UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

FACULTAD DE INGENIERÍA

Navegación de un robot móvil para la supervisión de una residencia monitoreada vía Internet

TESIS

Que para obtener el título de

Ingeniero Mecatrónico

P R E S E N T A

Jorge Luis Azuara Domínguez

DIRECTOR DE TESIS

Dr. Jesús Savage Carmona

Ciudad Universitaria, Cd. Mx., 2018

Agradecimientos

A mis papás por todo el apoyo que me han dado, gracias al esfuerzo y los sacrificios que han hecho durante muchos años tengo la oportunidad de lograr este objetivo.

A mis hermanos Ivan y Rubén, por ayudarme siempre que los he necesitado.

A mis amigos que conozco de primer semestre, Raúl, Alonso, Gustavo, Jessica y Rodrigo. También a mis amigos que fui conociendo a lo largo de la carrera, Juan, Luis, Nacho, Carlos y Nabila.

A mis amigos y compañeros del Laboratorio de Biorobótica, gracias por su amistad y por compartir sus conocimientos conmigo.

A la Universidad Nacional Autónoma de México por la formación que me brindó.

Al Dr. Savage por darme la oportunidad de realizar la tesis.

Al proyecto PAPIME PE102115 "*Prácticas para la materia de Diseño de Casas Inteligentes*" por el apoyo recibido.

Índice general

Índice de figuras

Capítulo 1

Introducción

SmartThings es una empresa de tecnología que cuenta con una plataforma abierta para desarrolladores de Internet Of Things, fue fundada en 2012 y en agosto de 2014 fue adquirida por Samsung. El presente trabajo propone la integración de un robot móvil autónomo como un nuevo dispositivo dentro del ecosistema de SmartThings capaz de interactuar con otros dispositivos comerciales, en él se expone la construcción del modelo funcional y el desarrollo hecho en ROS.

1.1. Objetivos

1.1.1. Objetivo general

Integrar un robot móvil como un nuevo dispositivo dentro del ecosistema de SmartThings. Así como construir y controlar el robot móvil, con la finalidad de implementar un sistema que le permita navegar de forma autónoma y tele operada en un ambiente de trabajo casero y estático.

1.1.2. Objetivos particulares

- Monitorizar un hogar mediante el procesamiento de alertas de los dispositivos de Internet of Things de la empresa SmartThings y, notificar al robot y al usuario sobre situaciones inesperadas como lo son fugas de agua, aperturas o cierres indeseados de puertas y ventanas, presencia de humo, detección de movimiento y actuación física sobre dispositivos en el hogar.

- Implementar un controlador de dispositivo (*Device Handler*: representación virtual de un dispositivo físico en SmartThings), en el que el usuario sea capaz de tele operar al robot móvil y enviarlo a lugares pre establecidos dentro del hogar.

- Crear una página web accesible para el usuario dentro de una red LAN, la cual contenga un visualizador 3D similar a la herramienta de visualización RViz para ROS.

- Aplicar leyes de control a partir del modelo cinemático del robot que le permitan lograr el seguimiento de una ruta calculada por medio de un algoritmo de búsqueda.

1.2. Alcances

Se construirá un robot móvil para interiores capaz de navegar de forma autónoma en ambientes de trabajo estáticos, por lo que se optará por utilizar únicamente el enfoque de comportamientos tradicionales como modelo de control para robots móviles. El laboratorio de Bio-Robótica cuenta con el robot de servicio Justina, por consiguiente, no resultará de interés los comportamientos reactivos, y por ende tampoco los híbridos. Se asumirá que el robot se localiza de manera ideal dentro del ambiente mediante el cálculo de la odometría, por lo que no se utilizará un método de localización activo.

Los componentes solicitados para la construcción del robot móvil son motores Dynamixel MX-12W, un chasis Dagu Wild Thumper 4WD, un sensor láser Hokuyo URG-04LX, una Cloud Camera de la empresa D-Link, una Raspberry Pi 3, un Arduino Uno y una Arduino ThingShield de la empresa SmartThings. Los dispositivos de Internet Of Things que se emplearán para la monitorización del hogar son simulados, sin embargo, se garantiza la funcionalidad de lo aquí planteado con dispositivos reales.

1.3. Organización del trabajo

En los Capítulos 2 y 3 se proporcionan los antecedentes y conceptos necesarios para poner al lector en contexto, y por tanto pueda comprender este escrito sin la necesidad de consultar otras fuentes de información o publicaciones anteriores.

En los Capítulos 4 y 5 se describen los componentes utilizados en la construcción del robot móvil, el desarrollo hecho en la plataforma de SmartThings y el desarrollo hecho en ROS. Así mismo, se describe el funcionamiento del sistema propuesto.

En el Capítulo 6 se presentan los resultados obtenidos en las pruebas experimentales realizadas al sistema propuesto, con el propósito de evaluar el desempeño del mismo.

Por último, en el Capítulo 7 se exponen las conclusiones y se propone el trabajo a futuro.

Capítulo 2

Generalidades: Internet de las Cosas

2.1. Antecedentes y estado de la técnica

El término "Internet" se refiere a las aplicaciones y protocolos construidos sobre redes informáticas sofisticadas e interconectadas, sirviendo a millones de usuarios en el mundo. Posteriormente, el enfoque se ha desplazado hacia una integración de personas y dispositivos para converger el ámbito físico con ambientes virtuales, creando el llamado Internet of Things (IoT). La conectividad es un requisito crucial de Internet of Things (Internet de las Cosas), para cumplir con ello, las aplicaciones necesitan soportar un conjunto diverso de dispositivos y protocolos de comunicación. [1]

El término Internet de las Cosas fue utilizado por primera vez por Kevin Ashton durante una presentación en 1999 sobre la gestión de la cadena de suministro. Él cree que la forma en la que interactuamos con las cosas y la manera en que vivimos dentro del mundo físico que nos rodea necesitan una seria reconsideración, debido a los avances en computación, Internet y la tasa de datos generados por dispositivos inteligentes. En aquel entonces, Ashton era director ejecutivo del centro de investigación Auto-ID Lab del MIT (Massachusetts Institute of Technology), lugar en el que se estaba haciendo un importante esfuerzo para identificar productos de manera única, y en donde Ashton contribuyó a la extensión de las aplicaciones de la Identificación por Radio-Frecuencia (RFID). En un principio, la RFID era la tecnología dominante para el desarrollo de Internet de las Cosas, pero con otros logros tecnológicos y los dispositivos Bluetooth aumentó la adopción general de la tendencia de Internet de las Cosas. [1] [2]

En 2005, la International Telecommunication Unit (ITU) mostró interés en las posibilidades de negocios de telecomunicaciones que podrían generarse gracias a la nueva conectividad de objetos a la red. La ITU elaboró un exhaustivo informe desde el punto de vista técnico, económico y ético. Introdujo un nuevo eje en el establecimiento de las redes, para completar la conectividad existente del "*any where*" (en cualquier lugar) y "*any time*" (en cualquier momento), se trata del eje "*any thing*" (cualquier cosa). Naturalmente, esto abre nuevas oportunidades de servicio.

En la Figura 2.1[1] se presenta la visión de la ITU, agregando la conexión de "*any thing*" a la conectividad "*any place*" y "*any time*". [2]

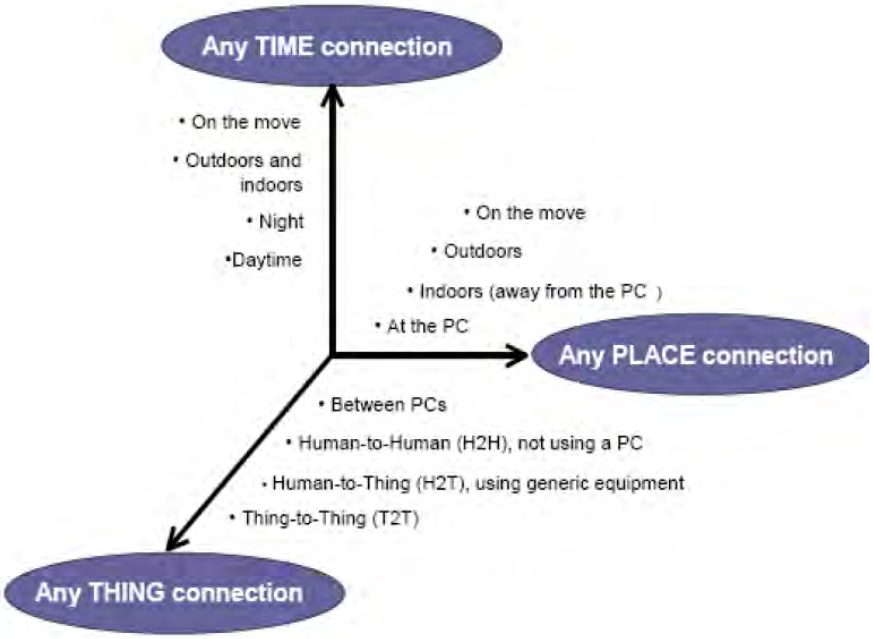

Figura 2.1: Visión de la ITU "*any place*", "*any time*" and "*any thing*".

Internet de las Cosas es un concepto que hace referencia a la interconexión de objetos de uso cotidiano, vehículos, edificios, cámaras, sensores, actuadores y otros dispositivos electrónicos para el intercambio y recopilación de datos. En IoT los elementos de nuestro ambiente y de la vida cotidiana son denominados "cosas", "objetos" o "máquinas", que se complementan con la tecnología informática y de comunicación. Internet de las Cosas considera la presencia de una variedad de objetos en el entorno, que por medio de conexiones inalámbricas y cableadas son capaces de interactuar entre sí y cooperar con otros objetos para alcanzar objetivos comunes. Este paradigma da pie a nuevas innovaciones que construirán interacciones novedosas entre las cosas y los seres humanos, permitirá la creación de ciudades inteligentes, infraestructuras y servicios para mejorar la calidad de vida y la utilización de los recursos. Un mundo donde lo real, lo digital y lo virtual convergen para crear entornos inteligentes para el transporte, ahorro de energía, entretenimiento, servicios de la salud, automatización del hogar, seguridad, entre otras áreas. IoT permite que los objetos sean identificables de forma única y controlados remotamente a través de la infraestructura de internet existente, resultando en una intervención humana reducida. [1–4]

Aunque parece que cada objeto capaz de conectarse a Internet caerá dentro de la categoría de "cosas", esta notación se utiliza para abarcar un conjunto más

[1]Fuente: [2]

genérico de entidades, incluyendo seres humanos y cualquier objeto que reconozca eventos y cambios en su alrededor y es capaz de comunicarse con otras entidades. El objetivo de IoT es el de permitir que las cosas se conecten en cualquier momento, situación y lugar, con cualquier cosa y cualquier persona idealmente empleando cualquier vía. Esto implica que los objetos deben ser accesibles sin restricciones de tiempo o lugar. Los objetos son reconocibles y se toman decisiones relacionadas con el contexto debido a su capacidad de transmitir información entre sí, es decir, los objetos pueden acceder a la información generada por otras cosas. [1] [3] [4]

El papel principal de IoT es la entrega de aplicaciones basadas en el conocimiento y orientadas a la acción en tiempo real. De alguna manera IoT es un camino a un mundo inteligente con objeto de facilitar tareas a los usuarios, dando un fácil seguimiento a los fenómenos que nos rodean para reaccionar con base en ello. Las tecnologías inalámbricas y cableadas ya proporcionan las capacidades de comunicación, satisfaciendo una variedad de servicios en las interacciones persona-persona, persona-máquina, máquina-persona y máquina-máquina. Estos objetos conectados son nuevos usuarios de la red y generan tráfico de datos en el Internet. Los servicios basados en IoT proporcionan automatización de varias tareas gracias a la identificación, detección, decisión y actuación. [1] [2]

El desarrollo de tecnologías como la nanotecnología, el procesamiento automático, los sistemas embebidos, la robótica y el software serán fundamentales para respaldar importantes innovaciones de los productos de IoT. Para aprovechar el potencial de IoT, es necesario desarrollar soluciones conceptuales y tecnológicas adecuadas. Esto incluye el desarrollo de arquitectura escalable, temas de privacidad, seguridad, ética, almacenamiento, procesamiento, diseño de protocolos de comunicación, administración de recursos y gestión de datos y redes. [1–3]

2.2. SmartThings

SmartThings es una plataforma de desarrollo abierta para Internet of Things, en la que se puede: [5]

- Crear aplicaciones que permitan a los usuarios finales conectar dispositivos, acciones y servicios externos para crear automatizaciones.
- Integrar nuevos dispositivos dentro del ecosistema de SmartThings.
- Publicar aplicaciones e integraciones de dispositivos al catálogo de SmartThings.

SmartThings busca ser intuitivo y amigable con los desarrolladores. Por consiguiente, entre sus características destacan el uso del lenguaje de programación Groovy; una arquitectura que permite a los desarrolladores controlar el hardware

mediante software simple; un IDE (Integrated Development Environment) web; un simulador para crear, probar y editar el código desarrollado, incluso cuando no se cuenta con los dispositivos físicos, estos pueden ser simulados y, por último, una comunidad de desarrolladores activa y creciente para colaborar y aprender.

Existen dos formas principales en las que se puede desarrollar dentro de SmartThings: las aplicaciones para obtener información generada por distintos dispositivos (*SmartApps*) y los controladores de dispositivos (*Device Handlers*).

2.2.1. SmartApps

Las *SmartApps* son programas que permiten a los usuarios finales conectar sus dispositivos entre sí, por medio de estas es posible conocer la información generada por los dispositivos que son de nuestro interés, es decir, se pueden ejecutar ciertas acciones cuando se producen ciertos eventos. A lo largo del trabajo se hace referencia a estas aplicaciones utilizando el término *SmartApp*, puesto que tanto en la documentación para desarrolladores y en la aplicación disponible para iOS, Android y Windows se utiliza el idioma inglés. Algunos ejemplos de estas aplicaciones son:

- Apagar los focos de una habitación después de no detectar movimiento durante un periodo de tiempo establecido.
- Enviar notificaciones en caso de que se abra una puerta cuando nadie está en casa.
- Apagar algún aparato conectado a la toma de corriente cuando alguien sale de casa.

Las *SmartApps* pueden comunicarse con servicios web externos, enviar notificaciones push y SMS. Las notificaciones push son mensajes que aparecen en un dispositivo móvil y entregan información desde una aplicación de software sin una solicitud específica del cliente, para recibir estas notificaciones no se requiere tener abierta la aplicación e incluso la pantalla del dispositivo puede estar bloqueada (Figura 2.2)[2]. SMS (Short Message Service) es un servicio disponible en los teléfonos móviles y permite enviar mensajes de texto cortos.

Las *Event Handler SmartApps* (aplicaciones controladoras de eventos) son las aplicaciones más desarrolladas, permiten suscribirse a `eventos` de los dispositivos y llamar a un método controlador, este método puede hacer una variedad de cosas, invocando comúnmente un `comando` en otro dispositivo. Una *SmartApp* no se está ejecutando continuamente; se ejecuta en respuesta a eventos suscritos o a su invocación en un momento particular. Los `eventos` y `comandos` están relacionados a los *Device Handlers* y se detallan en el siguiente apartado.

[2]Fuente: `http://appsbuilderblog.azurewebsites.net/wp-content/uploads/2015/05/Wonder-Shop-e1432547265382.jpg`

Figura 2.2: Ejemplo de notificación push.

2.2.2. Device Handlers

Es posible integrar a SmartThings nuevos dispositivos por medio de la creación de controladores de dispositivos (*Device Handlers*), los cuales son programas que encapsulan los detalles de la comunicación entre SmartThings y los dispositivos físicos.

En otras palabras, un *Device Handler* es la representación virtual de un dispositivo físico, y es responsable de la comunicación entre el dispositivo real y la plataforma de SmartThings. Un *Device Handler* también puede estar asociado a un dispositivo virtual cuando no está disponible aún un dispositivo físico. A lo largo del trabajo se hace referencia a estos programas utilizando el término *Device Handler*, debido a que en la documentación para desarrolladores se utiliza el idioma inglés.

Para entender cómo funcionan los *Device Handlers*, es necesario definir los conceptos de `capabilities` (capacidades), `commands` (comandos) y `attributes` (atributos).

Las `capabilities` proporcionan una capa de abstracción que permite trabajar con los dispositivos basándose en las funciones que admiten sin la necesidad de vincularse con un fabricante o producto en específico. Considérese la `capacidad`

"Switch", en términos simples, un switch es un dispositivo que puede encenderse o apagarse, pudiendo ser un interruptor para un foco o un reproductor de música. Esto permite a las *SmartApps* identificar a los dispositivos por sus capacidades, y por ende trabajar con una amplia variedad de dispositivos diferentes. Entonces una *SmartApp* puede interactuar con el dispositivo sabiendo que admite los comandos de "on" y "off". Cabe mencionar que los *Device Handlers* generalmente soportan más de una capacidad. Las capacidades se descomponen a su vez en commands (comandos) y attributes (atributos). Algunas capacidades tienen atributos, pero no comandos y viceversa.

Los commands representan las acciones que puede realizar un dispositivo. Por ejemplo, un switch puede encenderse o apagarse y una cerradura puede abrirse o cerrarse. Los comandos se implementan como métodos en el *Device Handler*, y en él deben encontrarse todos los métodos asociados a cada uno de los comandos admitidos.

Los attributes representan los valores de un estado particular de un dispositivo. Por ejemplo, la capacidad "Presence Sensor" tiene el atributo "presence", y los posibles valores que puede tomar son "not present" y "present". Los valores de los atributos se establecen por medio de events (eventos), en donde el nombre del evento es el nombre del atributo, y el valor del evento es el valor del atributo.

Existen dos capacidades que no tienen atributos ni comandos, "Actuator" y "Sensor". La capacidad "Actuator" indica que un dispositivo tiene comandos, y la capacidad "Sensor" establece que un dispositivo tiene atributos. Cuando se está desarrollando un *Device Handler*, es una práctica recomendada la admisión de las capacidades "Actuator" y "Sensor" si el dispositivo tiene comandos o atributos distintos de los ya establecidos.

Los events son fundamentales para la plataforma SmartThings, permiten que las *SmartApps* respondan a los cambios en el entorno físico, los eventos reportan el estado de un dispositivo, por ejemplo, la apertura de una puerta. Las instancias de evento son creadas internamente por la plataforma de SmartThings y son pasadas a los manejadores de eventos de las *SmartApps* que se suscriben a esos eventos.

2.2.3. Modos, ubicaciones y rutinas

Una ubicación representa la localización geográfica de un usuario, como la casa o la oficina. Las ubicaciones no tienen que tener un Hub, pero generalmente lo tienen. Las *SmartApps* y los *Device Handlers* están vinculados a la ubicación en la cual se encuentran instalados. Más adelante se explica la función del Hub dentro de SmartThings.

SmartThings permite a los usuarios especificar que las *SmartApps* se ejecuten

solamente en ciertos modos. Los modos pueden pensarse como filtros para el comportamiento de los dispositivos en el hogar. Los usuarios pueden cambiar la manera en las que las cosas actúan en función del modo en que se encuentre una ubicación. SmartThings viene con los modos pre-configurados "*Home*", "*Away*" y "*Night*". Por ejemplo:

- En el modo *Night*, un sensor de movimiento enciende una luz.
- En el modo *Away*, un sensor de movimiento envía un mensaje de texto y enciende una alarma.

El modo en el que se encuentra una ubicación se puede cambiar por medio de rutinas, y una rutina se puede activar de forma manual o automática. Las rutinas permiten realizar acciones personalizables en diferentes situaciones, como cuando uno duerme, despierta, está de vacaciones, haciendo compras o en la oficina. SmartThings tiene cuatro rutinas por defecto:

- *I'm Back!* Cuando regresas a casa activa el modo *Home*.
- *Good Night!* Cuando vas a dormir activa el modo *Night*.
- *Goodbye!* Cuando sales de casa activa el modo *Away*.
- *Good Morning!* Cuando despiertas activa el modo *Home*.

La rutina *Goodbye!*, por ejemplo, podría ajustar un termostato, la iluminación del hogar y establecer las configuraciones que se cree que son óptimas cuando no hay nadie en casa.

SmartThings permite crear rutinas y modos propios. Las rutinas y los modos pueden parecer redundantes, pero no siempre son lo mismo. Con una rutina se puede ajustar la iluminación y temperatura de una sola habitación en particular cuando alguien sale de ella, y no la temperatura e iluminación de toda la casa (que sería el caso de una automatización creada con el modo *Away*), ya que alguien podría estar saliendo de dicha habitación, pero no estar saliendo de la casa.

2.2.4. Arquitectura

SmartThings define a su Hub o concentrador como el cerebro de tu casa inteligente (Figura 2.3)[3], y es el encargado de la comunicación entre todos los dispositivos, la nube de SmartThings y la aplicación móvil disponible para iOS y Android. El Hub requiere un router conectado a Internet con un puerto Ethernet disponible. SmartThings actualmente admite los protocolos inalámbricos Z-Wave y Zig-Bee, además, es compatible con los estándares LAN y Cloud-to-cloud.

[3]Fuente: `https://theinsidersnet-xznojcftbaaxo07wu.stackpathdns.com/site/public/images/pl_12/70/201508/103702_12082015_314_siteheader.jpg`

Figura 2.3: Samsung SmartThings Hub.

La capa de gestión de conectividad (Connectivity Management) es la encargada de conectar el Hub y los dispositivos móviles del cliente a los servidores de SmartThings y la nube como un todo. En todas las redes existe latencia, es por ello que siempre existirá un retraso entre cuando se solicita que algo suceda y cuando sucede realmente. Para lidiar con esto, la plataforma de SmartThings asume que cuando un comando se activa se ejecutara a su debido tiempo, sin darle ningún tipo de seguimiento. No se puede garantizar la ejecución de un comando ya que de múltiples formas se puede estar interactuando con un dispositivo, así también, es imposible saber el exacto valor actual de un estado, las respuestas a la solicitud de un valor en las *SmartApps* pueden diferir a corto plazo, pero eventualmente son iguales.

La plataforma de SmartThings asume un enfoque de "Cloud First". Esto quiere decir que, para utilizar todos los dispositivos, automatizaciones y asegurar que la aplicación refleje el estado correcto de los dispositivos, el Hub deberá estar conectado a Internet y a la nube de SmartThings. Con el Hub original de SmartThings, todos los *Device Handlers* y las *SmartApps* se ejecutan en la nube de SmartThings. Con el nuevo Hub de SmartThings, ciertos *Device Handlers* y *SmartApps* se ejecutan localmente en el Hub o en la nube de SmartThings. La ubicación de la ejecución varía y es administrada por el equipo interno de SmartThings.

Recapitulando, SmartThings provee una capa de abstracción que es clave para la flexibilidad y funcionalidad de la plataforma; puesto que cuando una *SmartApp* interactúa con un dispositivo a través de su representación virtual, sabe que el dispositivo admite ciertas acciones y reporta ciertos estados con base en sus capacidades. Así que, las *SmartApps* saben que dispositivos son elegibles basándose en las capacidades que proporcionan.

La representación virtual de un dispositivo se denomina *Device Handler* y, desde el punto de vista de la arquitectura, es el puente entre las capacidades genéricas y el protocolo específico utilizado para comunicarse con el dispositivo físico.

En la Figura 2.4[4] se muestra en donde se sitúan los *Device Handlers* en la arquitectura de SmartThings. En el ejemplo mostrado, el *Device Handler* implementa la capacidad switch; se comunica con el dispositivo físico a través de la capa de conectividad (Connectivity Layer) y con las *SmartApps* y la aplicación móvil a través de la capa de gestión de datos (Data Management Layer). La función del *Device Handler* es analizar los mensajes de estado (Status Message) entrantes, los cuales reportan el estado del dispositivo físico, y los convierte en eventos normalizados (Normalized Status). Además, recibe comandos normalizados (Normalized Commands) y los convierte en comandos específicos del protocolo que utiliza el dispositivo físico para efectuar la acción deseada.

Los dispositivos son los componentes principales en la infraestructura de SmartThings y, al crear eventos no hacen otra cosa más que publicar lo que ha sucedido. Las *SmartApps* se configuran con suscripciones que escuchan ciertos eventos, el propósito de las suscripciones es el de coincidir los eventos generados por los *Device Handlers* con los que está usando una *SmartApp* en particular.

[4]Fuente: `http://docs.smartthings.com/en/latest/_images/smartthings-architecture.png`

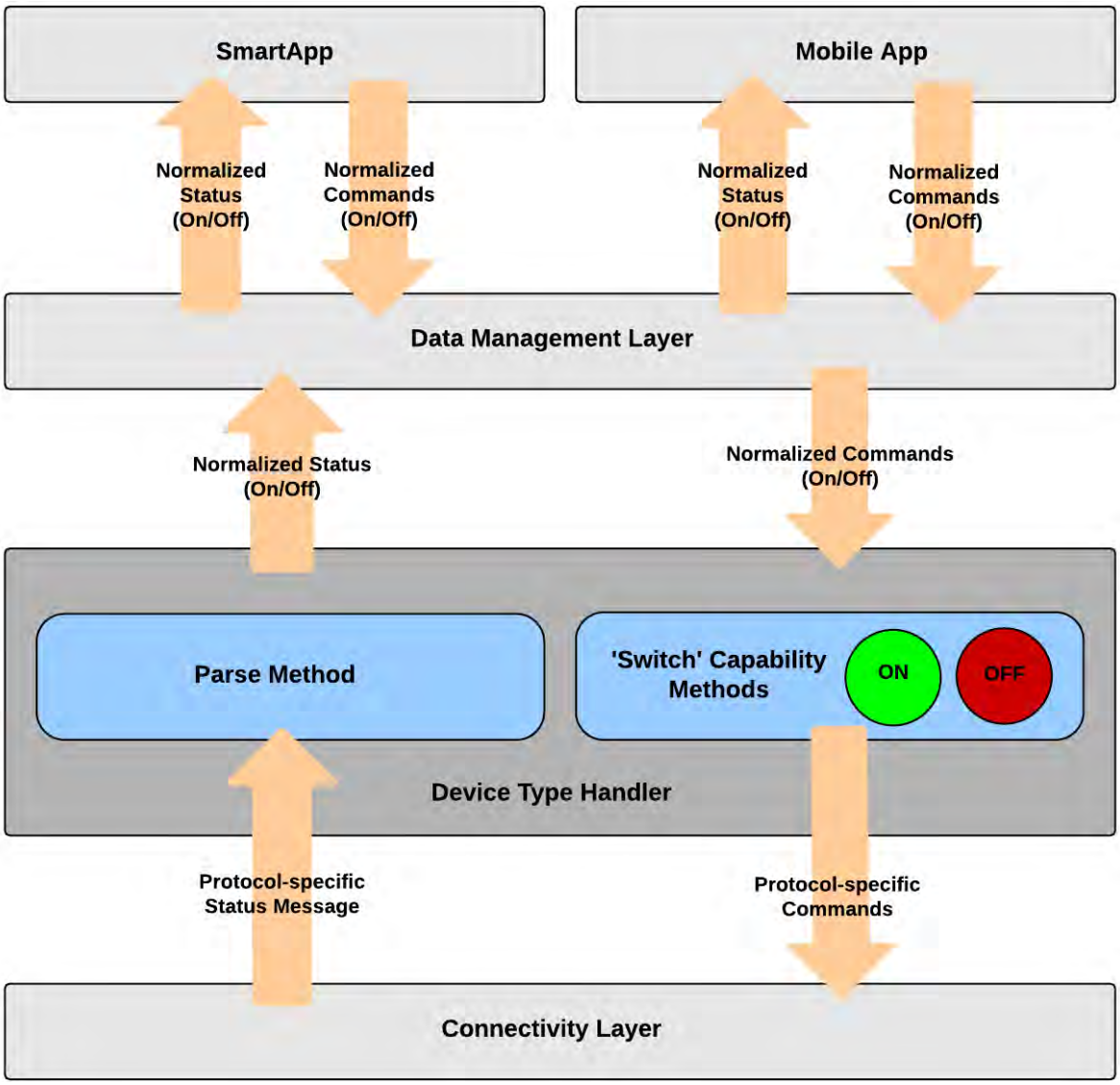

Figura 2.4: *Device Handler* dentro de la arquitectura de SmartThings.

Capítulo 3

Generalidades: Robótica

3.1. Antecedentes y estado de la técnica

Los robots han aparecido múltiples veces en la ciencia ficción; la palabra robot apareció por primera vez en 1920 en la obra Rossum's Universal Robots (R.U.R) del periodista y dramaturgo checoslovaco Karel Čapek (1890-1938), el término robot se utilizaba para nombrar a los humanoides artificiales que ayudaban a los seres humanos en tareas físicamente difíciles. La palabra robot es un neologismo derivado etimológicamente de la palabra checa robota, una palabra arraigada en el antiguo lenguaje eslavo y que significa algo así como el trabajo obligatorio de un siervo, trabajo forzado o esclavitud. El enfoque artístico que se le ha dado a los robots se ha centrado en bípedos mecánicos antropomorfos que pueden hablar, caminar, ver y escuchar, dicho enfoque ha sido el tema de producciones cinematográficas como I, Robot, basada en las historias de Isaac Asimov, la película A.I. de Stanley Kibric, y Star Wars, por mencionar algunos ejemplos. Los robots presentados en la ciencia ficción son como los mostrados en la Figura 3.1 [5]. [6–9]

En un contexto actual, y más allá de su definición literal, el término robot se usa para denotar máquinas compuestas por elementos mecánicos y electrónicos, y dotadas de un sistema informático para su programación, percepción del entorno y control en tiempo real. La robótica es la ciencia de percibir y manipular el mundo físico a través de dispositivos mecánicos controlados por computadora, estudia aquellas máquinas que pueden reemplazar a los seres humanos en la ejecución de tareas, tanto en lo referente a la actuación física como a la toma de decisiones. Un enfoque completo de la robótica yace en la intersección de la mecánica, electrónica, procesamiento de señales, ingeniería de control, computación y modelado matemático. [6] [10] [11]

[5] Fuente: [9]

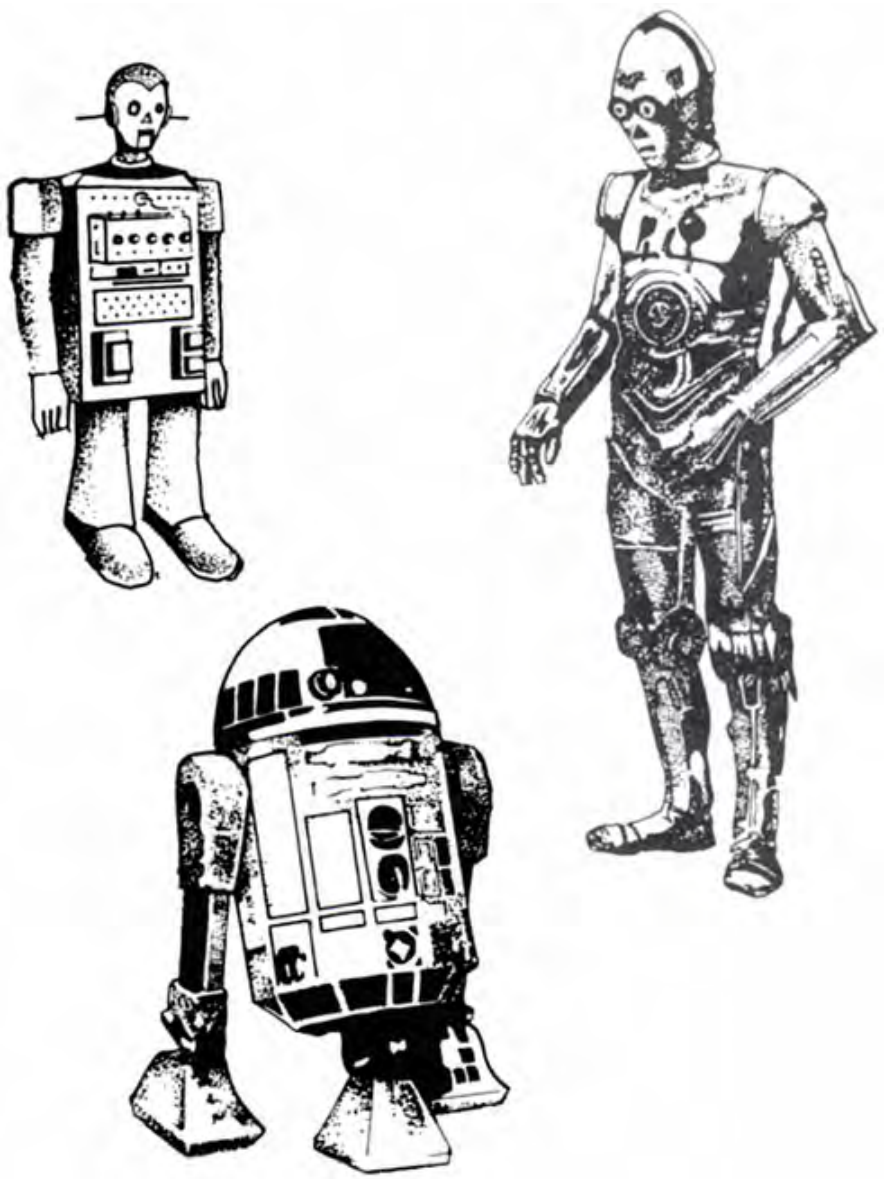

Figura 3.1: Robots en la ciencia ficción.

Es importante recordar que el término robot nace asociado a la idea de trabajo y producción. Los primeros robots esencialmente combinaron articulaciones mecánicas de teleoperadores con la autonomía y programabilidad de las máquinas CNC (Computer numerical control). Los teleoperadores (manipuladores teleoperados) se desarrollaron en los años cuarenta para manejar materiales radiactivos, consistían en un par de pinzas "maestra" y "esclava", acopladas por mecanismos que permitían que la pinza esclava reprodujera los movimientos de la pinza maestra accionada por un operador. Las máquinas herramienta de control numérico, son un tipo de máquina que se utiliza para dar forma a piezas sólidas, por lo general metales, y se desarrollaron a finales de los años cuarenta y principios de los cincuenta. En los años ochenta y noventa se diseñaron un gran número

de máquinas cuyo objetivo era realizar tareas en lugares difícilmente accesibles, en condiciones peligrosas para la salud o llevar a cabo trabajos difíciles para el ser humano. Así, se han abordado aplicaciones como manipulación y transporte de materiales peligrosos, construcción, cirugía, exploración espacial y actividades subacuáticas. A continuación, se enlistan algunos hitos en la historia de la robótica que nos llevan hasta la tecnología actual de los robots. [10] [12]

- 1947 - Se desarrolló el primer teleoperador accionado por energía eléctrica.
- 1954 – El ingeniero americano George Devol diseñó el primer robot programable, considerado como el primer robot industrial.
- 1946 - Joseph Engelberber, compró los derechos del robot de Devol y fundó la compañía Unimation.
- 1961 - Se desarrolló el primer robot que incorporó force feedback.
- 1963 - Se desarrolló el primer sistema de visión robótica.
- 1976 - Se desarrolló el dispositivo Remote Center Compliance (RCC) para la inserción de piezas para ensamblaje en Draper Labs en Boston.
- 1978 - Unimation presentó el robot PUMA (Programmable Universal Machine for Assembly), basado en diseños de un estudio de General Motors (Figura 3.2)[6].

Figura 3.2: Robot PUMA (Programmable Universal Machine for Assembly).

[6]Fuente: `http://datapeak.net/images/1978puma.jpg`

- 1979 - El diseño del robot SCARA (Selective Compliant Assembly Robot Arm o Selective Compliant Articulated Robot Arm) se introdujo en Japón (Figura 3.3)[7].

Figura 3.3: Robot SCARA (Selective Compliant Assembly Robot Arm o Selective Compliant Articulated Robot Arm).

- 1986 - El robot submarino Jason, del Instituto Oceanográfico de Woods Hole, exploró el naufragio del Titanic (Figura 3.4)[8].

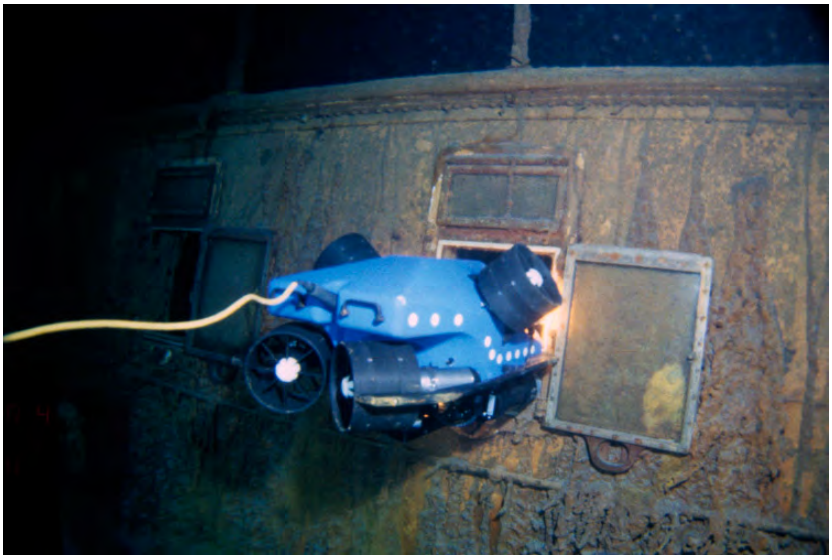

Figura 3.4: Robot Jason.

[7]Fuente: https://www.qyresearchgroups.com/media/backend/userfiles/1496652263.jpg

[8]Fuente: http://www.whoi.edu/cms/images/oceanus/1vehicleJJ_211755.jpeg

- 1993 - El robot experimental ROTEX de la Agencia Aeroespacial Alemana (DLR) voló a bordo del transbordador espacial Columbia y realizó una variedad de tareas bajo los modos teleoperado y fuera de línea basado en sensores.
- 1996 - Honda reveló su robot humanoide; un proyecto iniciado en secreto en 1986 (Figura 3.5)[9].

Figura 3.5: Robot ASIMO.

- 1996 - El laboratorio de Bio-Robótica adquirió el robot TX8.
- 1997 - Se celebró en Nagoya, Japón la primera competencia de robots de fútbol, RoboCup-97, y atrajo a 40 equipos de todo el mundo (Figura 3.6)[10].

Figura 3.6: RoboCup-97 Nagoya, Japón.

[9]Fuente: `http://world.honda.com/ASIMO/history/image/bnr/bnrL_history.jpg`
[10]Fuente: `https://www.sonycsl.co.jp/person/kitano/RoboCup/pict1.jpg`

- 1997 - El robot móvil Sojourner viajó a Marte para la misión NASA's Mars PathFinder.
- 2001 - Se realizó la primera tele-cirugía en Nueva York, se efectuó una extracción de la vesícula biliar laparoscópica de una mujer en Estrasburgo, Francia.
- 2006 - En el laboratorio de Bio-Robótica se desarrolló el robot TPR8.
- 2009 - En el laboratorio de Bio-Robótica se desarrolló el robot Pak-ito.
- 2011 - En el laboratorio de Bio-Robótica se desarrolló el robot Al-ita.
- 2011 - En el laboratorio de Bio-Robótica se comenzó a desarrollar el robot Justina (Figura 3.7).

Figura 3.7: Robot Justina.

La mayoría de los robots que existen hoy en día son los robots fijos o también llamados industriales, los cuales realizan tareas repetitivas, precisas y con gran velocidad. Estos robots se desempeñan en la automatización industrial moderna y se encargan de ejecutar tareas en un entorno controlado, sin embargo, estos robots

sufren la desventaja de la falta de movilidad; un manipulador fijo tiene un rango de movimiento delimitado, por el contrario, un robot móvil podría navegar a través de una planta de fabricación. Parece probable que conforme se desarrolla la tecnología, el número de robots móviles aumentará y será mucho más notable a medida que requieran ser atendidas tareas en un entorno no controlado. Los robots pueden clasificarse de acuerdo a su movilidad de la siguiente manera: [6] [8] [13]

- Robots manipuladores (fijos)
- Robots móviles
  - Robots de tierra
    - Robots con ruedas
    - Robots con patas
  - Robots submarinos
  - Robots aéreos

Las aplicaciones más importantes de la robótica móvil son fuera de las cadenas de fabricación industrial, tales como su uso en accidentes nucleares, localización de naufragios y viajes espaciales. Un sector creciente en las aplicaciones de la robótica es el de los robots de servicio, que incluye a los robots domésticos, de ayuda a discapacitados y asistencia en general. [10]

Tomando como referente la autonomía, los robots móviles tienen como precedente la tortuga de Walter presentada en 1948, la cual podía subir pendientes, reaccionar ante la presencia de obstáculos y dirigirse hacia una posición de recarga cuando la alimentación empezaba a ser insuficiente (Figura 3.8)[11]. En los años setenta se trabajó en el desarrollo de robots móviles empleando sistemas de visión, y en los ochenta el incremento de la capacidad computacional, el desarrollo de nuevos sensores y sistemas de control, permitieron mejorar la autonomía; en esta década cabe destacar los robots móviles para interiores y exteriores desarrollados en la Carnegie Mellon University, Pittsburgh, USA. [10]

Los robots pueden clasificarse de acuerdo a su grado de autonomía en teleoperados, autónomos y de funcionamiento repetitivo. En los robots teleoperados las tareas de planificación, percepción del entorno y manipulación son realizadas por los humanos. Algunos sistemas proveen al operador realimentación del entorno por medio de imágenes o fuerzas. Los robots de funcionamiento repetitivo son los que se utilizan en la producción industrial. Realizan comúnmente tareas invariantes y tienen una limitada o nula percepción del entorno. Son precisos, rápidos e incrementan la productividad, evitando al hombre en ocasiones trabajos peligrosos. Los robots autónomos son capaces de percibir, modelar el entorno, planificar y actuar para lograr objetivos sin la intervención del ser humano. Además, pueden trabajar en entornos poco estructurados y dinámicos. [10]

[11] Fuente: `http://davidbuckley.net/DB/HistoryMakers/Elsie.075.jpg`

Figura 3.8: Tortuga de Walter.

