simulada para no generarla artificialmente. Por último, notamos que muchos proyectos de navegación incorporan un modo de recuperación cuando el robot queda trabado en algún obstáculo, por lo que sería interesante analizar el agregado de este modo al proceso de navegación del robot desmalezador.

Con respecto a la simulación, el modelo utilizado plantea ejes virtuales los cuales luego se descomponen en los reales. Esta abstracción no sería del todo correcta modelarla en URDF. Esto es parte del modelo cinemático inverso que traduce velocidades requeridas del cuerpo rígido a velocidades angulares por ruedas. Se podrían implementar nodos de ROS por fuera de Gazebo que realicen dichas conversiones, dejando al *plugin* encargado de recibir las velocidades angulares por rueda y accionar los *joints* con controladores PID únicamente.

Por último, cabe destacar que el modelo de simulación desarrollado queda abierto a extensiones futuras para que pueda ser aprovechado en la validación de otros sistemas involucrados en el robot desmalezador, como son los métodos de localización, detección de maleza y detección de surcos. Por otra parte, si bien los resultados de la planificación de trayectorias fueron satisfactorios en la simulación, queda pendiente la verificación de los mismos en una prueba de campo con el prototipo real del robot.

Apéndice A

Diagrama simulación y navegación

La figura A.1 muestra el diagrama completo de la simulación y navegación del robot desmalezador, y está inspirado en los diagramas encontrados en el tutorial de ROS Control de Gazebo[1] y en el tutorial del *navigation stack* de ROS[2] combinados y adaptados al desarrollo actual.

[1] http://gazebosim.org/tutorials/?tut=ros_control
[2] http://wiki.ros.org/navigation/Tutorials/RobotSetup

