Proyecto de grado

EasyRobots: Control y Comportamiento de Robots Omnidireccionales

Documento Final

Santiago Martínez, Rafael Sisto
pgomni@fing.edu.uy
http://www.fing.edu.uy/inco/grupos/mina/pGrado/omnidireccionales.html

Tutor

Gonzalo Tejera

Cotutores

Facundo Benavides, Santiago Margni

Instituto de Computación
Facultad de Ingeniería - Universidad de la República
Montevideo - Uruguay

13 de marzo de 2010

Resumen

El gran avance de la tecnología en los últimos años ha posicionado a la robótica y la automatización de procesos como aspectos cotidianos en la vida de las personas, de manera de simplificar las tareas que se pueden delegar a robots, como pueden ser aquellas repetitivas o riesgosas para la vida humana. Para la automatización de éstas, se requieren personas cada vez más capacitadas para desarrollar nuevas tecnologías, debido a los crecientes conocimientos necesarios. Las instituciones académicas tratan de adaptarse a estos requerimientos mediante actividades recreativas que generen interés en jóvenes estudiantes, como pueden ser competencias de fútbol y sumo robótico. Si bien estas competencias generan mucho interés en el público en general, la cantidad de jóvenes que puede participar se ve acotada por los conocimientos técnicos requeridos en las diversas disciplinas que enmarca la robótica, como pueden ser programación, física, matemática, mecánica, electrónica, entre otras. Estas instituciones han intentado acercar nuevas generaciones de estudiantes mediante la facilitación de robots construídos, permitiendo que el participante pueda enfocarse en el desarrollo del comportamiento del robot. A pesar de estas simplificaciones, la implementación de estos comportamientos requieren, al menos, sólidos conocimientos en programación y matemática.

En el presente proyecto se desarrolla un sistema de *software* que abstrae al usuario de los problemas más comunes de la creación de sistemas para robots móviles. El sistema *EasyRobots* surge para complementar la carencia de estos conocimientos, facilitando comportamientos genéricos predefinidos y editores gráficos para la creación de los mismos. *EasyRobots* está orientado tanto a usuarios con escasos conocimientos de programación, quitando la necesidad de programar, así como a usuarios con conocimientos sólidos en esta área. Podría considerarse la vía de entrada al mundo de la robótica para muchos jóvenes investigadores, quitando inicialmente la necesidad de transitar el filtro de años de estudio de matemáticas y programación pudiéndose incluir en planes de estudios secundarios o incluso primarios, motivando de esta forma el estudio de matemáticas y física a fin de generar sistemas robóticos más complejos.

Para la creación de *EasyRobots* se contemplaron tres objetivos principales: investigación y construcción de robots omnidireccionales, estudio de técnicas de fusión de sensores e implementación de un sistema flexible y modular que permita la reutilización de código generado, proveyendo una mayor escalabilidad y simplificación en el mantenimiento de sistemas robóticos.

Es necesario considerar que a la fecha se han desarrollado distintos tipos de robots móviles y sistemas de tracción, que pueden variar de acuerdo al destino de uso y al ambiente. En ambientes altamente dinámicos y reducidos, por ejemplo un edificio de oficinas, los robots requiere una gran movilidad para poder evitar obstáculos que puedan interponerse dinámicamente en su camino. En este caso, un robot de dos ruedas debe rotar para evadir el obstáculo y luego continuar con su trayectoria original. Dentro de este marco surgen los robots omnidireccionales (también conocidos como robots holonómicos), que pueden dirigirse en cualquier sentido independientemente de su orientación. En el ejemplo anterior, un robot omnidireccional no debe variar su orientación para evitar el obstáculo. Esto motivó la investigación y la construcción de un robot omnidireccional de manera de evaluar las diferencias de uso con los robots no omnidireccionales. Para tales fines, se estudiaron las distintas ruedas y estructuras posibles y necesarias a utilizar para su construcción. Finalmente, se construyó un robot utilizando un *kit* de construcción *Lego Mindstorms NXT* y ruedas omnidireccionales compatibles adquiridas en el exterior para su posterior utilización en el sistema *EasyRobots*.

El posicionamiento y la navegación son las tareas más importantes de los robots móviles. Es de gran importancia conocer la posición del robot de forma de alcanzar un cierto destino. La utilización de un único sensor en un sistema robótico, presenta serios problemas asociados a los componentes físicas de los sensores como pueden ser puntos únicos de fallas, incertidumbres inherentes y rangos de medición acotados. Por ello, surgen distintas técnicas de fusión de sensores que permiten mitigar estos problemas haciendo uso de diversos tipos de sensores coordinados en un sistema robótico. Esta motivación derivó en el estudio de los principales conceptos de la fusión de sensores así como de las técnicas posibles de implementar para su posterior utilización en el sistema *EasyRobots*.

En este documento se describen los aspectos más importantes del desarrollo del sistema *EasyRobots*, su utilización e interacción con aplicaciones externas. Además, se detallan las consideraciones más importantes sobre el tipo de robots omnidireccionales y las técnicas de fusión de sensores, los experimentos realizados y finalmente las conclusiones obtenidas.

Índice

Índice de figuras

1. Introducción

1.1. Generalidades

Los robots han sido y son un disparador para la imaginación. Que el hombre sea capaz de construir un ente que emule al propio ser humano en sus capacidades, aunque sea parcialmente, sin duda pertenece también al campo de los sueños.

La palabra robot deriva de la palabra checa "robota", que significa trabajo forzoso o esclavo. Fue introducida por el escritor Karel Capek, cuyas obras de ficción robótica incluían, entre otros, criaturas creadas por elementos químicos y biológicos, en lugar de mecánicos. Pero los robots mecánicos actuales no son muy diferentes de estas creaciones biológicas ficticias. En la actualidad, los robots son creados con el fin de reemplazar al hombre en tareas que pueden llegar a ser peligrosas para la vida humana, tediosas o repetitivas.

Básicamente, un robot se compone de [14]:

- Una construcción mecánica, tal como una plataforma con ruedas, o un brazo mecánico, capaz de interactuar con su entorno.
- Los sensores acoplados al robot o externos, que permiten percibir el entorno.
- Sistemas de control que toman como entrada los datos sensoriales, de forma de instruir al robot para tomar acciones.

Estos tres componentes fueron los detonantes para el desarrollo del proyecto derivándose en tres grandes subáreas de estudio como son Robots Omnidireccionales (1.1.1), Fusión de Sensores (1.1.2) y Sistemas de Control de Robots (1.1.3) en general.

1.1.1. Robots Omnidireccionales

Muchas veces puede llegar a ser un problema muy complejo realizar trayectorias en las que haya que cambiar bruscamente de dirección, como pueden ser el recorrido de laberintos y tareas en ambientes muy dinámicos. Debido a esto surge la necesidad de contar con robots que puedan sortear fácilmente objetos móviles y poseer una gran maniobrabilidad.

Este proyecto surge motivado por la necesidad de investigar los robots omnidireccionales y el amplio espectro de aplicaciones que implica este tema. En el mundo existen variadas aplicaciones utilizando este tipo de vehículos con fines académicos, industriales, médicos e incluso lúdicos. En este caso, se hizo hincapié en las aplicaciones académicas, aunque sin quitar importancia a las demás aplicaciones.

Los robots omnidireccionales tienen la habilidad de desplazarse en cualquier dirección en el plano y rotar simultáneamente; esto significa que poseen tres grados de libertad en su movilidad. Dichos robots tienen mayor movilidad comparada a los robots más comunes, no-omnidireccionales. La diferencia principal, radica en la habilidad de éstos de desplazarse hacia los lados. La habilidad de moverse en cualquier dirección, independientemente de la orientación del robot los convierte en una atractiva opción para su aplicación en ambientes dinámicos. Las competencias anuales de RoboCup[1][15] son un ejemplo de ámbitos donde los robots omnidireccionales han sido utilizados.

En este proyecto se construyó un robot omnidireccional y se comprobaron las habilidades de realizar trayectorias complejas de forma simple, pudiendose así utilizar este robot como prototipo para un equipo de fútbol de robots u otras aplicaciones como un repartidor en una oficina o un montacargas omnidireccional.