3.2. Robots móviles

La creación de un sistema autónomo e inteligente capaz de integrarse en la vida cotidiana de los seres humanos, detectar, aprender e interactuar con su entorno ha sido de interés en campos como la inteligencia artificial, teoría de control y robótica. Los robots móviles surgen a partir de la necesidad de extender el alcance restringido de los robots industriales, los cuales son, en términos simples, una estructura mecánica anclada en uno de sus extremos. La autonomía de un robot móvil está basada en el sistema de navegación automática, lo que implica la realización de tareas de planificación, percepción y control. La percepción involucra la obtención de información, su tratamiento e interpretación. En los robots móviles, en el caso más general, la planificación puede descomponerse en encontrar una ruta y evadir obstáculos no esperados. La misión de un robot para interiores podría consistir en desplazarse hasta una habitación en particular, calculando la ruta a una posición final desde la posición inicial en la que se encuentra el robot. [10] [14]

La eficiente navegación de un robot móvil está dada principalmente por su capacidad para percibir e interactuar con el entorno que lo rodea. Una característica deseable que debe tener un robot móvil es poder localizarse en su espacio de trabajo, reconocer puntos de referencia y objetos que lo rodean. Este conocimiento es importante para la conclusión exitosa de tareas de navegación. Los comportamientos tradicionales, reactivos e híbridos son modelos que dictan la forma en la que los robots ejecutan sus tareas. [15]

En los comportamientos tradicionales se tiene una representación o modelo previo del medio ambiente, por lo que es necesario un método para la generación de uno. Al disponer de una descripción de los elementos presentes en el entorno, el problema reside en su interpretación y adaptación de las lecturas de los sensores a los modelos previamente dados. Asimismo, si de antemano se sabe que ciertos objetos son fijos y estarán siempre presentes, pueden ser utilizados como referencia para la localización. A partir de la representación del entorno, el robot planea las acciones y decide que ejecutar para cumplir con sus objetivos. La representación del ambiente siempre debe estar actualizada y bien definida, sin embargo, dicha representación no siempre es exacta debido a que los ambientes por lo general son dinámicos y a la existencia de ruido durante su construcción. [15]

En los comportamientos reactivos no es necesario tener un modelo del entorno, únicamente se utiliza la información del medio ambiente obtenida por los sensores para realizar una acción, esto permite tener una respuesta rápida en ambientes dinámicos. Los comportamientos reactivos se basan en responder de cierta forma ante cierto estímulo, es decir, si se percibe algo entonces se reacciona de una forma en particular. A diferencia de los comportamientos tradicionales no requiere una planeación de acciones y los requerimientos computacionales son menores.

Los comportamientos híbridos combinan los comportamientos tradicionales y los reactivos, es decir, se cuenta con una representación del medio ambiente para la toma de decisiones y la información obtenida por los sensores dispara ciertas acciones. Adicionalmente, se debe tener una coordinación de ambos comportamientos.

En la Figura 3.9 [12] se muestra el esquema básico de un robot móvil, y se observa un sistema mecánico, sensores, actuadores y un sistema de control. El sistema de control tiene bucles de realimentación de la información suministrada por los sensores internos. Los sensores internos (sensores propioceptivos) miden el estado de la estructura mecánica, en los robots móviles con ruedas uno de los sensores más utilizados son los encoders, con los que se puede calcular la posición del robot mediante la medición de la posición angular de sus ruedas. Los sensores externos (sensores exteroceptivos) suministran información del entorno, en los robots móviles los láseres son utilizados para la detección de obstáculos. Los actuadores son los encargados de generar fuerzas o pares necesarios para mover la estructura mecánica. [10]

[12] Fuente: [10]

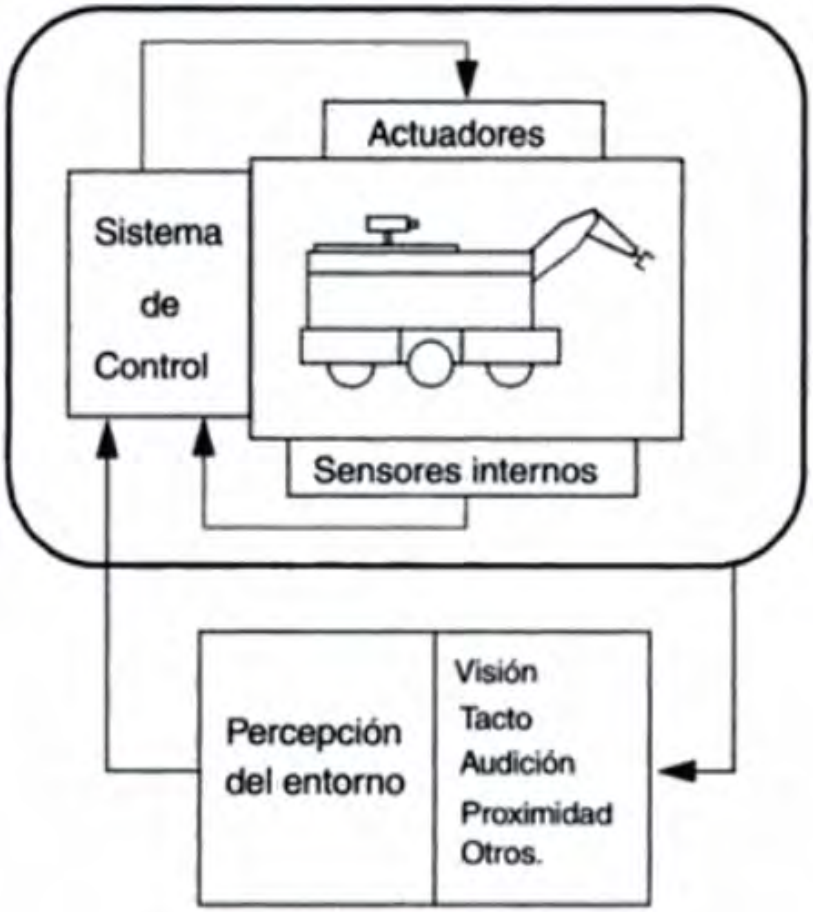

Figura 3.9: Esquema básico de un robot móvil.

Utilizando los comportamientos tradicionales se asume un entorno conocido y estático, en particular en este trabajo se modela el espacio en el medio ambiente descomponiéndolo en celdas de ocupación, lo cual permite encontrar una ruta libre de obstáculos, y cuando se planifica la ruta se tiene como objetivo de control el seguimiento de la misma. Es sabido que la aplicación de una ley de control en lazo cerrado requiere la medición de la o las variables a controlar, y en un robot móvil son aquellas variables que determinan su posición y orientación, una técnica para la medición de estas variables es la odometría, de igual forma es necesario resolver el problema de regulación de la rapidez del robot móvil. El control de un robot móvil puede resultar más complejo que el de los robots manipuladores, puesto que los lazos de control se plantean en el espacio de las variables articulares y en las coordenadas del mundo. En la actualidad las ecuaciones de movimiento que se emplean generalmente se basan en métodos geométricos y modelos cinemáticos simplificados. [10]

El sistema de percepción de un robot móvil autónomo tiene como objetivo estimar la posición y orientación del vehículo, lograr una navegación segura y modelar el entorno para obtener una representación o un mapa del mismo. [10]

3.2.1. Odometría y otros métodos de localización

A diferencia del caso de los robots manipuladores, en los que los encoders de las articulaciones proporcionan una medición directa de la configuración del robot, los robots móviles están instrumentados con encoders que miden la rotación de sus ruedas, pero no directamente la posición y la orientación del vehículo con respecto a un marco de referencia global. Es por lo tanto que es necesario un procedimiento de localización que estime en tiempo real la configuración del robot móvil. [6]

Dead reckoning, término derivado de "deduced reckoning" (cálculo deducido), es un método matemático para determinar la ubicación actual de un buque o barco mediante su posición previa y, por medio del rumbo conocido y la información de velocidad durante un periodo de tiempo determinado. Hoy en día, un gran número de robots móviles terrestres basan su estrategia de navegación en dead reckoning y, periódicamente anulan los errores acumulados con arreglos recurrentes de diversas ayudas de navegación. [16]

La implementación más simple de dead reckoning es denominada en ocasiones odometría, e implica que el desplazamiento de un vehículo a lo largo de la trayectoria desplazada se calcula a partir de un odómetro a bordo del mismo. Los encoders ópticos acoplados al eje de un motor o a los ejes de las ruedas son un medio común de instrumentación de odometría. Existen diferentes sensores que cuantifican con exactitud la posición angular entre los que se encuentran los potenciómetros, encoders magnéticos, encoders inductivos, encoders capacitivos, pero para aplicaciones de robots móviles, los encoders ópticos incrementales y absolutos son los más populares hoy en día. [16]

La exactitud de las mediciones de odometría es en gran medida una función directa del diseño cinemático del vehículo. Una ventaja de la odometría es que siempre es capaz de proporcionar una estimación de la posición del vehículo, pero tiene como desventaja que el error de posición crece sin límite a menos que se utilice periódicamente una referencia independiente para reducirlo. [16]

La odometría comúnmente se calcula en intervalos de tiempo periódicos, ya que se asume que el robot se mueve a velocidad lineal y velocidad angular constantes dentro de cada intervalo de muestreo. Otro método para estimar la ubicación del robot consiste en la integración el modelo cinemático mediante un método numérico, puesto que conocemos las velocidades linear y angular del robot, ya que son las velocidades con las que deseamos que se mueva el robot. La experiencia práctica sugiere que la odometría aun siendo errónea, suele ser más precisa que la integración del modelo cinemático. Ambos métodos sufren de deriva, pero el método de integración sufre adicionalmente de la falta de coincidencia entre las velocidades reales logradas por las leyes de control y las velocidades deseadas. [6] [11]

La navegación inercial es un método que utiliza acelerómetros y giroscopios para medir la aceleración y velocidad de rotación, las mediciones se integran una o dos veces, según sea el caso, para obtener la posición; sin embargo, tiene como desventaja que los datos fluctúan con el tiempo debido a la necesidad de integrar, por lo que la acumulación de error puede ser muy grande. No obstante, al combinar este método con la odometría puede dar buenos resultados en intervalos de tiempo no muy prolongados y en pequeñas distancias viajadas. [16]

La utilización de sensores internos (sensores propioceptivos) para la estimación de la ubicación se conoce como localización pasiva. En la práctica, la localización odométrica está sujeta a un error que crece a lo largo del tiempo (deriva) y se vuelve rápidamente significativo a lo largo de trayectos suficientemente grandes. Este error es resultado de varias causas, entre las que se encuentran el deslizamiento de las ruedas y la inexactitud de parámetros como el radio de las ruedas. [6]

Una solución más robusta se obtiene por medio de métodos de localización activa. Este tipo de solución se puede usar cuando el robot está equipado con sensores exteroceptivos de proximidad (como un láser) y además se conoce un mapa del espacio de trabajo, ya sea que sea dado previamente o sea construido por el robot durante la navegación (movimiento). Entonces es posible corregir la estimación que proporcionan los métodos de localización pasiva comparando las lecturas esperadas por los sensores exteroceptivos con las lecturas reales. Estas técnicas, tales como el Filtro Extendido de Kalman (Extended Kalman Filter) y el Filtro de Partículas (Particle Filter), proveen mayor precisión que la localización odométrica pura y, por ende, son esenciales en tareas de navegación a lo largo de trayectos largos. [6]

3.2.2. Locomoción

Un robot móvil requiere mecanismos de locomoción que le permitan moverse a través del entorno, hoy en día existe una gran variedad de posibles maneras de moverse. La mayoría de los mecanismos de locomoción han sido inspirados en seres biológicos, por lo que hay robots que pueden caminar, saltar, correr, deslizar, nadar y volar. Los sistemas biológicos tienen éxito al moverse en una amplia variedad de terrenos irregulares, por lo que resulta deseable copiar sus mecanismos de locomoción, por otro lado, resulta difícil replicar su naturaleza. En primer lugar, la complejidad mecánica se logra fácilmente en los seres biológicos, los insectos tienen un tamaño y peso muy pequeños, alcanzan un nivel de robustez que no se ha podido igualar con técnicas de fabricación humana, igualmente existen insectos que cuentan con un gran número de patas y tratándose de estructuras o mecanismos manufacturados, estos deben de fabricarse individualmente. Finalmente, los sistemas de almacenamiento de energía biológica y de activación muscular utilizados por animales e insectos superan los sistemas creados por el hombre. [13]

Sin embargo, existe una excepción a los mecanismos de locomoción inspirados en la naturaleza, la rueda es un invento humano que consigue buenos resultados en terreno plano. Por simplicidad, los robots móviles suelen utilizar mecanismos con ruedas o un pequeño número de patas articuladas. Pero en general, la locomoción con patas requiere más grados de libertad y por ende mayor complejidad mecánica que la locomoción con ruedas. La eficiencia de la locomoción con ruedas depende de las cualidades del ambiente, en particular de la dureza y la planicidad del suelo. La naturaleza favorece la locomoción con patas, puesto que en la naturaleza el terreno es áspero y no estructurado, mientras que en el entorno de los seres humanos las superficies son planas en interiores y exteriores, por lo que en aplicaciones de robótica móvil es común encontrar la locomoción con ruedas, aunque recientemente ha habido progreso en robots con patas para ambientes naturales al aire libre. [13]

La rueda ha sido el mecanismo de locomoción más popular en los robots móviles de tierra y, su implementación mecánica es relativamente simple. La tracción y la maniobrabilidad son dos puntos importantes que suelen tratarse en la locomoción con ruedas, así como el problema del deslizamiento que existe dependiendo de las características del terreno. Los robots móviles emplean diferentes tipos de configuración que les proporcionan distintas características. Los robots móviles omnidireccionales tienen la mayor maniobrabilidad en el plano, ya que son capaces de trasladarse independientemente y de forma simultánea en los ejes del sistema coordenado, así como rotar sobre su propio eje. A continuación, se mencionan las configuraciones comúnmente utilizadas en los robots móviles con ruedas. [10] [13]

3.2.2.1. Configuración single wheel drive

En la configuración single wheel drive o también conocida como triciclo clásico, se tiene una sola rueda al frente para la tracción y el direccionamiento y, requiere de dos ruedas giratorias pasivas en la parte trasera (Figura 3.10) [13]. Para conducir en línea recta la rueda delantera se posiciona en la orientación media y se acciona con la velocidad deseada. Para conducir en curva, la rueda se coloca en un ángulo que coincida con la curva deseada. En la Figura 3.11 [14] se muestra el movimiento del robot para diferentes ajustes de dirección, con la rueda ajustada a 90° el robot girará en torno a un punto medio entre las dos ruedas pasivas. Debido a la simplicidad en su control, es una configuración frecuentemente utilizada en robots móviles para interiores y exteriores con planicidad. El cálculo de la odometría es equivalente al de la configuración Ackerman, discutida más adelante, donde la rueda direccionable reemplaza la rueda imaginaria central en la configuración Ackerman. [10] [17]

[13] Fuente: [16]
[14] Fuente: [17]

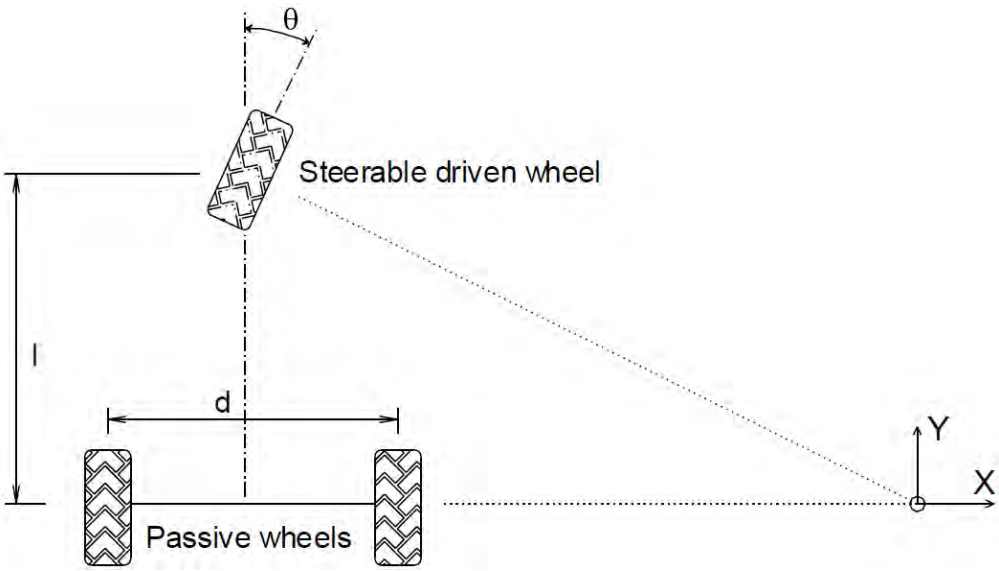

Figura 3.10: Configuración triciclo clásico.

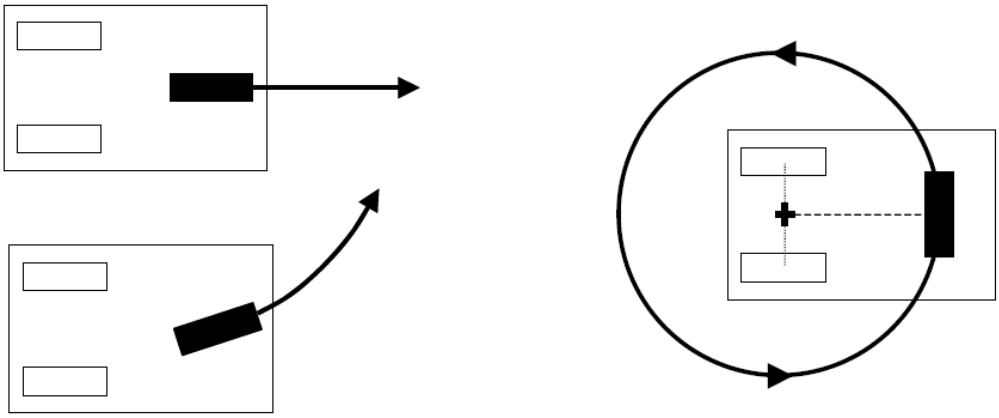

Figura 3.11: Direccionamiento en la configuración triciclo clásico.

3.2.2.2. Configuración differential drive

En la configuración de direccionamiento diferencial o differential drive se tienen dos motores en posiciones fijas en el lado izquierdo y derecho del robot, accionando independientemente una rueda cada uno. Este diseño además necesita de una, dos o cuatro ruedas pasivas adicionales, dependiendo de la ubicación de las ruedas motrices (Figura 3.12)[15]. Esta configuración es mecánicamente más simple que la de triciclo clásico, puesto que no requiere la rotación de la rueda accionada. Por otro lado, el control en la conducción es más complejo ya que requiere la coordinación de dos ruedas motrices. En esta configuración los encoders cuentan las revoluciones de las ruedas accionadas para calcular mediante ecuaciones geométricas la posición

[15]Fuente: [16]

del vehículo, es así que se estima la posición actual a partir de una posición previa conocida. [16] [17]

La configuración diferencial es la más utilizada en robots para interiores, y el direccionamiento viene dado por la diferencia de rapideces de las ruedas laterales de tracción. En la Figura 3.13 [16] se muestra las posibles formas de direccionamiento que se obtienen al modificar las rapideces de las ruedas laterales. Si ambos motores funcionan a una misma rapidez diferente de cero, el robot se mueve hacia adelante o hacia atrás, si un motor tiene una mayor rapidez con respecto del otro el robot se mueve en una curva a lo largo de una circunferencia y si ambos motores operan a la misma rapidez, pero en direcciones opuestas entonces, el robot gira sobre sí mismo. [10] [17]

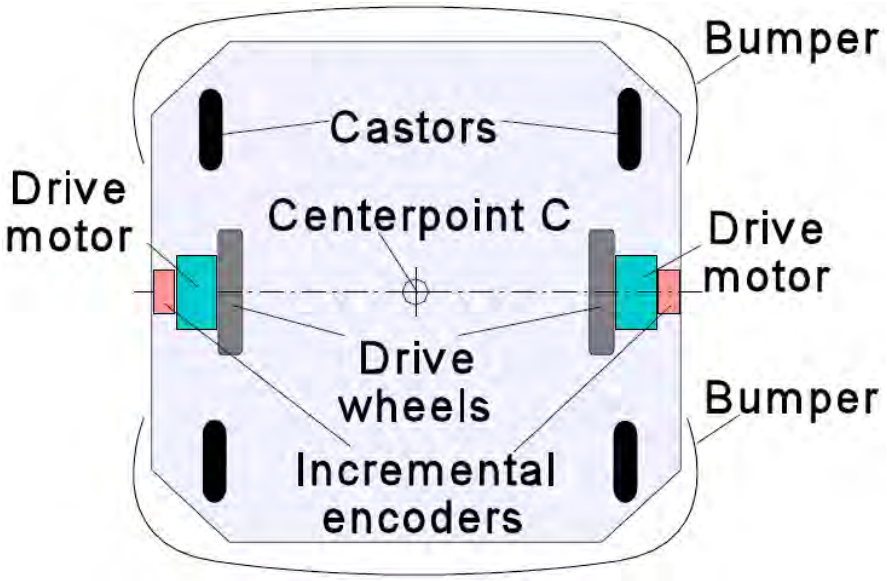

Figura 3.12: Direccionamiento diferencial.

[16]Fuente: [17]

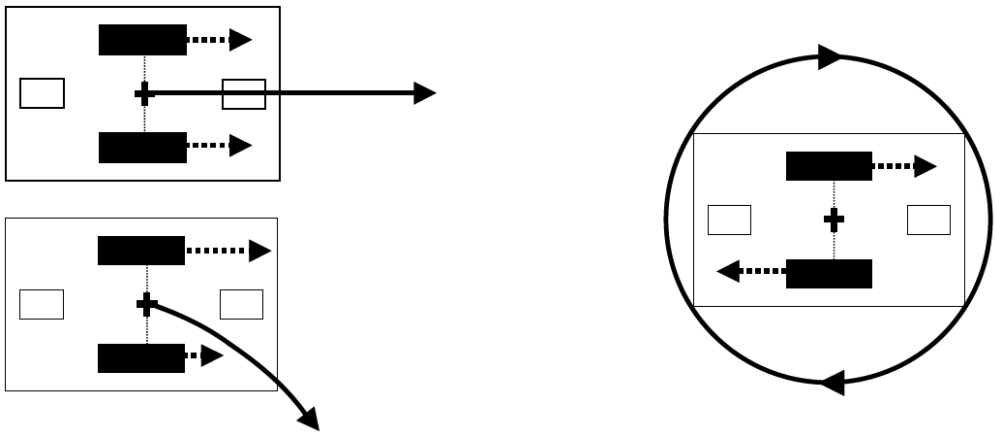

Figura 3.13: Direccionamiento en la configuración differential drive.

3.2.2.3. Configuración Ackerman

Utilizado casi exclusivamente en la industria del automóvil, la configuración Ackerman consiste en dos ruedas traseras conectadas para la tracción y dos ruedas delanteras conectadas para el direccionamiento. La dirección está diseñada para que cuando el vehículo gire la rueda delantera que queda en el interior gire un ángulo ligeramente mayor que la rueda exterior, eliminado de este modo el deslizamiento. Como se puede apreciar en la Figura 3.14 [17] los ejes extendidos de todas las ruedas se intersecan en un punto común, y el lugar geométrico que describen los puntos de contacto de cada una de las ruedas con el suelo es un conjunto de arcos concéntricos en el punto P_1, y todos los vectores instantáneos de velocidad son tangenciales a dichos arcos. [16] [17]

Por conveniencia, el ángulo de dirección del vehículo se considera como el ángulo asociado a la rueda imaginaria situada en el punto de referencia P_2 tal y como se muestra en la Figura 3.14. En comparación con la configuración de direccionamiento diferencial, en la configuración de Ackerman la conducción en línea recta no es un problema, ya que las ruedas traseras son accionadas a través de un eje en común, sin embargo, el vehículo no puede girar sobre sí mismo y las ruedas motrices traseras experimentan deslizamiento en las curvas. [16] [17]

[17] Fuente: [16]

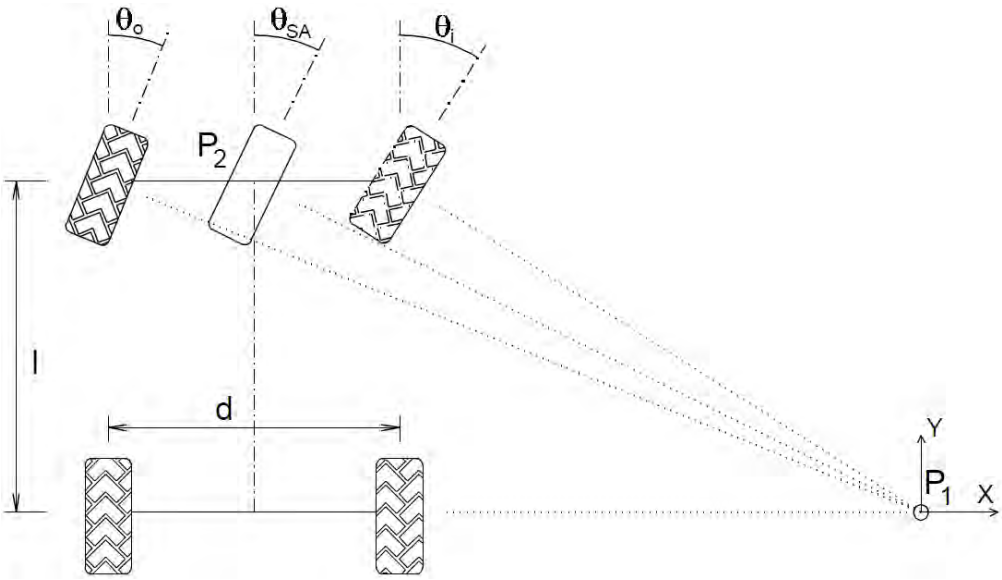

Figura 3.14: Configuración Ackerman.

3.2.2.4. Configuración Skid Steer

En la configuración skid steer se tienen dos o más ruedas en cada lado del robot y actúan de manera simultánea, es decir, el movimiento del robot resulta de combinar las velocidades de las ruedas de la izquierda con las velocidades de las ruedas de la derecha (Figura 3.15)[18]. Esta configuración es semejante a la configuración de direccionamiento diferencial, pero con la diferencia de que al agregarle más ruedas se tienen más puntos de contacto con el suelo y se produce deslizamiento cuando el robot móvil gira. Así mismo, en comparación con la configuración diferencial, resulta más difícil su control puesto que hay que lograr que todas las ruedas de cada lado se muevan con la misma rapidez. [10]

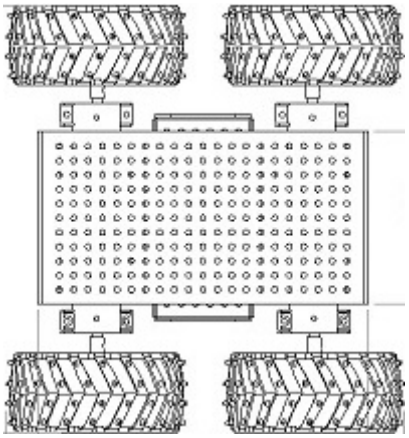

Figura 3.15: Configuración Skid Steer.

[18]Fuente: `http://klinikrobot.com/images/stories/4wd%20dagu%20wild%20thumper%20dimensions%20with%20wheels.jpg`

3.2.2.5. Configuración con pistas de deslizamiento

Esta configuración puede ser vista como una configuración funcionalmente análoga a un robot móvil con direccionamiento diferencial, o como la configuración skid steer. La tracción y el direccionamiento se obtienen por medio de pistas de deslizamiento y se catalogan como vehículos tipo oruga (Figura 3.16) [19]. En comparación con las dos configuraciones mencionadas, se tiene como diferencia una mejor maniobrabilidad del robot en terrenos irregulares, y mayor fricción durante los giros a causa de que con las pistas se tienen múltiples puntos de contacto con la superficie. Una de las aplicaciones de este tipo de robot es su uso como robot de rescate en zonas de desastre. [10] [17]

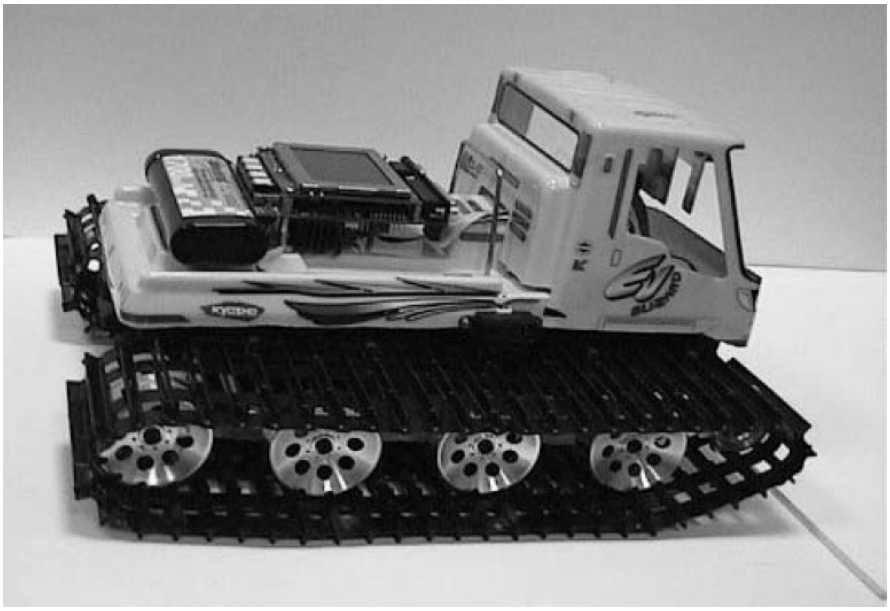

Figura 3.16: Configuración con pistas de deslizamiento.

3.2.2.6. Configuración Synchro Drive

La configuración síncrona es una extensión de la configuración con una sola rueda para tracción y direccionamiento. Sin embargo, se tienen tres ruedas que son actuadas y orientadas simultáneamente, las tres ruedas giran de manera que siempre apuntan a la misma dirección (Figura 3.17)[20]. La transmisión para la actuación se puede conseguir mediante coronas de engranajes, con correas concéntricas, o puede lograrse utilizando una cadena y un solo motor para la dirección y un único motor para accionar todas las ruedas. Este tipo de vehículo se puede conducir en cualquier dirección deseada, razón por la que por lo general tienen una forma cilíndrica. [10] [17]

[19]Fuente: [17]
[20]Fuente: [17]

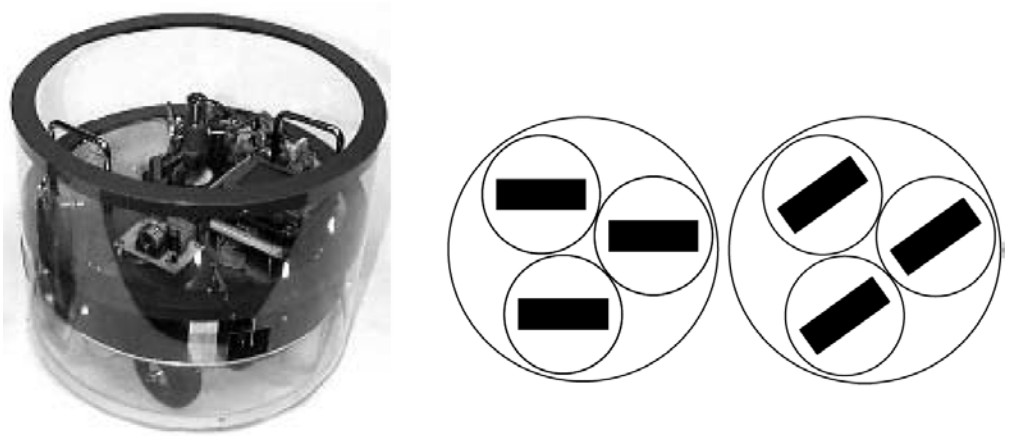

Figura 3.17: Configuración Synchro Drive.

3.2.2.7. Configuración Omnidireccional

Esta configuración requiere el empleo de ruedas especiales, llamadas ruedas suecas o ruedas Mecanum. La superficie de estas ruedas está cubierta por rodillos rodantes que se mantienen en su lugar por medio de rodamientos de bolas y pueden girar libremente alrededor de su eje. Es importante mencionar que la rueda es accionada por un motor, pero los rodillos en la superficie de la rueda no. En la Figura 3.18 [21] se muestran ruedas Mecanum con rodillos a $\pm 45°$ con respecto del eje de la rueda, y en la Figura 3.19 [22] se muestran ruedas Mecanum con rodillos a 90° con respecto del eje de la rueda. [10] [17]

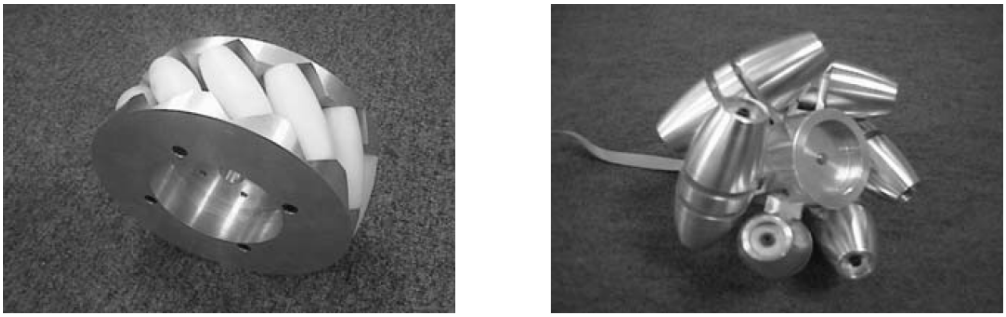

Figura 3.18: Ruedas Mecanum con rodillos a 45°.

[21] Fuente: [17]
[22] Fuente: [17]

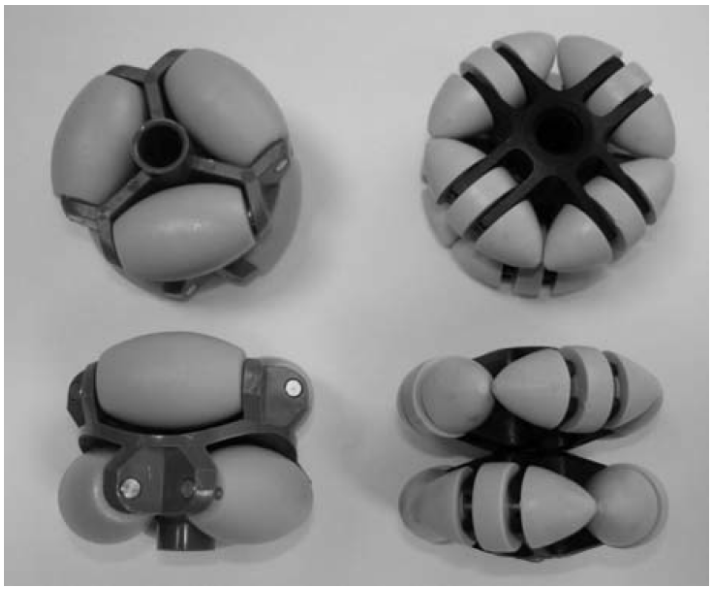

Figura 3.19: Ruedas Mecanum con rodillos a 90°.

Un robot omnidireccional puede ser construido con tres o cuatro ruedas Mecanum actuadas independientemente (Figura 3.20)[23]. Los diseños con tres ruedas requieren ruedas Mecanum con rodillos a 90° con respecto al eje de la rueda, y el diseño con cuatro ruedas necesita dos ruedas zurdas (rodillos a +45° con respecto al eje de la rueda) y dos ruedas derechas (rodillos a −45° con respecto al eje de la rueda). La configuración omnidireccional permite desplazarse en cualquier dirección en el plano, es decir, hacia adelante, hacia atrás, en curva, de forma lateral y girar sobre sí mismo (Figura 3.21)[24]. [17]

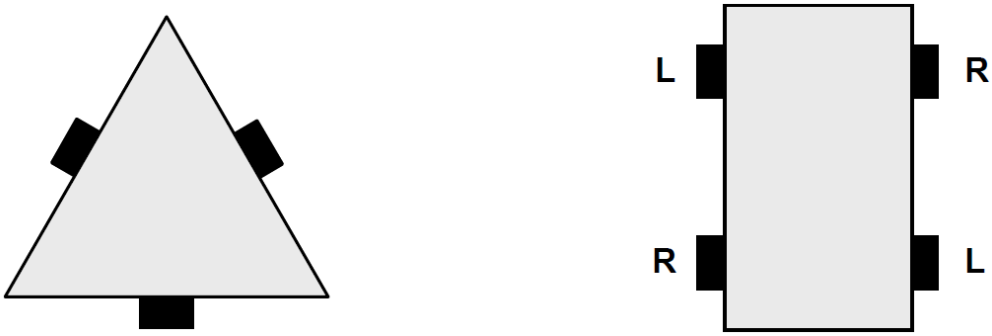

Figura 3.20: Configuración omnidireccional con tres y cuatro ruedas.

[23]Fuente: [17]
[24]Fuente: [17]

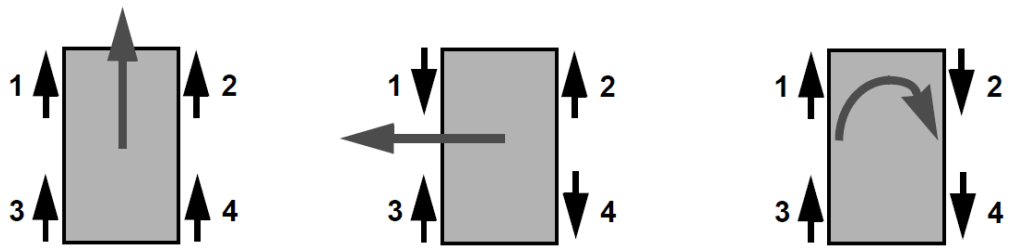

Figura 3.21: Direccionamiento en la configuración omnidireccional de cuatro ruedas.

3.2.3. Restricciones holonómicas y no holonómicas

Las ruedas son por mucho el mecanismo más común de locomoción en los robots móviles. Los vehículos con ruedas están sujetos a restricciones cinemáticas que reducen en general su movilidad local, dejando intacta la posibilidad de alcanzar cualquier configuración mediante maniobras apropiadas. Por ejemplo, cualquier conductor sabe por experiencia que, aunque es imposible mover un carro de forma lateral, aún es posible estacionarse arbitrariamente en cualquier lugar, al menos en ausencia de obstáculos. [6]

Antes de discutir acerca de las restricciones holonómicas y no holonómicas, es necesario definir algunos términos. Un cuerpo rígido es un sistema caracterizado por la restricción de que la distancia entre dos puntos cualquiera siempre es constante. Ahora considérese un sistema B_r de r cuerpos rígidos y asúmase que todos los elementos de B_r pueden alcanzar cualquier posición en el espacio. Para encontrar de manera única la posición de todos los puntos del sistema, es necesario asignar un vector $x = [x_1 \cdots x_p]^T$ de $6r = p$ parámetros, llamados configuración; por cada cuerpo son necesarios seis parámetros, tres para la posición y tres para la orientación. Estos parámetros son denominados coordenadas generalizadas del sistema sin restricciones B_r, y p determina el número de grados de libertad (GDL). Cualquier limitación en la movilidad del sistema B_r es llamada restricción. [6]

Los sistemas no lineales a menudo tienen restricciones que limitan el movimiento del sistema y restringen las trayectorias que puede seguir. Una restricción holonómica se define como cualquier restricción que puede ser expresada de la forma: [18] [19]

$$F(q,t) = 0 \tag{3.1}$$

Donde $q = [q_1, q_2, \cdots, q_n]^T$ es el vector de coordenadas generalizadas, y t es el tiempo. Ahora supóngase una restricción de la forma:

$$f(q,\dot{q},t) = 0 \tag{3.2}$$

Si esta última restricción se puede convertir a la forma (3.1), entonces decimos que es integrable. Por lo tanto, aunque f en (3.2) contiene derivadas con respecto

del tiempo ($\dot{q}$), puede ser expresada en la forma holonómica (3.1), y por lo tanto es realmente una restricción holonómica.

Se dice que una restricción de la forma (3.2) es no holonómica si no puede ser reescrita en la forma (3.1), tal que dependa sólo de las propias coordenadas generalizadas y del tiempo.

Los robots sub-actuados son sistemas típicos que están sujetos a restricciones no holonómicas, y por lo tanto son llamados sistemas no holonómicos. La no holonomicidad ocurre de varias maneras. Por ejemplo, cuando un robot tiene grados de libertad redundantes o cuando un robot tiene pocos motores, $k < n$, donde n es el número de grados de libertad. Entonces el robot puede producir cuando mucho k movimientos independientes. La diferencia $n - k$ indica la existencia de no holonomicidad. [19]

La restricción no holonómica encontrada en la robótica móvil es la restricción de movimiento de un disco que rueda sin deslizamiento sobre un plano (Figura 3.22)[25]. Su configuración está descrita por tres coordenadas generalizadas: las coordenadas cartesianas (x, y) del punto de contacto con el suelo, medidas en un marco de referencia fijo, y el ángulo ϕ de orientación del disco con respecto al eje x. El vector de coordenadas generalizadas es por lo tanto $q = [x, y, \phi]^T$. [6] [19]

Las ecuaciones cinemáticas se muestran a continuación, en donde r es el radio del disco y θ es la posición angular del disco, tal y como se observa en la Figura 3.22.

$$\dot{x} = r\dot{\theta}\cos\phi \tag{3.3}$$

$$\dot{y} = r\dot{\theta}\sin\phi \tag{3.4}$$

Estas ecuaciones expresan la condición de que el vector de velocidad del punto de contacto (del disco con el suelo) yace en el plano sagital del disco (el plano que contiene el disco) y, por lo tanto, no tiene una componente en la dirección ortogonal a su plano sagital. Eliminando $v = r\dot{\theta}$ en (3.3) y (3.4) resulta:

$$v = r\dot{\theta} = \frac{\dot{x}}{\cos\phi} = \frac{\dot{y}}{\sin\phi} \quad \text{ó} \quad \dot{x}\sin\phi - \dot{y}\cos\phi = 0 \tag{3.5}$$

Esta es la restricción no holonómica de movimiento del disco. Sin embargo, la velocidad angular del disco alrededor del eje vertical no está restringida.