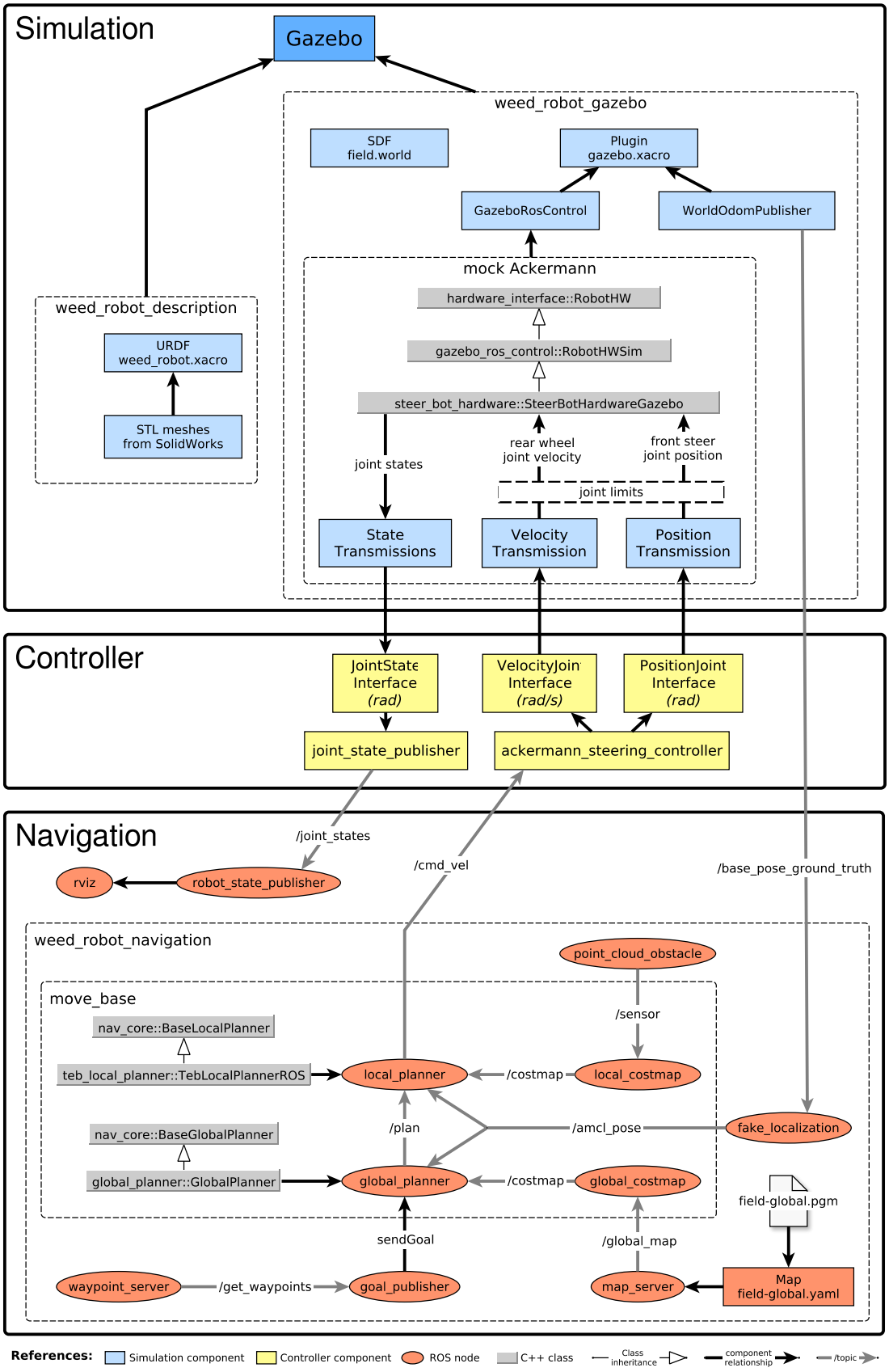


Figura A.1: Diagrama simulación, control y navegación.

Bibliografía

[1] Stavros G. Vougioukas. Agricultural Robotics. *Annual Review of Control, Robotics, and Autonomous Systems*, 2(1):365–392, 2019.

[2] Fernando Auat Cheein and Ricardo Carelli. Agricultural Robotics: Unmanned Robotic Service Units in Agricultural Tasks. *Industrial Electronics Magazine, IEEE*, 7:48–58, 2013.

[3] Marcel Bergerman, E.J. Van Henten, John Billingsley, John Reid, and Mingcong Deng. IEEE Robotics and Automation Society Technical Committee on Agricultural Robotics and Automation. *Robotics & Automation Magazine, IEEE*, 20:20–125, 2013.

[4] James Lowenberg-DeBoer, Iona Huang, Vasilis Grigoriadis, and Simon Blackmore. Economics of robots and automation in field crop production. *Precision Agriculture*, pages 278–299, May 2020.

[5] P. Parvatha Reddy. *Agro-ecological Approaches to Pest Management for Sustainable Agriculture. Precision Agriculture*, pages 295–309. Springer Singapore, 2017.

[6] John Stafford. Implementing Precision Agriculture in the 21st Century. *Journal of Agricultural Engineering Research*, 76:267–275, July 2000.

[7] Ulrich Weiss and Peter Biber. Plant detection and mapping for agricultural robots using a 3D LIDAR sensor. *Robotics and Autonomous Systems*, 59:265–273, 2011.

[8] Taihú Pire, Martín Mujica, Javier Civera, and Ernesto Kofman. The Rosario dataset: Multisensor data for localization and mapping in agricultural environments. *International Journal of Robotics Research*, 38:633–641, 2019.

[9] Marcelo Pistarelli, Taihú Pire, and Ernesto Kofman. Caracterización de un Sistema GPS RTK de Bajo Costo. *Revista Tecnología y Ciencia*, (35):94–107, May 2019.

[10] Jean-Claude Latombe. *Robot Motion Planning*. Kluwer Academic Publishers, 1991.

[11] Howie Choset, Kevin M. Lynch, Seth Hutchinson, George Kantor, Wolfram Burgard, Lydia Kavraki, and Sebastian Thrun. *Principles of Robot Motion: Theory, Algorithms, and Implementations*. MIT Press, 2005.

[12] Farbod Fahimi. *Autonomous Robots: Modeling, Path Planning, and Control*. Springer US, 2009.

[13] Bruno Siciliano, Lorenzo Sciavicco, Luigi Villani, and Giuseppe Oriolo. *Robotics: Modelling, Planning and Control*. Springer Publishing Company, Incorporated, 2008.