1.1.2. Fusión de Sensores

Otro de los temas tratados es la Fusión de sensores o *Sensor Data Fusion*. Existen en el mercado una variada gama de tipos de sensores y sistemas de sensado que pueden ser aplicados en diversas problemáticas. Sin embargo, muchos sistemas robóticos utilizan un único tipo de sensores por vez. Combinar las observaciones en una descripción coherente del mundo es un problema básico de la percepción tanto en humanos como en animales; esto a su vez lleva a evaluar la factibilidad de la fusión de datos sensoriales en sistemas robóticos, de forma de dotar a los robots

[1] Originalmente llamada *Robot World Cup Initiative*, es una iniciativa internacional de investigación y educación. En ese sentido intenta fomentar la investigación en Inteligencia Artificial y robótica proporcionando un problema estándar donde una amplia gama de tecnologías puede ser integrada y examinada, así como promover proyectos de educación integrada.

con mayor inteligencia de sensado. Se discute un *marco de trabajo*[2] para la fusión de datos sensoriales numéricos de forma de atacar el problema de fusión de sensores y finalmente se desarrolla una aplicación que integre el mismo.

1.1.3. Sistema de Control de Robots

Por último, pero no menos importante, se trató el problema de desarrollo y prototipación robótica en general. En la actualidad, una persona para poder desarrollar un sistema robótico académico debe preparar un ambiente de prueba (configurar sensado, actuado y construcción de robots). Desarrollar estas tareas puede insumir mucho tiempo y conocimiento, aunque actualmente se puede ver simplificada haciendo uso de *kits* de construcción, por ejemplo el *kit Lego Mindstorms NXT* [16] para la construcción, actuación y sensado del robot, y utilizando sistemas de visión global, como pueden ser *Ergo* o *Doraemon*[17]. Luego de resolver estos desafios, existe un problema mayor que es el desarrollo de *software* de las estrategias robóticas y comunicación hacia los robots, requiriendo conocimientos extensos de programación y algoritmos que dificultan el acercamiento a la robótica de personas jóvenes sin la formación adecuada. Por este motivo, en el presente proyecto se intentó atacar esta dificultad con un primer acercamiento de un *marco de trabajo* que simplifique el desarrollo y ejecución de sistemas robóticos. Los usuarios de este sistema pueden desarrollar sistemas robóticos a través de la definición de archivos XML[3] conteniendo información de los sistemas de sensado, actuación, robots y comportamientos, que pueden ser editados gráficamente. Esto permitiría que usuarios sin conocimientos de programación puedan adquirir una mejor comprensión de la robótica y generaran interés en el área.

1.2. Antecedentes

El Instituto de Computación de la Facultad de Ingeniería de la Universidad de la República Oriental del Uruguay ha promovido diversos proyectos en el área de la robótica e inteligencia artificial en los últimos años.

En el año 2003 se realizaron tres proyectos que abarcaban diversos aspectos de la robótica. En uno de ellos se trabajó con el objetivo de construir un sistema de visión para un equipo de fútbol de robots, enfrentando el problema de procesamiento y reconocimiento de patrones en imágenes en tiempo real. En otro de los proyectos del mismo año, se trabajó con el objetivo de construir un robot a bajo costo que fuera capaz de moverse en forma autónoma, recibiendo órdenes desde una computadora. Finalmente, en el proyecto restante se construyó un equipo de fútbol de robots denominado F.R.U.T.O., que se enfocó en el comportamiento del robot trabajando en un ambiente simulado. En el transcurso del desarrollo de dicho proyecto, se creyó conveniente la creación de un simulador de robótica, denominado GSim, con el objetivo de obtener mejores resultados en el desarrollo del equipo.

En el año 2004 se llevó a cabo un proyecto en el cual se construyó un prototipo de robot bípedo, capaz de caminar cuasi estáticamente.

Luego, en el año 2005 se presentaron dos proyectos que tenían como objetivo seguir trabajando en el área del comportamiento de los robots y en sistemas M.A.S.[4], donde se realizó la construcción de equipos de fútbol de robots para actuar en ligas simuladas. Ambos proyectos crearon equipos para la participación en ligas simuladas, uno para la federación FIRA y otro para la Robocup. Un poco al margen de estos proyectos, desde el año 2004, cada año el grupo MINA organiza el Campeonato Uruguayo de Sumo de Robots, donde los participantes utilizan un robot y compiten por la mejor estrategia programada. Este grupo también ha participado desde el año 2003 del CAFR[5], alternando entre la categoría real y la simulada.

En el año 2007, se realizó un proyecto de grado con el propósito de poseer un entorno de simulación robótica genérico, el cual pudiera ser una herramienta de utilidad para aplicaciones robóticas en general, particularmente para la creación de equipos de robots para distintos tipos de competencias y categorías.

Con estos antecedentes, en el año 2008 se propuso este proyecto de grado para extender los tipos de robots móviles utilizados a omnidireccionales y estudiar mecanismos de control y asignación de comportamiento, así como métodos de disminución de incertidumbre en los datos recibidos de los sensores del mundo físico.

[2] Estructura de soporte definida, mediante la cual otro proyecto de *software* puede ser organizado y desarrollado. Puede incluir soporte de programas, bibliotecas y un lenguaje interpretado por otros programas para ayudar a desarrollar y unir los diferentes componentes de un proyecto.

[3] Acrónimo inglés de *Extensible Markup Language*. Lenguaje extensible de etiquetas desarrollado por el *World Wide Web Consortium* (W3C). Es una simplificación y adaptación del SGML que permite definir la gramática de lenguajes específicos.

[4] Sistemas Multi-Agente (Multi-agent System)

[5] Campeonato Argentino de Fútbol de Robots

1.3. Objetivos del Proyecto

Los objetivos principales de este proyecto fueron:

- Estudio de robots omnidireccionales, involucrandose en este punto la construcción de un robot omnidireccional, el control del mismo y la evaluación de factibilidad y simplicidad de realizar trayectorias complejas en ambientes altamente dinámicos.
- Estudio de técnicas de fusionado de datos sensoriales, de forma de unificar mediciones de distintos tipos de sensores en un único modelo sensorial. Esto implica la aplicación de alguna de estas técnicas en un software, así como la validación de su funcionamiento.
- Desarrollo de un *marco de trabajo* para el modelado de sistemas robóticos, donde usuarios puedan definir mediante una interfaz gráfica, sin necesidad de conocimientos de programación, comportamientos para distintos tipos de robots (incluyendo robots omnidireccionales), pudiendo utilizar distintas técnicas de sensado incluyendo fusión de sensores. Estos sistemas robóticos deben incluir por lo menos los siguientes elementos:
  - Robots y entidades: Los robots definidos deben ser controlados por el propio sistema u otros. Además se definen entidades genéricas, como pueden ser obstáculos, de forma que los robots controlados puedan interactuar con los mismos.
  - Sensores y fusión de sensores: Los sensores y técnicas de fusión de sensores son utilizados para actualizar las posiciones de los robots y entidades definidos en el sistema.
  - Comportamientos: Se pueden definir comportamientos para los robots controlados por el usuario. Éstos son fácilmente configurables utilizando componentes predefinidos y es posible editarlos de forma gráfica.

1.4. Organización del Documento

Capítulo 2 - Marco Teórico Este capítulo presenta un resumen de la investigación realizada al principio del proyecto sobre la cual se desarrolló la solución. Se explica el relevamiento de la estructura física de los robots omnidireccionales y la técnica de fusión de sensores, que originaron la implementación de una aplicación de control y comportamiento de robots omnidireccionales.

Capítulo 3 - Construcción de un robot omnidireccional Este capítulo introduce los aspectos que refieren a la construcción de un robot. Se introducen las principales estructuras de construcción, los tipos de ruedas evaluados y las técnicas de control básico de los robots omnidireccionales.

Capítulo 4 - Solución EasyRobots Este capítulo introduce la aplicación *EasyRobots* desarrollada para controlar robots. Se explica la arquitectura de cada subsistema que compone la aplicación, así como las decisiones de diseño y la extensibilidad del sistema, detallando la extensibilidad de cada subsistema.

Capítulo 5 - Evaluación global Este capítulo explica las pruebas realizadas sobre la aplicación haciendo uso del robot de tres ruedas construido. Se exponen las pruebas planteadas, el desarrollo de éstas y los resultados. Finalmente se analizan dichos resultados considerando los esperados al momento de desarrollar las pruebas.

Capítulo 6 - Conclusiones Este capítulo explica las conclusiones derivadas del desarrollo del proyecto, así como los trabajos a futuro para enriquecer aspectos de la aplicación desarrollada.

Se incluyen apéndices para explicar detalles del desarrollo del proyecto que permiten profundizar ciertos temas expuestos a lo largo del documento.

Finalmente, se listan las referencias utilizadas en el desarrollo del documento así como un glosario conteniendo terminología, acrónimos y definiciones utilizadas en el documento.