[25]Fuente: [19]

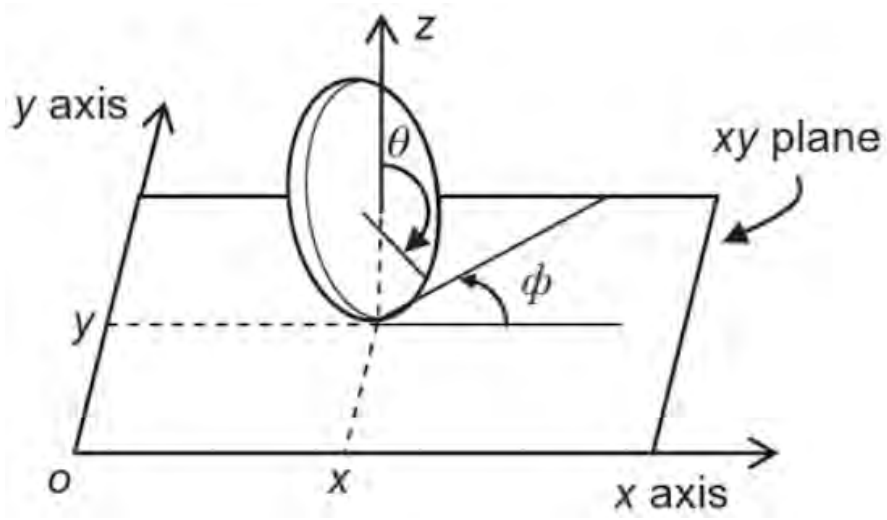

Figura 3.22: Coordenadas generalizadas de un disco rodando en un plano.

La restricción (3.5) es no holonómica porque no implica la perdida de accesibilidad en el espacio de configuración del disco. El disco puede alcanzar cualquier configuración final (x_f, y_f, ϕ_f), dada cualquier configuración inicial (x_i, y_i, ϕ_i) a través de la siguiente secuencia de movimientos que no violan la restricción (3.5):

1. Rotar el disco alrededor de su eje vertical hasta alcanzar una orientación $\phi_{intermedia}$ de tal forma que el disco apunte al punto de contacto final (x_f, y_f).

2. Rodar el disco con la orientación $\phi_{intermedia}$ hasta que el punto de contacto alcance la posición final (x_f, y_f).

3. Rotar el disco de nuevo alrededor de su eje vertical para cambiar su orientación final de $\phi_{intermedia}$ a ϕ_f.

3.2.4. SLAM

Para disponer de una descripción o mapa global de los elementos del entorno se requieren métodos para su construcción, por tanto, es deseable proporcionar al robot lo necesario para construir un mapa y ubicarse dentro de este mapa mientras navega por lugares previamente inexplorados, dado que los sensores de un robot móvil tienen un alcance limitado, es necesario explorar físicamente su entorno para construir dicho mapa. La localización y la navegación son tareas importantes para los robots móviles y, uno de los problemas más fundamentales es el de localización y mapeo simultáneo, este problema es comúnmente abreviado como SLAM (Simultaneous Localization And Mapping). [11] [13] [15] [17]

El término "localización y mapeo simultáneo" describe bastante bien el problema: partiendo de un punto inicial arbitrario, el robot crea un mapa de su

entorno al mismo tiempo que se localiza a sí mismo en este mapa. SLAM es un problema significativamente difícil, dado que la creación de un buen mapa dependerá en gran medida de las poses estimadas por el robot a lo largo del camino recorrido (el término pose se utiliza para referirse a la posición y orientación del robot móvil). El enfoque más simple para la construcción de mapas está basado en la estimación de la pose del vehículo obtenida a partir de dead reckoning, sin embargo, como se señaló previamente, este enfoque no es confiable en largos trayectos debido a la deriva (Figura 3.23a)[26]. Por consiguiente, surge un acoplamiento entre la construcción de mapas y la mejora de la estimación de la ubicación del robot móvil (Figura 3.23b). La percepción ocurre localmente en el marco de referencia del robot y, para garantizar la correspondencia entre la representación local del entorno y la representación global contenida en un mapa, es de particular interés el uso de filtros predictivos para estimar las poses y para actualizar dichas estimaciones a partir de las lecturas de los sensores al mismo tiempo que se construye un mapa incremental del entorno. Una limitante para la mejora de dicha estimación es el problema de asociación de datos, la asociación de datos se refiere al hecho de hacer coincidir las observaciones más recientes con los elementos previamente aprendidos del entorno y saber cuáles observaciones corresponden a características ambientales que no se observaron con anterioridad, sin embargo, hay situaciones que complican la tarea de asociación de datos como lo son el ruido presentado en los sensores y la dinámica del entorno. Esquemáticamente, el problema de la creación de mapas consiste en los siguiente pasos: [11] [14] [15]

1. Detectar el entorno del vehículo en el tiempo k utilizando sensores a bordo del robot, por ejemplo, un sensor láser.

2. Dar representación a los datos de los sensores.

3. Integrar las observaciones percibidas recientemente en el tiempo k con la estructura previamente estimada del entorno en el tiempo $k-1$.

Para que el SLAM sea exitoso debe incorporar técnicas de cierre y reubicación, éstas técnicas son aquellas que permiten determinar la localización del vehículo y corregir el mapa cuando la incertidumbre aumenta durante la exploración. No obstante, la interacción entre la corrección de la ubicación y el mapeo puede resultar problemática. El robot móvil podría localizarse de forma imprecisa basándose en una característica errónea en el mapa construido, de manera similar, se puede construir un mapa erróneo al agregar características desde una ubicación errada. El problema de construcción de mapas se puede ver como un símil del problema del huevo y la gallina: para tener una mejor localización, el robot necesita hacer coincidir sus más recientes observaciones con características en un mapa conocido, mientras que para construir un mapa el robot necesita saber en dónde se ubica con respecto al mapa. [13] [14]

[26]Fuente: [14]

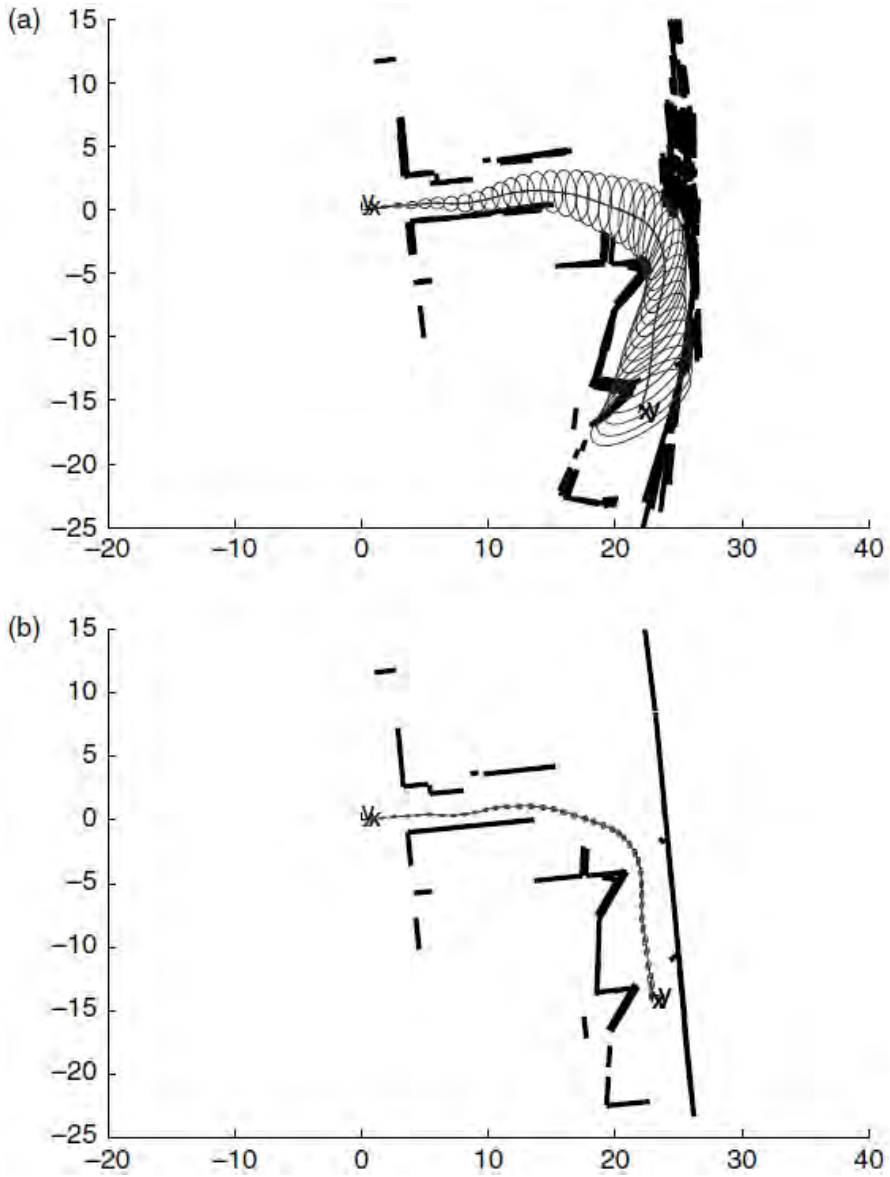

Figura 3.23: SLAM: (a) Mapeo utilizando dead reckoning en una trayectoria de 40m (b) Mapeo mejorando la estimación de la ubicación.

3.2.5. Mapas

Un mapa se puede definir como una lista de objetos en el entorno y su ubicación, los mapas generalmente se indexan en una de dos formas, conocidas como basados en características y basados en ubicación. En los mapas basados en características, el elemento n contiene las propiedades de una característica junto a la ubicación cartesiana de la característica. En los mapas basados en ubicación, el elemento n corresponde a una ubicación específica, y en particular, en los mapas 2D cada elemento está asociado a una coordenada específica x - y con respecto a un marco de referencia global. Los mapas basados en ubicación contienen no sólo información

sobre objetos en el entorno, sino que también contienen información sobre el espacio libre. Una representación clásica de mapas se conoce como occupancy grid map (mapa de cuadrícula de ocupación), este tipo de mapa se basa en la ubicación y asigna a cada coordenada x - y un valor de ocupación que especifica si una ubicación está libre u ocupada. Los mapas de cuadrícula de ocupación son adecuados para la navegación puesto que facilitan la búsqueda de rutas a través del espacio libre. [11]

Para elegir una representación de mapa adecuada es necesario tomar en cuenta lo siguiente: [13]

1. La precisión del mapa debe corresponder con la precisión que el robot necesita para lograr sus objetivos.
2. La complejidad de la representación del mapa tiene un impacto directo en la complejidad computacional del mapeo, localización y navegación.

Es común que los robots móviles para interiores cuenten con dispositivos láser que dan lecturas de distancia a objetos cercanos. El objetivo de crear un mapa con el robot es capturar solamente los objetos que pueden ser detectados por sus sensores, tomando en cuenta únicamente aspectos geométricos de los objetos, cualquier otra característica secundaria que no esté relacionada con la posición y la ocupación del espacio es irrelevante, tal es el caso de la textura o el color. [13]

Una forma de representar el entorno es mediante estrategias de descomposición para generar una abstracción del mismo. Pero una desventaja de la abstracción y descomposición es la pérdida de fidelidad entre el mapa y el mundo real hablando en términos de la estructura general y precisión geométrica. A pesar de ello, este tipo de estrategias puede resultar útil si la abstracción se planifica cuidadosamente para capturar características relevantes del mundo. [13]

En la Figura 3.24 [27] se observa el resultado del método de descomposición fija, en la que se transforma el mundo real continuo en una aproximación discreta y se puede observar lo que ocurre en áreas libres y con obstáculos. La desventaja de este enfoque es la inexactitud, ya que como se puede ver es posible que los espacios angostos sean representados como ocupados durante dicha transformación. La descomposición fija es enormemente popular en la robótica móvil, siendo quizá la técnica de representación más común actualmente utilizada. Una versión popular se conoce como cuadrícula de ocupación (occupancy grid), en la que el entorno está representado por una cuadrícula discreta y en donde cada celda está llena o vacía, es decir, ocupada o libre. Este método se suele utilizar cuando el robot está equipado con un sensor láser y al combinarse con la pose absoluta del robot se utiliza para actualizar el valor de cada celda. Este enfoque también tiene como desventaja que el tamaño del mapa crece en relación con el tamaño del entorno y crece aún más al utilizar un tamaño de celda pequeño. [13]

[27] Fuente: [13]

En un mapa de cuadrícula de ocupación cada celda tiene un valor que va desde 0 hasta un límite superior, el valor 0 indica que la celda no ha sido "atacada" por ninguna medición de distancia, a media que aumenta el número de ataques el valor se incrementa y, por encima de cierto umbral la celda es considerada como un obstáculo. [13]

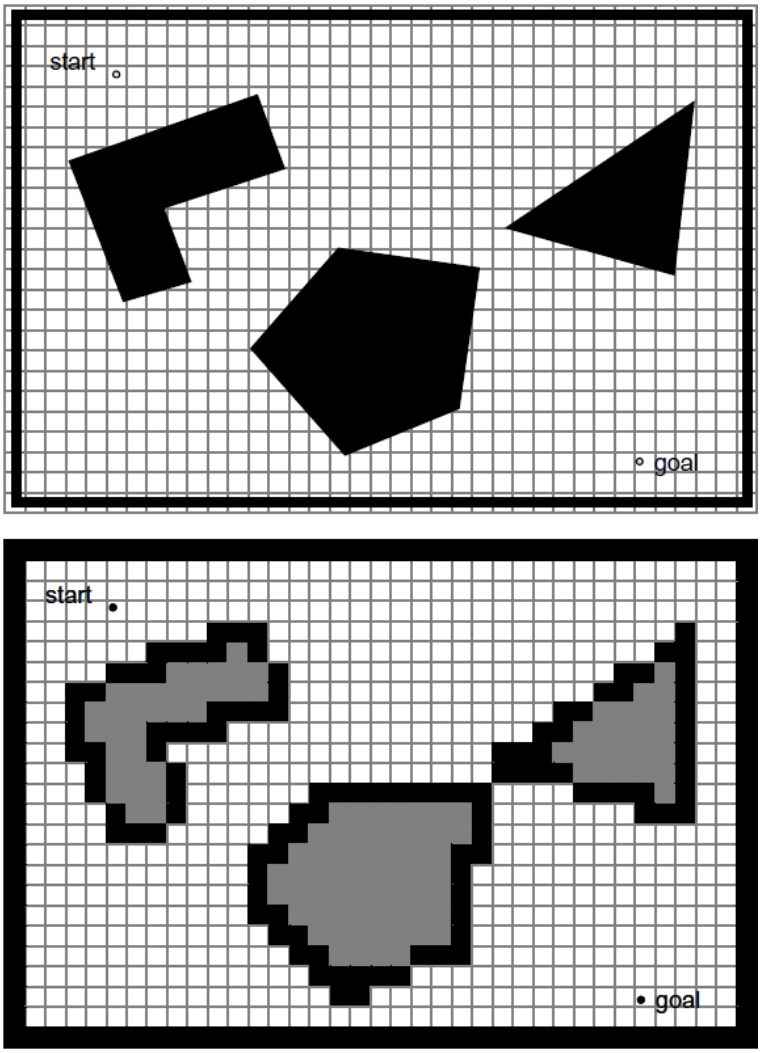

Figura 3.24: Descomposición fija del espacio.

3.3. ROS

ROS (Robot Operating System) es una plataforma de software que provee herramientas y bibliotecas para ayudar a los desarrolladores a crear aplicaciones para robots. ROS es open source, lo que quiere decir que el código fuente original está disponible y puede ser distribuido y modificado libremente. Es importante enfatizar que ROS no reemplaza, sino más bien funciona junto con un sistema operativo, utiliza el sistema de gestión de procesos, el sistema de archivos y la

interfaz de usuario de un sistema operativo. ROS proporciona los servicios que se esperarían en un sistema operativo como la implementación de funcionalidad de uso común, controladores de dispositivos, comunicación por mensajes entre procesos, administración de paquetes y abstracción de hardware. Además, cuenta con herramientas y bibliotecas para ejecutar código en múltiples computadoras. [20–22]

ROS fue diseñado para ser lo más modular y distribuido posible, de forma que los usuarios pueden utilizar tantos componentes de ROS según se crea conveniente, es decir, la modularidad permite elegir que partes aportadas por otros usuarios nos son útiles y que partes es mejor implementar por cuenta propia. ROS implementa diferentes estilos de comunicación, incluyendo la comunicación síncrona estilo RPC (llamada a procedimiento remoto) por medio de servicios (`services`), transmisión asíncrona de datos a través de temas (`topics`) y almacenamiento de datos en un servidor de parámetros (`parameter server`). [20] [21]

La computación distribuida es un modelo para resolver problemas, un problema se divide en muchas tareas que son resueltas por una o más computadoras; las computadoras están conectadas en red, se comunican y coordinan sus acciones por medio del paso de mensajes. RPC es un mecanismo usado por un proceso cliente para llamar a un procedimiento alojado remotamente (en un proceso servidor), el procedimiento remoto ejecuta y envía la respuesta de regreso al cliente, cada solicitud debe llevar la suficiente información requerida por el proceso servidor. La comunicación síncrona sigue un patrón de solicitud/respuesta (request/response) y está caracterizada por las respuestas inmediatas en los patrones de comunicación, requiere que tanto el cliente como el servidor estén siempre disponibles y en funcionamiento. Comúnmente esta forma de comunicación se implementa por RPC. Por otro lado, la comunicación asíncrona esta desacoplada y no sigue ningún patrón de solicitud/respuesta, típicamente, una de las partes participantes crea un mensaje que es entregado a un destinatario y no necesita una respuesta inmediata, de hecho, no hay necesariamente una respuesta. A diferencia de un mecanismo síncrono no se requiere que el servidor esté siempre disponible. [23]

Un desafío común en robótica es realizar el seguimiento de algunos de los componentes de un robot, tanto móviles como fijos. Afortunadamente ROS cuenta con una biblioteca llamada `tf` (transform) para la administración de las transformaciones entres los sistemas de referencia contenidos en un robot, esta biblioteca permite definir transformaciones estáticas, como por ejemplo la posición y orientación de una cámara o un sensor laser, y trasformaciones dinámicas, como en el caso de una articulación de un brazo robótico.

ROS proporciona herramientas para describir y modelar un robot, el formato para la descripción de robots es URDF (Unified Robot Description Format), consiste en un archivo XML en el que se describen las propiedades físicas de un robot, como longitudes de extremidades, tamaños de ruedas, ubicación de sensores y en general la apariencia visual de cada parte del robot. Una vez que el robot se define por este

medio puede ser renderizado en tres dimensiones por medio de la herramienta de visualización Rviz.

Probablemente la herramienta gráfica más utilizada en ROS es Rviz, la cual proporciona visualización tridimensional de varios tipos de datos obtenidos por los sensores de un robot, como las lecturas de un sensor láser y nubes de puntos tridimensionales, también es posible visualizar los marcos de referencia asociados a un robot, mapas, marcadores, entre otras cosas. Esto permite poder observar el comportamiento del robot, lo que el robot ve y como lo ve, haciendo de este modo más sencilla la identificación de problemas.

ROS tiene tres niveles de conceptos: Filesystem Level, Computation Graph Level y Community Level. A continuación, se explicarán algunos de los conceptos pertenecientes a cada nivel, los cuales son importantes para este trabajo.

3.3.1. Filesystem Level

De igual forma que un sistema operativo, los archivos de ROS también se organizan en el disco duro de una manera en particular. El software de ROS está organizado en paquetes. Los paquetes son la unidad principal de organización de software y pueden contener bibliotecas, nodos, archivos de configuración, software de terceros o cualquier otra cosa que forme parte de un módulo útil. El objetivo de los paquetes es proporcionar funcionalidad de una manera fácil de tal modo que puede reutilizarse fácilmente. En la Figura 3.25 [28] se muestra la estructura típica de un paquete de ROS. [24] [25]

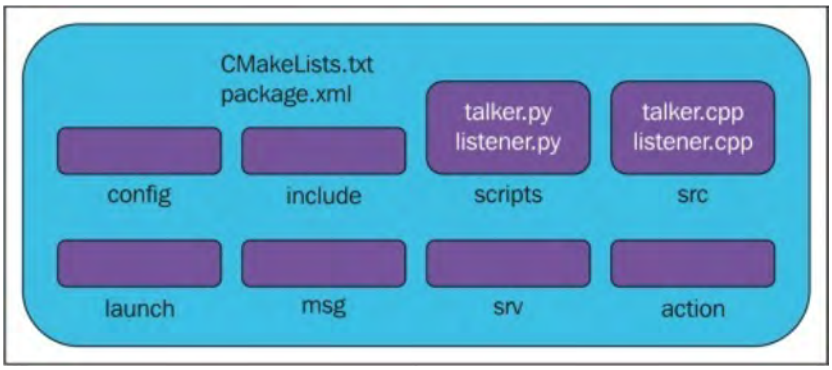

Figura 3.25: Estructura de directorios en un paquete.

ROS utiliza descripciones de mensajes denominadas tipos de mensaje (`msg`) y, definen la estructura de datos para los mensajes enviados en ROS. ROS usa una descripción simplificada de mensajes para describir los tipos de datos que se envían por medio de nodos, dicha descripción facilita que las herramientas de ROS generen automáticamente código fuente para cada tipo de mensaje en varios lenguajes de

[28] Fuente: [25]

programación. Las descripciones de los mensajes se almacenan en archivos `.msg` en un subdirectorio nombrado `msg` dentro de un paquete.

ROS emplea una descripción de servicios simplificada (`srv`) para describir los tipos de servicios. Los servicios están basados en el formato `msg` para permitir la comunicación entre nodos del tipo solicitud/respuesta. Las descripciones de servicios se almacenan en archivos `.srv` en un subdirectorio llamado `srv` dentro de un paquete.

3.3.2. Computation Graph Level

En ROS los cálculos se hacen utilizando una red de procesos denominados nodos (`nodes`), a esta red se le llama Computation Graph Level. Los conceptos principales de esta red son: nodos (`nodes`), `master`, servidor de parámetros (`parameter server`), mensajes (`messages`), temas (`topics`), servicios (`services`) y `bags`. [24] [25]

Los nodos son procesos que realizan cálculos. ROS está diseñado para ser modular y que el código desarrollado pueda ser útil para otros robots haciendo pequeños o nulos cambios. Los nodos se comunican entre sí utilizando temas, servicios y el servidor de parámetros. Un sistema de control para un robot generalmente está comprendido por muchos nodos, por ejemplo, un nodo controla un sensor láser, un nodo controla los motores de las ruedas, un nodo realiza la localización, un nodo realiza la planificación de rutas, un nodo provee un visualizador gráfico del sistema. El uso de nodos proporciona varios beneficios para el sistema en general, como lo es la tolerancia a fallas y la reducción de la complejidad del código en comparación con los sistemas monolíticos (formados por una sola pieza). Todos los nodos en ejecución tienen un nombre que los identifica de manera única, un nodo de ROS comúnmente está escrito en los lenguajes de programación C++ y Python.

El master actúa como un servicio de asignación de nombres, en ROS los nombres juegan un papel muy importante, los nodos, temas, servicios y parámetros tienen nombres. Esta arquitectura permite una operación desacoplada y, los nombres son el medio por el cual se pueden construir sistemas grandes y complejos. El master proporciona servicios de registro para los nodos y permite que los nodos se encuentren entre sí, una vez que se ubican es cuando la comunicación es posible y pueden realizar conexiones según sea el caso. Los nodos se conectan con otros nodos directamente, el master solamente proporciona información de búsqueda.

El servidor de parámetros es un diccionario multivariable y compartido que permite el almacenamiento de datos por medio de una llave en una ubicación central. Este servidor es accesible a través de la red y los nodos lo utilizan para almacenar y obtener parámetros en tiempo de ejecución. Como no está diseñado para alto rendimiento es mejor utilizarlo para datos estáticos, tales como las ganancias de un controlador PID para un sistema.

Los nodos se comunican entre sí mediante el paso de mensajes. Un mensaje es una estructura de datos comprendida por campos tipados, los tipos pueden ser entero, flotante, booleano, etcétera. De igual manera están permitidos los arreglos.

Los temas son buses sobre los cuales los nodos intercambian mensajes (información), los mensajes se enrutan a través de un sistema de transporte con una política de publicación/suscripción. Los nodos no saben con quien se están comunicando, un nodo simplemente envía un mensaje publicándolo en un tema determinado y un nodo que está interesado en recibir cierto tipo de información debe suscribirse al tema apropiado. El tema es un nombre que se utiliza para identificar un cierto tipo de mensaje, por lo que cada bus tiene un nombre, y cualquiera puede conectarse a un bus para enviar o recibir mensajes. Un nodo puede publicar y/o suscribirse a múltiples temas y, puede haber varios publicadores y suscriptores simultáneos para un tema. Los temas están destinados para la comunicación unidireccional.

El modelo publicación/suscripción es un tipo de comunicación flexible, pero su transporte unidireccional de muchos a muchos no es apropiado para una interacción del tipo solicitud/respuesta, que con frecuencia se requiere en un sistema distribuido. La interacción solicitud/respuesta se realiza por medio de un servicio. Un nodo proveedor ofrece un servicio y un cliente utiliza el servicio enviando un mensaje de solicitud y esperando por la respuesta.

Los nodos se pueden comunicar a través de temas y servicios, tal y como se muestra en la Figura 3.26 [29]. Cuando se comparte información por medio de la invocación de un servicio, el proceso (nodo) que realiza la petición debe esperar a que el proceso receptor reciba la solicitud, termine su ejecución y envié una respuesta, de este modo puede continuar con su ejecución. Por otro lado, si un nodo usa un tema como medio de transporte de información, puede continuar con su ejecución sin importar si existe o no un nodo servidor.

[29]Fuente: `http://wiki.ros.org/custom/images/wiki/ROS_basic_concepts.png`

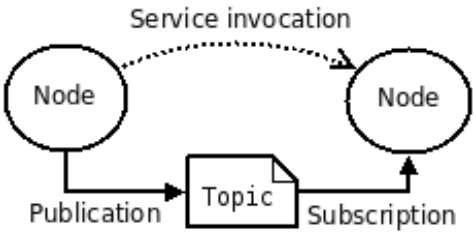

Figura 3.26: Comunicación entre nodos por medio de temas y servicios.

3.4. Teoría de control

El control automático es parte integral de la sociedad moderna y ha jugado un papel vital en el avance de la ingeniería y la ciencia. En particular el control automático se ha vuelto importante en procesos industriales y de fabricación en los que, por ejemplo, es necesario controlar la presión, temperatura, humedad, flujo o incluso vehículos guiados que entregan material en estaciones de trabajo, además de su importancia en vehículos espaciales, misiles guiados y sistemas robóticos. [26] [27]

Para hablar de sistemas de control primero es necesario definir algunos conceptos. Un sistema es un conjunto de componentes que interactúan para logar un objetivo. Un sistema se caracteriza por tener interrelaciones entre los componentes que lo conforman y, por tener límites reales o imaginarios que separan sus propios componentes de otros componentes externos. En cualquier etapa de análisis de un sistema podemos elegir solamente una parte del sistema original, y lo llamamos subsistema del sistema original. Así mismo, podemos decidir ampliar los límites del sistema original para incluir otros componentes. En la Figura 3.27 [30] se muestra la representación de un sistema. En particular, estamos interesados en los efectos de los estímulos o cantidades externas sobre el estado de un sistema. Se define como entradas al sistema a los estímulos o cantidades externas que actúan sobre el sistema y, la condición o estado del sistema se puede describir mediante variables de estado. Las variables de estado junto con el conocimiento de las entradas al sistema proveen información que permite determinar el estado futuro del sistema. En la práctica, en ocasiones no es posible medir todas las variables de estado, y las cantidades cuyo comportamiento puede observarse o medirse son denominadas salidas del sistema. Un sistema dinámico se define como un sistema cuyo comportamiento cambia con el tiempo, a menudo en respuesta a estímulos o fuerzas externas. Se define como variables de estado al conjunto del menor número de variables que determinan completamente el comportamiento de un sistema dinámico en todo momento. [28]

Una planta puede ser parte de un equipo o un conjunto de elementos de una máquina cuyo objetivo es efectuar un propósito determinado, en particular se

[30] Fuente: [28]

considera como una planta a cualquier objeto físico que se desea controlar. Una variable a controlar es una cantidad física que se mide o que se quiere controlar. Una variable de control es la cantidad física que un controlador debe manipular con el fin de modificar el valor de la variable a controlar. Una perturbación es una señal que tiende a afectar negativamente el valor de la salida de un sistema y sobre la cual por lo general no se tiene control. Controlar implica aplicar la variable de control requerida para corregir o limitar una desviación del valor de la variable a controlar con respecto a un valor deseado. [26]

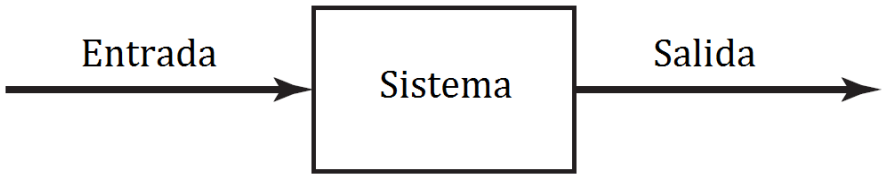

Figura 3.27: Representación de un sistema.

Un sistema de control está formado por subsistemas, procesos o plantas unidos con el propósito de obtener una salida deseada con un comportamiento deseado, dada una entrada específica. La Figura 3.28 [31] muestra un sistema de control en su forma más simple, en donde la entrada representa la salida deseada. En ingeniería, al tratar con sistemas dinámicos, es de interés especificar las entradas del sistema que fuerzan a los estados o salidas del sistema a comportarse en el tiempo de alguna manera específica, dicho de otra forma, el objetivo es controlar los estados o salidas del sistema. Esto se consigue por medio de un controlador cuya tarea es producir las entradas requeridas del sistema que den como resultado las salidas deseadas. Se le llama sistema de control a la interconexión de un sistema y un controlador. [27] [28]

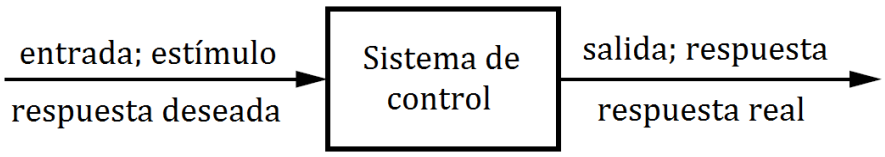

Figura 3.28: Descripción simplificada de un sistema de control.

Dos tipos de sistemas de control son: control en lazo abierto (Figura 3.29)[32] y control en lazo cerrado (Figura 3.30)[33]. Los sistemas de control en lazo abierto son aquellos en los que la salida no se mide ni se realimenta para compararla con la entrada de referencia, es decir, la salida no tiene efecto sobre la acción de control. El control en lazo abierto no puede compensar ninguna perturbación y, en la práctica se utiliza cuando se conoce la relación entre la entrada y la salida. Los sistemas de

[31] Fuente: [27]
[32] Fuente: [28]
[33] Fuente: [28]

control en lazo cerrado también se denominan sistemas de control realimentados, en este tipo de sistemas se alimenta al controlador la señal de error de actuación, que es la diferencia entre la señal de entrada o referencia y la señal de salida, con el propósito de reducir el error y lograr que la salida del sistema alcance un valor deseado. En los sistemas de control en lazo cerrado la salida se mide con un sensor o transductor; este tipo de sistema de control puede compensar las perturbaciones gracias al uso de la realimentación. [26] [27]

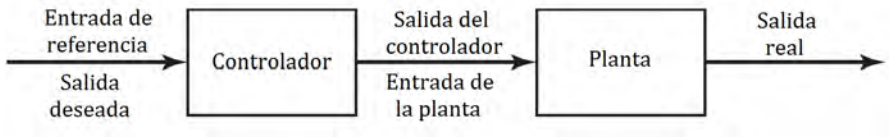

Figura 3.29: Sistema de control en lazo abierto.

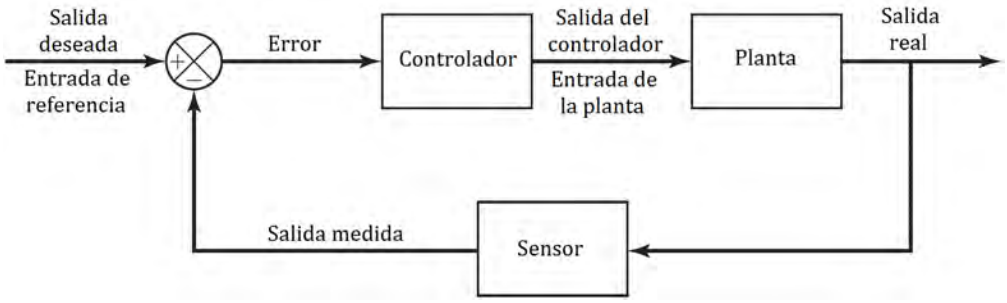

Figura 3.30: Sistema de control en lazo cerrado.

La respuesta en el tiempo de un sistema de control está constituida por dos partes, la respuesta transitoria y la respuesta en estado permanente. La respuesta transitoria es la que va desde un estado inicial hasta un estado final. La respuesta en estado permanente es la forma en la que la salida del sistema se comporta conforme el tiempo tiende a infinito. El tiempo de asentamiento es el tiempo requerido para que la curva de respuesta alcance y permanezca dentro de un rango cercano a su valor final, generalmente entre el 98 % y el 102 % del valor final. Se define al error en estado permanente como la diferencia entre el valor de referencia y la salida después que haya pasado la repuesta transitoria. [26] [27]

Para este trabajo son importantes dos objetivos de control, los cuales con regulación y seguimiento. El objetivo de regulación consiste en hacer que una variable de estado evolucione desde un valor inicial hasta un valor final, manteniéndose en el valor final. El objetivo de seguimiento consiste en hacer que las salidas del sistema evolucionen siguiendo trayectorias deseadas cualesquiera.

El modelado mediante variables de estado es una aproximación para entender el comportamiento de los sistemas físicos y, la estructura básica de un modelo en variables de estado está dada por las siguientes ecuaciones:

$$\dot{x} = f(x, u) \qquad x(t_0) = x_0$$
$$y = h(x, u)$$

En donde:

x	Es un vector n-dimensional de elementos reales que denota el estado del sistema, denominado vector de estados
n	Es la dimensión del vector de estados y determina el orden del sistema
$\dot{x}$	Es un vector que denota la derivada del vector de estados con respecto del tiempo
u	Es un vector m-dimensional de elementos reales que denota las variables de entrada o de control del sistema
y	Es un vector p-dimensional de elementos reales que denota la salida del sistema
$x(t_0)$	Determina las condiciones iniciales del sistema

El controlador PID es uno de los más utilizados en la industria, sintonizarlo implica ajustar los parámetros de control que cumplan con los requerimientos para el rendimiento deseado de un sistema. En la literatura se han propuesto muchos métodos para la sintonización, sin embargo, la reconfiguración de los parámetros de control se puede realizar experimentalmente mediante prueba y error. La utilidad de los controladores PID radica cuando el modelo matemático de la planta no se conoce y, por ende, no se pueden emplear otros métodos de diseño analíticos. [26]

La ley de control para el cálculo de la entrada al sistema que se desea controlar, está dada por la siguiente descripción matemática, en donde e es el error entre la señal de entrada o referencia y la señal de salida, las constantes P, I, D, son las tres acciones de corrección de error denominadas proporcional, integral y derivativa, respectivamente:

$$u = P\,e + I\int e\,dt + D\frac{de}{dt}$$

Cada acción de corrección tiene diferentes efectos en la respuesta del sistema o planta a controlar. En la Tabla 3.1 se muestran los efectos que tiene cada acción de corrección.

Acción	Tiempo de levantamiento	Sobrepaso	Tiempo de asentamiento	Error en estado permanente
P	Decrece	Aumenta	Poco cambio	Decrece
I	Decrece	Aumenta	Aumenta	Elimina
D	Poco cambio	Decrece	Decrece	Poco cambio

Tabla 3.1: Ganancias de los controladores PID de los motores.

Capítulo 4

Desarrollo: Internet de las Cosas

4.1. Necesidades del proyecto

Se desea integrar un robot móvil autónomo como un nuevo dispositivo en el ecosistema de SmartThings, el cual sea capaz de interactuar con otros dispositivos comerciales. Se cuenta con un Hub de la empresa SmartThings y con los componentes necesarios para la construcción del robot móvil. Se requiere que el robot móvil sea capaz de navegar de forma autónoma en un ambiente de trabajo casero, se desea que el robot reciba las notificaciones de varios tipos de sensores de la empresa SmartThings y navegue hasta ubicaciones en las que se detecte actividad por dichos sensores dentro de un hogar. Por ejemplo, al detectar la apertura de una puerta cuando nadie se encuentra en casa, se requiere que el robot navegue hasta el lugar donde ocurrió dicho evento. Por otro lado, se desea que el usuario final pueda enviar al robot móvil a lugares prestablecidos dentro del hogar, esto a través de la aplicación móvil de SmartThings.

En la Figura 4.1 se presenta un esquema de comunicación entre los elementos involucrados en el presente proyecto. Posteriormente, se solicitó que el robot cuente con un modo tele operado y el montaje de una cámara IP a bordo del robot. Un requerimiento adicional es que un usuario final pueda visualizar el modelo 3D del robot y su ubicación en tiempo real en el mapa conocido del lugar en el que opera el robot móvil.

Se sabe que desde la aplicación móvil se pueden enviar comandos a una placa Arduino UNO, teniendo como intermediarios a una placa llamada ThingShield de la empresa SmartThings y el esquema de comunicación utilizado por dicha empresa (servidor y Hub de SmartThings). Se solicita que la placa Arduino UNO se comunique con una Raspberry Pi 3, la cual será el módulo central de procesamiento de información y control del robot móvil.

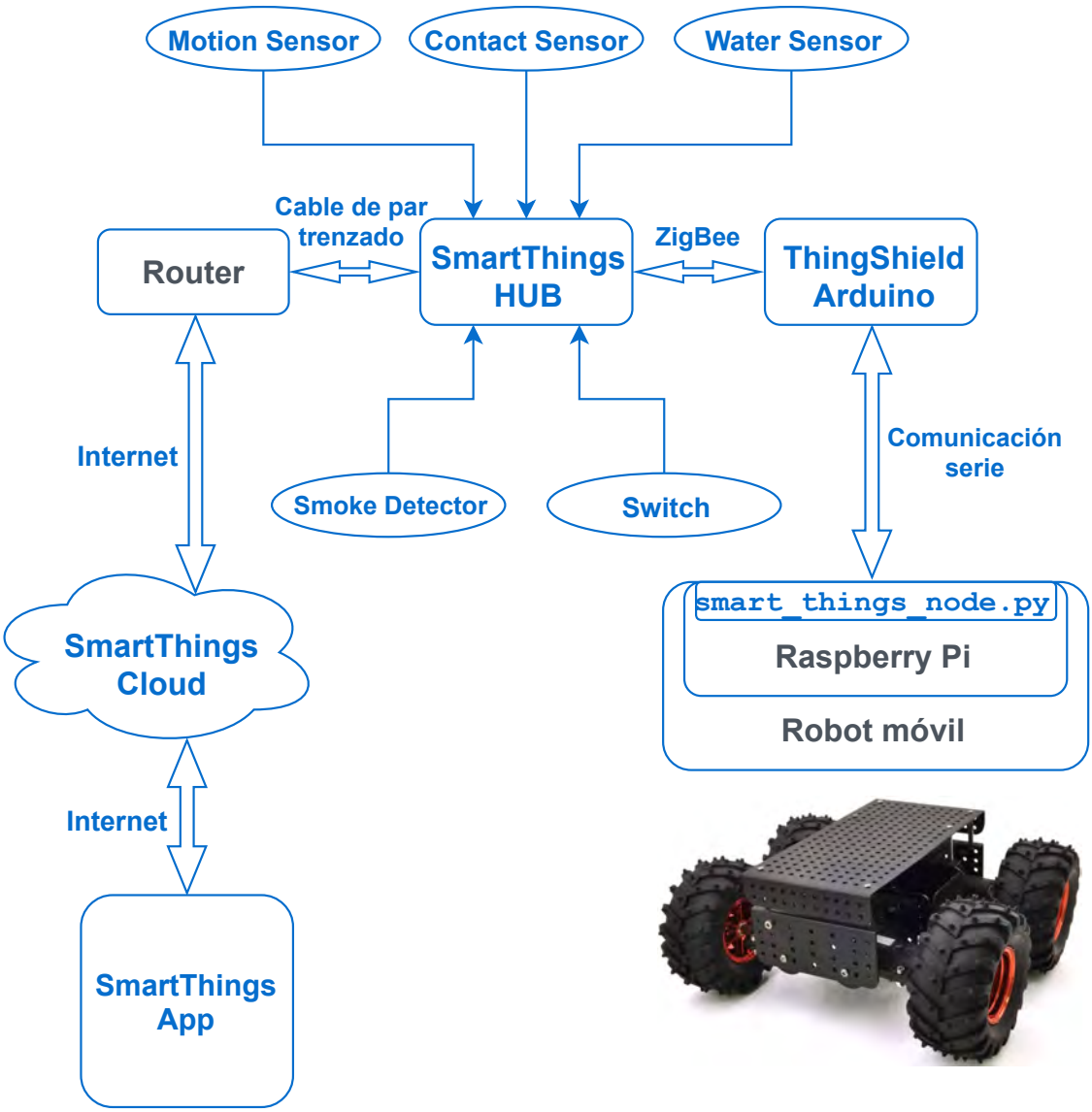

Figura 4.1: Esquema de comunicación.

4.2. SmartThings

El propósito de utilizar SmartThings es hacer posible que un usuario final sea capaz de controlar el robot móvil desarrollado en este trabajo, así como monitorizar una residencia a través de dispositivos comerciales de la empresa SmartThings, informando al robot móvil y al propio usuario sobre actividad inesperada, principalmente cuando no se encuentra nadie en el hogar.

Para logar esto es necesario la implementación en un *Device Handler*, con el cual sea posible controlar al robot móvil de forma tele operada y enviarlo a lugares preestablecidos dentro del hogar, de igual manera, por este medio es

posible informarle al robot móvil cuando se suscita un evento detectado por otros dispositivos instalados dentro del hogar.