[14] Kevin M. Lynch and Frank C. Park. *Modern Robotics: Mechanics, Planning, and Control*. Cambridge University Press, 2017.

[15] Howie Choset and Philippe Pignon. Coverage Path Planning: The Boustrophedon Cellular Decomposition. In *Field and Service Robotics*, pages 203–209, 1998.

[16] R. Neumann De Carvalho, H. A. Vidal, P. Vieira, and M. I. Ribeiro. Complete coverage path planning and guidance for cleaning robots. In *Proceeding of the IEEE International Symposium on Industrial Electronics*, volume 2, pages 677–682, 1997.

[17] Ercan Acar, Howie Choset, Yangang Zhang, and Mark Schervish. Path Planning for Robotic Demining: Robust Sensor-Based Coverage of Unstructured Environments and Probabilistic Methods. *The International Journal of Robotics Research*, 22:441–466, 2003.

[18] Marija Dakulović, Sanja Horvatić, and Ivan Petrović. Complete Coverage D* Algorithm for Path Planning of a Floor-Cleaning Mobile Robot. *IFAC Proceedings Volumes*, 44:5950–5955, 2011.

[19] Ercan Acar, Morgan Simmons, Michael Rosenblatt, Maayan Roth, Mary Berna, Yonatan Mittlefehldt, and Howie Choset. Sensor Based Coverage of Unknown Environments for Land Mine Detection. In *Sixteenth National Conference on Artificial Intelligence*, pages 932–933, 1999.

[20] S. X. Yang and C. Luo. A neural network approach to complete coverage path planning. *IEEE Transactions on Systems, Man, and Cybernetics*, 34:718–724, 2004.

[21] M. A. Farsi, Karl Ratcliff, Jeffrey P. Johnson, C. R. Allen, K. Z. Karam, and Richard Pawson. Robot control system for window cleaning. *Proceedings of American Control Conference*, 1:994–995, 1994.

[22] Ismael Ait, Taihú Pire, and Ernesto Kofman. Revisión de Métodos de Planificación de Camino de Cobertura para Entornos Agrícolas. *X Jornadas Argentinas de Robótica*, 2019.

[23] Bengt Erland Ilon. Wheels for a Course Stable Selfpropelling Vehicle Movable in any Desired Direction on the Ground or Some Other Base. U.S. Patent No. 3,876,255, 1975.

[24] Josef F. Blumrich. Omnidirectional Wheel. U.S. Patent No. 3,789,947, 1974.

[25] Georges Giralt, Raja Chatila, and Marc Vaisset. *An Integrated Navigation and Motion Control System for Autonomous Multisensory Mobile Robots*, pages 420–443. Springer New York, 1990.

[26] Sadegh Rabiee and Joydeep Biswas. A Friction-Based Kinematic Model for Skid-Steer Wheeled Mobile Robots. pages 8563–8569, May 2019.

[27] Tomás Lozano-Pérez. Spatial Planning: A Configuration Space Approach. *IEEE Transactions on Computers*, C-32(2):108–120, 1983.

[28] Steven M. LaValle. *Planning Algorithms*. Cambridge University Press, 2006.

[29] Gregor Klancar, Andrej Zdesar, Saso Blazic, and Igor Skrjanc. *Wheeled Mobile Robotics: From Fundamentals Towards Autonomous Systems*. Butterworth-Heinemann, 2017.

[30] James Crowley. Navigation of an Intelligent Mobile Robot. *Robotics and Automation, IEEE Journal of*, RA-1:31–41, April 1985.

[31] Thierry Siméon, Jean-Paul Laumond, and C. Nissoux. Visibility-based probabilistic roadmaps for motion planning. *Advanced Robotics*, 14(6):477–493, 2000.

[32] Yun-Hui Liu and Suguru Arimoto. Path Planning Using a Tangent Graph for Mobile Robots Among Polygonal and Curved Obstacles. *The International Journal of Robotics Research*, 11(4):376–382, August 1992.

[33] Der-Tsai Lee and Robert Drysdale. Generalization of Voronoi Diagrams in the Plane. *SIAM Journal on Computing*, 10:73–87, 1981.