2. Marco teórico

Esta sección presenta un resumen del estudio realizado sobre las herramientas y conocimientos que fueron utilizados para el desarrollo del proyecto de grado. Para comprender en mayor detalle los conceptos teóricos aquí expuestos, referirse al documento del Estado del Arte de este proyecto[18].

Debido a la orientación del proyecto de grado, se estudiaron herramientas y metodologías de dos áreas principalmente:

Robots omnidireccionales: Dentro del área de la robótica existen muchos tipos de robots; entre éstos se pueden destacar los robots móviles, que se caracterizan por tener la capacidad de desplazarse físicamente. Asimismo, pueden utilizar distintos sistemas de propulsión, como pueden ser ruedas, piernas mecánicas, entre otros. Ademas, los robots móviles pueden ser omnidireccionales; esto es que poseen la virtud de desplazarse en cualquier dirección en el plano, independientemente de su orientación, permitiéndoles así realizar movimientos omnidireccionales. Se definen movimientos omnidireccionales como aquellos compuestos por un desplazamiento lineal y rotación simultánea de forma de alcanzar un destino con el ángulo deseado[19]. En la figura 1 se pueden visualizar las diferencias entre las trayectorias para ubicar un vehículo no omnidireccional y uno omnidireccional; como puede observarse, el no omnidireccional debe realizar movimientos complejos para esta tarea, mientras un omnidireccional puede dirigirse directamente al objetivo. En esta sección, se estudian los robots omnidireccionales, las distintas estructuras utilizadas y los componentes principales utilizados para su construcción. Así mismo, se consideran los cálculos necesarios para el control cinemático de estos robots.

Algoritmos de fusión de sensores: En esta área se estudian las técnicas de sensado en sistemas robóticos, características de los algoritmos de fusión de sensores (*Sensor Data Fusion*) y principios que deben ser considerados al implementar estos algoritmos.

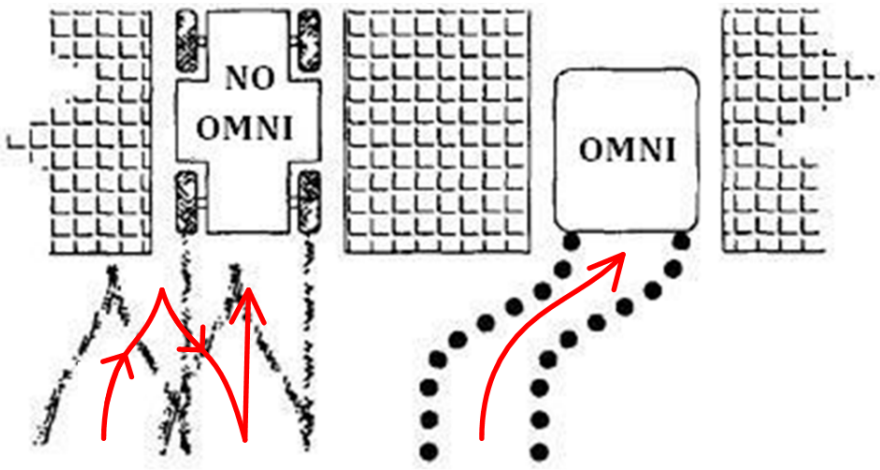

Figura 1: Movimientos no omnidireccionales y omnidireccionales [1].

2.1. Robots omnidireccionales

Para utilizar el sistema implementado, se construyó un robot de manera de evaluar el comportamiento en el mundo físico. Se consideraron las herramientas de construcción a utilizar así como las dimensiones del robot y la comunicación con *EasyRobots*.

En esta sección se detallan las herramientas utilizadas, así como los componentes más importantes para poder realizar las trayectorias omnidireccionales. Asimismo, se describen los componentes más importantes de un robot móvil, como son las ruedas y la estructura mecánica. Además se detallan las ecuaciones necesarias para el control cinemático del robot, brindando un panorama general de la construcción y utilización de robots omnidireccionales.

2.1.1. Ruedas estudiadas

Una de las principales características de los robots omnidireccionales son sus ruedas. Para poder lograr movimientos omnidireccionales, se requieren ruedas de determinados tipos. Los tipos de ruedas estudiadas podrían calificarse en Ruedas Convencionales y Ruedas Especiales. Si bien vehículos construidos utilizando ruedas convencionales permiten realizar movimientos omnidireccionales, se requiere mecánica y lógica adicional para su control, debido a

que se deben posicionar las ruedas en el sentido del desplazamiento. Por otro lado, las Ruedas Especiales permiten realizar los movimientos omnidireccionales de forma directa, es decir sin tiempo de posicionamiento previo, como se detalla en la sección 2.1.2. A continuación se detallan estos tipos de ruedas.

Ruedas Convencionales Dentro de esta categoría se pueden identificar tres tipos tal como se muestra en la figura 2. Como se puede observar, las Ruedas *Castor*, se construyen ubicando el eje vertical del soporte de la rueda desplazado sobre la dirección perpendicular al eje de la rueda. Las Ruedas *Convencionales* simples poseen el eje vertical del soporte alineado con el eje vertical de la rueda, mientras que las ruedas *Convencionales con desplazamiento* poseen el eje vertical desplazado respecto al eje de la rueda.

Estas ruedas no permiten realizar movimientos omnidireccionales, ya que requieren una rotación previa a comenzar el desplazamiento; sin embargo, el tiempo de orientación es pequeño respecto al tiempo total del desplazamiento. Por lo anterior, su movimiento se considera omnidireccional.

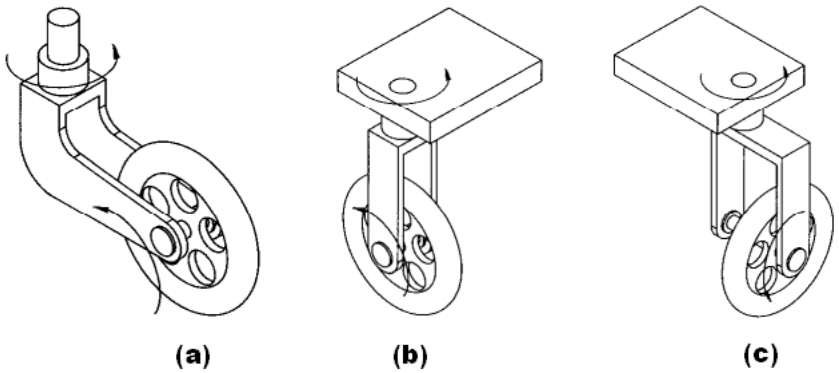

Figura 2: Tipos de ruedas convencionales. (a) Tipo *Castor (Convencional con desplazamiento frontal)*, (b) Tipo *Convencional simple*, (c) Tipo *Convencional con desplazamiento lateral*[2].

Considerando estas ruedas, la estructura de un robot de tres ruedas convencionales (cantidad de ruedas utilizadas en la construcción del robot utilizado, ver la sección 3) es similar a la de la figura 3. Si bien dicha disposición permite realizar movimientos omnidireccionales, posee la desventaja implícita de requerir dos motores por cada rueda generando así una complejidad extra en la construcción del robot.

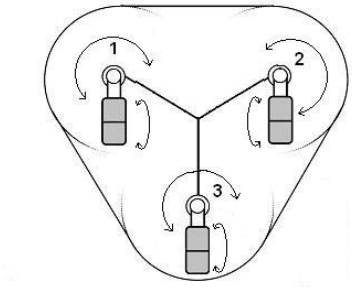

Figura 3: Ejemplo de configuración de ruedas de un robot omnidireccional de tres ruedas convencionales [3].

Ruedas especiales Se basan en la idea de poseer una componente activa que provee tracción en una dirección y una componente pasiva en otra dirección. Entre las ruedas estudiadas, se incluyen las ruedas *Universales*, las ruedas *Mecanum* y ruedas *Esféricas*. A continuación se explican las ruedas de tipo *Universales* que fueron las utilizadas en el robot desarrollado en el proyecto de grado.

Las ruedas especiales incluyen, entre otras, las siguientes ventajas:

1. Simplicidad en el control de movimiento.

2. Simplicidad en el diseño mecánico del robot ya que se requiere solamente un motor por rueda.

Entre las desventajas se encuentran:

1. Capacidad de carga limitada, debido a que el punto de apoyo son los rodillos de las ruedas, ejerciendo fuerza sobre el eje de los rodillos, que son más frágiles que lo que puede ser el eje de una rueda.

2. Radio de la rueda variable, como se puede ver en la figura 4.

3. Alta sensibilidad a desniveles y obstáculos, ya que la altura máxima de un obstáculo sobrepasable es igual al radio de los rodillos. Este caso se da cuando la rueda se desliza lateralmente al enfrentarse al obstáculo[20].