Por medio de una *SmartApp* es posible compartir información entre los dispositivos asociados al Hub de SmartThings y tomar acciones con base en ello, por lo cual fue necesario desarrollar una *SmartApp* en la que el usuario elige los sensores dentro del hogar que le interesa que reporten los eventos que detectan al robot móvil. Por ejemplo, se podría tener un sensor de contacto instalado en la puerta principal de una residencia, y el usuario final estaría interesado en informarle al robot móvil la apertura o cierre de dicha puerta cuando no hay nadie en casa.

SmartThings busca reducir la intervención humana, por lo cual se pueden programar o agendar acciones, dentro de las cuales está la detección automática de presencia o ausencia de personas dentro del hogar. A continuación, se presenta la forma en que SmartThings puede saber cuándo alguien está en el hogar, así como el *Device Handler* y la *SmartApp* desarrollada.

4.2.1. Device Handler

Al asociarle un *Device Handler* a un dispositivo se renderiza en la aplicación móvil una interfaz de usuario específica para dicho dispositivo, desde la cual el usuario puede visualizar el estado actual de su dispositivo y/o enviar comandos para que se realicen ciertas acciones. Para este caso se creó un *Device Handler* para la placa ThingShield.

En la Figura 4.2 se puede apreciar la interfaz desde la cual el usuario puede controlar al robot de forma tele operada y establecer en porcentaje la velocidad angular y lineal a la que se moverá el robot en este modo. De igual manera se puede enviar al robot móvil a lugares preestablecidos por medio de los botones correspondientes, así como visualizar la disponibilidad del robot, que indica `busy` cuando está navegando de forma autónoma y `stand-by` cuando no lo está. Finalmente, también se indica el estado de carga de la batería en porcentaje.

El robot móvil tiene como máxima prioridad atender las notificaciones que recibe de los dispositivos que monitorizan el hogar, su segunda prioridad es navegar a los lugares preestablecidos en la aplicación móvil y su última prioridad es el modo tele operado. Si el robot móvil recibe una o más notificaciones de los dispositivos instalados en el hogar mientras está ocupado, terminará de realizar la tarea que está realizando en ese momento y atenderá las notificaciones en el orden que llegaron. Por ejemplo, si el robot recibe primero una alerta del dispositivo C y luego otra del A, pero está ocupado en ese momento, entonces terminará la tarea en proceso y primero atenderá la notificación del dispositivo C y posteriormente la del dispositivo A. Si el robot móvil se encuentra ocupado hará caso omiso al recibir instrucciones de navegar a alguno de los lugares preestablecidos y también omitirá las instrucciones recibidas del modo tele operado.

El *Device Handler* desarrollado tiene las `capacidades` "`Actuator`" y "`Sensor`" para establecer `comandos` y `atributos` personalizados. Además cuenta con la `capacidad` "`Battery`" para indicar la carga actual de la batería. Uno de los comandos personalizados a destacar es el llamado `sendDeviceNameNotification`, el cual tiene definido un método con el mismo nombre, este método tiene un único parámetro de tipo String.

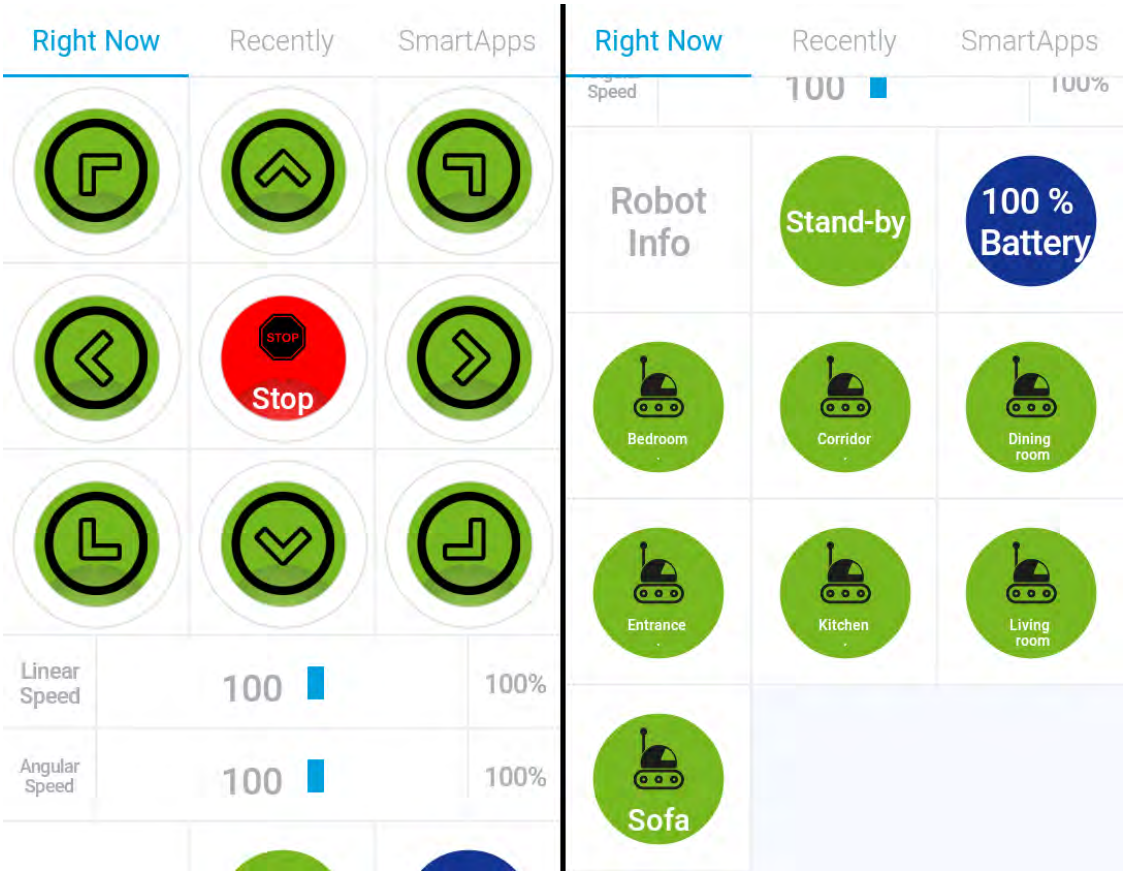

Figura 4.2: Renderización del *Device Handler* en la aplicación móvil.

El método `sendDeviceNameNotification` es el que se invoca desde la *SmartApp* creada, ya que por medio de él es posible pasar como valor del parámetro el nombre de cualquier dispositivo que haya detectado algún suceso inesperado dentro del hogar. Lo que se hace dentro del método `sendDeviceNameNotification` es enviarle a la ThingShield el valor que recibe el parámetro precedido por la cadena de caracteres `dev-`, es decir como resultado final lo que se envía a la ThingShield es lo siguiente: `dev-nombre_del_dispositivo`.

4.2.2. SmartApp

El objetivo de una *SmartApp* es automatizar y agendar ciertas tareas en el hogar, SmartThings cuenta por defecto con *SmartApps* para situaciones comunes en el hogar, por ejemplo, ajustar la iluminación en el hogar a ciertas horas del día, o simplemente apagar o encender focos cuando alguien entra o sale del hogar. Sin embargo, SmartThings también permite a los desarrolladores crear *SmartApps* propias para situaciones personalizadas, como en el caso del trabajo aquí presentado.

La *SmartApp* creada para este trabajo solicita al usuario seleccionar primero la placa ThingShield instalada en el robot móvil, la cual es un requerimiento obligatorio para la *SmartApp*, posteriormente solicita seleccionar los dispositivos que notificaran sobre su estado al robot móvil y, finalmente se selecciona el modo en el que trabajará la *SmartApp* (Figuras 4.3 y 4.4).

Entre los dispositivos que se pueden seleccionar en esta *SmartApp* están sensores de contacto que comúnmente se instalan en puertas y ventas, sensores de movimiento, dispositivos con atributos on/off, sensores para la detección de fugas de agua y sensores para la detección de humo.

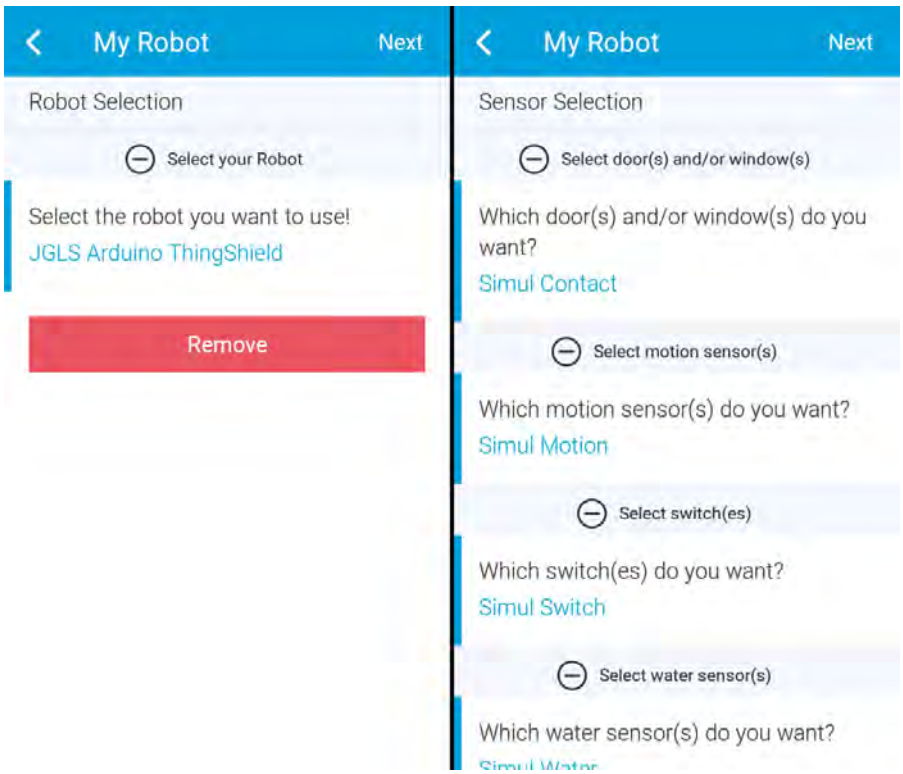

Figura 4.3: Instalación de la *SmartApp* (páginas 1 y 2).

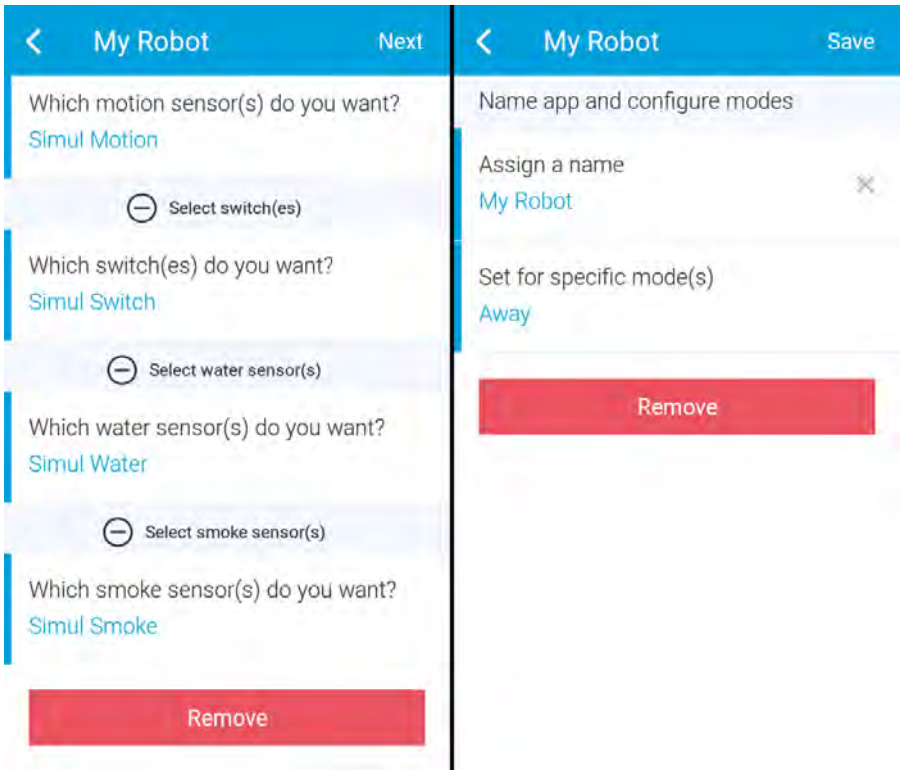

Figura 4.4: Instalación de la *SmartApp* (páginas 2 y 3).

La *SmartApp* se suscribe a los eventos generados por los dispositivos mencionados anteriormente y entra en acción cuando ocurren dichos eventos. Los eventos a los que se sucribe son: `motion.active`, `contact.closed`, `contact.open`, `switch.on`, `switch.off`, `water.wet` y `smoke.detected`; cada uno de estos eventos cuenta con un método controlador.

Por ejemplo, para el caso del sensor de movimiento, la *SmartApp* se suscribe al evento `motion.active` (este evento se dispara cuando se detecta movimiento) y una vez que el evento ocurre se hace una llamada a su respectivo método controlador. Dentro del método controlador se realizan dos cosas, lo primero que se hace es obtener el nombre que el usuario final le asignó al dispositivo que generó el evento; una vez que se tiene el nombre del dispositivo se invoca el método `sendDeviceNameNotification` implementado en el *Device Handler* de la ThingShield y se le pasa como valor del parámetro el nombre del dispositivo. Es decir, dentro del método controlador del evento `motion.active` se invoca el método del *Device Handler* de la siguiente forma: `sendDeviceNameNotification(nombre_del_dispositivo)`. Finalmente se envía una notificación push con una pequeña descripción de lo sucedido, en la Figura 4.5 se muestra un ejemplo de este tipo de notificaciones. Esto mismo sucede para cada uno los eventos generados por los dispositivos.

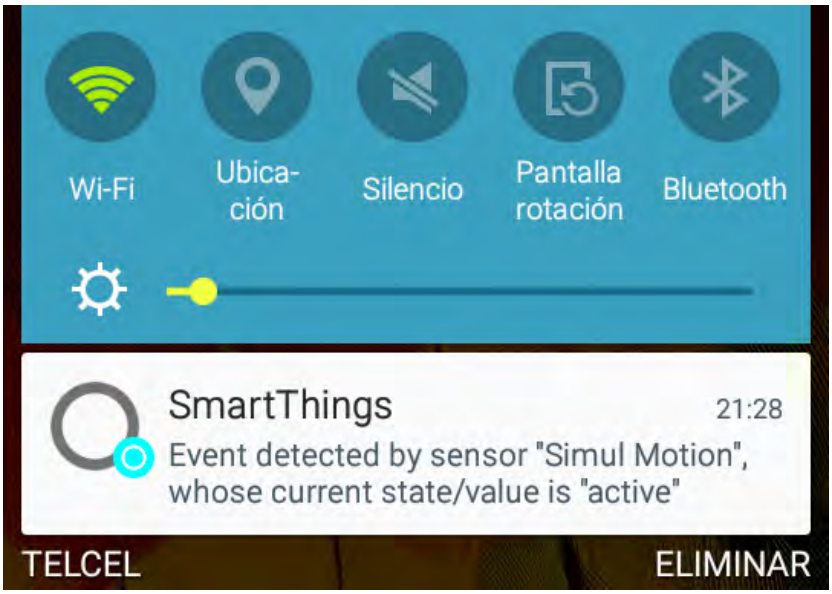

Figura 4.5: Ejemplo de notificación push de la aplicación móvil de SmartThings.

4.2.3. Activación automática de los modos Home y Away

El robot móvil se pensó para la monitorización del hogar cuando nadie está en casa, SmartThings puede saber si alguien está o no en casa mediante modos, el modo `Away` se activa con la rutina `Good Bye!` y el modo `Home` se activa con la rutina `I'm Back!`. El modo se puede cambiar de forma manual, pero también se puede cambiar de modo de manera automática por medio de sensores de arribo (Arrival Sensors) o con ayuda de nuestros dispositivos móviles.

Al crear una cuenta de SmartThings, de manera opcional se puede configurar la geolocalización del Hub en un mapa estableciendo un radio de ubicación (Figura 4.6)[34], de esta forma se puede utilizar un smartphone como un sensor de presencia. Dado que los smartphones tienen un GPS incorporado es posible saber cuándo alguien entra y sale del área señalada.

El sensor de arribo de SmartThings es un dispositivo del tamaño de un llavero (Figura 4.7)[35], permite realizar el seguimiento de personas, mascotas y vehículos para saber cuándo llegan o salen de casa, en sí, de cualquier cosa que pueda portarlo. Como recomendación, SmartThings sugiere colocarlo dentro del automóvil, en un llavero, en una mochila o en el collar de una mascota. El sensor detecta el ir y regresar cuando se está fuera y dentro del alcance del Hub.

[34]Fuente: `https://i0.wp.com/blog.smartthings.com/wp-content/uploads/2014/01/rsz_photo_2_1-1.jpg`

[35]Fuente: `https://smartthings.imgix.net/app/public/spree/products/1122/original/ST-Arrival-Box-600x374.jpg`

Figura 4.6: Geolocalización del Hub.

Figura 4.7: SmartThings Arrival Sensor.

4.3. SmartThings node

El paquete `smart_things` contiene el nodo `smart_things_node.py`, este nodo se comunica con la tarjeta Arduino Uno R3 para recibir y enviar información al

servidor de SmartThings. La comunicación se realiza por medio de cadenas de caracteres utilizando el protocolo de comunicación serial; desde el *Device Handler* de la placa ThingShield se envían seis tipos diferentes de comandos al robot móvil, estos se enlistan a continuación:

1. El primer tipo es una cadena para informar sobre un evento detectado por alguno de los dispositivos de SmartThings, un ejemplo de este tipo de evento puede ser la detección de humo por parte de un sensor. Entonces la cadena será: `dev-nombre_del_dispositivo`.

2. El segundo tipo es cuando el usuario desea que el robot móvil navegue hasta alguno de los lugares predeterminados, por ejemplo, el usuario solicita que el robot móvil navegue hasta la cocina. La cadena correspondiente será: `loc-nombre_del_lugar`.

3. El tercer tipo de comando es la cadena `stop`. Esta cadena sirve para detener al robot móvil y aplica únicamente para el modo tele operado.

4. Cuando se quiere utilizar al robot en modo tele operado se envía una cadena con la dirección en la que se desea que el robot se mueva. La cadena es la siguiente: `mov-dirección_de_movimiento`.

5. En el modo tele operado el robot móvil tiene una velocidad angular máxima a la que se mueve, por ello el usuario sólo puede indicar en porcentaje esta velocidad. La cadena para ajustar la velocidad angular es: `ang-porcentaje`.

6. De igual forma en el modo tele operado el robot tiene establecida una velocidad lineal máxima. La cadena para ajustar dicha velocidad es: `lin-porcentaje`.

Desde la placa ThingShield se envían cadenas de caracteres al servidor de SmartThings para informar al usuario sobre la disponibilidad actual del robot y el nivel de carga de su batería. Las cadenas de caracteres enviadas son las siguientes:

1. Para informar que el robot móvil está ocupado realizando una tarea se envía la cadena `rS-busy`. Para indicar que el robot móvil está disponible se envía la cadena `rS-onSB`.

2. Para informar sobre la carga de la batería se envía una cadena con el porcentaje de carga actual en ella. La cadena es la siguiente: `bL-porcentaje_de_carga`.

Este nodo se suscribe y publica temas para la gestión de la información recibida y enviada al servidor de SmartThings, a continuación, en la Tabla 4.1 se muestran los temas relacionados a este nodo.

Temas publicados	/navigation/goal_location	Por medio de este tema se publica el nombre del lugar al que el usuario desea enviar al robot.
	/navigation/smart_things /alerts_devices	En este tema se publica el nombre del dispositivo de SmarThings que detectó un suceso inesperado.
	/hardware/mobile_base /speeds	Este tema publica las velocidades de las ruedas del robot. Únicamente aplica para el modo tele operado.
Temas suscritos	/hardware/robot_state /base_battery	Voltaje actual en la batería.
	/hardware/robot_state /availability	Disponibilidad actual del robot.

Tabla 4.1: SmartThings node: Temas publicados y suscritos.

4.4. Lugares predeterminados y dispositivos simulados

Desde la aplicación móvil es posible cambiar el estado de los dispositivos simulados que se utilizaron en este trabajo, en la Figura 4.8 se muestran estos dispositivos. Los dispositivos *Simul Contact*, *Simul Motion*, *Simul Smoke*, *Simul Switch* y *Simul Water* son los utilizados en la *SmartApp* creada, en tanto que el dispositivo *Simul Presence* se utiliza para cambiar entre los modos `Home` y `Away`.

En el nodo `planner_node` se establece la ubicación cartesiana de cada uno de los dispositivos simulados y lugares predeterminados dentro de un mapa estático con respecto del sistema de referencia `map`, la ubicación de los dispositivos simulados se muestra en la Tabla 4.2 y la ubicación de los lugares predeterminados se muestra en la Tabla 4.3. En la Figura 4.9 se muestran algunas de las ubicaciones de los lugares y los dispositivos, las cajas en color verde corresponden a los lugares predeterminados y los cilindros en color azul corresponden a los dispositivos simulados.

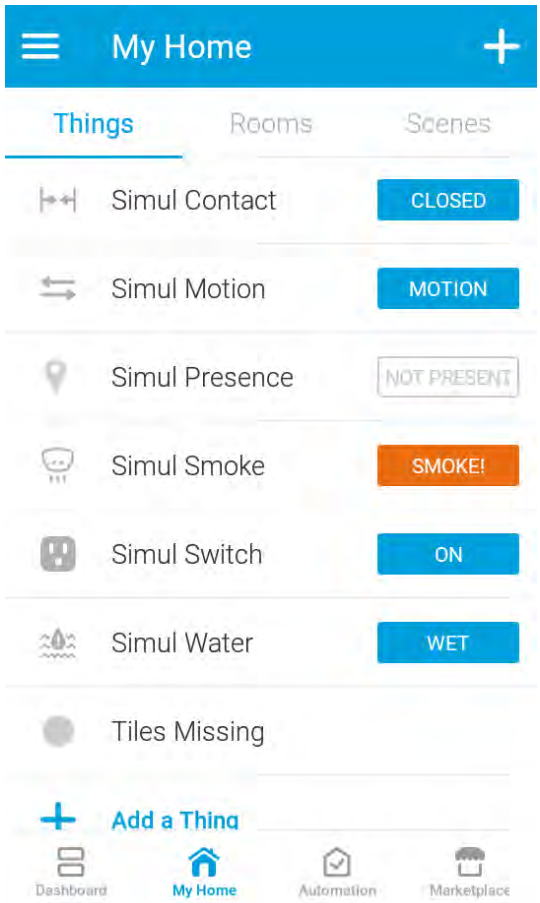

Figura 4.8: Dispositivos simulados en la aplicación móvil.

Nombre del dispositivo	Ubicación
Simul Contact	$(0.0, 0.05)$
Simul Smoke	$(0.0, 1.25)$
Simul Switch	$(6.25, 6.00)$
Simul Motion	$(3.0, -1.0)$
Simul Water	$(4.0, 1.2)$

Tabla 4.2: Ubicación de los dispositivos simulados.

Nombre del lugar	Ubicación
Living room	$(3.75, -1.70)$
Corridor	$(-0.5, 3.0)$
Bedroom	$(3.0, 6.0)$
Kitchen	$(3.75, 2.50)$
Sofa	$(2.0, -2.0)$
Dining room	$(6.0, -9.0)$
Entrance	$(0.0, -10.0)$

Tabla 4.3: Ubicación de los lugares predeterminados.

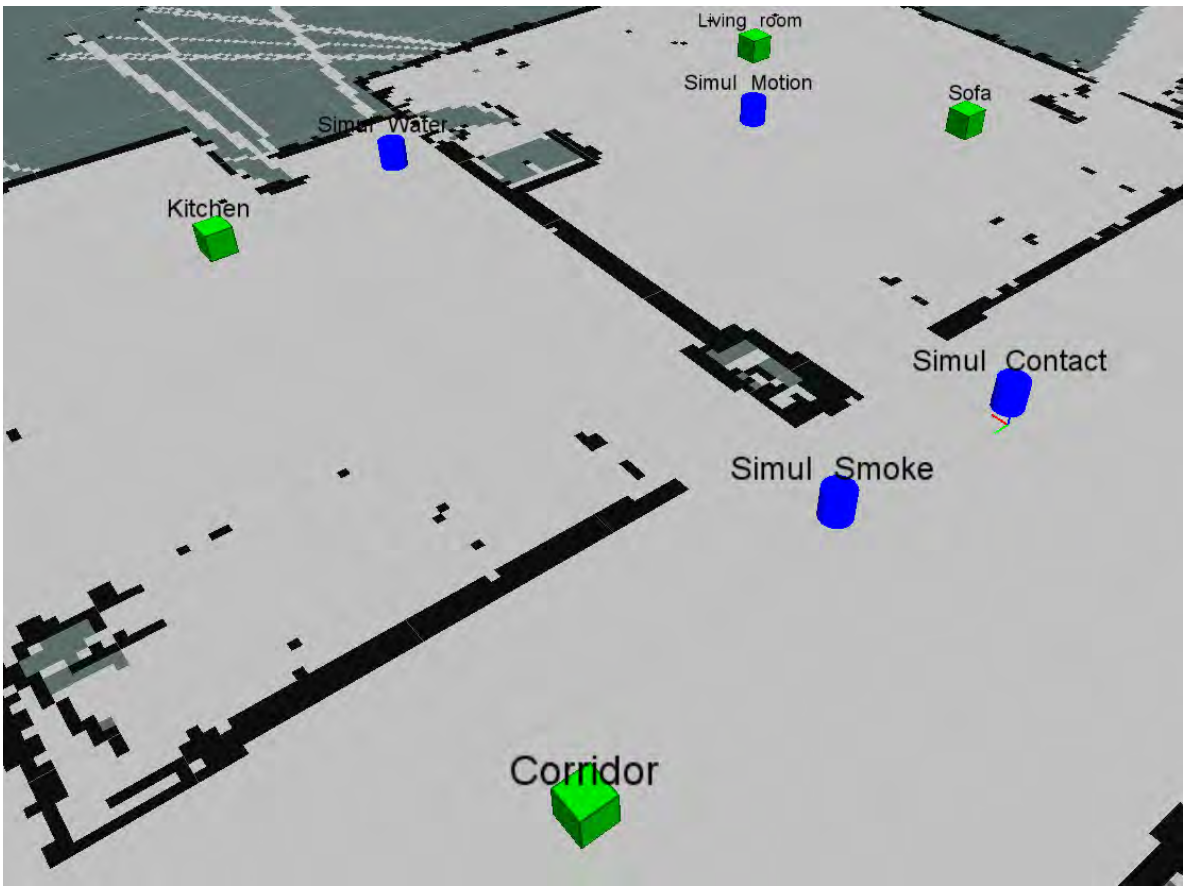

Figura 4.9: Ubicación de lugares predeterminados y dispositivos simulados.

Capítulo 5

Desarrollo: Robot móvil

5.1. Construcción del robot móvil

El kit Wild Thumper 4WD utilizado en este proyecto es muy fácil de ensamblar, cuenta con un chasis y cuatro motores previamente montados junto con su respectivo sistema de suspensión, únicamente toma unos minutos colocar las cuatro ruedas en su lugar con ayuda de una llave allen, la cual viene incluida en el kit. Sin embargo, dichos motores no cuentan con un encoder, los cuales son necesarios para la estimación de la pose del robot por medio del cálculo de la odometría. Por tal motivo se utilizaron los motores Dynamixel MX-12W, los cuales además de contar con un encoder con el suficiente número de bits (4096 ticks), tienen la ventaja de que son fáciles de controlar por medio de un paquete de instrucciones, enviando dicho paquete de instrucción es posible leer el estado del encoder y establecer la velocidad de giro deseada en el eje del motor, entre otras cosas.

Para colocar los motores Dynamixel, fue necesario retirar los motores originales junto con su respectivo sistema de suspensión, dicho sistema no es de utilidad para el robot, puesto que su objetivo es navegar en interiores con superficies planas. Para fijar los motores Dynamixel se optó por diseñar unos soportes con ayuda de un software de CAD (Figura 5.1), los soportes están hechos de acrílico de 4mm de espesor y se manufacturaron por medio de corte láser, el corte láser tiene la ventaja de ser un método de manufactura exacto y relativamente económico. Por otro lado, el acrílico de 4mm es un material con suficiente resistencia mecánica para este proyecto, puesto que el robot móvil tiene una masa de aproximadamente dos kilogramos. La exactitud en la manufactura es esencial para el correcto alineamiento de las ruedas, puesto que para el cálculo de la odometría se asume que las ruedas están perfectamente alineadas una con respecto de otra y que giran concéntricamente.

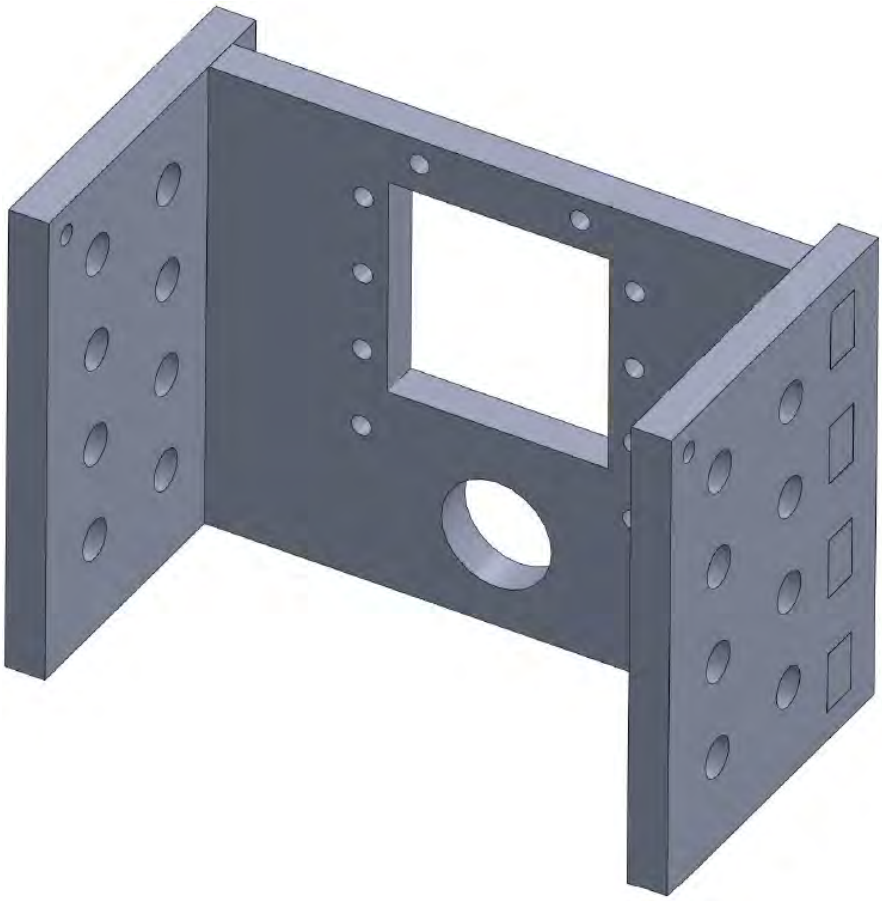

Figura 5.1: Soportes para la sujeción de motores Dynamixel MX-12W.

Inicialmente se tenía planeado construir el robot móvil con una configuración de locomoción Skid Steer, sin embargo, al realizar las pruebas pertinentes, el robot no era capaz de girar sobre su propio eje debido al bajo par de los motores Dynamixel. Por dicha razón, se optó por la configuración differential drive, sustituyendo las dos ruedas de tracción traseras por dos ruedas giratorias pasivas y, dejando las dos ruedas delanteras para la tracción y el direccionamiento. En la Figura 5.2 se muestra el robot móvil construido y la aplicación de SmartThings utilizada para operarlo.

Figura 5.2: Robot móvil construido.

5.1.1. Componentes del robot móvil

Para la construcción del robot móvil presentado en este trabajo se solicitó utilizar componentes con los que ya contaba el laboratorio de Bio Robótica de la División de Ingeniería Eléctrica (DIE) de la Facultad de Ingeniería. Los componentes utilizados fueron un chasis Dagu Wild Thumper 4WD, motores Dynamixel MX-12W, un sensor láser Hokuyo URG-04LX, una cloud Camera de la empresa D-Link, una Raspberry Pi 3, una batería LiPo, un regulador de voltaje ajustable S18V20VALV de la empresa pololu, un Arduino Uno y una tarjeta Arduino ThingShield de la empresa SmartThings y, a continuación se describen.

5.1.1.1. Chasis Dagu Wild Thumper 4WD

Wild Thumper 4WD es kit de robótica, incluye 4 ruedas con puntas, un chasis y cuatro motores con suspensión independiente como se puede ver en la Figura 5.3 [36]. El chasis está hecho de aluminio de 2mm de espesor y tiene una rejilla de agujeros de 4mm de diámetro espaciados cada 10mm, lo que permite el montaje de diversos componentes y accesorios, el interior del chasis cuenta con terminales de tornillo para realizar conexiones eléctricas. [29]

Figura 5.3: Dagu Wild Thumper 4WD todo terreno.

Con este kit la configuración para la locomoción es Skid Steer, los cuatro motores con los que cuenta tienen una reducción de 75:1 y tensión nominal de 7.2V, sin embargo, estos motores no cuentan con encoders, los cuales son necesarios para el cálculo de la odometría, razón por la cual se optó utilizar otros motores.

5.1.1.2. Motor Dynamixel MX-12W

El motor MX-12W es fabricado por la empresa ROBOTIS (Figura 5.4)[37], una de sus principales características es que puede operar en tres modos: Wheel Mode, Joint Mode y Multi-turn Mode. En los modos Joint y Multi-turn el motor se comporta como un servo convencional para el control de posición angular, con la diferencia de que en el modo Multi-turn se permiten múltiples giros. El modo Wheel o modo de rotación continua permite que el motor pueda tener revoluciones ilimitadas, este es el modo utilizado para el presente trabajo. Otra característica importante es que el motor utiliza un controlador PID para el control de posición y el control de velocidad, según sea el modo de operación, el cual es posible sintonizar. [30]

[36]Fuente: `https://a.pololu-files.com/picture/0J2895.1200.jpg`

[37]Fuente: `http://support.robotis.com/en/images/product/actuator/dynamixel/mx_series/mx12.jpg`

Figura 5.4: Motor Dynamixel MX-12W.

El usuario puede controlar este motor modificando los datos contenidos en una tabla de control, dichos datos se almacenan dentro de una memoria EEPROM y una memoria RAM, a través de un paquete de instrucciones enviado por medio del protocolo de comunicación serial half dúplex es posible leer y escribir los datos contenidos en dichas memorias.

Este motor cuenta con un encoder absoluto de 360° y 12 bits, lo que permite una resolución de aproximadamente 0.088°. Sus dimensiones son de 32mm x 50mm x 40mm, masa de 54.6 g, voltaje de alimentación entre 10V y 14.8V (se recomienda alimentación de 12V), velocidad a rotor libre de 470RPM con alimentación de 12V y temperatura de operación permisible entre -5°C y 80°C, cabe mencionar que el fabricante no especifica el par que entrega el motor. La tabla de control almacena datos acerca del estado actual y la operación dentro del motor Dynamixel, entre esos datos se encuentra el voltaje de alimentación y la temperatura interna.

5.1.1.3. Raspberry Pi 3

La Raspberry Pi 3 Model B (Figura 5.5)[38] es el módulo central de procesamiento de información para el robot móvil presentado en este trabajo, se encarga de comunicarse con los motores, el sensor láser y el microcontrolador Arduino Uno R3. La Raspberry Pi 3 es una computadora construida en una placa del tamaño de una tarjeta de crédito, fue desarrollada en Reino Unido por la Fundación Raspberry Pi para promover la enseñanza de ciencias de la computación en escuelas y países en

[38]Fuente: `https://www.raspberrypi.org/app/uploads/2017/05/Raspberry-Pi-3-1-1619x1080.jpg`

desarrollo, no obstante, el modelo se hizo más popular de lo previsto, vendiéndose en un mercado más amplio del mercado objetivo para usos como la robótica.

Figura 5.5: Raspberry Pi 3 Model B.

La Raspberry Pi 3 es la tercera generación de Raspberry Pi y substituyó a la Raspberry Pi 2 Model B en febrero de 2016. Incorpora un CPU Quad Core a 1.2GHz de 64 bits y 1GB de memoria RAM, los cuatro núcleos ARM Cortex A53 hacen que el dispositivo sea aproximadamente 50 % más rápido en comparación con la Raspberry Pi 2. Cuenta con periféricos para conectividad Wifi inalámbrica y Bluetooth, puerto Ethernet, puerto HDMI, cuatro puertos USB 2.0, puerto de video compuesto, puerto para conectar una cámara Raspberry Pi, puerto para conectar una pantalla táctil, puerto micro SD para cargar el sistema operativo y almacenar datos. La Fundación Raspberry Pi recomienda el uso de Raspbian, un sistema operativo Linux basado en Debian. [31]

La Raspberry Pi 3 cuenta con 40 pines de entrada/salida llamados GPIO (General Purpose Input/Output), los pines GPIO pueden configurarse como entrada de propósito general, salida de propósito general y en algunos casos con funciones especiales que dependen del pin en específico, entre las funciones alternativas están los buses I2C. Los pines suministran 3.3V, y la conexión de un pin GPIO a un voltaje superior a 3.3V podría causar daños, lo que significa que se debe de tener cuidado adicional cuando se conecta a un sistema RS232 o algún otro que utilice diferentes niveles de voltaje. Cuando un pin GPIO se configura como entrada de propósito general se puede configurar como una fuente de interrupción como se haría en un microcontrolador. Los pines GPIO son de utilidad para la adquisición de datos de sensores, o para la comunicación con otros dispositivos, por ejemplo, en este trabajo se utilizaron para la comunicación con motores Dynamixel MX-12W.

5.1.1.4. Sensor láser Hokuyo URG-04LX-UG01

El Hokuyo URG-04LX-UG01 es un telémetro láser de escaneo diseñado para su uso en interiores, es principalmente usado por estudiantes e investigadores para aplicaciones en robótica (Figura 5.6)[39]. Este sensor es utilizado para determinar la distancia hasta un objeto, su área de escaneo es un semicírculo de 240° con una resolución angular de 0.352° (683 pasos) aproximadamente y puede reportar distancias en un rango entre 20mm y 5600mm con una resolución de 1mm (Figura 5.7)[40]. Se alimenta por medio de un bus USB (5V y 500mA), tiene una masa aproximada de 160g, el tiempo de escaneo es de 100ms y puede operar a una temperatura ambiente entre -10°C y 50°C. [32]

Figura 5.6: Hokuyo URG-04LX-UG01.

[39]Fuente: http://www.robotshop.com/media/catalog/product/cache/1/image/900x900/9df78eab33525d08d6e5fb8d27136e95/h/o/hokuyo-urg-04lx-ug01-scanning-laser-rangefinder.jpg

[40]Fuente: http://www.robotshop.com/media/files/pdf/hokuyo-urg-04lx-ug01-specifications.pdf

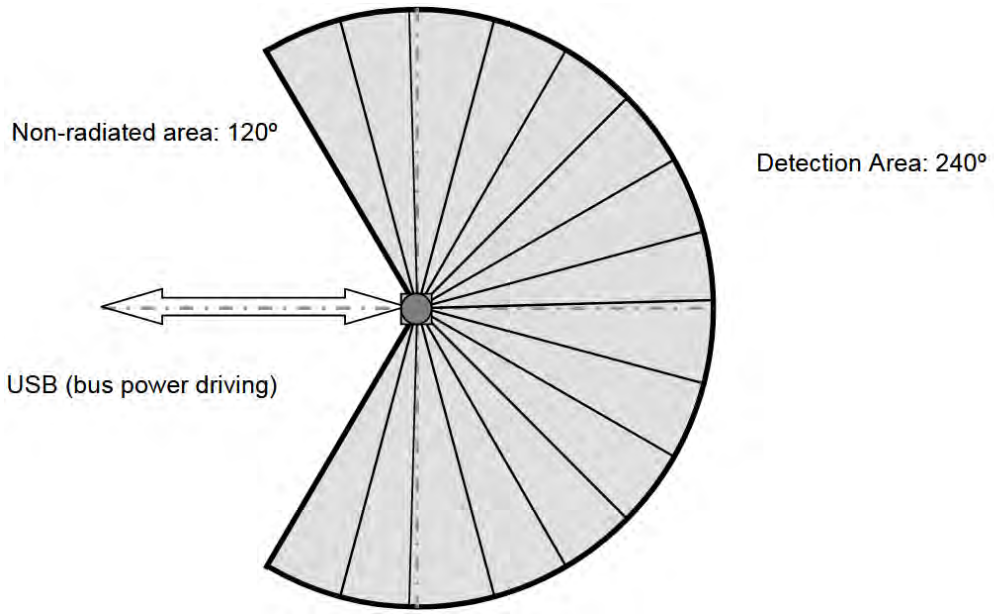

Figura 5.7: Área de escaneo del Hokuyo URG-04LX-UG01.

5.1.1.5. Arduino Uno R3 y Arduino ThingShield

Arduino UNO R3 es una placa para el microcontrolador ATmega328P (Figura 5.8)[41], tiene 14 pines digitales de entrada/salida de los cuales 6 se pueden usar como salidas PWM, cuenta con 6 pines para entradas analógicas, cada pin puede suministrar hasta 20mA, la frecuencia de reloj es de 16MHz, su masa es de 25g, sus dimensiones son 68.6mm x 53.4 mm, el voltaje de alimentación recomendando es de entre 7V y 12V (el voltaje de alimentación límite es de 6V-20V), el voltaje de operación es de 5V; la placa tiene un conector de alimentación, conexión USB y un botón de reinicio. [33] [34]

Arduino UNO es una de las placas más populares en el mercado, es la primera de una serie de placas y es el modelo de referencia para la plataforma Arduino, R3 es la tercera y última revisión de Arduino UNO. Arduino tiene una amplia comunidad de contribuidores y de soporte, lo cual hace que sea una manera muy fácil de trabajar con componentes electrónicos, a lo largo de los años se han hecho contribuciones por terceros para facilitar la generación de conocimiento, el cual es accesible y puede ser de gran ayuda para principiantes y expertos. Arduino surgió como una herramienta fácil para el prototipado rápido, dirigida a estudiantes sin experiencia en electrónica y programación. El software de Arduino (IDE, Integrated Development Environment) es fácil de utilizar para principiantes y es igualmente útil para usuarios avanzados, se puede ejecutar en Mac, Windows y Linux. La placa Arduino UNO tiene todo lo necesario para comenzar a desarrollar, simplemente basta con conectarla a una computadora con un cable USB.