[34] Jean-Claude Latombe. *Exact Cell Decomposition*, pages 200–247. Springer US, 1991.

[35] J. Milnor, M. Spivak, and R. Wells. *Morse Theory*, volume 51. Princeton University Press, 1969.

[36] Edsger W. Dijkstra. A Note on Two Problems in Connexion with Graphs. *Numer. Math.*, pages 269–271, December 1959.

[37] Peter E. Hart, Nils J. Nilsson, and Bertram Raphael. A Formal Basis for the Heuristic Determination of Minimum Cost Paths. *IEEE Transactions on Systems Science and Cybernetics*, 4(2):100–107, July 1968.

[38] Steven M. Lavalle. Rapidly-exploring random trees: a new tool for path planning. *The annual research report*, 1998.

[39] Oussama Khatib. Real-time obstacle avoidance for manipulators and mobile robots. In *Proceedings. 1985 IEEE International Conference on Robotics and Automation*, volume 2, pages 500–505, March 1985.

[40] Dieter Fox, Wolfram Burgard, and Sebastian Thrun. The Dynamic Window Approach to Collision Avoidance. *IEEE Robotics Automation Magazine*, 4(1):23–33, March 1997.

[41] Sean Quinlan and Oussama Khatib. Towards Real-Time Execution of Motion Tasks. In *The 2nd International Symposium on Experimental Robotics II*, pages 241–254. Springer-Verlag, 1991.

[42] Christoph Rösmann, Wendelin Feiten, Thomas Woesch, Frank Hoffmann, and Torsten Bertram. Trajectory modification considering dynamic constraints of autonomous robots. pages 1–6, January 2012.

[43] Christoph Rösmann, Wendelin Feiten, Thomas Wosch, Frank Hoffmann, and Torsten Bertram. Efficient trajectory optimization using a sparse model. pages 138–143, September 2013.

[44] Christoph Rösmann, Frank Hoffmann, and Torsten Bertram. *Online Trajectory Planning in ROS Under Kinodynamic Constraints with Timed-Elastic-Bands*, pages 231–261. May 2017.

[45] Lester E. Dubins. On Curves of Minimal Length with a Constraint on Average Curvature, and with Prescribed Initial and Terminal Positions and Tangents. *American Journal of Mathematics*, 79(3):497–516, July 1957.

[46] Sertac Karaman, Matthew Walter, Alejandro Perez, Emilio Frazzoli, and Seth Teller. Anytime Motion Planning using the RRT*. pages 1478–1483, June 2011.

[47] James A. Reeds and Lawrence A. Shepp. Optimal paths for a car that goes both forwards and backwards. *Pacific Journal of Mathematics*, 145:367–393, 1990.

[48] Jorge Nocedal and Stephen J. Wright. *Penalty and Augmented Lagrangian Methods*, pages 497–528. Springer New York, 2006.

[49] Rainer Kümmerle, Giorgio Grisetti, Hauke Strasdat, Kurt Konolige, and Wolfram Burgard. G2o: A general framework for graph optimization. In *2011 IEEE International Conference on Robotics and Automation*, pages 3607–3613, 2011.

[50] Redmond Shamshiri, Ibrahim Hameed, Lenka Pitonakova, Cornelia Weltzien, Siva Balasundram, Ian Yule, Tony Grift, and Girish Chowdhary. Simulation software and virtual environments for acceleration of agricultural robotics: Features highlights and performance comparison. *International Journal of Agricultural and Biological Engineering*, 11:12–20, January 2018.

[51] Carlo Pinciroli, Vito Trianni, Rehan O'Grady, Giovanni Pini, Arne Brutschy, Manuele Brambilla, Nithin Mathews, Eliseo Ferrante, Gianni Di Caro, Frederick Ducatelle, Timothy Stirling, Alvaro Gutiérrez, Luca Maria Gambardella, and Marco Dorigo. ARGoS: A modular, multi-engine simulator for heterogeneous swarm robotics. pages 5027–5034, September 2011.