Estas desventajas, sin embargo, no representan un problema a los efectos de este proyecto de grado, debido a que al tratarse de un prototipo, el robot construido no posee un peso considerable (0,8 Kg) (desventaja 1), el radio de la rueda variable se puede mitigar utilizando ruedas dobles desfasadas (desventaja 2) y no se pretendía sortear obstáculos (desventaja 3).

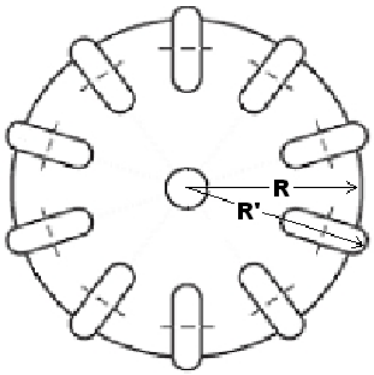

Figura 4: Radio variable en una rueda universal. Se puede observar como el radio R es mayor que el radio R' *[2]*

Las ruedas Universales poseen rodillos pasivos ubicados en la periferia de la rueda principal que brindan una componente pasiva perpendicular a la activa. La figura 5 ejemplifica estas ruedas.

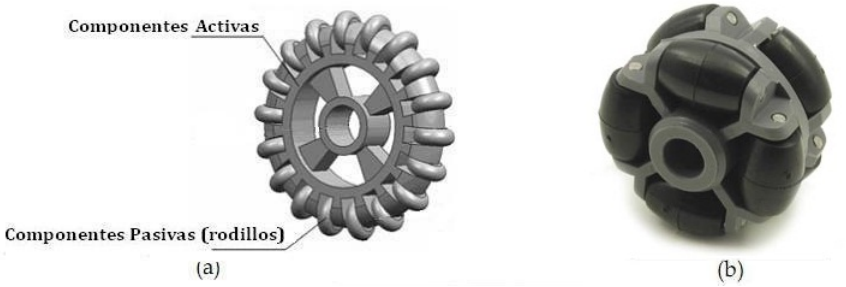

Figura 5: Ruedas Universales. (a) Rueda Simple, (b) Rueda doble[4, 5].

Cuando tres o más de estas ruedas son utilizadas en un robot, sus componentes activos y pasivos combinados permiten realizar movimientos omnidireccionales.

El cuadro 2 detalla las principales diferencias entre los distintos tipos de ruedas especiales estudiadas.

En la figura 6 se puede observar el tipo de rueda *Mecanum* también incluído en la comparación anterior. Como se puede apreciar, este tipo de rueda es construído de forma similar a las Ruedas Universales, con la salvedad que los rodillos no se encuentran perpendiculares a la rueda.

Tipo de Rueda	Característica	Capacidad de Carga	Diseño	Radio de la rueda	Fricción	Sensible al suelo
Rueda Universal		Baja	Simple	Variable	Baja	Si
Rueda *Mecanum*		Baja	Complejo	Constante	Baja	Si
Rueda *Convencional*		Alta	Complejo	Constante	Alta	No
Rueda *Castor*		Alta	Complejo	Constante	Alta	No

Cuadro 1: Tabla comparativa de tipos de ruedas [2].

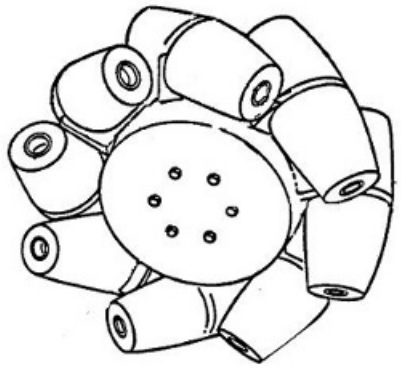

Figura 6: Tipo de Rueda Mecanum, también estudiada. *[6]*

2.1.2. Estructuras de robots omnidireccionales

Una decisión importante a ser considerada al construir un robot es la cantidad de ruedas a utilizar. Las decisiones más comunes a este problema son utilizar dos, tres o cuatro ruedas, presentando cada una de éstas ventajas y desventajas: los vehículos de dos ruedas tienen la ventaja de que el control es simple, pero se ven reducidos en maniobrabilidad (no pueden ser omnidireccionales salvo que se utilicen dos motores por rueda, introduciendo complejidad en el diseño). Los vehículos de tres ruedas tienen control y dirección simple, pero estabilidad y tracción limitada [21]. Los de cuatro ruedas tienen mayor tracción y estabilidad, pero mecánica y controles más complejos.

A continuación se presenta la estructura de un vehículo omnidireccional de tres ruedas, que corresponde a la cantidad utilizada en el proyecto. La justificación de esta elección se detalla en la sección 3.2. Se estudiaron además otras estructuras como son robots de cuatro ruedas dirigibles, ruedas *Castor* y genéricos de K ($K \geq 3$) ruedas omnidireccionales o *Mecanum*.

La estructura de tres ruedas consiste en tres ruedas omnidireccionales dispuestas como se muestra en la figura 7. Ésta consiste en tres ruedas universales colocadas en los vértices de un triángulo equilátero. Como ejemplo, en la figura se puede observar que para que el robot se desplace a lo largo del vector *1*, la rueda A debe rotar de forma de generar una componente lineal en el sentido del vector *2*. De forma análoga, la rueda C debe rotar generando una componente como el vector *3*. Los componentes verticales de estos vectores se anulan, generando un desplazamiento en el sentido del vector buscado. Cabe destacar que la rueda B se desliza a lo largo de sus rodillos, debido al uso de ruedas especiales.

Este tipo de estructuras posee la ventaja de que su control es simple, ya que para cada velocidad deseada del vehículo existe una única combinación en las velocidades de las ruedas (a diferencia de los vehículos omnidireccionales de más de tres ruedas).

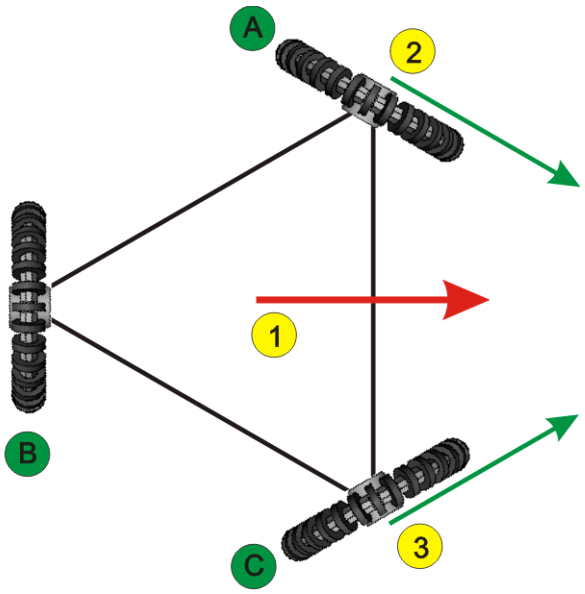

Figura 7: Estructura del robot omnidireccional de 3 ruedas

2.1.3. Control cinemático de robots omnidireccionales

A continuación se explica el control cinemático de robots de tres ruedas que es utilizado para controlar el robot desarrollado. Así mismo se introduce el control cinemático de robots con un número genérico de ruedas (mayor a tres) permitiendo deducir el control cinemático de otros robots.

Se representan las velocidades del robot deseadas como el vector $\{\dot{x}, \dot{y}, \dot{\theta}\}$ en coordenadas cartesianas, luego se describe como llegar a la configuración de las ruedas $\left\{\dot{\phi_1}, \dot{\phi_2}, ..., \dot{\phi_n}\right\}$, determinando sus velocidades angulares.

Control cinemático de robots omnidireccionales de tres ruedas El esquema de la disposición de las ruedas del robot se puede ver en la figura 8.