[41] Fuente: `https://store-cdn.arduino.cc/usa/catalog/product/cache/1/image/500x375/f8876a31b63532bbba4e781c30024a0a/A/0/A000066_front_2.jpg`

Figura 5.8: Arduino UNO R3.

Con la SmartThings Arduino Shield (Arduino ThingShield), es posible agregar capacidades (funciones) de SmartThings a cualquier Arduino compatible con el pinout de la placa Arduino UNO, incluyendo Arduino Mega, Arduino Duemilanove y Arduino Leonardo. Las dimensiones de esta placa (Figura 5.9)[42] son 2.5in x 1.9in x 0.3in (6.35mm x 4.82mm x 0.76mm aproximadamente), tiene una masa de 8 onzas (aproximadamente 170g), cuenta con un led RGB y dos botones pulsadores. [35]

Para comenzar a utilizar la placa ThingShield sólo basta con montarla en alguna de las placas de Arduino antes mencionadas, en este caso en particular en la placa Arduino Uno R3, y alimentar la placa de Arduino como se hace usualmente. Emparejar la ThingShield con el Hub de SmartThings es un procedimiento simple, basta con abrir en un smartphone la aplicación móvil de SmartThings e ir al modo "Add SmartThings", uno vez hecho esto es necesario presionar el botón Switch en la ThingShield, entonces aparecerá este nuevo dispositivo en la aplicación móvil y estará vinculado a la respectiva cuenta.

Cuando se empareja la placa por primera vez no tiene ninguna funcionalidad, únicamente es el proceso por el cual se hace identificable para la aplicación móvil y para el entorno de desarrollo web. Para agregar funcionalidad a la ThingShield es necesario asociarle un *Device Handler*, SmartThings tiene varios ejemplos en el entorno de desarrollo web, pero para comenzar a hacer pruebas sugiere utilizar uno en particular, llamado "On/Off Shield (example)". Por otro lado, también es necesario programar el microcontrolador, y se requiere descargar la biblioteca que

[42]Fuente: `https://support.smartthings.com/hc/article_attachments/208683723/front-right-arduino-shield.png`

provee SmartThings para Arduino, una vez hecho esto bastará con abrir la IDE de Arduino y cargar el ejemplo nombrado "stLED". Este ejemplo en particular sirve para encender y apagar el led color verde de la ThingShield. La placa Arduino UNO seguirá teniendo la misma funcionalidad que siempre, solamente utilizará los pines digitales 2 y 3 para comunicarse con la ThingShield.

Figura 5.9: Arduino ThingShield.

5.1.1.6. Cloud Camera D-Link

La cámara IP D-Link DSC-932L es una solución de vigilancia para el hogar o la oficina, tanto para el día como para la noche (Figura 5.10)[43]. Cuenta con LEDs infrarrojos que permiten el monitoreo en condiciones de baja o nula luminosidad y su visión nocturna puede tener un alcance de hasta 5 metros. Por medio de un portal web o de una aplicación para smartphones o tabletas con iOS y Android, se puede configurar, ver y administrar la cámara. Tradicionalmente este tipo de cámaras requiere una configuración compleja, sin embargo, D-Link ofrece el servicio mydlink con el que el acceso y la gestión se facilita y no se requieren conocimientos técnicos. [36] [37]

[43]Fuente: `http://www.dlink.com/fi/fi/products/-/media/product-pages/dcs/932l/b1/dcs932lb1image-lfront.png`

Figura 5.10: Day/Night Cloud Camera DCS-932L.

La cámara se puede conectar con un cable Ethernet a un router, o se puede optar por conectarla de forma inalámbrica por WiFi. El tamaño de la imagen y los cuadros por segundo son configurables, el formato del video es MJPEG y el formato de las imágenes es JPEG. La resolución del video es de 640 x 480 pixeles a 20 cuadros por segundo y, 320 x 240 / 160 x 112 pixeles a 30 cuadros por segundo. Requiere alimentación de 5V y corriente de 1A, puede operar en un rango de temperaturas entre 0°C y 40°C.

5.1.1.7. Batería LiPo

Las baterías de Litio Polímero o baterías LiPo son un tipo de batería recargable y son utilizadas regularmente en los sistemas de radio control. Este tipo de baterías ha ganado popularidad debido a su capacidad, larga duración de la capacidad, altas tasas de descarga y en comparación con baterías como las NiCd (Níquel Cadmio) son más ligeras. Sin embargo, tienen como desventaja que el equipo para cargarlas y descargarlas puede ser costoso, así también, el darles un mal uso o trato puede provocar que se incendien o exploten, por ello necesitan un cuidado especial en la forma en que se cargan, descargan y almacenan.

Las baterías de LiPo tienen un sistema de clasificación, esto nos permite comparar las características de cada batería y nos ayuda a determinar cuál es la adecuada para nuestras necesidades, las tres clasificaciones principales son la tasa de descarga, la capacidad y el voltaje/número de celdas. Las baterías LiPo indican

con un número y una letra la cantidad de celdas y el tipo de conexión entre las celdas, S para conexión en serie y P para conexión en paralelo, por ejemplo, 3S significa que la batería tiene 3 celdas conectadas en serie. Cada celda tiene un voltaje nominal de 3.7V, por lo que en el caso del paquete de 3 celdas (3S) el voltaje nominal es de 11.1V. La capacidad es una medida de cuanta energía puede contener una batería, y la unidad para su medición es el mAh (miliamperio hora), dicha unidad indica el consumo de corriente que se requiere para descargar la batería en un lapso de tiempo de una hora. La tasa de descarga es una medida de que tan rápido se puede descargar una batería de forma segura y sin dañarse, se hace referencia a esta tasa como la clasificación C (de capacidad), para calcular el consumo máximo y seguro de amperios también es necesario conocer la capacidad. Supóngase una batería con tasa de descarga 10C y capacidad de 3000mAh, por lo tanto, 10 x 3000mAh = 30A, entonces la batería podrá manejar una carga máxima y continua de 30A. [38]

La tensión nominal de las celdas no es el voltaje de carga completo, las celdas están completamente cargadas cuando alcanzan 4.2V, y su voltaje mínimo seguro es de 3.0V. Descargar las celdas a un voltaje inferior puede degradar de forma permanente la capacidad de retener la carga.

5.1.1.8. Regulador de Voltaje

El regulador de voltaje ajustable S18V20VALV, es un regulador step-up/step-down (Figura 5.11)[44], lo que significa que aumenta o reduce la tensión de entrada según sea necesario, esto lo hace muy útil para aplicaciones en donde el voltaje de la fuente de alimentación puede variar mucho, como por ejemplo en las baterías que comienzan por arriba del voltaje deseado en la salida y se descargan por abajo del voltaje regulado. [39]

Figura 5.11: Regulador de voltaje ajustable S18V20VALV.

[44]Fuente: `https://a.pololu-files.com/picture/0J5031.600x480.jpg`

La tensión en la salida se puede ajustar en un rango de 4V a 12V, la entrada admite un rango de 3V a 30V, la corriente disponible en la salida depende principalmente del voltaje a la entrada y el voltaje a la salida, pero es de alrededor de 2A cuando el voltaje de entrada es cercano al voltaje de salida. Este regulador tiene protección contra sobre voltaje, protección contra sobre corriente, apagado térmico (típicamente se activa a 165°C) y bloqueo cuando el voltaje de entrada está por debajo de 2.5V, causando que el regulador se apague.

5.1.2. Placa para conexiones y comunicación con los motores

Para la comunicación con los motores Dynamixel el fabricante sugiere utilizar los circuitos integrados mostrados en la Figura 5.12[45]. Los motores utilizan el mismo canal para la transmisión y recepción de datos, por lo cual es necesario utilizar dos buffers tri estado (74HC126) y una compuerta NOT (74HC04) para conmutar entre los modos transmisión y recepción. Sin embargo, esto implica el uso de dos circuitos integrados, por lo cual se optó por utilizar el circuito integrado 74HC241 y agregar los circuitos integrados 74LS14 y LM339, estos dos últimos para filtrar los datos enviados y recibidos y, garantizar los niveles de voltaje de alto y bajo. Se tomó esta decisión debido a que el robot estará sometido a vibraciones mecánicas al desplazarse, las cuales pueden influir de forma negativa en la comunicación con los motores. Así mismo es necesario adaptar los niveles de voltaje ya que los motores Dynamixel y la Raspberry Pi 3 trabajan con distintos niveles. En la Figura 5.13 se muestra el diagrama de conexiones para los componentes que se soldaron en una tarjeta perforada.

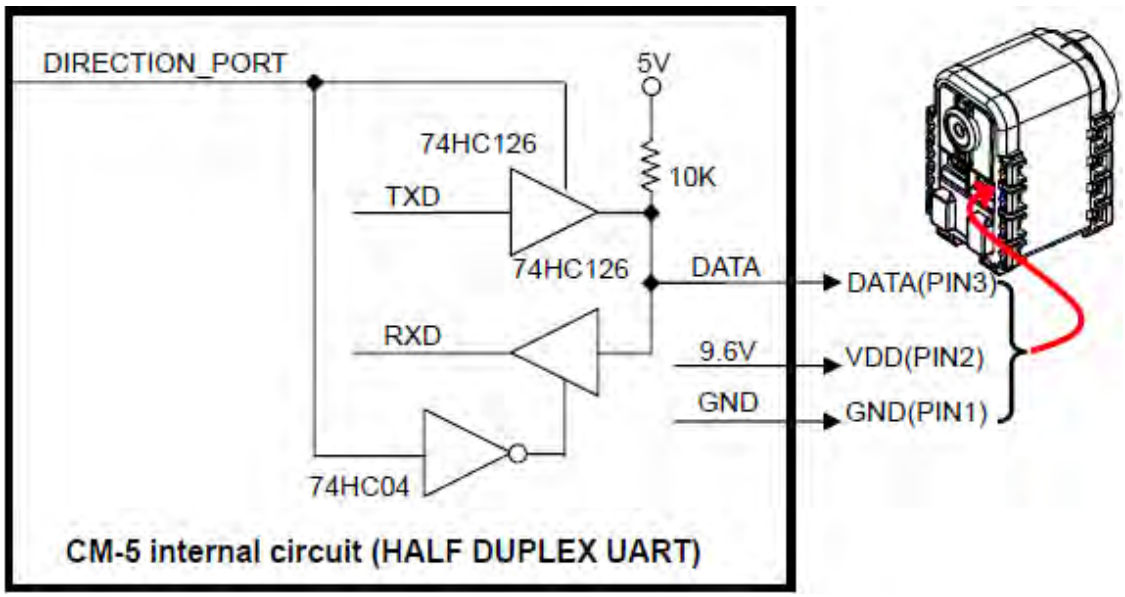

Figura 5.12: Diagrama de conexiones sugerido por el fabricante.

[45]Fuente: `http://support.robotis.com/en/images/product/actuator/dynamixel/ax_series/ax_series_circuit.png`

La placa elaborada cuenta con conectores para el regulador de voltaje, la Raspberry Pi 3, los motores Dynamixel y una batería LiPo. Para la identificación de los cables se utilizó una convención de colores, el color blanco es para cables de tierra, rojo para cables de 5V, negro para cables de 12V y gris para cables de transmisión de datos. Cabe mencionar que el fabricante de los motores Dynamixel proporciona cables para su conexión, por lo que estos cables no siguen la convención antes mencionada y hay que tener particular cuidado al momento de conectarlos.

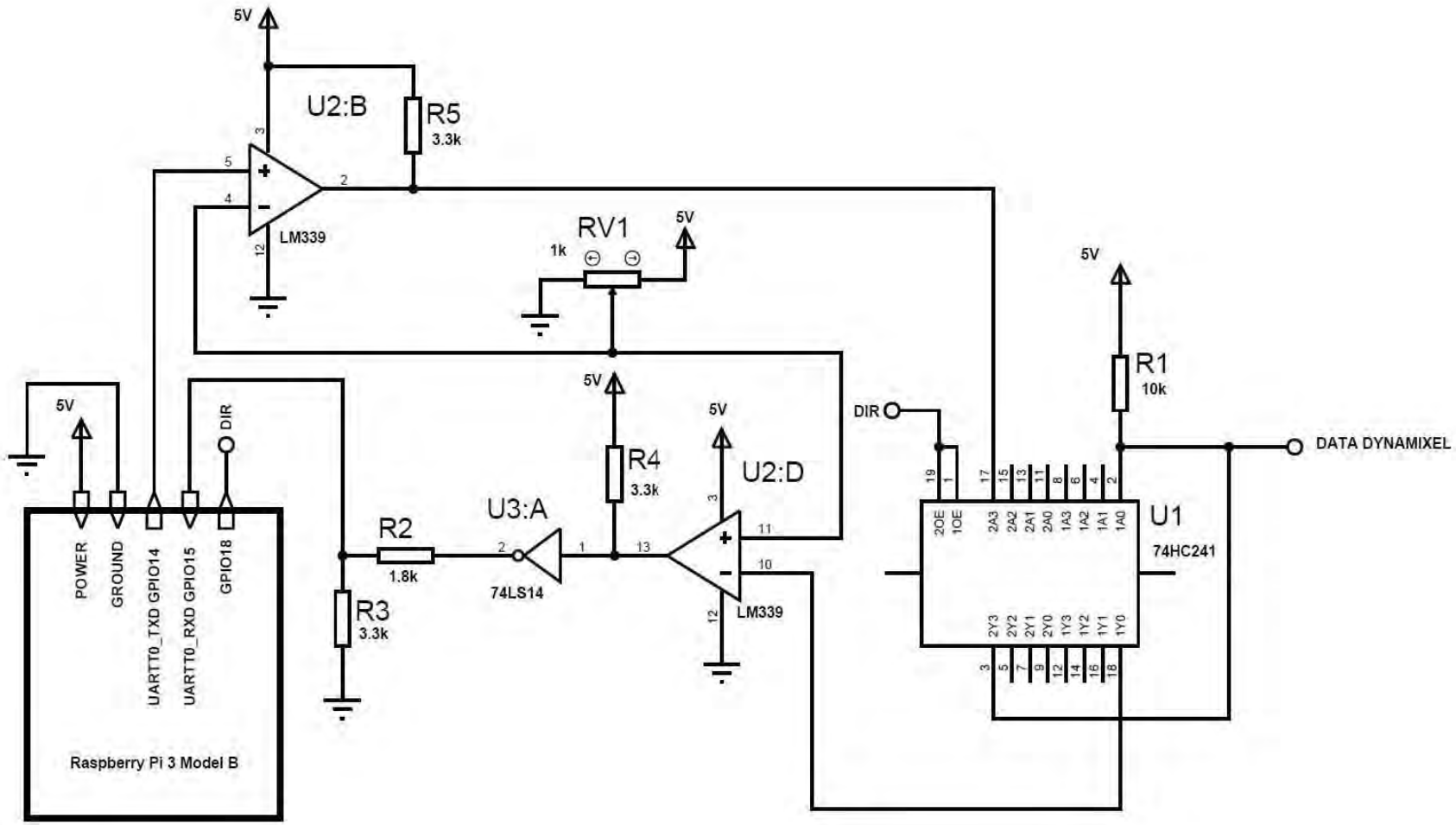

Figura 5.13: Diagrama de conexiones.

5.2. Modelo cinemático del robot

Las ecuaciones de movimiento que se emplean comúnmente para el control de un robot móvil se basan en modelos cinemáticos. La cinemática es la ciencia que estudia el movimiento sin considerar las fuerzas que lo ocasionan. Dentro de la cinemática se estudian la posición, la velocidad y la aceleración. La configuración diferencial utiliza dos ruedas motrices montadas en un eje de giro común (Figura 5.14)[46], y cada rueda puede ser accionada independientemente hacia adelante o hacia atrás. Al variar la velocidad de cada rueda, el robot gira alrededor de un punto que yace sobre el eje de giro común de ambas ruedas, ese punto se denomina centro instantáneo de curvatura (ICC, Instantaneous Center of Curvature) o centro instantáneo de rotación (ICR, Instantaneous Center of Rotation). La posición del centro instantáneo de rotación depende de las velocidades de cada rueda, lo que permite al robot girar sobre su propio eje, trazar arcos de circunferencia y desplazarse en línea recta. Esto representa una restricción de movimiento, por la cual el robot únicamente puede moverse en una dirección perpendicular a el eje de giro común de ambas ruedas.

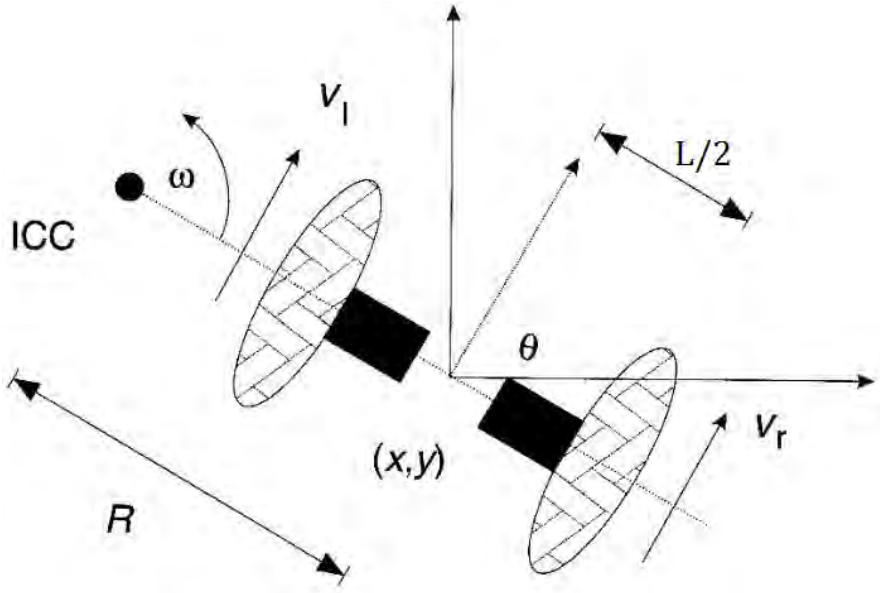

Figura 5.14: Robot móvil con configuración diferencial.

Cuando un cuerpo gira alrededor de un punto, cada punto del cuerpo tiene la misma velocidad angular, la velocidad tangencial de cualquier punto del cuerpo es proporcional a la distancia entre dicho punto y el punto de rotación (ICR). Dado que la velocidad angular es la misma para todo el robot móvil y, bajo las consideraciones de que el robot es un cuerpo rígido, el robot opera en un plano horizontal y no existe deslizamiento entre las ruedas y el suelo, se pueden deducir las siguientes ecuaciones:

[46]Fuente: `https://chess.eecs.berkeley.edu/eecs149/documentation/differentialDrive.pdf`

$$\omega\left(R+\frac{L}{2}\right)=v_R \tag{5.1}$$

$$\omega\left(R-\frac{L}{2}\right)=v_L \tag{5.2}$$

En donde L es la distancia entre los centros de las dos ruedas motrices (en la Figura 5.14 se muestra la longitud $\frac{L}{2}$), v_R y v_L son las velocidades lineales de las ruedas derecha e izquierda respectivamente, ω es la velocidad angular y, finalmente, R es la distancia entre el ICR y el punto medio entre las ruedas motrices. Despejando R y ω de las ecuaciones (5.1) y (5.2), tenemos:

$$R=\frac{L\left(v_R+v_L\right)}{2\left(v_R-v_L\right)}$$

$$\omega=\frac{v_R-v_L}{L}$$

Con estas últimas ecuaciones podemos observar los siguientes casos de movimiento:

1. Cuando la velocidad de las dos ruedas es la misma, $v_R=v_L$, el robot se mueve en línea recta hacia adelante o hacia atrás. R tiende a infinito y la velocidad angular es cero.

2. Cuando las velocidades de las ruedas son distintas, pero su sentido de giro es el mismo (hacia adelante o hacia atrás), $v_R \neq v_L$, el robot se moverá a lo largo de un arco de circunferencia. Entonces la velocidad angular y R serán distintos de cero.

3. Si la velocidad de la rueda izquierda es cero y la velocidad de la rueda derecha es distinta de cero, entonces el robot girará apoyándose sobre la rueda izquierda. Por lo tanto, $R=\dfrac{L}{2}$ y $\omega=\dfrac{v_R}{L}$. Lo mismo sucederá en el caso opuesto.

4. Por último, si las velocidades son de igual magnitud, pero el sentido de giro de las ruedas es opuesto, $v_R=-v_L$, el robot girará alrededor del punto medio entre las ruedas motrices. $R=0$ y $\omega=\dfrac{2v_R}{L}$.

La configuración de un robot móvil que se mueve en el plano está dada por tres coordenadas generalizadas (o variables de estado), es decir, dos variables para la posición y una para la orientación. Para determinar la pose del robot en el plano es necesario establecer un marco de referencia global del plano y un marco de referencia local del robot. En la Figura 5.15 [47] los ejes X_I y Y_I definen un marco de referencia

[47] Fuente: [13]

inercial en el plano (marco de referencia global); para especificar la pose del robot en el plano, se elige un punto P en el robot, este punto conviene que sea el punto medio entre las ruedas motrices del robot y, será el punto que establezca el origen del sistema de referencia local asociado al robot.

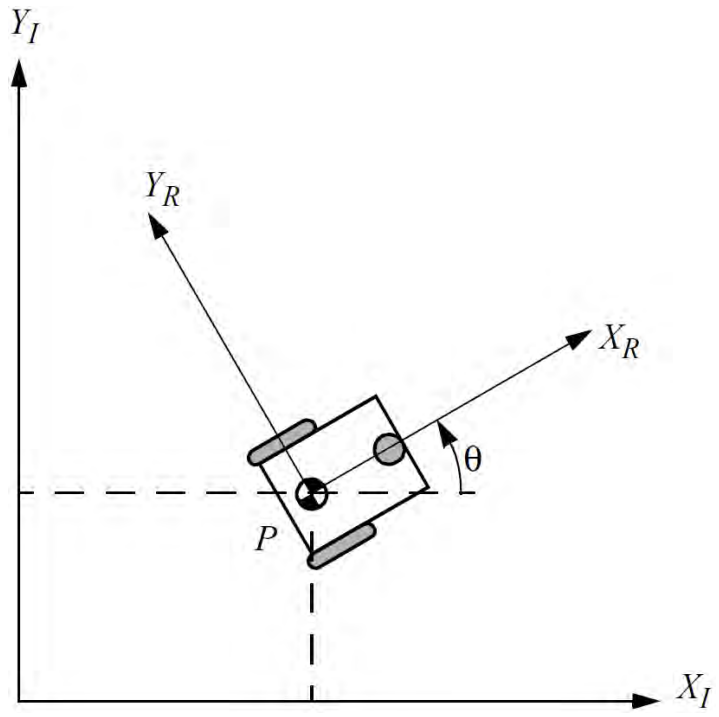

Figura 5.15: Marco de referencia global y marco de referencia local del robot.

La pose del robot en el plano con respecto del marco de referencia global está especificada por dos coordenadas cartesianas (x, y) y por el ángulo de orientación (θ) del marco de referencia local del robot con respecto del marco de referencia global, por lo tanto, se puede definir la pose del robot mediante el siguiente vector de estados de tres elementos (para no confundir la coordenada cartesiana x con el vector de estados, el vector de estados se denota como: $\overline{x}$):

$$\overline{x} = [x, y, \theta]^T$$

Como ya se dijo el punto P es el punto de referencia en el robot, la velocidad lineal de este punto se considerará como la velocidad lineal del robot, en las ecuaciones (5.1) y (5.2) el producto de la velocidad angular y la distancia R da como resultado la velocidad lineal de dicho punto. Por lo tanto, se pueden rescribir dichas ecuaciones de la siguiente manera:

$$v_R = v + \omega \frac{L}{2} \tag{5.3}$$

$$v_L = v - \omega \frac{L}{2} \tag{5.4}$$

En donde v es la velocidad lineal del robot. Despejando de estas últimas ecuaciones la velocidad lineal del robot se tiene:

$$v = \frac{v_R + v_L}{L}$$

Finalmente, el modelo cinemático del sistema en variables de estado con respecto del marco de referencia global queda de la siguiente forma:

$$\dot{x} = v\cos\theta = \frac{v_R + v_L}{L}\cos\theta \tag{5.5}$$

$$\dot{y} = v\sin\theta = \frac{v_R + v_L}{L}\sin\theta \tag{5.6}$$

$$\dot{\theta} = \omega = \frac{v_R - v_L}{L} \tag{5.7}$$

Y el vector de entradas del sistema es el siguiente:

$$u = [v_R, v_L]^T$$

5.2.1. Restricción no holónomica

Como se dijo en la Subsección 3.2.3, la diferencia $n - k$ indica la existencia de no holonomicidad. Un robot móvil con configuración diferencial puede producir dos movimientos independientes por medio de las velocidades de sus dos ruedas (entradas al sistema), es decir, $k = 2$, y tiene tres grados de libertad (coordenadas generalizadas), es decir, $n = 3$. Por lo tanto, tiene una restricción no holonómica ($n - k = 1$).

La restricción no holónomica se puede obtener a partir de las ecuaciones (5.5) y (5.6) del modelo cinemático:

$$v = \frac{\dot{x}}{\cos\theta} = \frac{\dot{y}}{\sin\theta}$$

Y se puede reescribir de la siguiente manera:

$$\dot{x}\sin\theta - \dot{y}\cos\theta = 0 \tag{5.8}$$

La Ecuación 5.8 expresa el hecho de que el punto P en el robot móvil se mueve únicamente en la dirección del eje X_R del marco de referencia local asociado al robot (Figura 5.15).

5.2.2. Representación de la posición y orientación en ROS y cálculo de la odometría

Los sistemas de coordenadas en ROS siempre están en 3D y son diestros, es decir, el eje X hacia adelante, el eje Y hacia la izquierda y el eje Z hacia arriba. La

relación entre dos marcos de referencia está representada por una pose relativa de 6 GDL, primero una traslación seguida de una rotación. Si W y U son dos marcos de referencia, la pose de U en W está dada por la traslación del origen de W hasta el origen de U, y la rotación de los ejes coordenados de U en W. Entonces dicha relación se lee como la pose del marco de referencia U con respecto del marco de referencia W.

En ROS existen convenciones de nomenclatura y significados para marcos de referencia utilizados en los robots móviles, con el fin de integrar y reutilizar los aportes de software hechos por la comunidad de ROS.

El sistema de referencia llamado `base_link` es el marco coordenado asociado a la base del robot móvil. `base_link` puede tener cualquier origen, orientación y posición en la base del robot móvil, cada robot tendrá un punto en la base que proporciona un lugar de referencia adecuado. En el caso del robot móvil desarrollado en este trabajo, dicho punto es el punto medio entre las ruedas motrices.

`odom` es idealmente un marco de referencia global fijo, sin embargo, la pose de un robot móvil en el marco `odom` puede sufrir de deriva sin límite a lo largo del tiempo. En una configuración típica, la pose del robot con respecto del marco de referencia `odom` se calcula mediante una fuente de odometría como, por ejemplo, una unidad de medición inercial o con los encoder en las ruedas del robot móvil. `odom` es un marco de referencia útil a corto plazo, pero debido a la deriva se convierte en un marco de referencia poco útil a largo plazo. La pose de un robot en el marco `odom` es continua, lo que quiere decir que la pose siempre evoluciona de forma fluida, sin saltos discretos.

El sistema de coordenadas llamado `map` es un marco de referencia fijo global, con el eje Z apuntando hacia arriba. La pose de una plataforma móvil con respecto del marco de referencia `map` no debería de sufrir significativamente de deriva con respecto del tiempo. La pose del robot con respecto del marco de referencia `map` no es continua, es decir, puede tener saltos discretos en cualquier momento. En una configuración típica, existe un medio de localización que constantemente recalcula la pose del robot con respecto de un mapa basándose en las observaciones más recientes de un sensor exteroceptivo, eliminando de este modo la deriva, pero causando saltos discretos cuando se obtiene nueva información de tal sensor. Haciendo de esta forma al marco de referencia `map` un marco de referencia global útil a largo plazo.

Cada marco de referencia tiene un marco de referencia padre (excepto el primero) y sólo puede tener un único padre, tal y como se puede observar en la Figura 5.16 [48]. El marco de referencia `map` es el padre de `odom`, y `odom` es el padre de `base_link`.

[48] Fuente: `http://www.ros.org/reps/rep-0105.html`

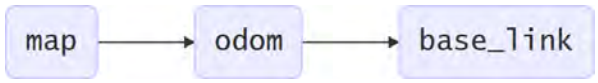

Figura 5.16: Representación de árbol de los marcos de referencia.

Se sabe que al variar la velocidad de cada rueda el robot gira alrededor del centro instantáneo de rotación trazando arcos de circunferencia. La longitud de arco es una medida de la longitud de un segmento de una curva cualquiera, para el caso de una circunferencia de radio r la longitud de arco dada por un ángulo φ es igual al producto de r por el ángulo φ. Si durante un periodo de tiempo T las velocidades de las ruedas derecha e izquierda del robot son constantes, con base en lo anterior y en la Figura 5.17 (y bajo las mismas consideraciones que se tuvieron para obtener el modelo cinemático), se pueden plantear las siguientes ecuaciones:

$$\Delta\theta\left(R+\frac{L}{2}\right)=S_R$$
$$\Delta\theta\left(R-\frac{L}{2}\right)=S_L$$

En donde R es la distancia entre el ICR y el punto medio entre las ruedas motrices, S_L y S_R son las distancias recorridas por cada una de las ruedas, $\Delta\theta$ es el ángulo que define a las distancias recorridas por cada una de la ruedas y L es la distancia entre los centros de las dos ruedas motrices. Para calcular la pose del robot con respecto del marco de referencia `odom` se puede calcular R y $\Delta\theta$ a partir de las anteriores ecuaciones, quedando de la siguiente manera:

$$\Delta\theta=\frac{S_R-S_L}{L}$$
$$R=\frac{L}{2}\frac{S_R+S_L}{S_R-S_L}=\frac{S_R+S_L}{2\Delta\theta}$$

La distancia L es un parámetro conocido del robot y las distancias recorridas por cada una de las ruedas se pueden determinar por medio del número de ticks contados por sus respectivos encoder. Los motores utilizados cuentan con un encoder de 4096 ticks por revolución, el radio de las ruedas utilizadas es $r=0.0625m$, sabiendo esto, la distancia recorrida por la rueda derecha se calcula de la siguiente manera (lo mismo aplica para la rueda izquierda):

$$S_R=\left(\frac{\text{número de ticks}}{4096\dfrac{\text{ticks}}{\text{revolución}}}\right)(2\pi\text{rad})\,(0.0625\text{m})$$

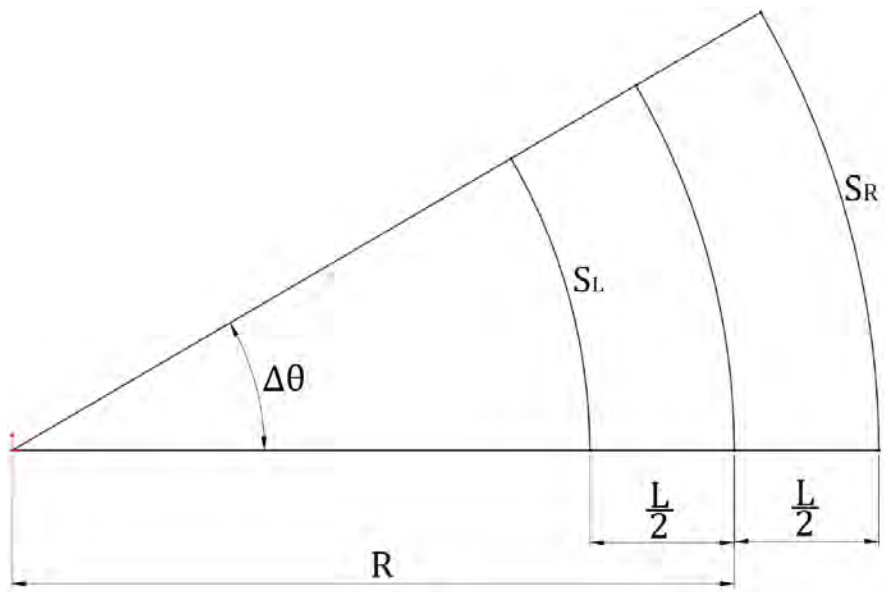

Figura 5.17: Distancias recorridas por las ruedas derecha e izquierda en un periodo de tiempo.

Considérese la Figura 5.18, para calcular la odometría es necesario conocer la pose del robot en el instante de tiempo k, es decir, se debe conocer (x_k, y_k, θ_k), durante un periodo de tiempo T las velocidades de las ruedas derecha e izquierda son constantes, por consiguiente la trayectoria descrita por el robot es un arco de circunferencia de radio R y la orientación del robot en el instante de tiempo $k+1$ es la siguiente:

$$\theta_{k+1} = \theta_k + \phi$$

Sin embargo, el ángulo que se calcula por medio de los encoder es $\Delta\theta$, tal y como se observa en la Figura 5.18, pero para conocer la nueva orientación del robot se requiere conocer el ángulo ϕ, no obstante, es posible deducir su valor. La suma de los ángulos internos de un triangulo es $180°$, por lo tanto $\Delta\theta + 90° + \omega = 180°$. La suma de los ángulos ω, ϕ y el ángulo recto formado por los ejes del sistema de referencia `base_link` en el instante de tiempo $k+1$ es igual a $180°$, es decir, $\omega + 90° + \phi = 180°$. En consecuencia:

$$\Delta\theta + 90° + \omega = \omega + 90° + \phi$$
$$\Delta\theta = \phi$$

De modo que la orientación del robot en el instante de tiempo $k+1$ se calcula como:

$$\theta_{k+1} = \theta_k + \Delta\theta$$

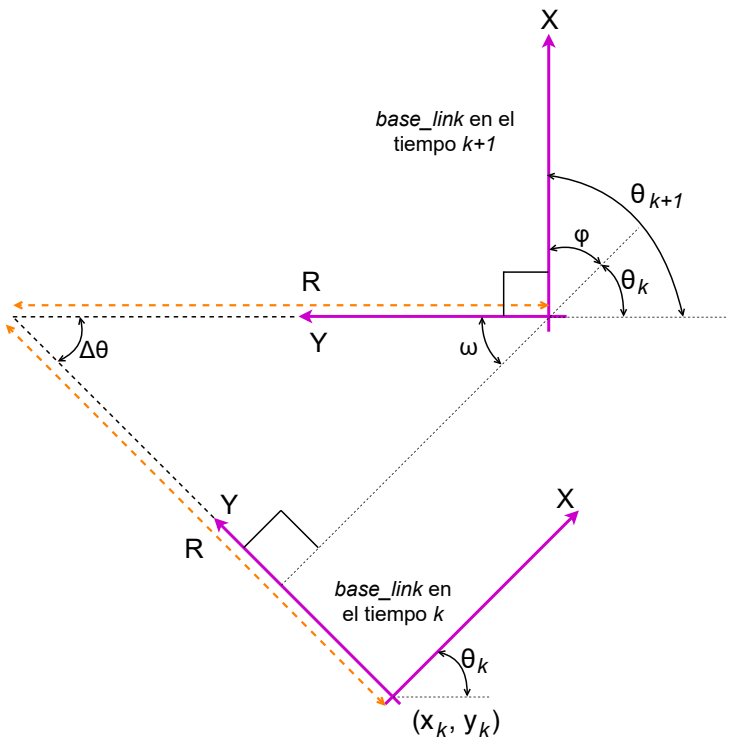

Figura 5.18: Cambio en la orientación del robot en un periodo de tiempo.

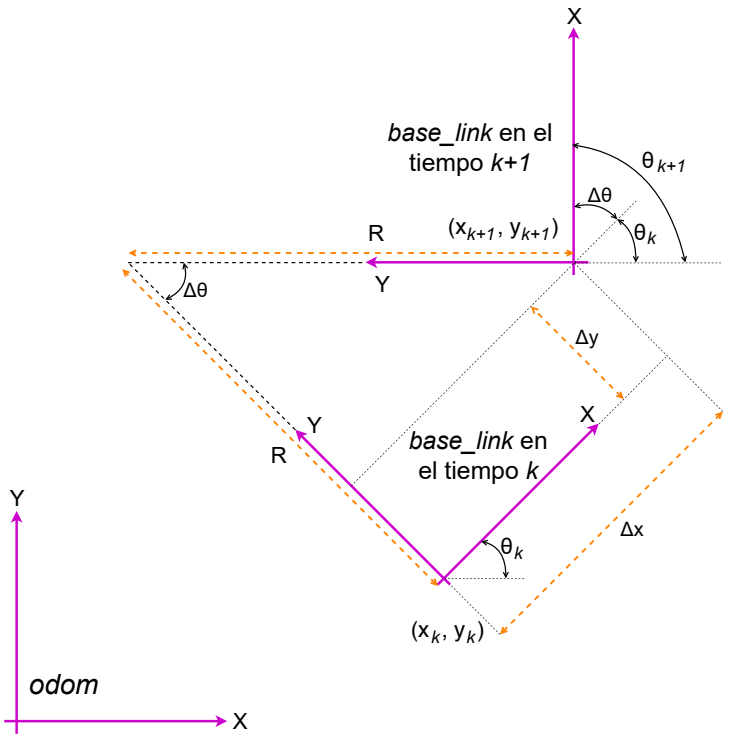

Figura 5.19: Cambio en la posición del robot en un periodo de tiempo.

Considérese la Figura 5.19, en el instante de tiempo $k+1$ el robot se ha desplazado las distancias Δx y Δy, las cuales se pueden calcular de la siguiente manera:

$$\begin{aligned}\Delta x &= R\sin(\Delta\theta)\\ \Delta y &= R - R\cos(\Delta\theta) = R\left[1-\cos(\Delta\theta)\right]\end{aligned}$$

Para conocer la posición del robot en el instante de tiempo $k+1$ con respecto de sistema del referencia `odom` se puede aplicar la matriz de transformación homogénea al vector en coordenadas homogéneas $[\Delta x, \Delta y, 1]^T$:

$$\begin{bmatrix} x_{k+1} \\ y_{k+1} \\ 1 \end{bmatrix} = \begin{bmatrix} \cos(\theta_k) & -\sin(\theta_k) & x_k \\ \sin(\theta_k) & \cos(\theta_k) & y_k \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \Delta x \\ \Delta y \\ 1 \end{bmatrix}$$

Finalmente, la pose del robot con respecto del sistema de referencia `odom` se calcula como se muestra a continuación:

$$\begin{aligned}x_{k+1} &= x_k + \Delta x\cos(\theta_k) - \Delta y\sin(\theta_k)\\ y_{k+1} &= y_k + \Delta x\sin(\theta_k) + \Delta y\cos(\theta_k)\\ \theta_{k+1} &= \theta_k + \Delta\theta\end{aligned}$$

5.3. Nodos de ROS

El software desarrollado para este trabajo está organizado en varios nodos y paquetes, en los cuales se implementa la comunicación con el hardware, el cálculo de rutas, leyes de control para el robot móvil y la toma de decisiones. Cabe mencionar que también se utilizaron nodos aportados por terceros (la comunidad de ROS).

5.3.1. Mobile base node

El nodo `mobile_base_node.py` se encuentra en el paquete `mobile_base`, este nodo se encarga de controlar el movimiento del robot móvil estableciendo las velocidades deseadas en cada uno de los dos motores Dynamixel, dichas velocidades se establecen en metros por segundo. Además, este nodo se encarga de determinar la posición y orientación del robot móvil con respecto del marco de referencia `odom` utilizando `tf`. Para estimar la pose primero se lee el estado actual de cada encoder, después se calcula la distancia recorrida por cada rueda y por último se calcula la odometría. Cabe mencionar que cuando el robot móvil comienza a operar las condiciones iniciales son nulas, es decir, el robot comienza en el origen del marco de referencia global y no está rotado con respecto de este. En la Tabla 5.1 se muestran los temas que publica y a los que se suscribe este nodo.

Temas publicados	`/hardware/robot_state /base_battery`
	
	`/tf`
		Voltaje actual de la batería.
		
		Transformación de `odom` a `base_link`.

Tópicos suscritos	`/hardware/mobile_base /speeds`	Velocidades deseadas en las ruedas derecha e izquierda.

Tabla 5.1: Mobile base node: Temas publicados y suscritos.

5.3.1.1. Comunicación con los motores Dynamixel

El nodo `mobile_base_node.py` se encarga de la comunicación con los motores Dynamixel, los cuales se controlan a través de tramas de bytes y comparten el mismo bus de comunicación, sin embargo, cada uno de los motores puede ser identificado por un ID. La comunicación entre este nodo de ROS y los motores se hace enviando y recibiendo paquetes de datos. Los paquetes son de dos tipos: se denomina Instruction Packet al paquete de datos que reciben los motores, y Status Packet, es el paquete con el que responden los motores Dynamixel. La estructura de Instruction Packet se muestra en la Figura 5.20[49] y la estructura de Status Packet en la Figura 5.21[50]. Adicionalmente existe un paquete de instrucciones especial para la serie MX de Dynamixel, se trata de Bulk Read, y sirve para leer de forma simultánea los registros de la tabla de control de varios motores Dynamixel mediante un único paquete de instrucciones, lo cual es una ventaja ya que reduce el tiempo de comunicación.

Figura 5.20: Instruction Packet.

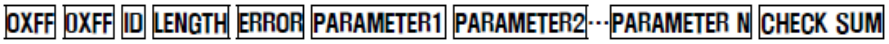

Figura 5.21: Status Packet.