[52] Carlo Pinciroli, Vito Trianni, Rehan O'Grady, Giovanni Pini, Arne Brutschy, Manuele Brambilla, Nithin Mathews, Eliseo Ferrante, Gianni Di Caro, Frederick Ducatelle, Mauro Birattari, Luca Maria Gambardella, and Marco Dorigo. ARGoS: a Modular, Parallel, Multi-Engine Simulator for Multi-Robot Systems. *Swarm Intelligence*, 6(4):271–295, 2012.

[53] B. P. Gerkey, R. T. Vaughan, K. Stoy, A. Howard, G. S. Sukhatme, and M. J. Mataric. Most valuable player: a robot device server for distributed control. In *Proceedings 2001 IEEE/RSJ International Conference on Intelligent Robots and Systems. Expanding the Societal Role of Robotics in the the Next Millennium (Cat. No.01CH37180)*, volume 3, pages 1226–1231, February 2001.

[54] Olivier Michel. Webots^{TM}: Professional Mobile Robot Simulation. *International Journal of Advanced Robotic Systems*, 1, March 2004.

[55] Eric Rohmer, Surya Singh, and Marc Freese. V-REP: A versatile and scalable robot simulation framework. pages 1321–1326, November 2013.

[56] Nathan Koenig and Andrew Howard. Design and use paradigms for Gazebo, an open-source multi-robot simulator. In *2004 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, volume March, pages 2149–2154, 2004.

[57] Morgan Quigley, Ken Conley, Brian P. Gerkey, Josh Faust, Tully Foote, Jeremy Leibs, Rob Wheeler, and Andrew Y. Ng. ROS: an open-source Robot Operating System. In *ICRA Workshop on Open Source Software*, 2009.

[58] Kenta Takaya, Toshinori Asai, Valeri Kroumov, and Florentin Smarandache. Simulation environment for mobile robots testing using ROS and Gazebo. In *2016 20th International Conference on System Theory, Control and Computing (ICSTCC)*, pages 96–101, 2016.

[59] Tully Foote. tf: The Transform Library. In *2013 IEEE Conference on Technologies for Practical Robot Applications (TePRA)*, 2013.

[60] Sachin Chitta, Eitan Marder-Eppstein, Wim Meeussen, Vijay Pradeep, Adolfo Tsouroukdissian, Jonathan Bohren, David Coleman, Bence Magyar, Gennaro Raiola, Mathias Lüdtke, and Enrique Perdomo. ros_control: A generic and simple control framework for ROS. *The Journal of Open Source Software*, 2, December 2017.

[61] Stephanie Bonadies and S. Andrew Gadsden. An overview of autonomous crop row navigation strategies for unmanned ground vehicles. *Engineering in Agriculture, Environment and Food*, 12(1):24–31, 2019.

[62] Avital Bechar and Clément Vigneault. Agricultural robots for field operations: Concepts and components. *Biosystems Engineering*, 149:94–111, 2016.

[63] Mark Post, Alessandro Bianco, and Xiu-Tian Yan. *Autonomous Navigation with Open Software Platform for Field Robots*, pages 425–450. January 2020.

[64] Arno Ruckelshausen, Peter Biber, Michael Dorna, H. Gremmes, Ralph Klose, Andreas Linz, R. Rahe, Rainer Resch, M. Thiel, Dieter Trautz, and Ulrich Weiss. Bonirob: An autonomous field robot platform for individual plant phenotyping. *Precision Agriculture*, 9:841–847, January 2009.

[65] Tijmen Bakker, Kees van Asselt, Jan Bontsema, Joachim Müller, and Gerrit van Straten. Autonomous navigation using a robot platform in a sugar beet field. *Biosystems Engineering*, 109(4):357–368, 2011.

[66] Jawad Iqbal, Rui Xu, Shangpeng Sun, and Changying Li. Simulation of an Autonomous Mobile Robot for LiDAR-Based In-Field Phenotyping and Navigation. *MDPI Robotics*, 9(2), 2020.

[67] Diego Aghi, Vittorio Mazzia, and Marcello Chiaberge. Local Motion Planner for Autonomous Navigation in Vineyards with a RGB-D Camera-Based Algorithm and Deep Learning Synergy. *Machines*, May 2020.