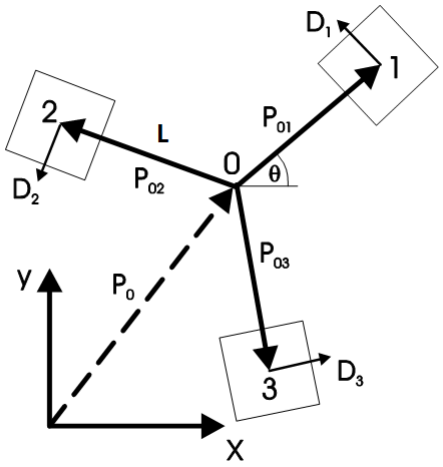

Figura 8: Diagrama de cinemática del Robot[7]

La posición de las ruedas tomando como origen el centro de masa del robot O y utilizando como eje $\overrightarrow{OX}$ el vector $\overrightarrow{P_{01}}$, puede obtenerse fácilmente con la ayuda de la matriz de rotación (θ es el ángulo de giro medido en sentido antihorario)

$$R(\theta) = \begin{pmatrix} cos(\theta) & -sin(\theta) \\ sin(\theta) & cos(\theta) \end{pmatrix} \quad (1)$$

por lo tanto:

$$P_{01} = L \begin{pmatrix} 1 \\ 0 \end{pmatrix} \quad P_{02} = R\left(\frac{2\pi}{3}\right) \quad P_{01} = \frac{L}{2} \begin{pmatrix} -1 \\ \sqrt{3} \end{pmatrix} \tag{2}$$

$$P_{03} = R\left(\frac{4\pi}{3}\right) \quad P_{01} = \frac{L}{2} \begin{pmatrix} 1 \\ \sqrt{3} \end{pmatrix} \tag{3}$$

donde L es la distancia desde el centro del robot a las ruedas. Los vectores unitarios D_i determinan la dirección de avance de la rueda i. Estas se determinan por (tomando en cuenta el mismo sistema de coordenadas usado anteriormente)

$$D_i = \frac{1}{L} R\left(\frac{\pi}{2}\right) P_{0i} \tag{4}$$

$$D_1 = \begin{pmatrix} 0 \\ 1 \end{pmatrix} \quad D_2 = \frac{1}{2} \begin{pmatrix} \sqrt{3} \\ 1 \end{pmatrix} \quad D_3 = \frac{1}{2} \begin{pmatrix} \sqrt{3} \\ -1 \end{pmatrix} \tag{5}$$

El vector $P_0 = \begin{pmatrix} x & y \end{pmatrix}^T$ es la posición del centro de masa en el sistema de coordenadas visto en la figura 8. La posición de cada rueda y su velocidad en este sistema está determinada por

$$r_i = P_0 + R(\theta) P_{0i} \tag{6}$$

$$\mathrm{v}_i = \dot{P}_0 + \dot{R}(\theta) P_{oi} \tag{7}$$

mientras que las velocidad individual de cada rueda (componente de velocidad de la rueda en el sentido del vector D_i) es

$$v_i = \mathrm{v}_i^T (R(\theta) D_i) \tag{8}$$

Sustituyendo la ecuación (7) en la ecuación (8) resulta en

$$v_i = \dot{P_0^T} R(\theta) D_i + P_{oi}^T \dot{R}^T(\theta) R(\theta) D_i \tag{9}$$

El segundo término de la ecuación es simplemente la velocidad tangencial del robot y se puede escribir de la siguiente forma:

$$P_{oi}^T \dot{R}^T(\theta) R(\theta) D_i = L\dot{\theta} \tag{10}$$

Por lo tanto la velocidad de las ruedas es una función lineal de la velocidad y la velocidad angular del robot es

$$\begin{pmatrix} v_1 \\ v_2 \\ v_3 \end{pmatrix} = \begin{pmatrix} -\sin(\theta) & \cos(\theta) & L \\ -\sin(\frac{\pi}{3} - \theta) & -\cos(\frac{\pi}{3} - \theta) & L \\ \sin(\frac{\pi}{3} + \theta) & -\cos(\frac{\pi}{3} + \theta) & L \end{pmatrix} \begin{pmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \end{pmatrix} \tag{11}$$

Finalmente partiendo de la ecuación (11) y sabiendo que la velocidad angular de una rueda es $\phi_i = \frac{v_i}{r}$ (siendo r el radio de las ruedas) , para lograr un movimiento en coordenadas cartesianas a partir de $\{\dot{x}, \dot{y}, \dot{\theta}\}$, se obtienen las velocidades angulares de las ruedas mediante la siguiente transformación lineal:

$$\begin{pmatrix} \dot{\phi}_1 \\ \dot{\phi}_2 \\ \dot{\phi}_3 \end{pmatrix} = \frac{1}{r} \begin{pmatrix} -\sin(\theta) & \cos(\theta) & L \\ -\sin(\frac{\pi}{3} - \theta) & -\cos(\frac{\pi}{3} - \theta) & L \\ \sin(\frac{\pi}{3} + \theta) & -\cos(\frac{\pi}{3} + \theta) & L \end{pmatrix} \begin{pmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \end{pmatrix} \tag{12}$$

Control cinemático de robots omnidireccionales de K ruedas Las diferentes estructuras de vehículos omnidireccionales construidos con una cantidad (K) de ruedas, se ven dotados de diversos rendimientos cinemáticos al desplazarse en distintas direcciones, efecto conocido como anisotropía[4]. A mayor cantidad de ruedas, disminuye la vibración y aumenta la capacidad de maniobrabilidad. También, con el incremento de la cantidad de ruedas devienen desventajas. Por ejemplo, si $K \geq 4$, se necesita un mecanismo elástico para mantener las ruedas en contacto con la superficie, si ésta no es completamente plana. Es importante por lo tanto construir un modelo cinemático para diseñar un robot con buen desempeño.

En esta sección se tratan las ruedas Mecanum en el sentido más amplio. En la figura 9 se pueden ver el eje de la rueda activa (S_i) y el eje de las ruedas pasivas (E_i). Claramente estos ejes siempre se intersectan definiendo el ángulo α ($\alpha \neq 0$), lo que implica que no son paralelos. En el ejemplo de la figura α vale $\frac{\pi}{2}$.

Para analizar el diseño general hay que definir una serie de parámetros que determinen la configuración de las ruedas.

Supóngase que el vehículo esta constituido por K ruedas distribuidas alrededor del mismo. La figura 10 muestra los parámetros relativos a la rueda $i$ésima. Lsa direcciones de rotación de las ruedas activas y pasivas son S_i y E_i, y la dirección de la velocidad de traslación es T_i y F_i. O_i es el centro de la rueda y su velocidad es V_{0_i}. C es el centro del robot, y su velocidad es (c, ω), donde ω es la velocidad angular y c es la velocidad de traslación en la dirección θ.

Sin tomar en cuenta factores de performance, las ruedas pueden ser fijadas al robot en posiciones arbitrarias, lo que significa que se puede asignar cualquier valor a los parámetros de la figura 10. $d_i$denota el vector desde el punto C a O_i. β denota el ángulo entre el vector d_i y el eje X. γ_i denota el ángulo entre S_i y el eje X. Al determinar el valor de los parámetros mencionados anteriormente, se está determinando la configuración de las ruedas del vehículo.

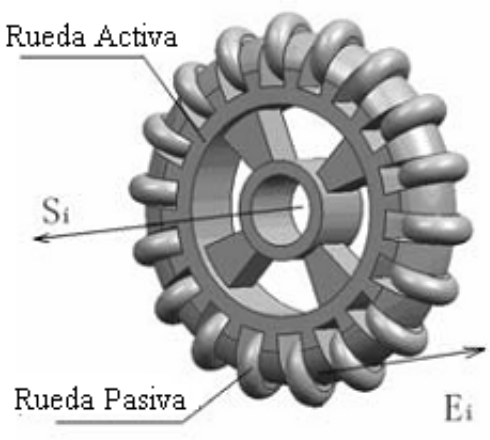

Figura 9: Esquema de rueda omnidireccional[4]

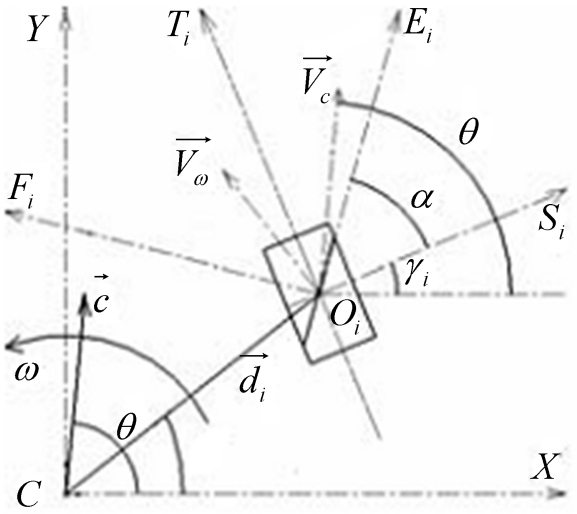

Figura 10: Parámetros de rueda i[4]

La velocidad del centro de la rueda O_i se determina por la velocidad de la rueda activa y el rodillo pasivo (ver ecuación (13)). Tomando en cuenta el vehículo como un cuerpo rígido, se puede obtener la velocidad O_i (ver ecuación (14)). Partiendo de (13) y (14), se puede deducir (15).