El protocolo de comunicación implementado por los motores Dynamixel es half dúplex, lo que significa que no se puede transmitir y recibir datos de forma

[49]Fuente: `http://support.robotis.com/en/images/product/actuator/dynamixel/communication/comm01.png`

[50]Fuente: `http://support.robotis.com/en/images/product/actuator/dynamixel/communication/comm02.png`

simultánea. Este método es el que se utiliza comúnmente cuando varios dispositivos comparten un mismo bus, y mientras un dispositivo está transmitiendo todos los demás dispositivos deben estar en modo de entrada o escucha. Los motores Dynamixel poseen un controlador que establece la dirección de la comunicación en modo entrada, y sólo cambia a modo salida cuando está transmitiendo un paquete de datos.

Los motores Dynamixel cuentan con un controlador PID para regular la velocidad y la posición (Figura 5.22)[51], según corresponda el modo de operación. Las ganancias se pueden ajustar al escribir los registros correspondientes de las tablas de control de cada uno de los motores. Las ganancias que se obtuvieron en la sintonización de los controladores son las mostradas en la Tabla 5.2.

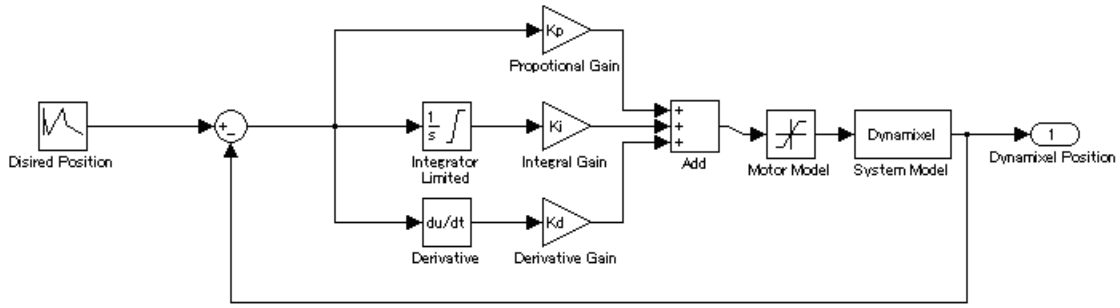

Figura 5.22: Controlador PID de los motores Dynamixel MX-12W.

Ganancias	Motor izquierdo	Motor derecho
Ganancia P	12	8
Ganancia I	2	0
Ganancia D	14	10

Tabla 5.2: Ganancias de los controladores PID de los motores.

Los paquetes de datos que este nodo de ROS envía a los motores son para configurar las ganancias de los controladores PID, establecer las velocidades en los motores y leer el estado actual de los encoder.

5.3.2. Low level control node

El objetivo de control para el robot móvil es el seguimiento de una ruta para poder llegar desde su ubicación actual hasta una ubicación destino. El nodo `low_level_control_node` está alojado dentro del paquete `low_level_control`, su propósito es especificar las entradas del sistema que le permitan al robot móvil llegar a una posición deseada (x_g, y_g). En la Tabla 5.3 se muestran los temas que publica y a los que se suscribe este nodo.

[51]Fuente: `http://support.robotis.com/en/images/product/actuator/dynamixel/mx_series/pidcontrol.png`

Temas publicados	`/hardware/mobile_base /speeds`	Tema para publicar las velocidades calculadas por las leyes de control.
	`/navigation/goal_reached`	Tema por el cual se publica una bandera que indica que el robot móvil llegó a su destino.
Temas suscritos	`/navigation/a_star_path`	Tema para obtener la ruta calculada por el nodo `path_calculator_node`.
	`/navigation/start_path`	Tema por el cual el nodo recibe una bandera que indica que el robot puede comenzar a seguir una ruta.

Tabla 5.3: Low level control node: Temas publicados y suscritos.

5.3.2.1. Leyes de control

El modelo obtenido para el robot móvil no considera la parte dinámica, y las velocidades lineales de las ruedas derecha e izquierda son las entradas al sistema. Realmente esto no sucede de esta forma, ya que las señales de control son los voltajes aplicados a cada uno de los motores, pero dado que estos tienen un propio controlador PID para la regulación de la velocidad, se asume que el tiempo de asentamiento es despreciable, es decir, que la curva de respuesta de las velocidades de los motores llega lo suficientemente rápido al estado estable y que la respuesta transitoria no afecta.

El sistema de control aplicado al robot móvil es realimentado, lo cual permite lidiar con incertidumbres y dinámicas omitidas en el modelado, así como atenuar el efecto de las perturbaciones. Tomando en cuenta la Figura 5.23[52], el ángulo θ_g corresponde al ángulo formado por el vector $[x_g - x_r, y_g - y_r]^T$, la posición (x_r, y_r) corresponde a la ubicación actual del robot, θ_r es el ángulo que define la orientación actual del robot, el punto (x_g, y_g) corresponde a la posición deseada y e_θ es el error de ángulo.

[52]Fuente: `https://github.com/mnegretev/RoboticsCourses/blob/master/Exercises/RMYAI/Prac06/Prac06.pdf`

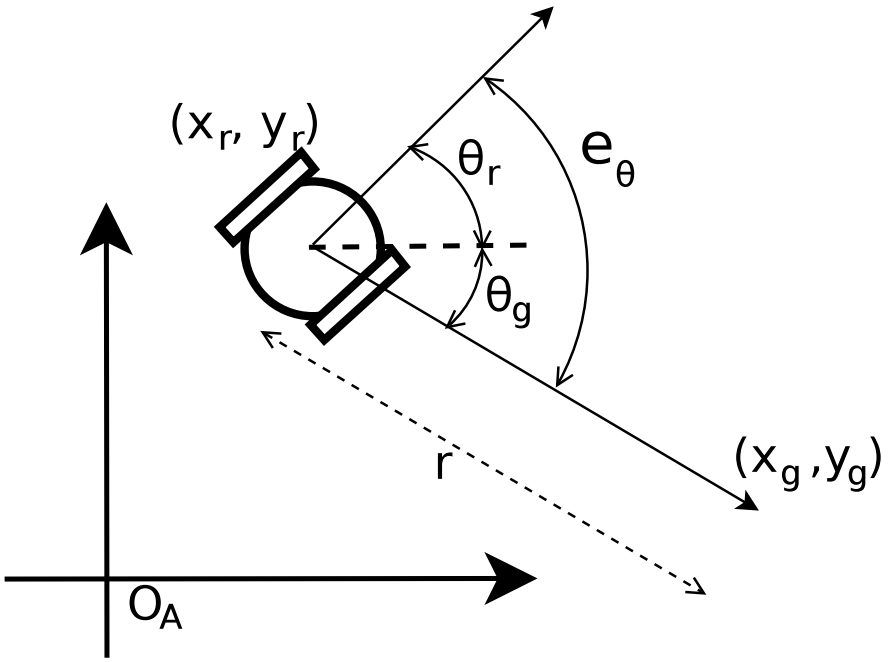

Figura 5.23: Posición deseada y pose del robot.

El error de ángulo siempre debe estar en el intervalo $(-\pi, \pi]$, si su valor es mayor que π o menor que $-\pi$ se debe calcular el ángulo equivalente que esté dentro del intervalo antes mencionado, ya que es importante dicho intervalo para el cálculo de las leyes de control. El error de ángulo se calcula de la siguiente manera:

$$e_\theta = \theta_g - \theta_r = \operatorname{atan2}(y_g - y_r, x_g - x_r) - \theta_r$$

Las leyes de control surgen a partir de las siguientes ecuaciones:

$$v = v_{max} e^{-\left(\frac{e_\theta^2}{\alpha}\right)} \tag{5.9}$$

$$\omega = \omega_{max} \left(\frac{2}{1 + e^{-\left(\frac{e_\theta}{\beta}\right)}} - 1 \right) \tag{5.10}$$

Estas ecuaciones determinan la velocidad lineal y angular del robot, y su objetivo es logar que primero el robot gire para orientarse con dirección hacia la posición deseada, y una vez que ha girado lo suficiente comience a moverse hacia el frente. v_{max} y ω_{max} son la velocidad lineal y angular máxima que el robot podrá alcanzar cuando se está moviendo, y las constantes α y β son parámetros que deben de sintonizarse.

Como se puede observar en la Figura 5.24, el valor de α determina que tan rápido aumenta la velocidad lineal cuando el error de ángulo decrece. Un valor

pequeño de α hará que el robot avance hasta que prácticamente esté orientado hacia la ubicación deseada. Observando la Figura 5.25, podemos notar que el valor de β determina que tanto aumenta la velocidad angular cuando el error de ángulo es crece, un valor pequeño de β hará que el robot gire lo suficiente para que se oriente hacia la ubicación deseada antes de que comience a avanzar hacia el frente.

Sustituyendo (5.9) y (5.10) en (5.3) y (5.4) se obtienen las leyes de control.

$$v_R = v_{max} e^{-\left(\frac{e_\theta^2}{\alpha}\right)} + \frac{L}{2}\omega_{max}\left(\frac{2}{1+e^{-\left(\frac{e_\theta}{\beta}\right)}} - 1\right) \tag{5.11}$$

$$v_L = v_{max} e^{-\left(\frac{e_\theta^2}{\alpha}\right)} - \frac{L}{2}\omega_{max}\left(\frac{2}{1+e^{-\left(\frac{e_\theta}{\beta}\right)}} - 1\right) \tag{5.12}$$

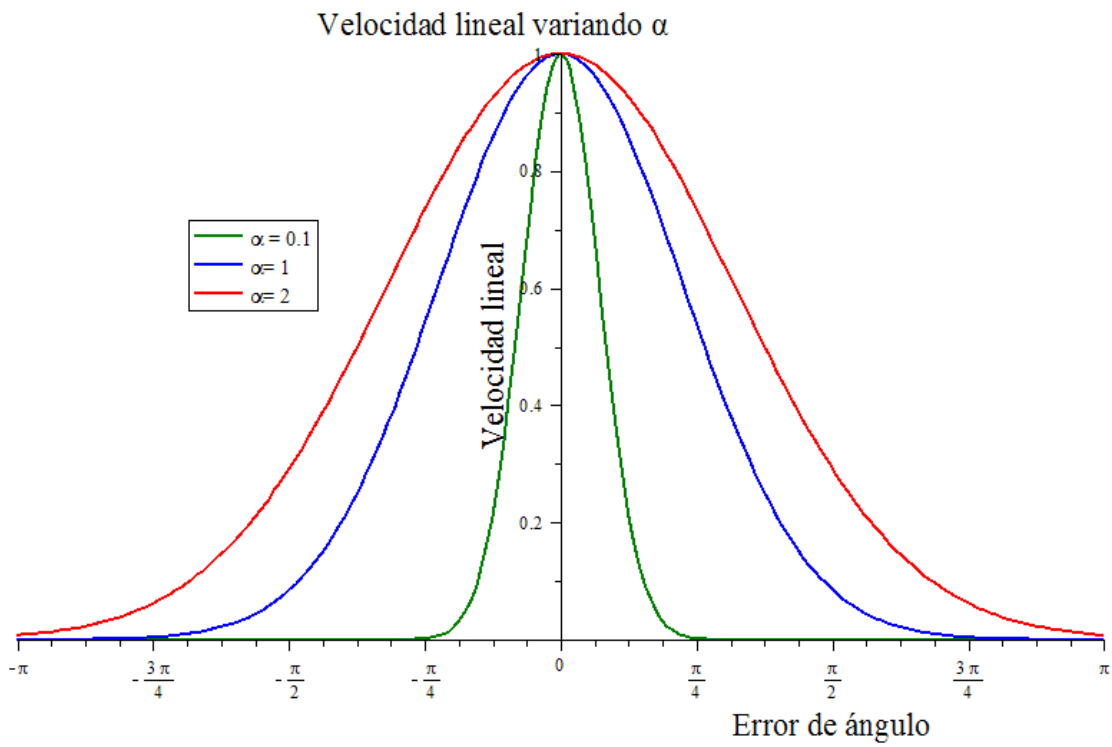

Figura 5.24: Velocidad lineal para distintos valores de α.

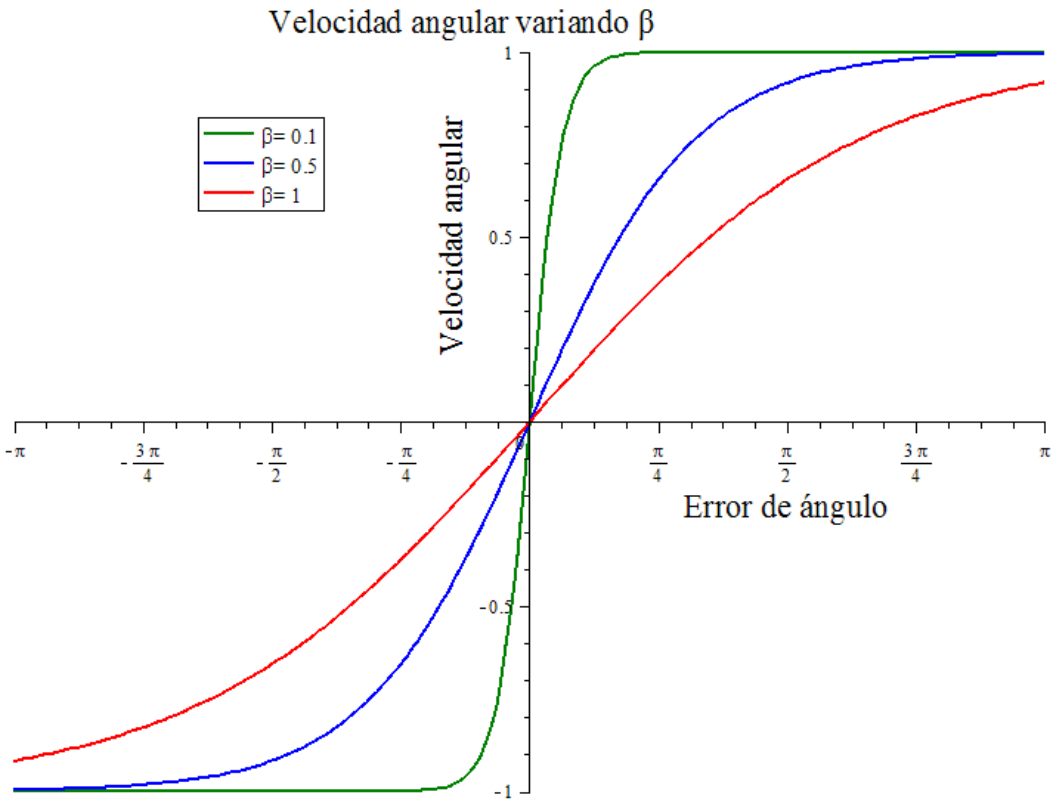

Figura 5.25: Velocidad angular para diferentes valores de β.

Las leyes de control dadas por las ecuaciones (5.11) y (5.12) son las utilizadas por el robot Justina, cabe mencionar que tienen el inconveniente de que dependen únicamente del error de ángulo, por lo que el robot no se detendrá ni disminuirá su velocidad conforme se acerca a la ubicación deseada y presentará oscilaciones alrededor de la vecindad cercana a dicha ubicación. Para realizar el seguimiento de una ruta, el robot móvil establece como ubicación deseada a un punto de la ruta que se encuentre a una distancia de 15cm con respecto de su ubicación actual y se detiene aproximadamente a una distancia de 5cm del último punto de la ruta que está siguiendo.

v_{max} no puede ser muy grande, puesto que otro problema que presentan las leyes de control es que el robot puede experimentar la velocidad lineal máxima desde el comienzo de su trayectoria, aunado al hecho de que dicha velocidad no disminuye cuando el robot se acerca al final de una ruta, lo cual repercute directamente en la transición entre velocidad lineal cero y velocidad lineal igual a v_{max} cuando el robot comienza a moverse y cuando se detiene.

En la Tabla 5.4 se muestran los valores de las constantes que se obtuvieron durante la sintonización de las leyes de control y en la Figura 5.26 se muestra el sistema de control implementado, en el que la variable q se refiere a la posición angular de las dos ruedas motrices del robot móvil.

Constantes	Valores
v_{max}	0.22
ω_{max}	1.5
α	0.2
β	0.2

Tabla 5.4: Constantes para las leyes de control.

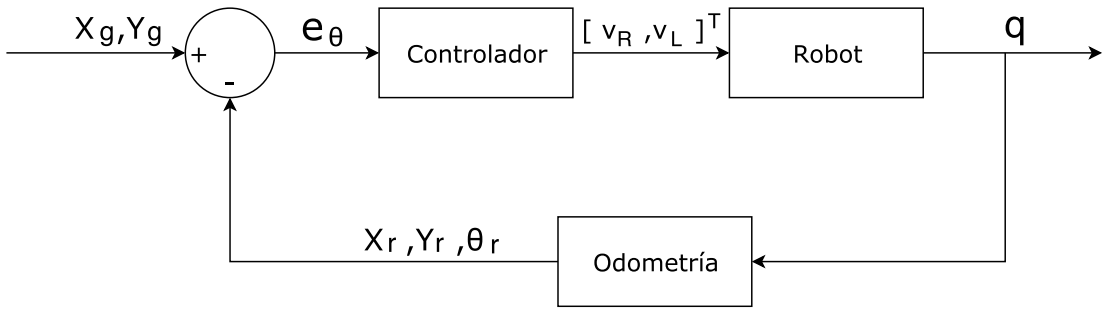

Figura 5.26: Diagrama de bloques del sistema de control.

5.3.3. Path calculator node

La planeación de rutas consiste en encontrar una ruta que conecte una posición inicial con una posición final, para ello, en este trabajo se asume que el robot se mueve en un plano, que la representación o modelo previo del entorno siempre está actualizado y que el ambiente es estático. El paquete `path_calculator` contiene el nodo `path_calculator_node`, este nodo se encarga de atender un servicio que recibe un mapa, una ubicación inicial y una ubicación final, como resultado entrega una ruta que une la ubicación inicial con la ubicación final a través del espacio libre. Este nodo primero utiliza un algoritmo de búsqueda en grafos llamado A* para encontrar una ruta, posteriormente suaviza la ruta encontrada.

En la Tabla 5.5 se muestra el tema que publica y el servicio que atiende el nodo `path_calculator_node`.

Tema publicado	`/navigation/a_star_path`	Ruta suavizada.
Servicio	`/navigation/a_star`	Servicio para el cálculo de una ruta.

Tabla 5.5: Path calculator node: Tema publicado y servicio.

5.3.3.1. Algoritmo de búsqueda A*

Problem solving intenta encadenar pasos para encontrar una solución a un problema. Teniendo formulado un problema y un objetivo definido, un algoritmo de búsqueda toma como entrada dicho problema y devuelve una solución en forma de una secuencia de acciones.

La búsqueda es un proceso para encontrar una secuencia de acciones que alcance un objetivo, los algoritmos de búsqueda trabajan considerando varias posibles secuencias de acciones. Los algoritmos de búsqueda no informados (también llamados de búsqueda ciega) sólo reciben la definición del problema, por otro lado, los algoritmos de búsqueda informados (también llamados de búsqueda heurística) además de recibir la definición del problema cuentan con una guía para saber en dónde buscar soluciones.

Considerando que se tiene una representación del entorno que consiste en un mapa de celdas de ocupación, el problema de planeación de rutas se puede definir formalmente de la siguiente manera:

- **Estados**: las celdas de ocupación dentro del mapa que pertenecen al espacio libre se consideran como nodos en un grafo, por lo tanto, un estado es estar en cualquier nodo que pertenezca al espacio libre.

- **Estado inicial**: es la posición inicial asociada a un nodo que pertenece al espacio libre dentro del mapa.

- **Descripción de las posibles acciones**: dado un estado en particular se tienen un conjunto de acciones que se pueden ejecutar en dicho estado. Considerando la Figura 5.27, supóngase que el estado actual es estar en la celda o nodo A, utilizando conectividad cuatro las posibles acciones son ir al nodo B, ir al nodo C, ir al nodo D, ir al nodo E; cabe recalcar que solamente se consideran aquellas celdas vecinas que pertenecen al espacio libre.

- **Modelo de transición**: es la descripción de lo que hace cada acción, es decir, el estado que resulta de ejecutar una acción. Para este caso la consecuencia de estar en una celda A e ir a una celda B, resulta en estar en la celda B.

- **La prueba de objetivo**: determina si un estado en particular es el estado objetivo. Para este caso, el estado objetivo es una celda de ocupación en particular asociada a una posición final.

- **Función de costo de una ruta**: es la función que asigna un costo numérico a una ruta, esta función refleja una medida de rendimiento. Tradicionalmente se denota como $g(n)$, que es el costo de la ruta desde el estado inicial al nodo n. Para este caso el costo para cada nodo n, es el número de celdas recorridas para llegar a dicho nodo desde el estado inicial.

El espacio de estados del problema es el conjunto de todos los estados alcanzables por cualquier secuencia de acciones partiendo desde el estado inicial. El espacio de estados forma una red o grafo dirigido en el que los nodos son estados y los enlaces entre los nodos son acciones. Una ruta en el espacio de estados es una secuencia de estados conectados por una secuencia de acciones.

Se dice que un algoritmo de búsqueda es completo si garantiza encontrar una solución cuando la hay, por otro lado, se dice que es óptimo si la solución que encuentra tiene el costo de ruta más bajo entre todas las soluciones posibles.

Expandir el estado actual significa generar un nuevo conjunto de estados aplicando cada acción legal al estado actual, dicho de otra forma, un nodo padre genera nuevos nodos hijo disponibles para la expansión. El conjunto de todos los nodos disponibles para la expansión se llama frontera, aunque muchos autores llaman a este conjunto lista abierta. El proceso de expansión de nodos en la lista abierta continúa hasta que se encuentra una solución o no hay más nodos disponibles para expandir, todos los algoritmos de búsqueda comparten esta estructura básica, con la diferencia de que utilizan una estrategia de búsqueda distinta, es decir, la forma en que eligen el estado a expandir. Para evitar explorar rutas redundantes es necesario recordar los nodos que han sido expandidos, para ello se utiliza un conjunto llamado lista cerrada o conjunto explorado, el cual está conformado por todos los nodos expandidos.

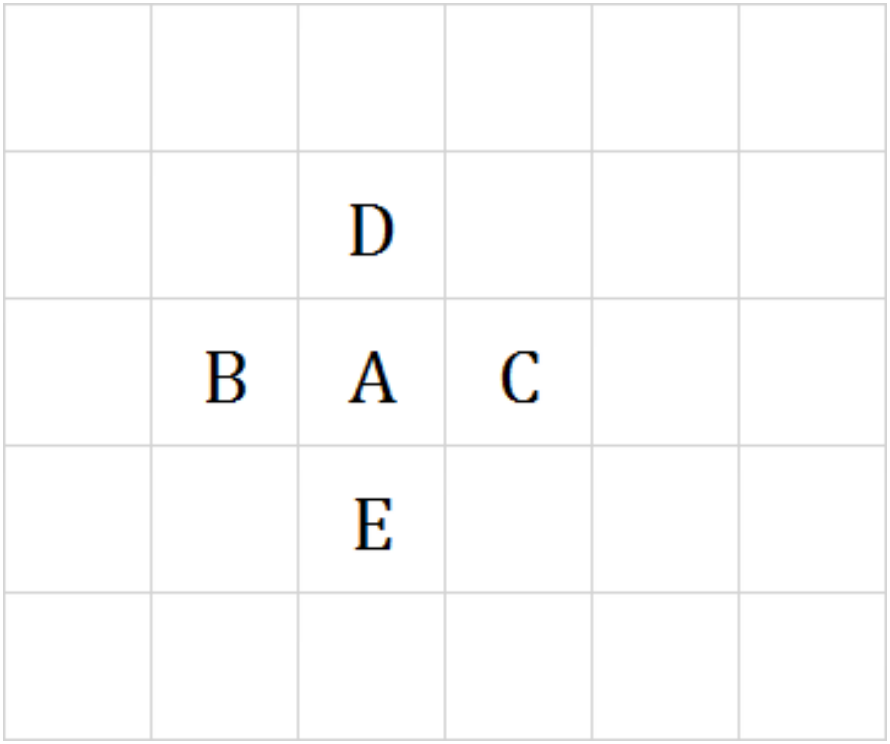

Figura 5.27: Expansión de nodos.

La búsqueda primero el mejor (best-first search) es una estrategia de búsqueda informada en la que se selecciona un nodo para la expansión con base en una función de evaluación, denotada como $f(n)$. Esta función se interpreta como una estimación de costo, así que el nodo con la evaluación más baja se expande primero. La mayoría de los algoritmos primero el mejor incluyen como componente de $f(n)$ una función heurística, denotada como $h(n)$. Las funciones heurísticas son una forma de proveer conocimiento adicional del problema a un algoritmo de búsqueda.

La búsqueda A* (pronunciada como búsqueda A estrella) es la forma más conocida del tipo de búsqueda primero el mejor. Evalúa los nodos utilizando $g(n)$, el costo para llegar al nodo, y la función heurística $h(n)$, el costo estimado de la ruta más barata desde el nodo n hasta un estado objetivo.

$$f(n) = g(n) + h(n)$$

Por lo tanto, $f(n)$ se interpreta como el costo estimado de la solución más barata a través de n. La búsqueda A* es completa y una condición para que sea óptima es que $h(n)$ sea una heurística admisible, una heurística admisible es aquella que nunca sobrestima el costo de alcanzar la meta. Si $h(n)$ es una heurística admisible y dado que $g(n)$ es el costo real para alcanzar n por medio de la ruta actual, entonces como consecuencia se tiene que $f(n) = g(n) + h(n)$ nunca sobrestima el costo real de una solución a lo largo de la ruta actual a través de n. Por lo tanto, para encontrar la solución más barata, una cosa razonable es expandir primero el nodo con el valor más bajo de $f(n)$. La prueba de objetivo se aplica a un nodo cuando se selecciona para la expansión, la razón es que la primera vez que se genera el nodo objetivo puede estar en una ruta subóptima, además de que cuando se selecciona un nodo n para la expansión es porque se ha encontrado la ruta óptima a ese nodo.

Una segunda condición para que la búsqueda A* sea óptima es la consistencia, se dice que una heurística $h(n)$ es consistente si para cada nodo n y para cada sucesor n' de n generado por cualquier acción a, el costo estimado de alcanzar el objetivo desde n no es mayor que el costo de llegar a n' desde n más el costo estimado de alcanzar el objetivo desde n':

$$h(n) \leq c\,(n, a, n') + h(n')$$

Dicho de otra forma, si $h(n)$ es consistente, los valores de $f(n)$ a lo largo de cualquier ruta no decrecen. Supóngase que n' es un sucesor de n, entonces $g(n') = g(n) + c\,(n, a, n')$ para alguna acción a, y se tiene:

$$f(n') = g(n') + h(n') = g(n) + c\,(n, a, n') + h(n') \geq g(n) + h(n) = f(n)$$

El algoritmo 1 [53] muestra en pseudocódigo la implementación de la búsqueda A* para la planeación de rutas, el valor de $g(n)$ se calcula sumando el costo para

[53] Fuente: `https://github.com/mnegretev/RoboticsCourses/blob/master/Exercises/RMYAI/Prac05/Prac05.pdf`

alcanzar un nodo a partir de otro $c(n_1, n_2)$, este costo se podría calcular utilizando la distancia euclidiana, no obstante, dado que se trabaja con celdas de ocupación y se utiliza conectividad cuatro para la expansión, el costo para alcanzar un nodo se calcula como un entero que incrementa en uno su valor con respecto del valor de $g(n)$ del nodo que lo genera. La heurística se puede calcular como la distancia euclidiana entre celdas, pero como se está utilizando conectividad cuatro, el costo computacional se reduce si en su lugar se utiliza la distancia de Manhattan, la cual es admisible y consistente. La distancia de Manhattan se calcula como la suma de las distancias horizontal y vertical; considérese la Figura 5.28, para el caso mostrado la distancia de Manhattan entre las celdas A y B es $h(n_A, n_B) = 7$, ya que hay tres celdas de distancia vertical y cuatro celdas de distancia horizontal.

Un problema que se tiene es que las celdas libres que están junto al espacio ocupado no se pueden considerar como parte del espacio navegable, ya que si el robot se mueve hacia ellas entonces colisionará con obstáculos. Una solución a esto es crecer los obstáculos, de forma que las celdas libres que se encuentran en una vecindad cercana al espacio ocupado se consideren también como ocupadas. En la Figura 5.29 se muestra un ejemplo de crecimiento de obstáculos, para hacer crecer los obstáculos la vecindad que se considera es con conectividad ocho, es decir, si una celda libre se encuentra en la vecindad de una celda ocupada entonces dicha celda libre se considera como parte del espacio ocupado.

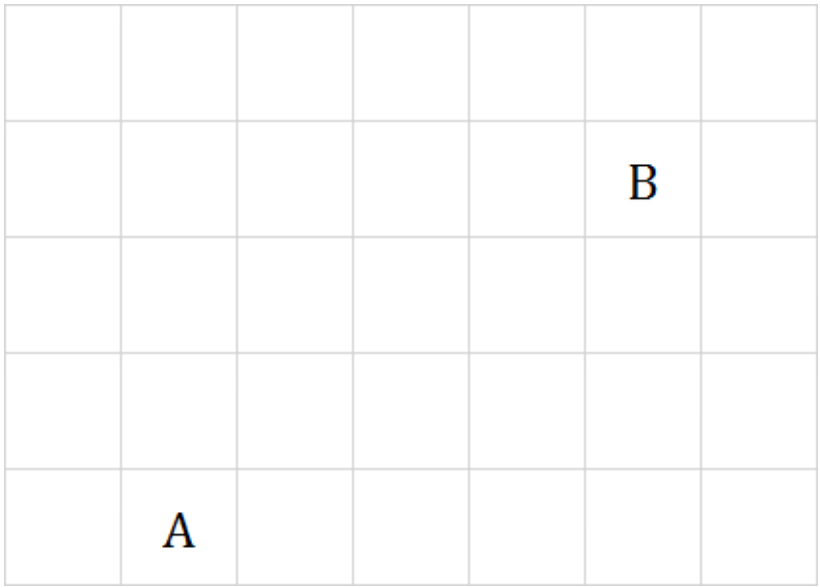

Figura 5.28: Distancia de Manhattan.

Algoritmo 1: Búsqueda con A*

```
Datos: Grafo, nodo inicial, nodo meta
Resultado: Ruta óptima expresada como una secuencia de nodos
Cerrado ← ∅
Abierto ← {nodo_inicial}
previo(nodo_inicial) ← ∅
mientras Abierto ≠ ∅ hacer
    nodo_actual ← nodo con el menor valor f del conjunto Abierto
    Abierto ← Abierto - {nodo_actual}
    Cerrado ← Cerrado ∪ {nodo_actual}
    si nodo_actual es nodo_meta entonces
        Anunciar éxito y salir de este ciclo
    fin
    para cada nodo_vecino de nodo_actual hacer
        si nodo_vecino ∈ Cerrado entonces
            Continuar con el siguiente nodo_vecino
        fin
        si nodo_vecino ∈ Abierto entonces
            costo_temporal ← g(nodo_actual) + c(nodo_actual, nodo_vecino)
            si costo_temporal < g(nodo_vecino) entonces
                g(nodo_vecino) ← costo_temporal
                f(nodo_vecino) ← costo_temporal + heurística(nodo_vecino,
                nodo_meta)
                previo(nodo_vecino) ← nodo_actual
            fin
        en otro caso
            g(nodo_vecino) ← g(nodo_actual) + c(nodo_actual, nodo_vecino)
            f(nodo_vecino) ← g(nodo_vecino) + heurística(nodo_vecino,
            nodo_meta)
            previo(nodo_vecino) ← nodo_actual
            Abierto ← Abierto ∪ {nodo_vecino}
        fin
    fin
fin
si nodo_actual ≠ nodo_meta entonces
    Anunciar falla
en otro caso
    RutaOptima ← ∅
    mientras nodo_actual ≠ ∅ hacer
        //El nodo actual se inserta al principio de la ruta
        RutaÓptima ← {nodo_actual} ∪ RutaÓptima
        nodo_actual ← previo(nodo_actual)
    fin
    Regresar RutaÓptima
fin
```

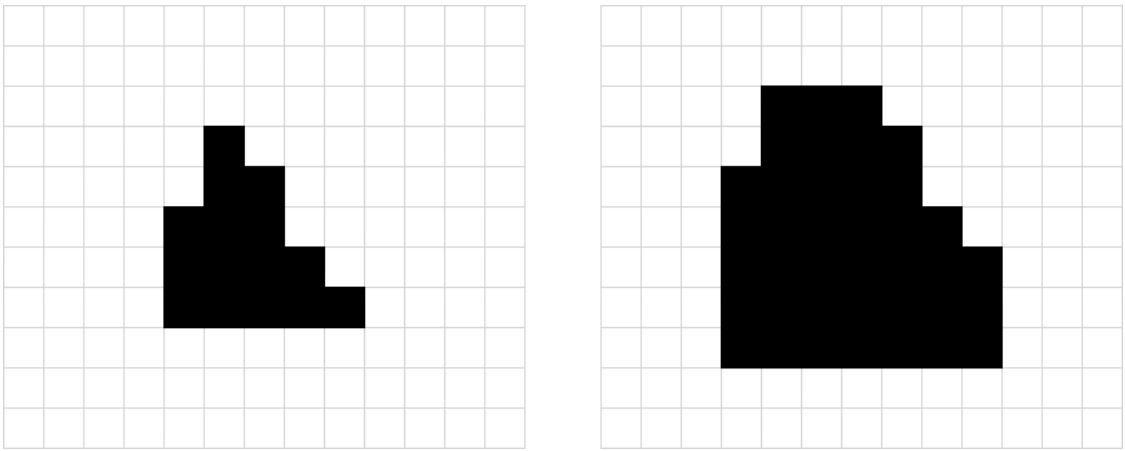

Figura 5.29: Crecimiento de obstáculos.

5.3.3.2. Suavizado de la ruta

Utilizar conectividad cuatro en el algoritmo A* tiene como consecuencia que la ruta encontrada tendrá cambios de dirección con ángulos rectos, lo cual no es deseable puesto que el robot no puede dar una vuelta de 90° de forma instantánea. Dicho lo anterior, se entiende que es conveniente suavizar la ruta encontrada por el algoritmo A*, de tal modo que se obtenga una nueva ruta parecida a la original. En la Figura 5.30 se muestra un ejemplo de suavizado de una ruta, la ruta azul es la ruta original, la ruta roja es una ruta sin cambios de dirección, pero no es de utilidad, y la ruta en color verde es una ruta suavizada.

En este trabajo, para poder obtener una ruta como la de color verde en la Figura 5.30, se utiliza el algoritmo de suavizado que usa el robot Justina del laboratorio de Bio Robótica. La ruta suavizada se obtiene al modificar la ruta original con base en una función de costo que minimice dos criterios: la distancia entre puntos consecutivos en la ruta suavizada y, la distancia del i-ésimo punto de la ruta original con el i-ésimo punto de la ruta suavizada, es decir, este último criterio toma en cuenta el parecido entre la ruta suavizada y la ruta original. La ruta se suaviza al disminuir la distancia entre puntos consecutivos, por lo que es claro que ambos criterios entran en conflicto: mientras más se suavice la ruta implica que menor será el parecido entre la ruta original y la ruta suavizada y viceversa. Por ende, la función de costo será grande si se suaviza demasiado la ruta y también será grande si la ruta obtenida es muy parecida a la original; así que lo que se hace es asignar un cierto peso a cada criterio para optimizar uno de ellos más que el otro.

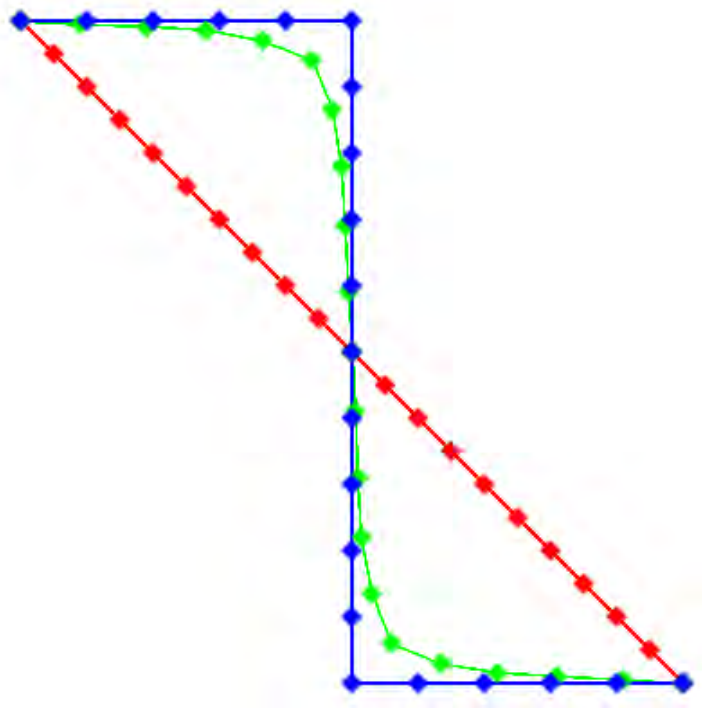

Figura 5.30: Ejemplo de suavizado de una ruta.

Para el suavizado se utiliza la siguiente función de costo:

$$V = \frac{1}{2}\alpha \sum_{i=1}^{n} (p_i - q_i)^2 + \frac{1}{2}\beta \sum_{i=1}^{n-1} (p_i - p_{i+1})^2 \tag{5.13}$$

En donde $Q = \{q_1 \dots q_n\}$ es el conjunto de todos los puntos de la ruta original y $P = \{p_1 \dots p_n\}$ es el conjunto de todos los puntos de la ruta suavizada. La constante α es el peso que se le da al parecido entre la ruta original y la ruta suavizada, y la constante β es el peso que se le da al suavizado, cabe mencionar que ambas constantes deben ser positivas. Para el suavizado no se consideran el primer y el último punto de la ruta, la optimización se aplica a todos los puntos intermedios. Tómese en cuenta la Figura 5.30, si α es igual a cero se obtiene la ruta roja, y si β es igual a cero se obtiene la ruta original (ruta azul).

Utilizando el descenso del gradiente se puede encontrar el argumento que minimiza la función de costo mostrada en la Ecuación 5.13, este método consiste en mover el argumento en pequeñas cantidades proporcionales y en sentido contrario al gradiente de la función V. Se puede asumir que cuando el cambio total observado en el argumento es menor que una tolerancia, entonces se ha alcanzado el mínimo y el algoritmo ha convergido.

Para suavizar la ruta se aplica iterativamente el descenso del gradiente a todos los puntos de la ruta, exceptuando el primero y el último. Entonces, la ecuación que se aplica a cada uno de los puntos mencionados es la siguiente:

$$p_i = p_i - \delta \nabla V(p_i)$$

En donde δ es una constante positiva que debe ser lo suficientemente pequeña para evitar oscilaciones al tratar de minimizar la función de costo, pero además se debe considerar que si es muy pequeña tendrá como consecuencia el aumento del costo computacional. El gradiente de la función de costo se muestra a continuación:

$$\begin{aligned}
\nabla V &= \left[\frac{\partial V}{\partial p_1}, \ldots, \frac{\partial V}{\partial p_i}, \cdots, \frac{\partial V}{\partial p_n}\right] \\
\frac{\partial V}{\partial p_1} &= \alpha(p_1 - q_1) + \beta(p_1 - p_2) \\
\frac{\partial V}{\partial p_i} &= \alpha(p_i - q_i) + \beta(2p_i - p_{i-1} - p_{i+1}) \\
\frac{\partial V}{\partial p_n} &= \alpha(p_n - q_n) + \beta(p_n - p_{n-1})
\end{aligned}$$

5.3.4. Planner node

El nodo `planner_node` se encuentra dentro del paquete `planner`, este nodo se encarga de la toma de decisiones del robot móvil. Básicamente es un nodo cliente que le solicita por medio de un servicio al nodo `path_calculator_node` el cálculo de una ruta de un punto a otro, y una vez que la ruta se ha obtenido con éxito el robot comenzará a seguir dicha ruta.

En este nodo se establece la ubicación cartesiana de cada uno de los dispositivos de SmartThings y lugares predeterminados dentro de un mapa estático. Cada vez que se le solicita al robot navegar hasta una cierta ubicación, se obtiene la pose del robot con respecto del marco de referencia global y también el mapa estático del lugar en el que está operando el robot, estos datos son importantes para solicitar el cálculo de una ruta.

Se tiene como prioridad atender las notificaciones que se reciben de los dispositivos de SmartThings, si se reciben notificaciones de tales dispositivos cuando el robot se encuentra realizando una tarea, primero terminará la tarea que está en proceso y posteriormente atenderá todas las notificaciones según el orden en que llegaron. Si el robot se encuentra ocupado hará caso omiso a la orden de navegar hasta alguno de los lugares predeterminados.

En la Tabla 5.6 se muestran los temas y servicios para este nodo.

Temas publicados	/navigation/start_path	Tema utilizado para indicar que el robot debe comenzar a moverse.
	/hardware/robot_state /availability	Tema usado para señalar la disponibilidad del robot.
Temas suscritos	/navigation/smart_things /alerts_devices	Tema para recibir el nombre del dispositivo de SmarThings que detectó un suceso inesperado.
	/navigation/goal_location	Tema para recibir el nombre del lugar al que el usuario desea enviar al robot.
	/navigation/goal_reached	Tema para saber cuándo el robot ha llegado a la ubicación deseada.
	/tf	Obtiene la transformación de map a base_link.
Servicios	/navigation/a_star	Servicio para solicitar el cálculo de una ruta.
	/navigation/static_map	Servicio para obtener el mapa estático.

Tabla 5.6: Planner node: Servicios, temas publicados y suscritos.

5.3.5. Joystick teleop node

El nodo joystick_teleop_node.py se encuentra dentro del paquete joystick_teleop. Este nodo se subscribe a un tema de tipo Joy.msg para mover mediante un joystick el robot móvil. Para controlar la velocidad angular se utiliza el stick izquierdo (únicamente el eje horizontal) y para controlar la velocidad lineal se usa el stick derecho (únicamente el eje vertical). Este nodo es útil cuando se requiere la creación de un mapa. En la Tabla 5.7 se muestran los temas para este nodo.

Tema publicado	/hardware/mobile_base /speeds	Velocidades deseadas para las ruedas del robot.
Tema suscrito	/hardware/joy	Estado de los ejes y botones de un joystick.

Tabla 5.7: Joystick teleop node: Tema publicado y suscrito.