[68] Flavio B.P. Malavazi, Remy Guyonneau, Jean-Baptiste Fasquel, Sebastien Lagrange, and Franck Mercier. LiDAR-only based navigation algorithm for an autonomous agricultural robot. *Computers and Electronics in Agriculture*, 154:71–79, 2018.

[69] Xiang Yin, Juan Du, Noboru Noguchi, T.X. Yang, and C.Q. Jin. Development of autonomous navigation system for rice transplanter. *International Journal of Agricultural and Biological Engineering*, 11:89–94, January 2018.

[70] Xiang Yin, Yanxin Wang, Yulong Chen, Chengqian Jin, and Juan Du. Development of autonomous navigation controller for agricultural vehicles. *International Journal of Agricultural and Biological Engineering*, 13:70–76, January 2020.

[71] Zhengduo Liu, Zhaoqin Zheng, Wanzhi Zhang, and Xiangxun Cheng. Design of obstacle avoidance controller for agricultural tractor based on ROS. *International Journal of Agricultural and Biological Engineering*, 12:58–65, January 2019.

[72] Santosh A. Hiremath, Gerie W.A.M. van der Heijden, Frits K. van Evert, Alfred Stein, and Cajo J.F. ter Braak. Laser range finder model for autonomous navigation of a robot in a maize field using a particle filter. *Computers and Electronics in Agriculture*, 100:41–50, 2014.

[73] Raphael Linker and Tamir Blass. Path-planning algorithm for vehicles operating in orchards. *Biosystems Engineering*, 101(2):152–160, 2008.

[74] T. Mueller-Sim, M. Jenkins, J. Abel, and G. Kantor. The Robotanist: A ground-based agricultural robot for high-throughput crop phenotyping. In *2017 IEEE International Conference on Robotics and Automation (ICRA)*, pages 3634–3639, 2017.

[75] Rainer Kümmerle, Michael Ruhnke, Bastian Steder, Cyrill Stachniss, and Wolfram Burgard. Autonomous Robot Navigation in Highly Populated Pedestrian Zones. *Journal of Field Robotics*, 32, September 2014.

[76] B. Cybulski, A. Wegierska, and G. Granosik. Accuracy comparison of navigation local planners on ROS-based mobile robot. In *2019 12th International Workshop on Robot Motion and Control (RoMoCo)*, pages 104–111, July 2019.

[77] Alexandros Filotheou, Emmanouil Tsardoulias, Antonis Dimitriou, Andreas Symeonidis, and Loukas Petrou. Quantitative and Qualitative Evaluation of ROS-Enabled Local and Global Planners in 2D Static Environments. *Journal of Intelligent & Robotic Systems*, 98, June 2020.

[78] Pablo Marín, Ahmed Hussein, David Martín Gómez, and Arturo de la Escalera. Global and Local Path Planning Study in a ROS-Based Research Platform for Autonomous Vehicles. *Journal of Advanced Transportation*, 2018, February 2018.

[79] Sebastian Thrun, Wolfram Burgard, and Dieter Fox. *Probabilistic Robotics (Intelligent Robotics and Autonomous Agents)*. The MIT Press, 2005.

[80] Dionysis Bochtis and Stavros Vougioukas. Minimising the non-working distance travelled by machines operating in a headland field pattern. *Biosystems Engineering*, 101:1–12, September 2008.

[81] Dionysis Bochtis, Stavros Vougioukas, and Hans W. Griepentrog. A Mission Planner for an Autonomous Tractor. *Transactions of the ASABE*, 52, September 2009.

[82] Gilbert Laporte. The traveling salesman problem: An overview of exact and approximate algorithms. *European Journal of Operational Research*, 59(2):231–247, 1992.

[83] Rainer Kümmerle, Bastian Steder, Christian Dornhege, Michael Ruhnke, Giorgio Grisetti, Cyrill Stachniss, and Alexander Kleiner. On Measuring the Accuracy of SLAM Algorithms. *Autonomous Robots*, 27(4):387–407, November 2009.

[84] Jürgen Sturm, Nikolas Engelhard, Felix Endres, Wolfram Burgard, and Daniel Cremers. A benchmark for the evaluation of RGB-D SLAM systems. pages 573–580, October 2012.