$$V_{0_i} = V_{T_i} + V_{F_i} \tag{13}$$

$$V_{0_i} = V_c + V_\omega \text{ donde } V_c = c \text{ , } V_\omega = \omega \times d_i \tag{14}$$

$$V_{T_i} + V_{F_i} = V_c + V_\omega \tag{15}$$

Proyectando V_c, V_ω, V_{T_i} y V_{F_i} sobre los ejes X e Y, se puede obtener la relación de la velocidad por T_i y F_i y la velocidad (c, ω) en (16) y (17). Donde V_{xc} denota la proyección de c al eje X, y de la misma manera se definen $V_{x\omega}$, V_{xT}, V_{xF}, V_{yc}, $V_{y\omega}$, V_{yT} y V_{yF}.

$$\begin{cases} V_{xc} + V_{x\omega} = V_c\cos(\theta) - V_\omega\sin(\beta) \\ V_{yc} + V_{y\omega} = V_c\sin(\theta) + V_\omega\cos(\beta) \\ V_{xT} + V_{xF} = -V_T\sin(\gamma_i) - V_F\sin(\alpha+\gamma_i) \\ V_{yT} + V_{yF} = V_T\cos(\gamma_i) + V_F\cos(\alpha+\gamma_i) \end{cases} \tag{16}$$

$$\begin{cases} V_T = V_c\cos(\alpha+\gamma_i-\theta)/\sin\alpha + V_\omega\sin(\alpha+\gamma_i-\beta) \\ V_F = V_c\cos(\theta-\gamma_i)/\sin\alpha + V_\omega\sin(\gamma_i-\beta) \end{cases} \tag{17}$$

Debido a que los rodillos pasivos no se controlan por medio de ningún motor, se puede ignorar a V_F durante el análisis de cinemática. Por lo tanto (18) es una ecuación general de cinemática del vehículo omnidireccional, donde V_{T_i} es la velocidad de la i-ésima rueda a lo largo de T_i. Obviamente, una vez obtenido los parámetros de control de cada rueda, se determina (c, ω). De forma inversa, si (c, ω) son conocidos, se puede deducir la velocidad angular de cada rueda.

$$V_{T_i} = V_c\cos(\alpha+\gamma_i-\theta)/\sin\alpha + \omega d_i\sin(\alpha+\gamma_i-\beta) \tag{18}$$

Donde $V_{T_i} = \omega' r$, ω' es la velocidad angular de la $i$ésima rueda, r es el radio de la rueda; V_c ($V_c = |c|$) y ω son las velocidades de traslación y angular del vehículo respectivamente.

Este cálculo anterior sirve para cuando el sistema de visión está centrado en el robot. Sin embargo, si se desea controlar más robots, generalmente se opta por tener una visión global del sistema, y controlar a los robots de una forma general. A continuación se deriva la ecuación 18 para un sistema con visión global.

En la figura 11 se detallan los valores a tener en cuenta para el cálculo de velocidades de las ruedas.

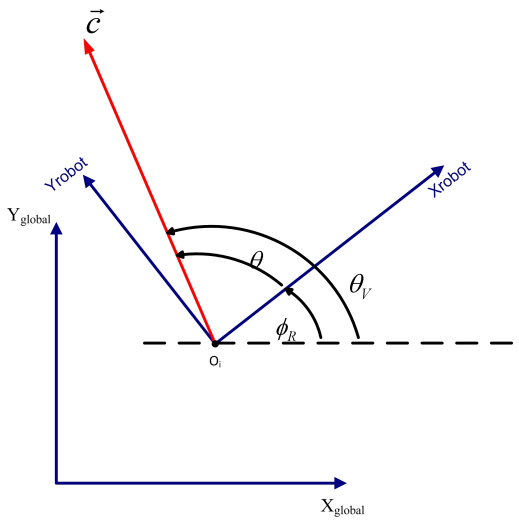

Figura 11: Ubicación del robot en un sistema de posicionamiento global

En el nuevo sistema de coordenadas, el ángulo θ (ángulo de la velocidad respecto al robot) se convierte en θ_V mediante la siguiente transformación

$$\theta_V = \theta - \phi_R \tag{19}$$

donde ϕ_R es la rotación del robot, como se puede ver en la figura 11.
Desarrollando el término $\cos(\alpha + \gamma_i - \theta)$ de la ecuación 18, se obtienen los siguientes términos:

$$\begin{aligned} \cos(\alpha + \gamma_i - \theta) &= \cos(\alpha + \gamma_i - \theta_V + \phi_R) \\ &= \cos(\alpha + \gamma_i + \phi_R)\cos(\theta_V) + \sin(\alpha + \gamma_i + \phi_R)\sin(\theta_V) \end{aligned}$$

Sustituyendo el término desarrollado en la ecuación 18 se obtiene la siguiente ecuación

$$V_{T_i} = V_c \cos(\theta_V)\frac{\cos(\alpha + \gamma_i + \phi_R)}{\sin\alpha} + V_c \sin(\theta_V)\frac{\sin(\alpha + \gamma_i + \phi_R)}{\sin\alpha} + \omega d_i \sin(\alpha + \gamma_i - \beta) \tag{20}$$

Sabiendo que

$$\dot{x} = V_c \cos(\theta_V) \tag{21}$$

y

$$\dot{y} = V_c \sin(\theta_V) \tag{22}$$

la ecuación 20 que define la velocidad en cada rueda queda simplificada como se describe a continuación, quedando definida la matriz general de transformación de velocidad en el plano a velocidad angular de las ruedas del robot, para robots omnidireccionales de K ruedas.

$$V_{T_i} = \dot{x}.\left[\frac{\cos(\alpha + \gamma_i + \phi_R)}{\sin\alpha}\right] + \dot{y}.\left[\frac{\sin(\alpha + \gamma_i + \phi_R)}{\sin\alpha}\right] + \omega\left[d_i \sin(\alpha + \gamma_i - \beta)\right] \tag{23}$$

Aplicación de matriz general de transformación a velocidad angular de ruedas Como ejemplo de aplicación de la matriz desarrollada en la sección anterior, se utiliza un robot de 3 ruedas separadas $\frac{2\pi}{3}$ radianes entre si, como se muestra en la figura 12. En este ejemplo, las ruedas utilizadas utilizan el ángulo $\alpha = \frac{\pi}{2}$, y los ángulos $\gamma_i = 0$, $\frac{2\pi}{3}$ y $\frac{4\pi}{3}$ respectivamente, para las ruedas $i = 1, 2\, y\, 3$. Además, los ángulos $\beta_i = 0$, $\frac{2\pi}{3}$ y $\frac{4\pi}{3}$ están definidos por la posición de los ejes de las ruedas.

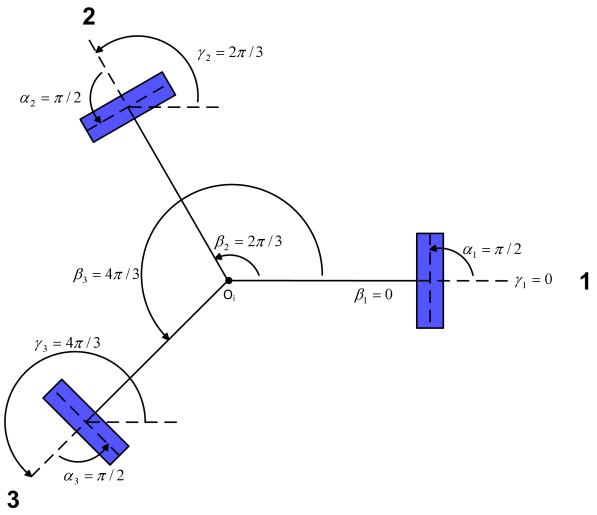

Figura 12: Robot omnidireccional de 3 ruedas

La matriz resultado de transformación de velocidad en el plano a velocidades angulares de las ruedas se detalla a continuación:

$$\begin{pmatrix} \dot{\phi}_1 \\ \dot{\phi}_2 \\ \dot{\phi}_3 \end{pmatrix} = \begin{pmatrix} -\sin(\phi_R) & \cos(\phi_R) & d_i \\ -\sin(\frac{\pi}{3} - \phi_R) & -\cos(\frac{\pi}{3} - \phi_R) & d_i \\ \sin(\frac{\pi}{3} + \phi_R) & -\cos(\frac{\pi}{3} + \phi_R) & d_i \end{pmatrix} \begin{pmatrix} \dot{x} \\ \dot{y} \\ \dot{\omega} \end{pmatrix} \tag{24}$$

A modo de verificación, se puede notar que la ecuación 24 coincide con la derivada anteriormente para tres ruedas (ecuación 12).