5.3.6. Joy node

El nodo joy_node se ubica en el paquete joy y es un nodo aportado por la comunidad de ROS. Se utiliza para conectar un joystick a ROS y a través de un mensaje de tipo Joy.msg publica el estado actual de cada uno de los botones y ejes de un joystick. En la Tabla 5.8 se muestran el tema y el parámetro para este nodo.

Tema publicado	/hardware/joy	Tema para reportar el estado de los ejes y botones de un joystick.
Parámetro	dev	Puerto asociado al joystick desde el que se leen los eventos.

Tabla 5.8: Joy node: Parámetro y tema publicado.

5.3.7. Hokuyo node

El nodo hokuyo_node se encuentra en el paquete hokuyo_node, este nodo es un aporte de la comunidad de ROS. Se encarga de la adquisición de datos de un sensor Hokuyo láser y los publica en un mensaje de tipo LaserScan.msg. En la Tabla 5.9 se muestran el tema y los parámetros para este nodo.

Tema publicado	`/hardware/scan`	Datos obtenidos en un escaneo.
Parámetros	`port`	Puerto en donde se encuentra el sensor Hokuyo láser.
	`frame_id`	Nombre del marco de referencia asociado al sensor láser.

Tabla 5.9: Hokuyo node: Parámetros y tema publicado.

5.3.8. Map server

`map_server` es un paquete aportado por la comunidad de ROS, y proporciona un nodo llamado `map_server`, el cual carga un mapa y ofrece dicho mapa en forma de datos por medio de un servicio. También proporciona una utilidad de línea de comandos (`map_saver`) que permite guardar los mapas generados dinámicamente por medio de SLAM.

Los mapas se almacenan en dos archivos: un archivo YAML con los metadatos del mapa y una imagen que describe el estado de ocupación de cada celda del mundo con un color para cada píxel según corresponda. Cabe mencionar que el mundo real se representa por medio de una aproximación discreta, conocida como occupancy grid. Los píxeles más blancos son espacio libre, los píxeles más oscuros son espacio ocupado y los píxeles entre estos colores son desconocidos, es decir, no se tiene información sobre su respectiva celda.

Cuando un mapa es representado por medio de un mensaje de ROS, la ocupación de cada celda se representa con un número entero entre 0 y 100, 0 significa que el espacio está completamente libre y 100 significa que el espacio está completamente ocupado, el valor -1 indica que la ocupación es completamente desconocida.

En la Tabla 5.10 se muestran los temas publicados y el servicio que atiende este nodo.

Temas publicados	/navigation/map_metadata /navigation/map	Metadatos del mapa. Mapa.
Servicio	/navigation/static_map	Obtención del mapa cargado a través de este servicio.

Tabla 5.10: Map server: Servicio y temas publicados.

5.3.9. Gmapping

gmapping es un paquete aportado por la comunidad de ROS y tiene un nodo llamado slam_gmapping, el cual puede crear un mapa 2D de tipo occupancy grid a partir de los datos obtenidos por un sensor láser y la pose del robot.

Para usar el nodo slam_gmapping, además de una fuente que calcule la odometría y un sensor láser montado en el robot de forma horizontal y fija, se requiere la transformación de odom a base_link y la trasformación de base_link a el marco de referencia asociado al sensor láser. Para corregir el error acumulado por la localización odométrica este nodo provee la transformación de map a odom, es decir, la estimación actual de la pose del robot con respecto del marco de referencia map.

En la Tabla 5.11 se muestran los temas publicados, los temas suscritos y algunos de los parámetros para el nodo slam_gmapping.

Temas publicados	/map_metadata /map	Metadatos del mapa. Mapa.
Temas suscritos	/tf /hardware/scan	Transformación entre los marcos de referencia laser_link, base_link y odom. Escaneos del sensor láser para la creación del mapa.
Parámetros	map_update_interval maxUrange linearUpdate angularUpdate xmin ymin xmax ymax maxRange	Tiempo en segundos que tardará en actualizarse el mapa. Rango máximo utilizable en el láser. Cada que el robot avance esta distancia se procesará un escaneo. Cada que el robot gire este ángulo se procesará un escaneo. Menor valor de x en el mapa. Menor valor de y en el mapa. Mayor valor de x en el mapa. Mayor valor de y en el mapa. Rango máximo del sensor láser.

Tabla 5.11: Gmapping: Temas publicados, temas suscritos y parámetros.

5.3.10. Robot state publisher

El paquete robot_state_publisher es aportado por la comunidad de ROS y por medio del nodo state_publisher permite publicar el estado de un robot a tf.

Una vez que el nodo inicia su ejecución, dicho estado está disponible para todos los componentes del sistema que usan `tf`. El nodo toma los ángulos o desplazamientos de las articulaciones del robot como entrada y publica la pose 3D de los eslabones del robot, utilizando un modelo de árbol cinemático. `tf` es un paquete que permite realizar el seguimiento de varios marcos de referencia a lo largo del tiempo.

El nodo antes mencionado utiliza el URDF especificado por el parámetro `robot_description` y el tema `joint_states` para calcular la cinemática directa del robot y publicar los resultados a `tf`. URDF (Unified Robot Description Format) es un formato XML para representar el modelo de un robot.

En la Tabla 5.12 se muestran los temas y parámetros para el nodo `state_publiser`.

Tema publicado	`/tf`	Publica el estado del robot.
Tema suscrito	`/joint_states`	Se suscribe a la información de la posición de las articulaciones.
Parámetros	`robot_description`	Archivo XML del modelo del robot.
	`publish_frequency`	Frecuencia a la que publica el nodo.

Tabla 5.12: Robot state publisher:Temas y parámetros.

5.3.11. Robot Web Tools

Robot Web Tools permite que los navegadores web puedan interactuar con ROS, rosbridge es una capa que permite conectar dispositivos y aplicaciones externas, es decir, es una capa de transporte que proporciona lo necesario para que los clientes se comuniquen. [40]

En la capa del navegador se tienen las bibliotecas de JavaScript: roslibjs, ros2djs y ros3djs. Estas bibliotecas se comunican con ROS por medio del servidor WebSocket de rosbridge (WebSocket es una tecnología que hace posible la comunicación entre el navegador del usuario y un servidor). Las bibliotecas proporcionan una abstracción de la funcionalidad principal de ROS. [40]

Rviz es una herramienta de visualización 3D, permite visualizar marcadores, mapas, información de sensores, entre otros elementos que son de utilidad al momento de desarrollar y probar código para un robot móvil. Con la biblioteca ros3djs se puede crear una herramienta de visualización similar a RViz, pero en una página web. El objetivo de crear una interfaz web con una funcionalidad parecida a Rviz es poder visualizar el estado del robot móvil en cualquier dispositivo que tenga un navegador web sin la necesidad de que dicho dispositivo cuente con ROS. En la Figura 5.31 se muestra la página web creada para el presente trabajo, la cual está alojada en un servidor que permite visualizarla únicamente dentro una red LAN.

Rosbridge proporciona la funcionalidad de ROS a los navegadores web. El protocolo de rosbridge envía comandos a ROS y cubre la publicación y subscripción a temas, llamadas a servicios, obtención y configuración de parámetros, entre otras cosas. El paquete `rosbridge_suite` es una colección de paquetes, que en conjunto permiten enviar y recibir información de ROS.

El paquete `tf2_web_republisher` es desarrollado como parte de las herramientas de Robot Web Tools, y permite calcular las transformaciones de `tf` para enviarlas a través de `rosbridge_suite` a un cliente web.

5.3.12. Roswww

El paquete `roswww` permite tener un servidor web para ROS, únicamente basta con guardar las páginas web en el directorio pertinente y para acceder a ellas basta con ingresar la siguiente dirección en un navegador web: `http://host:port/roswww/page_name.html`.

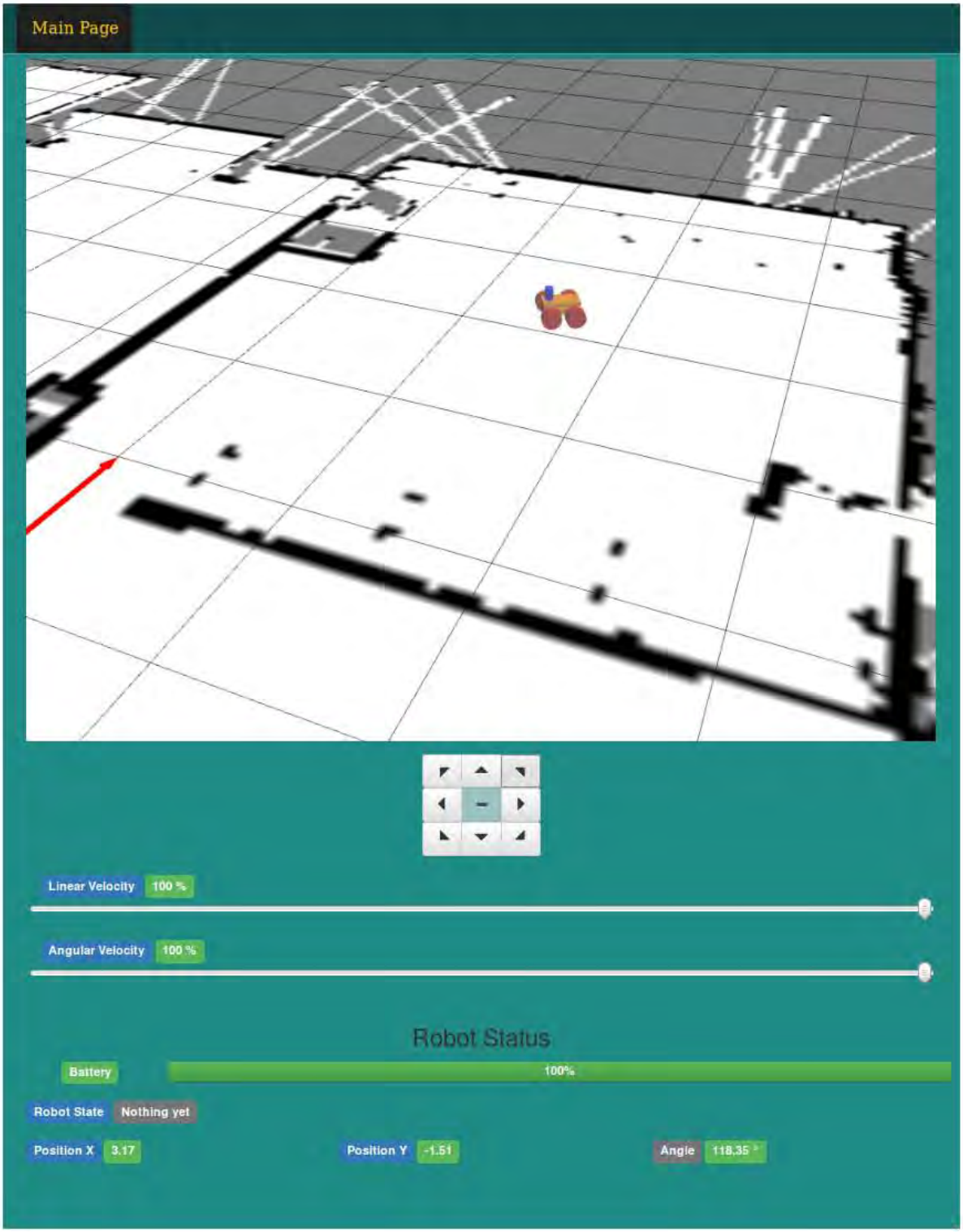

Figura 5.31: Página web creada.

Capítulo 6

Pruebas experimentales y resultados

6.1. SmartThings: envió de comandos

El Hub de SmartThings es el encargado de la comunicación entre todos los dispositivos y la nube, para garantizar el correcto funcionamiento es necesario que el Hub siempre esté conectado a Internet. La información que llega a la ThingShield es procesada por el microcontrolador de la tarjeta Arduino UNO, el nodo `smart_things_node.py` utiliza el protocolo de comunicación serial para el intercambio de datos entre la Raspberry Pi 3 y la tarjeta Arduino UNO.

Para hacer el intercambio de información, el *Device Handler* creado recibe y envía cadenas de caracteres a la Raspberry Pi, utilizando como intermediarios la tarjeta Arduino UNO y la Thingshield, estas cadenas corresponden a los comandos mostrados en la Sección 4.3. Para probar el envío y la recepción de las cadenas de caracteres se realizó una serie de pruebas, en cada prueba se enviaron 100 cadenas de caracteres de forma consecutiva desde el *Device Handler* hacia la ThingShield y viceversa, corroborando que los datos llegaran correctamente.

Las cadenas de caracteres se enviaron aproximadamente a intervalos de tiempo de entre uno y dos segundos. Para enviar las cadenas del *Device Handler* hacia la Raspberry Pi se utilizó la App de SmartThings para dispositivos móviles, para el envío de cadenas de la Raspberry Pi hacia el *Device Handler* se usó un programa hecho en Python. Los resultados obtenidos se muestran en las Tablas 6.1 y 6.2, se consideró como fracaso el hecho de que las cadenas de caracteres llegaran incorrectamente, llegaran repetidas o que simplemente no llegaran.

Número de prueba	Éxitos	Fracasos	Porcentaje de éxito
1	90	10	90 %
2	96	4	96 %
3	91	9	91 %
4	90	10	90 %
5	94	6	94 %
6	97	3	97 %
7	95	5	95 %
8	92	8	92 %
9	97	3	97 %
10	94	6	94 %

Tabla 6.1: Envío de cadenas de caracteres del *Device Handler* hacia la ThingShield.

Número de prueba	Éxitos	Fracasos	Porcentaje de éxito
1	89	11	89 %
2	94	6	94 %
3	94	6	94 %
4	97	3	97 %
5	98	2	98 %
6	98	2	98 %
7	95	5	95 %
8	89	11	89 %
9	94	6	94 %
10	97	3	97 %

Tabla 6.2: Envío de cadenas de caracteres de la ThingShield hacia el *Device Handler*.

6.2. Controladores PID para los motores Dynamixel

Para determinar los valores de las constantes P, I, D se recurrió a una sintonización experimental, para ello se definieron como objetivos conseguir que la curva de respuesta tenga poco sobrepaso, minimizar las oscilaciones y lograr el menor tiempo de asentamiento posible. Sintonizando correctamente un controlador PD se pueden conseguir los objetivos deseados, ya que estas dos acciones en conjunto decrecen el tiempo de levantamiento, el sobrepaso, el tiempo de asentamiento y el error en estado permanente.

Durante el proceso de sintonización se utilizó una entrada escalón para observar el desempeño de los controladores. Lo primero que se hizo fue modificar el valor de la ganancia de un controlador P hasta conseguir un balance entre tiempo de levantamiento y sobrepaso, es decir, lograr el menor tiempo de levantamiento posible de tal modo que el sobrepaso no fuera muy grande. Una vez hecho lo

anterior se añadió la ganancia derivativa para disminuir el sobrepaso y el tiempo de asentamiento.

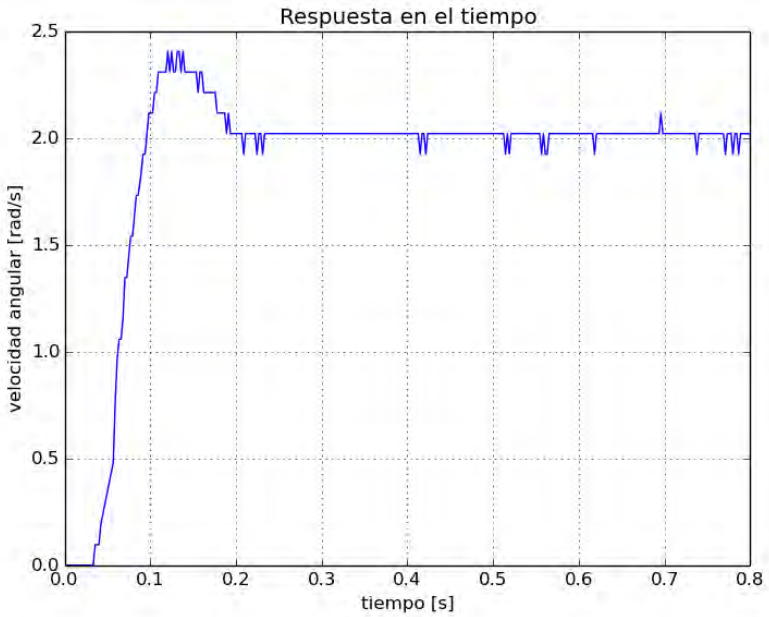

Figura 6.1: Motor derecho. Salida del sistema de control para una salida deseada de $2\left[\frac{rad}{s}\right]$.

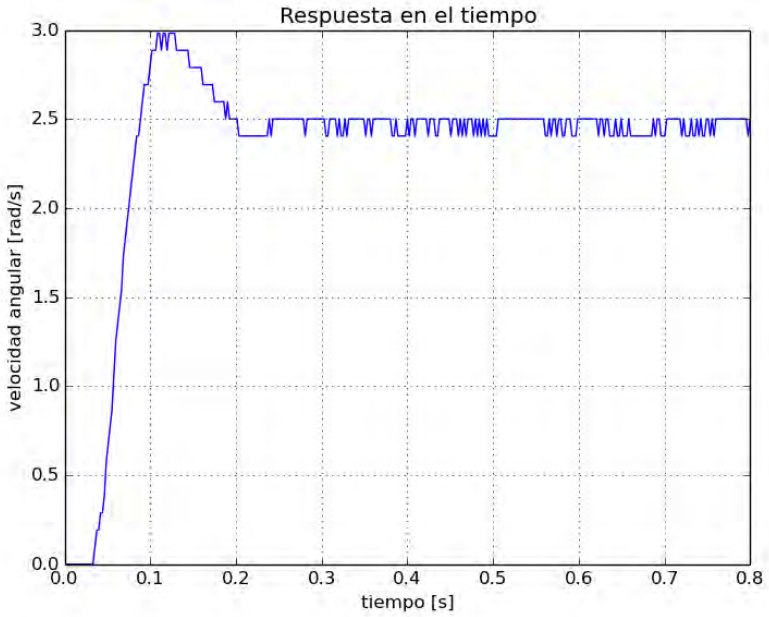

Figura 6.2: Motor derecho. Salida del sistema de control para una salida deseada de $2.5\left[\frac{rad}{s}\right]$.

Como ya se mencionó, con un controlador PD se pueden lograr los objetivos deseados, sin embargo, un inconveniente que se presentó en el motor izquierdo fue que tiene mayor dificultad para moverse en el arranque, por ello en este motor en particular se decidió implementar un controlador PID, ya que la acción integral incrementa la magnitud de la variable de control con el paso del tiempo debido al error acumulado. Así que una vez que se sintonizó el controlador PD para el motor izquierdo, se procedió a agregar el término integral y la ganancia se modificó hasta lograr que el motor se moviera, procurando minimizar las oscilaciones y evitando incrementar el tiempo de asentamiento. De igual manera, durante la sintonización de la ganancia integral se consideró el hecho de que si es grande tiene como resultado un aumento en el sobrepaso, lo cual provoca cambios bruscos en la velocidad que propician el deslizamiento entre la rueda correspondiente y el suelo.

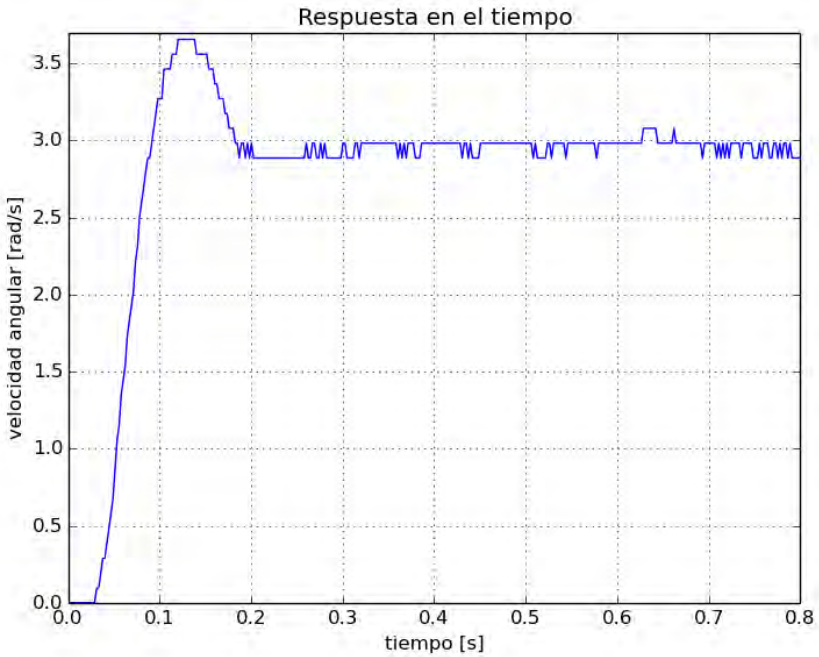

Figura 6.3: Motor derecho. Salida del sistema de control para una salida deseada de 3 $\left[\frac{rad}{s}\right]$.

$\omega_{deseada}\left[\dfrac{rad}{s}\right]$	$t_r\,[s]$	$\%S_p$	$t_s\,[s]$
2.0	0.062	20.62	0.156
2.5	0.056	16.66	0.156
3.0	0.062	21.66	0.153

Tabla 6.3: Motor derecho. Características de la respuesta transitoria.

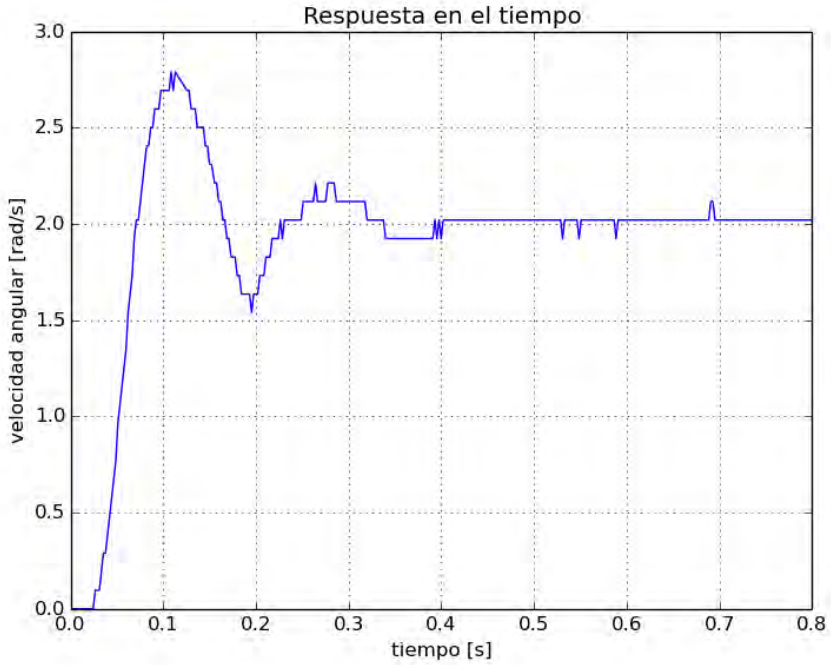

Figura 6.4: Motor izquierdo. Salida del sistema de control para una salida deseada de $2\left[\frac{rad}{s}\right]$.

En las Figuras 6.1, 6.2 y 6.3 se muestra la salida del sistema de control del motor derecho para distintas entradas según se indica en cada Figura, algunas de las características de las curvas de respuesta en el tiempo para este sistema de control se muestran en la Tabla 6.3. Las Figuras 6.4, 6.5 y 6.6 corresponden a la salida del sistema de control del motor izquierdo para las entradas que se indican en cada una de ellas, las características para estas curvas se muestran en la Tabla 6.4. Tanto en las Figuras como en las Tablas se observa el desempeño de cada sistema de control, las ganancias obtenidas para cada controlador son las mostradas en la Tabla 5.2. Es importante mencionar que el valor actual de la velocidad angular puede ser leído en la tabla de control de cada uno de los motores y que la resolución en la medición de la velocidad angular es de aproximadamente $0.096\left[\frac{rad}{s}\right]$. También cabe decir que existe ruido en la medición de la velocidad angular, además en las Figuras mostradas las curvas de respuesta no empiezan en el tiempo cero ya que el muestreo se realizó antes de que se aplicara la variable de control a cada uno de los motores.

$\omega_{deseada}\left[\frac{rad}{s}\right]$	$t_r\,[s]$	$\%S_p$	$t_s\,[s]$
2.0	0.043	40	0.201
2.5	0.056	40	0.221
3.0	0.050	38.33	0.206

Tabla 6.4: Motor izquierdo. Características de la respuesta transitoria.

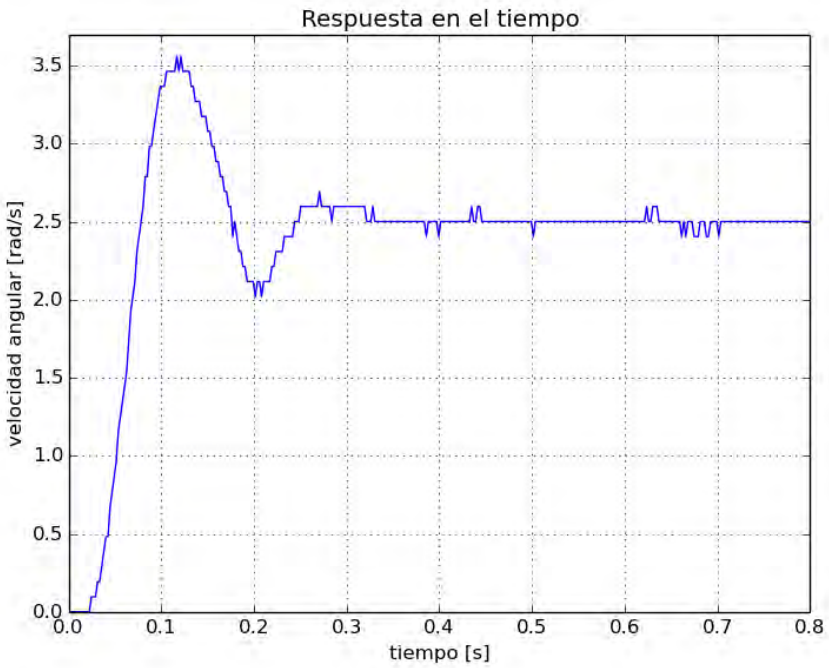

Figura 6.5: Motor izquierdo. Salida del sistema de control para una salida deseada de $2.5\left[\frac{rad}{s}\right]$.

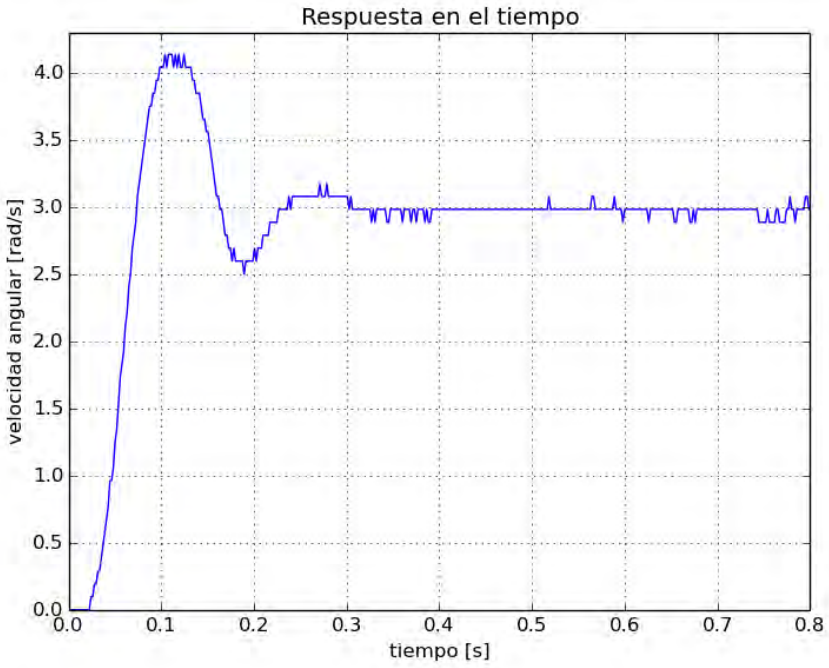

Figura 6.6: Motor izquierdo. Salida del sistema de control para una salida deseada de $3\left[\frac{rad}{s}\right]$.

6.3. Odometría

Para probar la localización del robot móvil por medio del cálculo de la odometría se comparó la pose estimada con la pose real del robot. Lo que se hizo fue hacer navegar al robot con ayuda del modo teleoperado hasta tres distintas ubicaciones, tales ubicaciones son las mostradas en la Figura 6.7 y se indican con los números dos, tres y cuatro, la ubicación marcada con el número uno es el sitio de donde partió el robot en cada una de las pruebas realizadas. En la Figura 6.8 se muestra el lugar en donde se llevaron acabo las pruebas, debido al espacio disponible solamente se establecieron tres ubicaciones, las ubicaciones se encuentran separadas entre sí a una distancia de un metro.

En total se llevaron a cabo tres pruebas, en la primera prueba el robot navegó desde la ubicación uno hasta la ubicación dos, en la segunda prueba el robot navegó desde la ubicación uno hasta la ubicación tres (pasando por la ubicación dos) y en la tercera prueba el robot navegó desde la ubicación uno hasta la ubicación cuatro (pasando por las ubicaciones dos y tres). La ubicación uno se encuentra en la posición $(0, 0)$, la ubicación dos se encuentra en la posición $(1, 0)$, la ubicación tres se encuentra en la posición $(2, 0)$ y la ubicación cuatro se encuentra en la posición $(2, -1)$.

Figura 6.7: Visualización de las ubicaciones en Rviz.

Figura 6.8: Visualización de ubicaciones en el lugar de pruebas.

En las Figuras 6.9, 6.11 y 6.13 se muestra la pose estimada del robot, estas imágenes fueron obtenidas por medio del visualizador Rviz, en ellas se observa parte de una cuadrícula uniforme de un metro por lado. Mientras que en las Figuras 6.10, 6.12 y 6.14 se muestran fotografías de la pose real del robot, en las que las losetas en el suelo sirven como una referencia visual tanto para la posición como para la orientación.

Si bien no se tiene un comparativo exacto entre la pose real del robot móvil con la pose estimada, un análisis a simple vista es útil; para hacer un comparativo exacto entre la pose estimada y la pose real habría que utilizar un método de localización activo. En las imágenes obtenidas en Rviz la cuadrícula uniforme sirve como guía visual para la pose, y en el caso de las fotografías tomadas al robot las losetas son de utilidad para este mismo fin, además en cada una de las imágenes se trazaron líneas auxiliares para tener una mejor referencia. Las Figuras 6.9 y 6.10 corresponden a la primera prueba, las Figuras 6.11 y 6.12 corresponden a la segunda prueba y las Figuras 6.13 y 6.14 corresponden a la última prueba. En todas las pruebas realizadas se operó al robot móvil con un joystick, teniendo particular cuidado que existiera poco deslizamiento entre las ruedas del robot y el suelo durante la navegación.

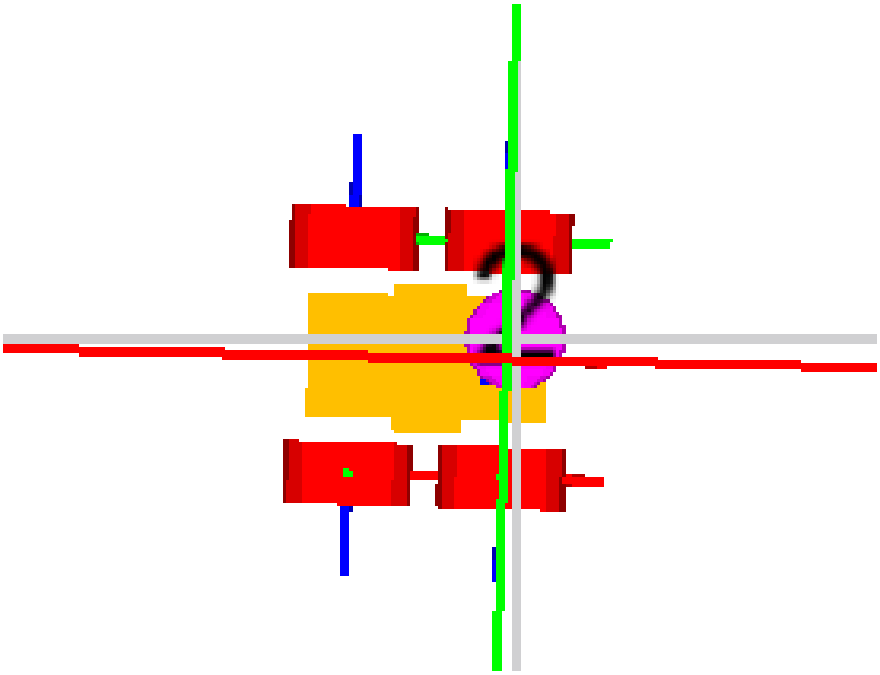

Figura 6.9: Pose estimada en la primera prueba.

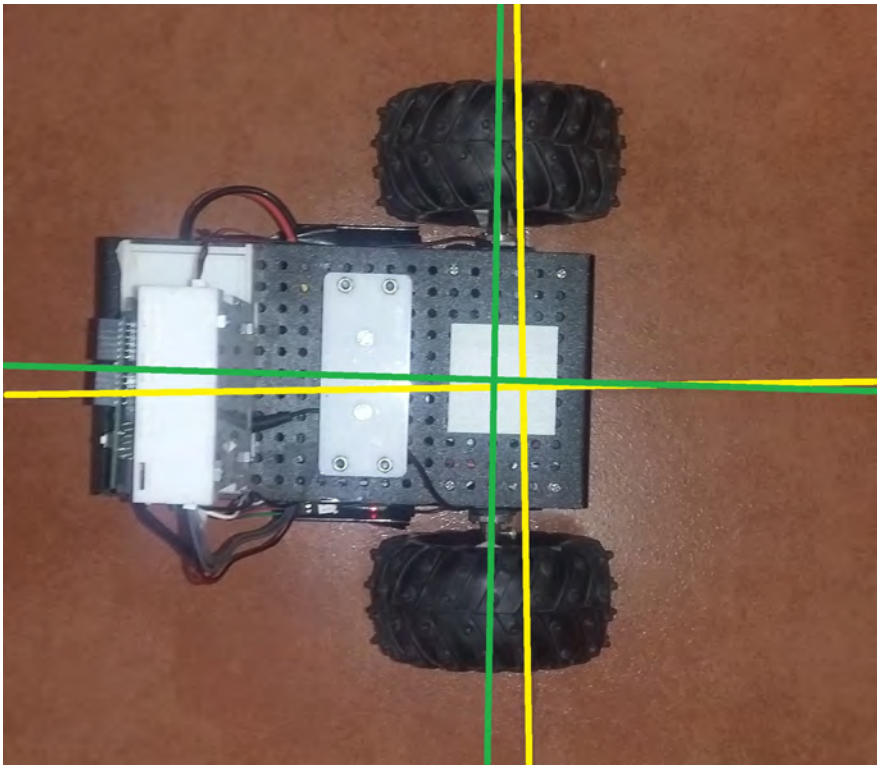

Figura 6.10: Pose real en la primera prueba.

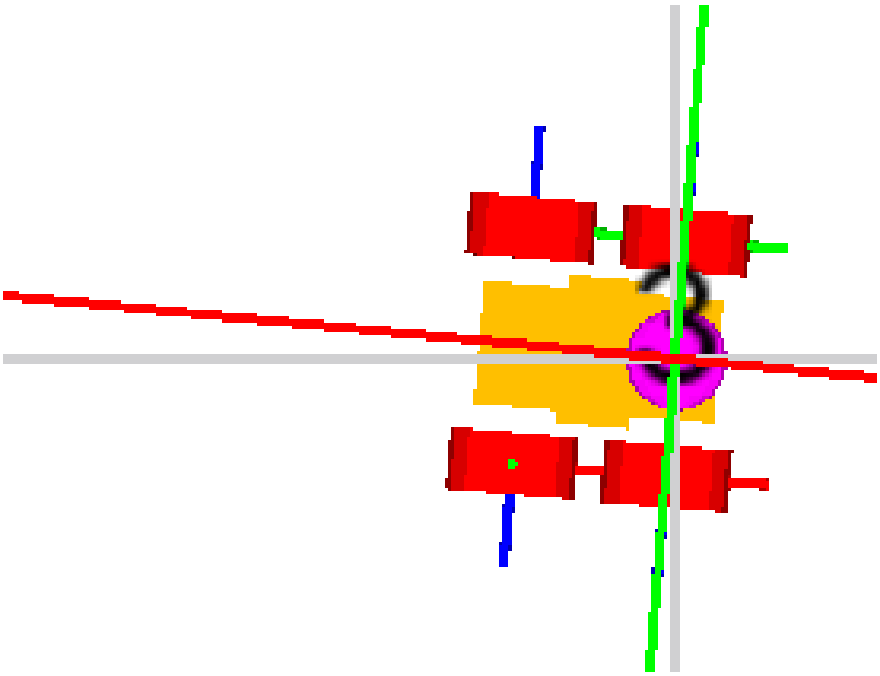

Figura 6.11: Pose estimada en la segunda prueba.

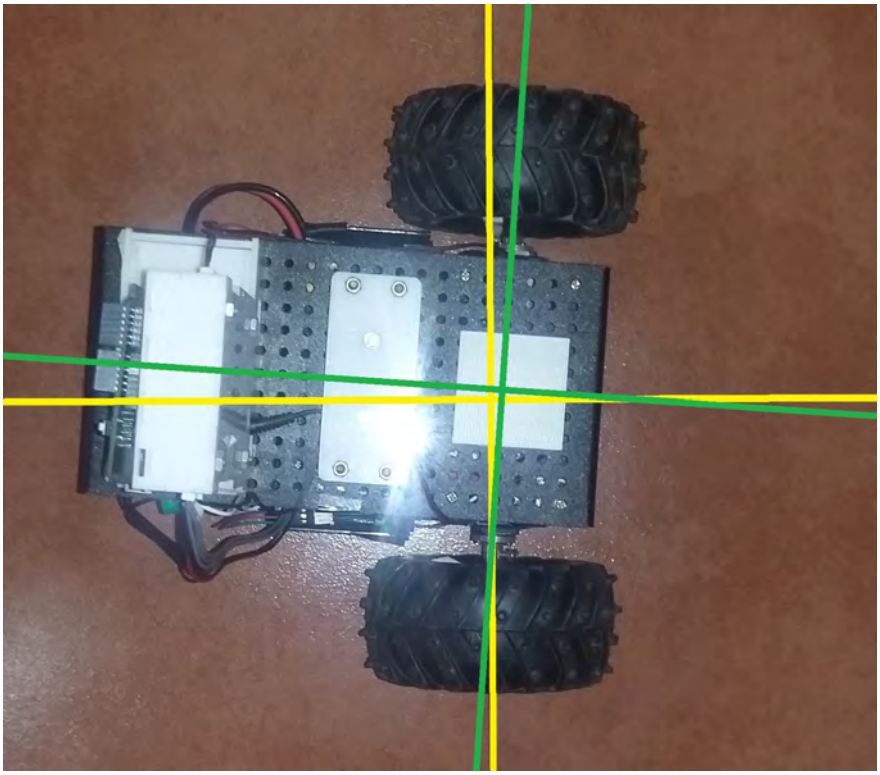

Figura 6.12: Pose real en la segunda prueba.

El robot móvil presentado en este trabajo es altamente susceptible al deslizamiento entre sus ruedas y el suelo, lo cual repercute directamente en la estimación de la pose del robot, esto se debe principalmente a que el motor izquierdo tiene problemas para moverse en el arranque lo cual favorece el deslizamiento entre la rueda derecha del robot y el suelo, además de que los cambios bruscos de velocidad también propician el deslizamiento en ambas ruedas.

En la primera y tercera prueba se obtuvo un resultado similar, ya que al observar las imágenes correspondientes se puede apreciar apreciar una diferencia entre la pose estimada y la pose real, esto se debe a que existió mayor deslizamiento durante la navegación del robot móvil. En tanto que en la segunda prueba al hacer un comparativo entre la pose estimada y la pose real se puede apreciar que la pose estimada se aproxima a la real, puesto que existió poco deslizamiento entre las ruedas del robot y el suelo durante la navegación.

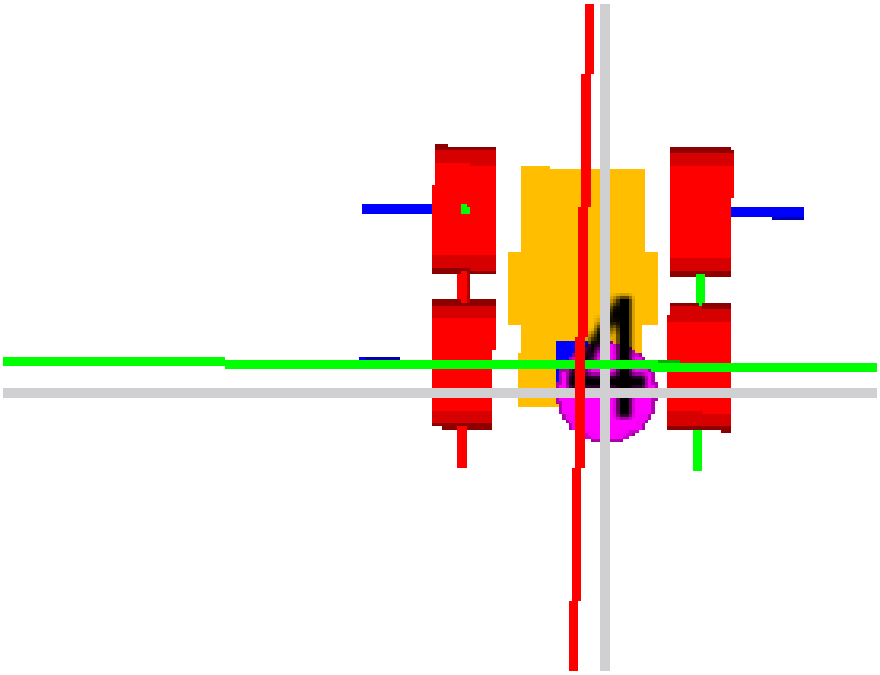

Figura 6.13: Pose estimada en la tercera prueba.