2.2. Fusión de sensores

En esta subsección, se introduce el concepto fusión de sensores y la forma general de un *marco de trabajo* de implementación de dicho concepto.

Uno de los principales problemas de los robots móviles es el posicionamiento. Para muchas aplicaciones, se necesita conocer la posición del robot y la orientación en todo momento. Por ejemplo, un robot de limpieza necesita asegurarse de cubrir toda la superficie sin repetir zonas o incluso perderse. Otro ejemplo podría ser un robot encargado de realizar las entregas en una oficina, el cual necesita poder navegar a través de un edificio y además conocer su posición y orientación relativa a su punto de comienzo. Para conocer las posiciones de los objetos, los sistemas disponen de un conjunto de sensores que registran los estados del mundo físico. La técnica de Fusión de Sensores encuentra su principal motivación en los errores implícitos en las mediciones de los sensores, como pueden ser ruido e incertidumbre.

2.2.1. Concepto y aplicación de fusión de sensores

La fusión de sensores es un término amplio que se utiliza para cualquier proceso que involucre combinar información de múltiples sensores en una única medición. Existen tres estrategias básicas para combinar sensores: *redundante* (sensores que compiten), *complementario* y *coordinado*.

En algunos casos se utilizan múltiples sensores cuando un sensor en particular es muy impreciso o ruidoso como para dar datos confiables. Al agregar un segundo sensor, se puede obtener otro "voto" para una cierta percepción.

Cuando dos sensores retornan una misma percepción, éstos son considerados *redundantes*. Un ejemplo de *redundancia física* se puede ver en la figura 13 donde el robot tiene dos anillos de sonares. El sistema retorna la menor distancia sensada de los dos, proveyendo una lectura más confiable para objetos bajos para los que el anillo superior normalmente no sensaría debido a la reflexión especular de las ondas. Los sensores pueden tener además redundancia lógica, cuando sensan el mismo objetivo, pero se utilizan distintos algoritmos para procesar los datos sensados. Un ejemplo de esto es cuando se extrae la distancia a un objeto utilizando un telémetro óptico[6] y un

[6] Un telémetro óptico consta de 2 objetivos separados una distancia conocida. Con ellos se apunta un objeto y la distancia se puede calcular mediante el uso de trigonometría

Figura 13: Ejemplo de anillos de sonares redundantes[8]

telémetro láser[7]. Los sensores redundantes se pueden denominar *competitivo*, ya que pueden ser vistos como que compiten para publicar la percepción "ganadora".

Los sensores *complementarios* proveen tipos disjuntos de información sobre un objeto.

Los sensores *coordinados* utilizan una secuencia de sensores, usualmente de forma de proveer un foco de atención. Por ejemplo, un depredador detecta movimiento, causando que éste pare para examinar con detención la escena en busca de señales de una presa[8].

Actualmente se le han dado dos usos distintos a la fusión de sensores. Por un lado se puede utilizar para generar un modelo del mundo con mayor precisión (este puede ser a su vez numérico ó simbólico), o puede ser incorporado a los comportamientos a través de *Fisión de Sensores*, *Orientado a Acciones* y *Sensado en Secuencia*. Para más información sobre estos conceptos, consultar el estado del arte [18] y el documento [8].

A continuación se describe la fusión de sensores orientada al modelado del mundo, utilizado para la implementación del módulo de fusión de sensores en este proyecto.

2.2.2. Fusión de sensores orientado al modelado del mundo

En esta sección se describe un *marco de trabajo* general para el modelo dinámico del mundo desarrollado por Crowley[9].

El modelado dinámico del mundo es un proceso iterativo que implica convertir las observaciones físicas en una descripción interna. Este proceso se compone de tareas generales, que permiten la creación de un *marco de trabajo* (ó *framework*) para el modelado del mundo, al cual pueden incorporarse técnicas de fusión de sensores.

El *marco de trabajo* presentado puede aplicarse a fusión de datos numéricos (como pueden ser posiciones en ejes cartesianos o distancias) y para fusión de información simbólica (descripción del mundo en términos de objetos, relaciones y eventos). En el caso de la fusión de datos numéricos se pueden utilizar técnicas de la teoría de la estimación[8].

Marco de trabajo general para el modelado dinámico del mundo La fusión de mediciones de los sensores implica utilizar algoritmos más complejos a diferencia de cuando se utiliza un único sensor. Primero, porque los ambientes con varios sensores incluídos brindan mediciones de diferentes tipos y por lo tanto se necesita un procesamiento sobre éstas para uniformizar las unidades. Segundo, porque se necesitan estrategias para combinar las mediciones. Estos métodos pueden variar desde una simple intersección de valores, operaciones lógicas hasta métodos más complejos como fusión mediante mínimos cuadrados no lineales, estimaciones mediante probabilidades, etc. A pesar de lo anterior, la fusión de sensores provee redundancia en los datos obtenidos por lo que se toleran de mejor forma las posibles fallas de los sensores y permiten recuperar mediciones perdidas[22].

[7] El telémetro láser es capaz de realizar medidas de distancia como el ultrasónico

[8] Rama de la estadística que se encarga de la estimación de parámetros basado en datos empíricos.

En la figura 14 se ilustra un *marco de trabajo* para el modelado dinámico del mundo. En éste, observaciones independientes son "transformadas" en un espacio de coordenadas y vocabulario común. Posteriormente, estas observaciones son integradas (fusionadas) en un modelo (descripción interna) por un proceso cíclico compuesto de tres fases: *Predicción*, *Correspondencia* y *Actualización*.

Predicción: en esta etapa, se utiliza el estado actual del modelo para predecir el estado del mundo en el momento que se realiza la siguiente medición.

Correspondencia: etapa en la que se corresponden las mediciones transformadas con las estimaciones. Para corresponder las predicciones y las últimas mediciones se debe contar con datos cualitativamente similares; para esto existe la etapa *Transformación*, para llevar las mediciones al mismo espacio de coordenadas y a un vocabulario común.

Actualización: la etapa de actualización sirve para agregar nueva información al modelo, así como quitar la información "antigua". Durante esta fase, son eliminadas del modelo la información que no es el "foco de atención" del sistema, así como la información que se entiende como errónea. Este proceso de eliminar los datos antiguos es necesario para prevenir que el modelo interno crezca de forma no acotada.

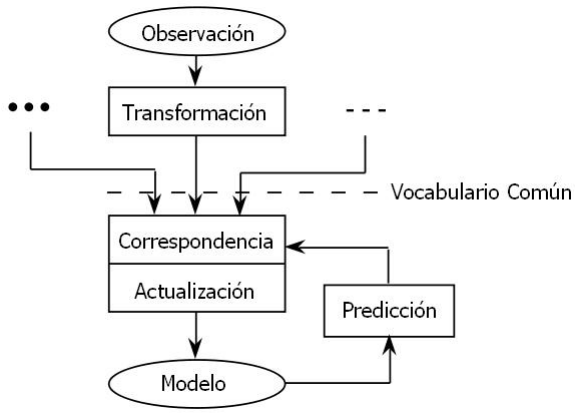

Figura 14: Un modelo para el modelado dinámico del ambiente[9].

En [9] se identifican un conjunto de principios para integrar información perceptual, basados en la experiencia de construcción de sistemas utilizando este *marco de trabajo*. Estos principios se derivan directamente de la naturaleza cíclica del proceso del modelado dinámico del mundo. A continuación se describen en detalle estos principios.

Principios para la fusión de sensores

Principio 1) *Las primitivas del modelo del mundo deben ser expresadas como un conjunto de propiedades.* Una primitiva del modelo expresa una asociación de propiedades que describen el estado de alguna parte del mundo. Esta asociación está basada típicamente en la posición espacial. Por ejemplo, un color amarillo o cierta temperatura. Para valores numéricos, cada propiedad puede ser el valor de una estimación adjunta a su precisión. Para entidades simbólicas, la propiedad puede ser una lista de posibles valores de un vocabulario finito. Esta asociación de propiedades es conocida como el "*vector de estado*" en la teoría de la estimación.

Principio 2) *Las observaciones y el modelo deberían ser expresadas en el mismo sistema de coordenadas.* Para poder corresponder una observación con el modelo, la observación debe poder ser registrada con el modelo. Esto típicamente involucra transformar las observaciones sensadas, implicando tener un modelo confiable de la geometría y función del sensor.
El sistema de coordenadas común puede ser fijo en la escena o en el observador. La elección del marco de referencia debe determinarse considerando el costo total de las transformaciones involucradas en cada ciclo. Por ejemplo, en el caso de un único observador estacionario (como puede ser una cámara global), es más eficiente transformar el modelo a las coordenadas del sensor, en vez de tener que transformar todas las mediciones a cordenadas relativas a los robots. Este caso se da por ejemplo en una competencia de fútbol robótico, en la cual el eje de coordenadas se encuentra fijo en la cancha.