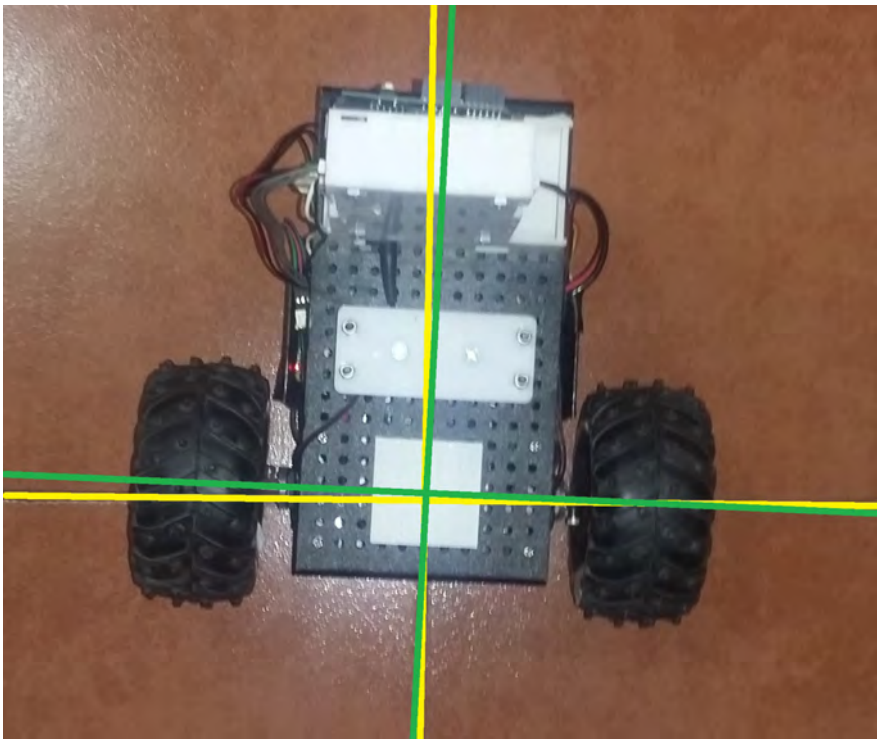

Figura 6.14: Pose real en la tercera prueba.

6.4. SLAM

En las Figuras 6.15 y 6.16 se muestran mapas generados en la planta baja del edificio U del conjunto sur de la Facultad de Ingeniería. El mapa mostrado en la Figura 6.15 es un mapa generado por el robot presentado en este trabajo, y el mapa de la Figura 6.16 es un mapa generado por el robot Justina del Laboratorio de Bio-Robótica.

Después de múltiples intentos la mejor representación del ambiente que se pudo obtener con el robot presentado en este trabajo es el mapa mostrado en la Figura 6.15, puesto que el problema que se presentó durante el mapeo fue el deslizamiento que existe entre las ruedas del robot y el suelo, lo cual tuvo como consecuencia una mala estimación de la pose del robot por medio del cálculo odometría y por ende resultó en no poder obtener una buena representación del entorno. Debido a lo anterior, en las pruebas hechas en el presente trabajo se optó por utilizar el mapa generado por el robot Justina, ya que además de ser un mapa más grande se tiene una mejor representación del entorno.

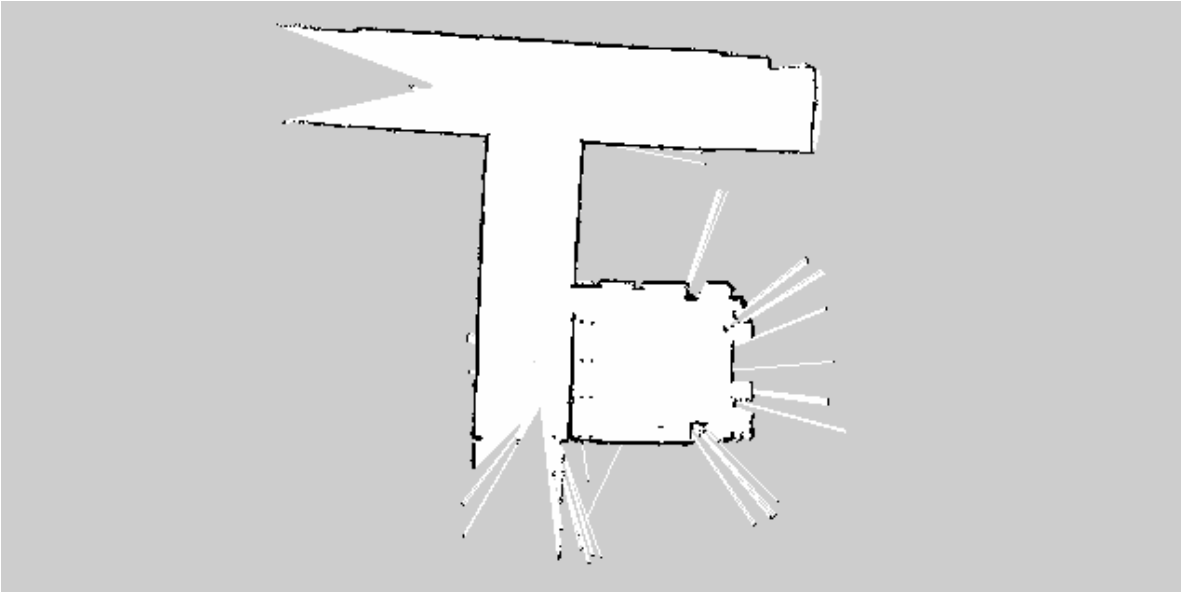

Figura 6.15: Mapa generado por el robot presentado en este trabajo.

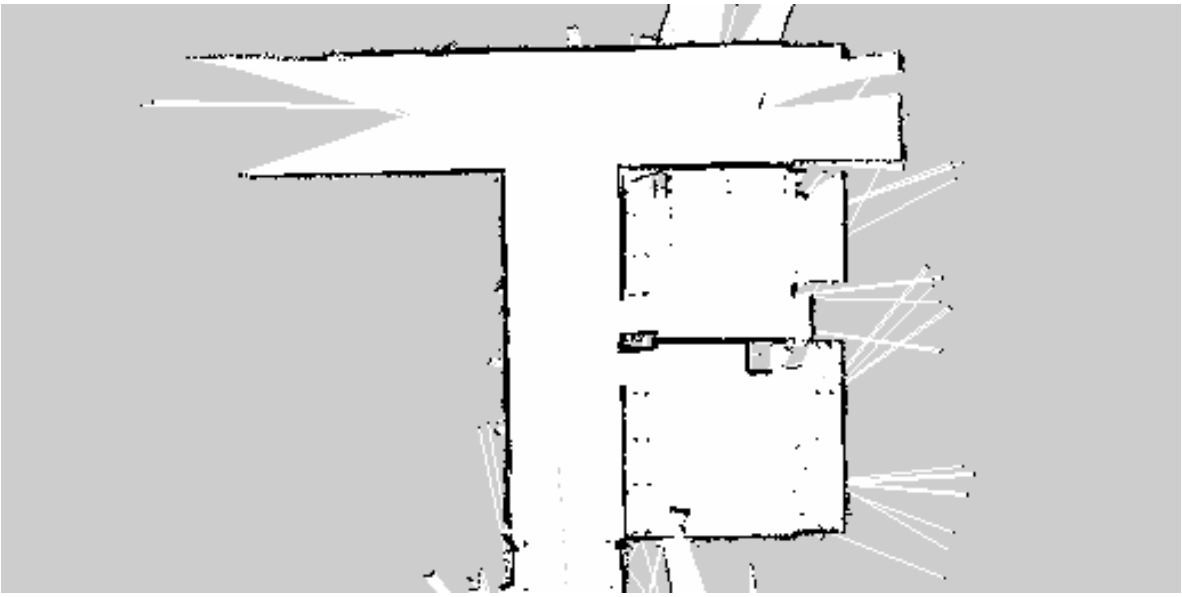

Figura 6.16: Mapa generado por el robot Justina.

6.5. Cálculo de rutas

En la Figura 6.17 se observa el resultado obtenido al crecer los obstáculos, el espacio libre y el espacio ocupado, cabe mencionar que los píxeles en color negro sólido representan el espacio ocupado, los píxeles más claros corresponden al espacio libre, y los píxeles transparentes en color negro son los que también se consideran como parte del espacio ocupado como resultado de crecer los obstáculos. Así mismo, en las Figuras 6.18, 6.19, 6.20 y 6.21 se muestran los resultados obtenidos por el nodo `path_calculator_node`, el cual se encarga de la planeación de rutas. En cada una de las imágenes mostradas se puede apreciar el sistema de referencia global `map` (eje x en color rojo, eje y en color verde y eje z en color azul), un mapa contenido en el plano xy y las rutas encontradas para distintos casos. En cada una de las imágenes se muestra en color verde la ruta obtenida por el algoritmo A* y en color rojo la ruta suavizada.

En la Figura 6.18 se muestra una ruta que conecta el punto $(0, -3)$ con el punto $(5, -1)$ y en la Figura 6.19 se muestra una ruta que conecta el punto $(5, -1)$ con el punto $(0, -3)$, es claro que las rutas encontradas son distintas, sin embargo, ambas tienen el mismo costo. Lo mismo sucede para el otro caso mostrado, esto se debe a que existen múltiples soluciones óptimas, es decir, múltiples soluciones con el menor costo posible.

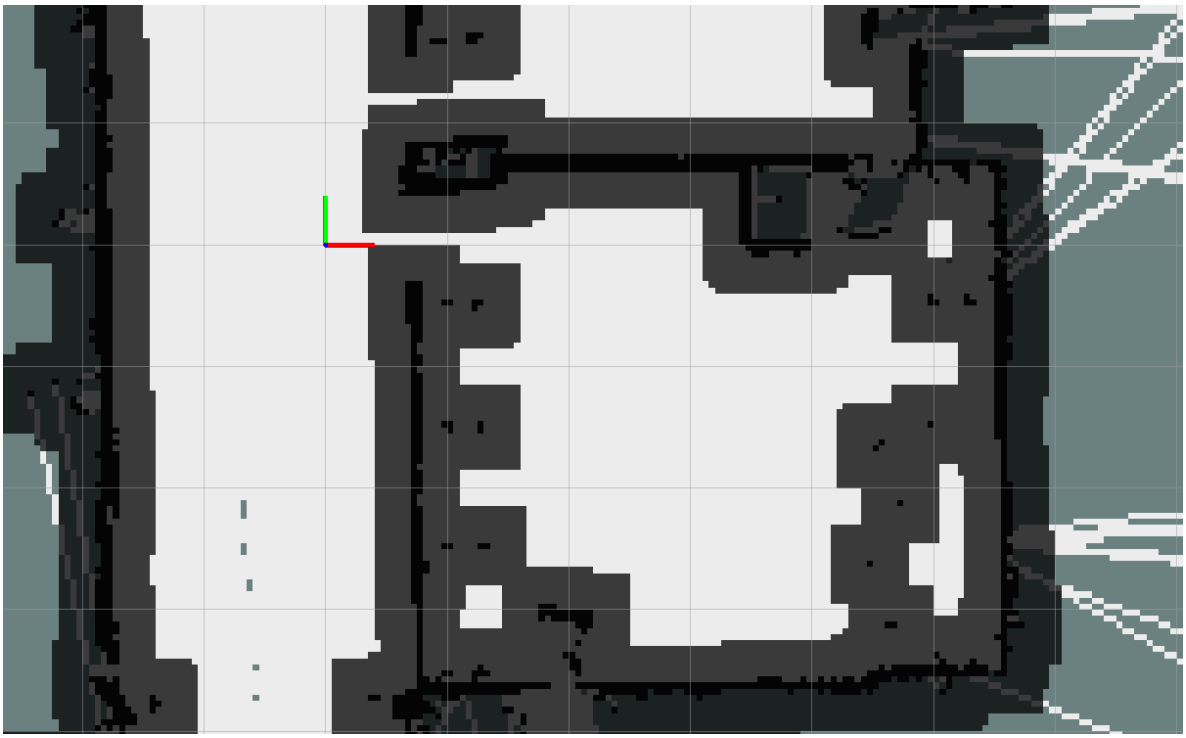

Figura 6.17: Crecimiento de obstáculos en el mapa.

Figura 6.18: Ruta (con 181 puntos) del punto $(0, -3)$ al punto $(5, -1)$.

Figura 6.19: Ruta (con 181 puntos) del punto $(5, -1)$ al punto $(0, -3)$.

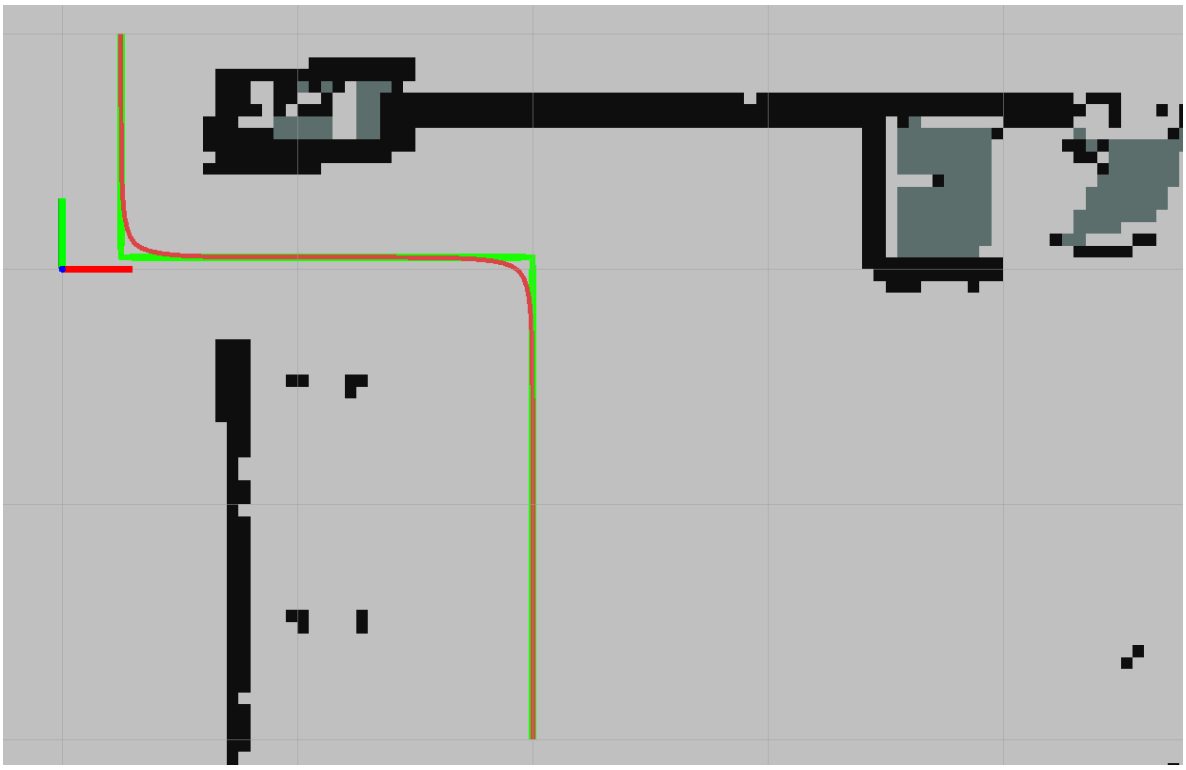

Figura 6.20: Ruta (con 96 puntos) del punto $(0.25, 1)$ al punto $(2, -2)$.

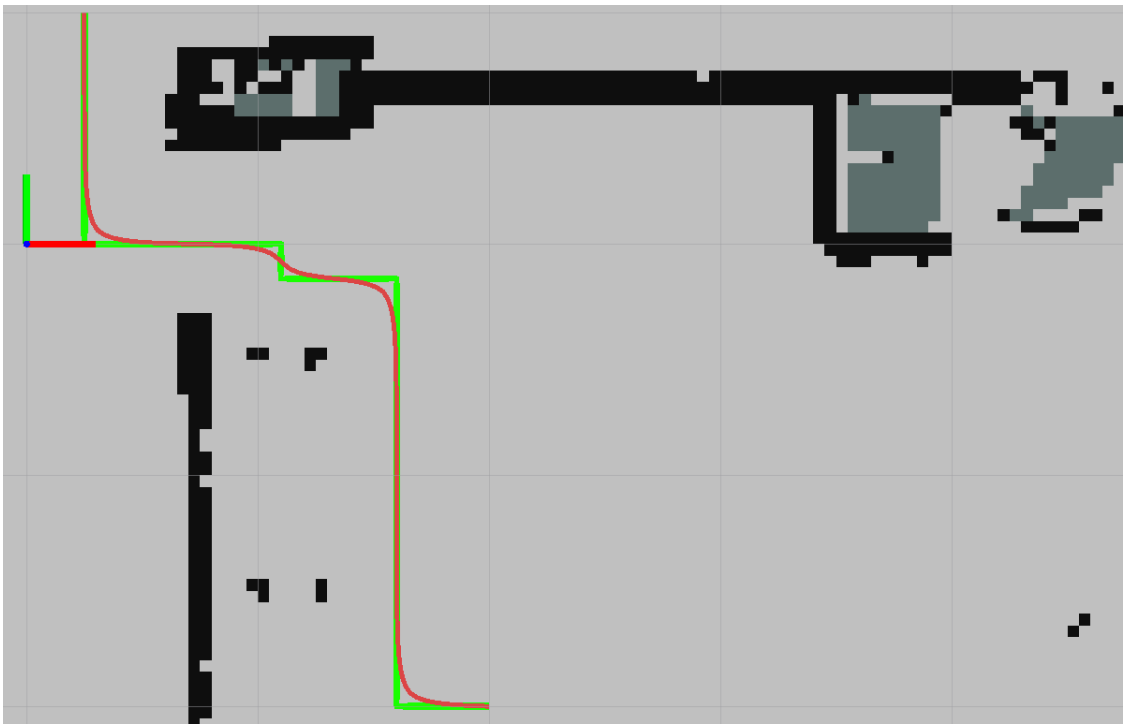

Figura 6.21: Ruta (con 96 puntos) del punto $(2, -2)$ al punto $(0.25, 1)$.

6.6. Seguimiento de rutas

Debido a que el error existente en la pose estimada crece excesivamente durante la navegación aún cuando se recorren distancias cortas se optó por sintonizar las leyes de control de posición con los motores a rotor libre, es decir, durante el proceso de sintonización el robot móvil no navegó en el lugar correspondiente al mapa utilizado, simplemente se consideró la pose calculada por medio de las lecturas dadas por los encoder de las ruedas. Esta prueba no es contundente, pero dado que los motores utilizados presentan los inconvenientes ya mencionados no es posible realizar una prueba de navegación autónoma sin que el robot tenga un alto error acumulado en la pose estimada.

Las pruebas realizadas fueron de utilidad para visualizar el comportamiento del robot móvil durante el seguimiento de rutas al variar las constantes presentes en las leyes de control, dichas constantes son v_{max}, ω_{max}, α y β, otra constante es la distancia a la que se encuentra la ubicación deseada que pertenece a un punto de la ruta a seguir, de aquí en adelante se hará referencia a tal distancia como el parámetro d.

Para los fines de navegación no es crucial que el robot móvil haga un seguimiento exacto de las rutas obtenidas, las rutas sirven para llegar a una ubicación destino a partir de la ubicación actual del robot, así como una guía para la evasión de obstáculos, por lo que es importante que la trayectoria descrita por el robot durante la navegación se asemeje en medida de lo posible a la ruta correspondiente obtenida.

El comportamiento del robot durante la navegación depende en conjunto de las cuatro constantes presentes en las leyes de control, pero en general resulta que cuando el valor de β es grande en comparación con el valor de α se presentan oscilaciones alrededor de la ruta cuando existen cambios de dirección de movimiento (en las curvas), esto se debe a que la magnitud de la velocidad angular no es la suficiente para cambiar la dirección del robot y se le da mayor peso a la magnitud de la velocidad lineal. De igual manera, cuando el valor de α es grande en comparación con el valor de β existen oscilaciones en los cambios de dirección de movimiento, las oscilaciones se presentan puesto que el efecto de la velocidad lineal hace que el robot avance en linea recta sin antes haberse orientado correctamente en la dirección de la ubicación deseada.

Las leyes de control utilizadas sirven para alcanzar un punto meta y a la distancia d se encuentra tal punto, de modo que para realizar el seguimiento de una ruta el valor del parámetro d debe ser una distancia cercana al origen del marco de referencia `base_link`. Cuando la distancia es chica el robot hace un mejor seguimiento de rutas, en tanto que al aumentar el valor del parámetro d el robot toma curvas más abiertas, pero si el valor es muy grande podría causar colisiones con obstáculos.

Los valores obtenidos para las constantes del controlador son los mostrados en la Tabla 5.4, con estos valores se logró hacer un seguimiento de tal forma que el robot puede evadir obstáculos y además se evitan oscilaciones en la curvas. El robot se detiene cuando se ubica aproximadamente a 5cm del último punto de la ruta que está siguiendo, para el presente trabajo no es de suma importancia que el robot llegue con exactitud al último punto de la ruta en cuestión.

En total se llevaron a cabo seis pruebas en las que se "simuló" el seguimiento de rutas, en estas pruebas se hizo navegar al robot hasta seis ubicaciones distintas de forma consecutiva partiendo del origen del sistema de referencia fijo (`map`), en la Figura 6.22 se muestra la ruta obtenida y la ubicación del robot móvil al final de la primera prueba. En la Tabla 6.5 se muestran las ubicaciones finales a las que arribó el robot en cada una de las pruebas, se puede ver que efectivamente el robot llegó a una ubicación que se encuentra a una distancia de aproximadamente 5cm de la ubicación destino.

Prueba	Ubicación origen	Ubicación destino	Ubicación del robot al final de la prueba
1	$(0.0, 0.0)$	$(3.75, 2.5)$ Kitchen	$(3.75, 2.45)$
2	$(3.75, 2.45)$	$(-0.5, 3.0)$ Corridor	$(-0.5, 2.96)$
3	$(-0.5, 2.96)$	$(3.0, -1.0)$ Simul Motion	$(3.0, -0.95)$
4	$(3.0, -0.95)$	$(4.0, 1.2)$ Simul Water	$(3.95, 1.2)$
5	$(3.95, 1.2)$	$(3.0, 6.0)$ Bedroom	$(3.0, 5.95)$
6	$(3.0, 5.95)$	$(0.0, 0.05)$ Simul Contact	$(0.0, 0.09)$

Tabla 6.5: Resultados del seguimiento de rutas.

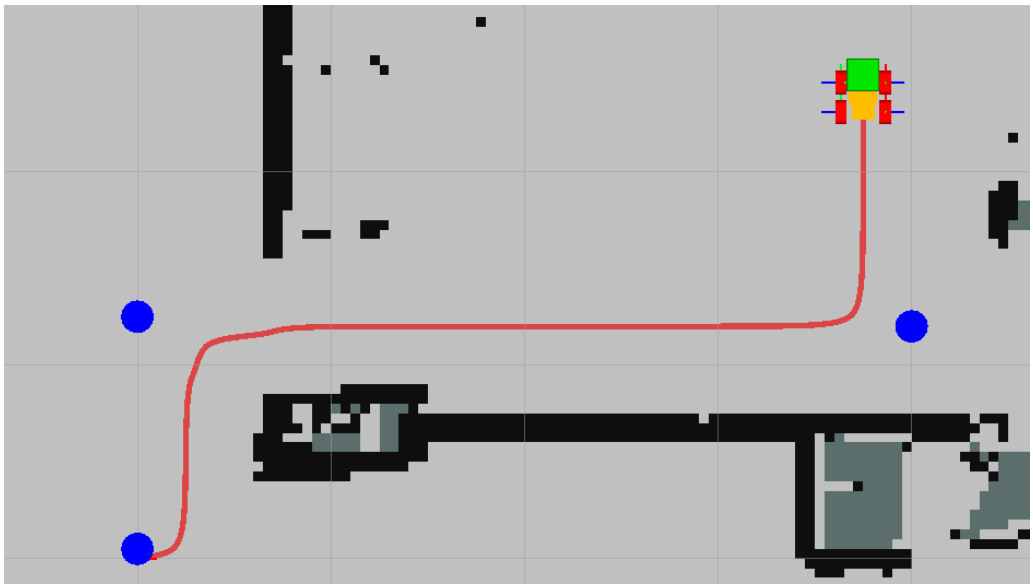

Figura 6.22: Ejemplo de seguimiento de rutas.

Capítulo 7

Conclusiones y trabajo a futuro

7.1. Conclusiones

El principal objetivo del presente trabajo fue la integración de un robot móvil como un nuevo dispositivo dentro del ecosistema de SmartThings, la propuesta inicial fue construir dicho robot móvil y hacerlo interactuar con dispositivos comerciales. La forma en la que se consiguió incorporar al robot en el entorno de SmartThings fue por medio de una placa ThingShield que se comunica utilizando el protocolo ZigBee.

En este trabajo se logró implementar el sistema de comunicación propuesto, se creó un *Device Handler* en el que se informa al usuario sobre la disponibilidad del robot y la carga actual de la batería, también se desarrolló una *SmartApp* en la que el usuario elige los dispositivos que se desea que reporten actividad inesperada tanto al robot como al mismo usuario por medio de notificaciones push en dispositivos móviles. De igual manera, se integró con éxito un modo tele operado y la capacidad de indicarle al robot que navegue hasta lugares preestablecidos dentro de una residencia.

Es importante mencionar que aún no es posible que un usuario final pueda hacer uso por completo del robot móvil creado. Como desventajas se tiene que es necesario contar con un mapa actualizado y bien definido del lugar en donde operará el robot móvil, igualmente se requiere editar el código fuente hecho en ROS puesto que es necesario que el robot tenga previo conocimiento de las ubicaciones en coordenadas cartesianas de los dispositivos seleccionados en la *SmartApp* y los lugares preestablecidos con respecto del sistema de referencia global `map`, esto con el fin de que el robot conozca la ubicación hasta la cual tiene que navegar. Aparte de esto, es necesario modificar el *Device Handler* creado para cambiar los nombres de los lugares preestablecidos dentro de una residencia, ya que por ahora se tienen etiquetas como "`Entrance`" y "`Living room`", y desde luego un usuario final deseará establecer sus propios lugares.

Un problema que se tiene con SmartThings es que los dispositivos únicamente pueden recibir un conjunto discreto o un conjunto continuo de valores conocidos, por ejemplo, se puede establecer la temperatura deseada en un termostato o encender y apagar un foco. Esto entra en conflicto al momento de querer hacer que el robot navegue hasta lugares preestablecidos dentro de una residencia, ya que este sería un conjunto discreto de valores desconocidos, y la forma en que se resolvió esto fue colocando un botón por cada uno de los lugares, lo cual claramente es una solución forzada.

Las pruebas realizadas al esquema de comunicación se hicieron considerando el peor de los escenarios, esto seguramente se suscitará en el modo tele operado, ya que en este modo se espera que se actúe al robot con mayor frecuencia debido a que continuamente se querrá cambiar la dirección de movimiento del robot móvil o simplemente será deseable detenerlo. De acuerdo a los resultados obtenidos en la pruebas correspondientes, se observa que en el peor de los casos se tuvo un 90 % de éxito en el envío de comandos al robot móvil y en el mejor de los casos se tuvo un éxito del 97 %. Aunque en los resultados obtenidos se tuvo un porcentaje de éxito alto, si una instrucción no llega de forma correcta al robot móvil en el modo tele operado podría causar que el robot colisione con algún objeto, también se darían los casos en los que el robot no se entere de la actividad detectada por los dispositivos instalados dentro del hogar o que no navegue hasta alguno de los lugares preestablecidos.

En los resultados obtenidos en las pruebas realizadas en el envío de datos al servidor de SmartThings se tuvo en el mejor de los casos un porcentaje de éxito del 98 % y en el peor de los casos se obtuvo un éxito de 89 %. En estas pruebas se envió información a intervalos de 1.5 segundos por medio de un programa hecho en Python. En el software desarrollado no se envía información al servidor de SmartThings de forma consecutiva en periodos de tiempo tan cortos, de hecho sólo se envía información sobre la carga actual de la batería y la disponibilidad actual del robot, cabe mencionar que esta información no varia constantemente puesto que la disponibilidad del robot cambia únicamente cuando el robot empieza a navegar de forma autónoma y cuando llega a su destino, en tanto que el voltaje actual en la batería se verifica cada 60 segundos. Si los datos llegan de forma incorrecta al servidor de SmartThings se tendría como consecuencia que no se refleje el estado correcto del robot en la aplicación para dispositivos móviles.

En general, la mayoría de las veces los comandos recibos por el robot móvil llegan de forma correcta siempre y cuando estos no sean enviados de forma consecutiva en intervalos de tiempo menores a dos segundos, lo mismo sucede cuando el robot envía información al servidor de SmartThings, sin embargo, ocasionalmente se tiene perdida de datos en ambos sentidos de la comunicación. En sí ninguno de los dispositivos con los que actualmente cuenta SmartThings requiere de una gran transferencia de datos en intervalos de tiempo tan cortos, por ejemplo, una cerradura o una válvula de agua no se están abriendo y cerrando cada dos

segundos, y la temperatura ambiental que mide un sensor no está variando cada uno o dos segundos. Una cuestión adicional a considerar es la latencia ya que en cualquier red existen retrasos, sin embargo, en SmartThings el retardo es perceptible para el humano, es decir, es notorio el tiempo que transcurre entre que se solicita que ocurra algo y cuando realmente ocurre, esto podría generar la sensación de que los comandos no se envían correctamente y provocar que un usuario ejecute múltiples comandos en periodos de tiempo cortos.

Otro punto a considerar es el alcance que tiene el protocolo ZigBee, el cual está entre 10 y 20 metros pero puede ser menor debido a interferencias generadas por teléfonos celulares, hornos de microondas, luces fluorescentes, entre otros aparatos electrónicos; esto representa una desventaja para el robot móvil dado que los dispositivos sólo se pueden conectar con un Hub y en grandes residencias se podría perder la comunicación. Aunque es importante mencionar que ciertos dispositivos de SmartThings actúan como repetidores, los cuales sirven como retransmisores y son de utilidad para mejorar el alcance, de forma que si un dispositivo está lejos del Hub entonces puede comunicarse con un repetidor. Cabe recalcar que para que todo dentro de SmartThings funcione se requiere que el Hub cuente con conexión a internet en todo momento, de lo contrario nada estará disponible, esto rompe con el objetivo de IoT, ya que en IoT se busca conseguir que las cosas estén disponibles en cualquier circunstancia.

En este trabajo no se consiguió construir un robot móvil capaz de localizarse correctamente por medio de la odometría, la cual es sumamente importante puesto que el éxito en la planificación de rutas depende de una representación precisa del área de navegación, y el éxito en la navegación de un robot móvil autónomo depende de una estimación confiable de la pose. Como se dijo en el capítulo anterior el principal factor que afecta el cálculo de la odometría en el robot construido es el deslizamiento que se produce entre las ruedas del robot y el suelo a causa del bajo par que entregan los motores utilizados, lo cual provoca que el error en la localización crezca demasiado aún cuando se han recorrido distancias muy cortas. Otros factores que afectan en el cálculo de la odometría son irregularidades en el suelo, colisiones con obstáculos, mal alineamiento de las ruedas y el hecho de que las ruedas no giran concéntricamente.

Bajo las suposiciones de un entorno estático y contar con un mapa actualizado y bien definido se logró obtener exitosamente rutas óptimas y suaves libres de obstáculos, dichas rutas unen una ubicación inicial con una ubicación final siempre y cuando exista la posibilidad de alcanzar la ubicación final desde la ubicación inicial y tales ubicaciones se encuentren dentro del espacio libre.

Debido al error en la pose estimada a causa del deslizamiento que existe entre las ruedas del robot y el suelo no fue posible construir un mapa con una buena precisión, si bien la representación obtenida se aproxima al entorno real, en los resultados mostrados en el capítulo anterior se puede apreciar que existe cierta

inexactitud. Entre los factores que incrementan la dificultad del mapeo están el tamaño del entorno y la ambigüedad perceptual, mientras más grande sea el entorno en relación con el rango de percepción del robot será más difícil relocalizarse correctamente y aumenta la posibilidad de crear un mapa erróneo, la ambigüedad perceptual se refiere a que mientras más se parezcan distintos lugares será más difícil diferenciarlos. En el mapa mostrado en el capítulo anterior se tuvieron estos dos problemas al momento de navegar por un pasillo, ya que el alcance del sensor láser es limitado y al momento de navegar por el pasillo el robot prácticamente percibe las mismas características del entorno, por lo que esto propició una mala estimación de la pose cuando existió deslizamiento entre las ruedas del robot y el suelo, de esta forma se tuvo como consecuencia la construcción de un mapa inexacto.

Es sabido que un sistema de control en lazo cerrado permite compensar perturbaciones, así como lidiar con dinámicas no modeladas e incertidumbres, pero para poder implementar este tipo de sistema de control es necesario medir la salida de la planta, es decir, se requiere conocer el valor de las variables a controlar. Por ello, controlar adecuadamente los estados de una planta dependerá de una correcta medición de los mismos. A diferencia de los robots manipuladores, los encoder en los robots móviles no proveen una medición directa de la configuración del robot, por esta razón se utilizó la odometría para conocer el estado del sistema, pero como ya se dijo, en el robot presentado en este trabajo el error en la pose estimada crece rápidamente, lo cual imposibilita la navegación del robot.

El robot móvil presentado no es capaz de navegar de forma autónoma, pero en la prueba diseña para el seguimiento de rutas se pudo determinar el efecto de las constantes en las leyes de control sobre el comportamiento del robot, lo cual será de utilidad para el trabajo a futuro.

7.2. Trabajo a futuro

Una vez descritos los problemas en el presente trabajo es necesario proponer las modificaciones necesarias para lograr el funcionamiento deseado del robot móvil. El modelo presentado fue de utilidad para identificar las áreas de oportunidad, y dado que ya se contaba con todo lo necesario para su construcción no fue posible plantear otra propuesta aún ante la detección de inconvenientes.

El principal inconveniente que se tiene son los motores utilizados por las razones ya mencionadas, así que una mejora sería cambiarlos por unos motores DC que cuenten con encoders.

Para dar solución al problema de la deriva en la localización odométrica se propone utilizar un método de localización activo, ya que permite corregir la pose estimada dados un mapa conocido del lugar en que opera el robot y las lecturas de

los sensores exteroceptivos (lecturas del sensor láser). Una ventaja de utilizar ROS es que existe software aportado por la comunidad y, actualmente existe el paquete `amcl` que tiene un nodo que implementa el enfoque de localización de Monte Carlo, por lo que basta con descargar el paquete y ajustar correctamente los parámetros del nodo, claro que primero habría que entender el algoritmo y la función de cada uno de los parámetros.

Para evitar tener que escribir todos los comandos necesarios en una terminal para la puesta a punto del robot móvil, se puede crear un shell script en el que se configuren correctamente las variables de entorno de ROS y se ejecuten todos los nodos requeridos para su funcionamiento, de modo que el script se ejecute en alguno de los runlevels a los que entra el sistema operativo durante el inicio.

Para hacer uso de la interfaz web creada se utiliza un websocket, de manera que se requiere especificar una dirección IP y un puerto, los cuales se encuentran establecidos en el código fuente creado, cuando cualquier dispositivo se conecta a una red la asignación IP es dinámica, por lo que habría que asignarle una IP estática a la Raspberry Pi para evitar modificar el código fuente constantemente.

Para que el robot navegue hasta alguno de los lugares predeterminados o a un lugar en donde se encuentre algún sensor necesita conocer la ubicación correspondiente en coordenadas cartesianas, estas ubicaciones están definidas en el nodo `planner_node`. Como se pretende que un usuario final haga uso del robot móvil, otra modificación sería proporcionar una forma en la que el usuario pueda editar tales ubicaciones, una posible solución podría ser implementar una base de datos, de tal forma que el usuario pueda tener acceso a ella por medio de la página web creada y que el nodo `planner_node` obtenga esa información.

Bibliografía

[1] Rajkumar Buyya and Amir Vahid Dastjerdi. *Internet of Things Principles and Paradigms*. Elsevier, USA, 2016.

[2] Hakima Chaouchi. *The Internet of Things Connecting Objects*. ISTE, London, UK, 2010.

[3] Ovidiu Vermesan and Peter Friess. *Internet of Things - Global Technological and Societal Trends*. River Publishers, Denmark, 2011.

[4] Ovidiu Vermesan and Peter Friess. *Internet of Things - Converging Technologies for Smart Environments and Integrated Ecosystems*. River Publishers, Denmark, 2013.

[5] SmartThings. SmartThings Developer Documentation. `http://docs.smartthings.com/en/latest/`. [En línea]. [Fecha de consulta: 2017].

[6] Bruno Siciliano, Lorenzo Sciavicco, Luigi Villani, and Giuseppe Oriolo. *Robotics - Modelling, Planning and Control*. Springer, London, 2009.

[7] Philip Husbands, Owen Holland, and Michael Wheeler. *The Mechanical Mind in History*. The MIT Press, Cambridge, Massachusetts, USA, 2008.

[8] Rafael Kelly, Victor Santibáñez Davila, and Antonio Loría. *Control of Robot Manipulators in Joint Space*. Springer, London, 2005.

[9] V. Daniel Hunt. *Industrial Robotics Handbook*. Industrial Press, Inc., New York, USA, 1th edition, 1983.

[10] Aníbal Ollero Baturone. *Robótica: manipuladores y robots móviles*. Marcombo, Barcelona, España, 2001.

[11] Sebastian Thrun, Wolfram Burgard, and Dieter Fox. *Probabilistic Robotics*. The MIT Press, Cambridge, Massachusetts, USA, 2006.

[12] Mark W. Spong, Seth Hutchinson, and M. Vidyasagar. *Robot Dynamics and Control*. Wiley, USA, 2nd edition, 2004.

[13] Roland Siegwart and Illah R. Nourbakhsh. *Introduction to Autonomous Mobile Robots*. The MIT Press, Cambridge, Massachusetts, USA, 2004.

[14] Shuzhi Sam Ge and Frank L. Lewis. *Autonomous Mobile Robots - Sensing, Control, Decision Making and Applications*. CRC Press, Florida, USA, 2006.

[15] Juan Andradem Cetto and Alberto Sanfeliu. *Environment Learning for Indoor Mobile Robots - A Stochastic State Estimation Approach to Simultaneous Localization and Map Building*. Springer, Germany, 2006.

[16] J. Borenstein, H. R. Everett, and L. Feng. *Where am I? Sensors and Methods for Mobile Robot Positioning*. University of Michigan, USA, 1996.

[17] Thomas Bräunl. *Embedded Robotics - Mobile Robot Design and Applications with Embedded Systems*. Springer, Germany, 2nd edition, 2006.

[18] Béla Lantos and Lőrinc Márton. *Nonlinear Control of Vehicles and Robots*. Springer, London, UK, 2011.

[19] Spyros G. Tzafestas. *Introduction to Mobile Robot Control*. Elsevier, London, UK, 1th edition, 2014.

[20] ROS. Documentation - ROS wiki. `http://wiki.ros.org/`. [En línea]. [Fecha de consulta: 2017].

[21] ROS. ROS Introduction. `http://wiki.ros.org/ROS/Introduction`. [En línea]. [Fecha de consulta: 2017].

[22] Morgan Quigley, Brian Gerkey, and William D. Smart. *Programming Robots with ROS*. O'Reilly Media, Inc., USA, 2016.

[23] Dirk Krafzig, Karl Banke, and Dirk Slama. *Enterprise SOA: Service-oriented Architecture Best Practices*. Pearson Education, USA, 2005.

[24] ROS. ROS Concepts. `http://wiki.ros.org/ROS/Concepts`. [En línea]. [Fecha de consulta: 2017].

[25] Lentin Joseph. *Mastering ROS for Robotics Programming*. Packt Publishing, Birmingham, UK, 2015.

[26] Katsuhiko Ogata. *Modern Control Engineering*. Prentice Hall, New Jersey, USA, 4th edition, 2002.

[27] Norman S. Nise. *Control Systems Engineering*. Wiley, USA, 6th edition, 2011.

[28] Stanislaw H. Żak. *Systems and Control*. Oxford University Press, New York, USA, 2003.

[29] Pololu. Dagu Wild Thumper 4WD All-Terrain Chassis. `https://www.pololu.com/product/1567`. [En línea]. [Fecha de consulta: 2017].

[30] Dynamixel. MX-12W. `http://support.robotis.com/en/product/actuator/dynamixel/mx_series/mx-12w.htm`. [En línea]. [Fecha de consulta: 2017].

[31] Raspberry Pi. Raspberry Pi 3 Model B. https://www.raspberrypi.org/products/raspberry-pi-3-model-b/. [En línea]. [Fecha de consulta: 2017].

[32] Hokuyo. Hokuyo URG-04LX-UG01 Scanning Laser Rangefinder. http://www.robotshop.com/en/hokuyo-urg-04lx-ug01-scanning-laser-rangefinder.html. [En línea]. [Fecha de consulta: 2017].

[33] Arduino. Arduino Introduction. https://www.arduino.cc/en/Guide/Introduction. [En línea]. [Fecha de consulta: 2017].

[34] Arduino. Arduino Uno Rev3. https://store.arduino.cc/usa/arduino-uno-rev3. [En línea]. [Fecha de consulta: 2017].

[35] SmartThings. Arduino ThingShield. http://docs.smartthings.com/en/latest/arduino/. [En línea]. [Fecha de consulta: 2017].

[36] D-Link. DCS-932L. http://www.dlinkla.com/dcs-932l. [En línea]. [Fecha de consulta: 2017].

[37] D-Link. DCS-932L Day/Night Cloud Camera. http://www.dlink.com/fi/fi/products/dcs-932l-day-night-cloud-camera. [En línea]. [Fecha de consulta: 2017].

[38] Brian Schneider. A Guide to Understanding LiPo Batteries. https://rogershobbycenter.com/lipoguide/. [En línea]. [Fecha de consulta: 2017].

[39] Pololu. Pololu Adjustable 4-12V Step-Up/Step-Down Voltage Regulator. https://www.pololu.com/product/2572. [En línea]. [Fecha de consulta: 2017].

[40] Robot Web Tools. http://robotwebtools.org. [En línea]. [Fecha de consulta: 2017].