Las transformaciones entre marcos de coordenadas generalmente requieren un modelo preciso de los sensores. Esta descripción, comúnmente llamada "modelo sensorial", es esencial para transformar una medición para ser incluida en el modelo. Determinar y mantener los parámetros de este "modelo sensorial" es un problema importante a tratar para un correcto funcionamiento de la fusión de sensores.

Principio 3) *Las observaciones y el modelo deberían ser expresadas utilizando un vocabulario común.* Un modelo perceptual puede ser pensado como una base de datos. Cada elemento de la base de datos es una colección de propiedades asociadas. Si se desea corresponder o agregar información al modelo, una observación debe ser transformada en términos de la base de datos, de forma de poder almacenarlas correctamente.
Una forma eficiente de integrar información de diferentes sensores es definir primitivas, compuesto por un conjunto de propiedades (por ejemplo, el conjunto de propiedades x, y y z definen una primitiva para el posicionamiento en 3 dimensiones). Un sensor puede brindar observaciones para un subconjunto de estas propiedades (para el ejemplo anterior, una cámara que brinda datos en x e y, dejando indefinida la propiedad z). Transformar las observaciones a un vocabulario común permite que el proceso de fusión ocurra independientemente de las fuentes de observaciones.

Principio 4) *Las propiedades deben incluir una representación explícita de la incertidumbre.* El modelado del mundo involucra dos tipos de incertidumbre: precisión y confianza. La precisión puede pensarse como una forma de incertidumbre espacial.

Principio 5) *Las primitivas deben ser acompañadas por un factor de confianza.* Las primitivas del modelo nunca son certeras; siempre deben ser consideradas como hipótesis. En aras de tener un mejor manejo de estas hipótesis, cada primitiva debería incluir un estimativo de la probabilidad de su existencia. Esto puede implementarse utilizando un factor de confianza en el rango $[-1, 1]$, una probabilidad, o incluso un conjunto de estados (por ejemplo "*existe*", "*no existe*").
Un factor de confianza provee al modelado un comportamiento dinámico (no se procesan todas las observaciones de igual forma). Como ejemplo de aplicación de lo anterior, las observaciones que no se corresponden con las expectativas pueden ser consideradas inicialmente como inciertas. Si se recibe una confirmación de una observación posterior, su factor de confianza se incrementa. Si no se recibe confirmación, la observación puede ser descartada.

2.2.3. Fusión de sensores orientado a comportamiento

En los primeros desarrollos de sistemas reactivos se utilizaron robots con una poca cantidad de sensores sencillos (por ejemplo un sonar o un anillo de sonares para distancias y una cámara para color, textura y ayuda al movimiento), lo que llevó a una filosofía de *un sensor por comportamiento*. Los comportamientos podían compartir flujos de sensores, pero sin estar enterados de ello. Esta filosofía llevo a un enfoque de fusión de sensores, en que la fusión a nivel de comportamiento sería una fantasía. Por el contrario, la fusión de sensores es básicamente la combinación de distintas instancias del mismo comportamiento, cada uno utilizando distintos sensores. De hecho, este proceso es una simple competencia entre los distintos comportamientos generados por las distintas instancias. A esta fusión de comportamientos se le denomina *Fisión de Sensores*. Este nombre surge de conceptos de física nuclear: la fusión en la cual se crea energía al fusionar dos núcleos atómicos, mientras que en la fisión, se crea energía separando el núcleo atómico en dos. La figura 15 muestra un diagrama de *Fisión de Sensores* (en inglés, *Sensor Fission*).

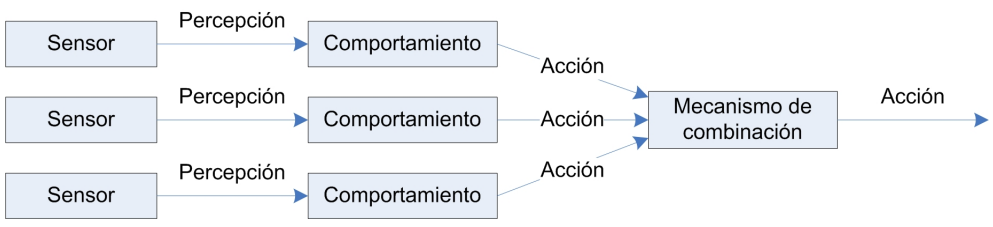

Figura 15: Diagrama de flujo de datos de Fisión de Sensores[8]

Otro tipo de fusión de sensores, que se puede observar por ejemplo en los animales, es *el Orientado a Acciones.* Por ejemplo, si un gato oye un sonido y ve un movimiento, reaccionará con mayor firmeza que si percibe un solo estímulo. Este tipo de fusión recibe su nombre para enfatizar que los datos de sensores son transformados de forma

de soportar una determinada acción específica y no para construir un modelo del mundo. En la figura 16 se muestra un diagrama un diagrama del flujo de datos de este tipo de fusión.

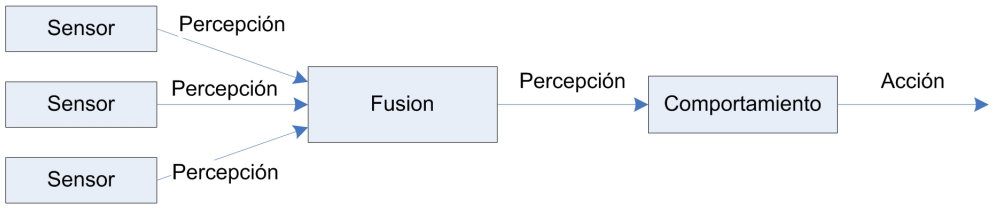

Figura 16: Diagrama de flujo de datos de *Fusión de Sensores*[8]

El último tipo de fusión de sensores que se presenta es el denominado *Sensado en Secuencia* (en inglés, *Sensor Fashion*). Este tipo de fusión intenta transmitir con su nombre que el robot cambia los sensores ante cambios de circunstancias. Por ejemplo, cuando el robot utiliza una cámara de video para su visión y durante su desplazamiento se encuentra con una zona de oscuridad, puede alternar el uso de la cámara a otro tipo de sensor como puede ser de ultrasonido. El diagrama de flujo de datos del *Sensado en Secuencia* se puede observa en la figura 17.

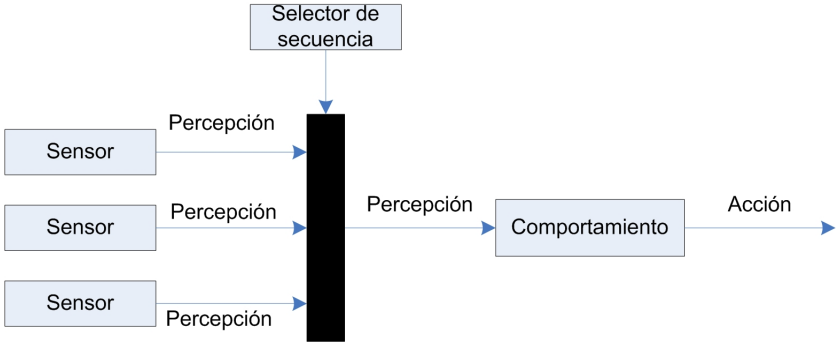

Figura 17: Diagrama de flujo de datos del Sensado en Secuencia[8]

La Fisión de Sensores, Fusión de Sensores Orientado a Acciones y el Sensado en Secuencia cubren los tipos de sensado *competitivo*, *complementario* y *coordinado* respectivamente.

3. Construcción de un robot omnidireccional

Esta sección detalla la construcción de un robot omnidireccional a fin de comprobar las ventajas y desventajas de este tipo de robots frente a vehículos no omnidireccionales, así como para poder evaluar su uso en ambientes reducidos y dinámicos como puede ser un depósito. Se describen los aspectos que motivaron a la construcción del robot, la estructura principal considerada, el estudio que se realizó sobre las principales ruedas disponibles en el mercado para seleccionar aquellas que se utilizarían y finalmente se describen los cálculos necesarios para el control del robot construido.

3.1. Desafíos y motivación

Para utilizar el sistema *EasyRobots* detallado en la sección 4 y para evaluar el uso de robots omnidireccionales, se requiere de la construcción de un robot, de forma de poder estudiar el comportamiento en el mundo físico. Para ello se deben considerar las herramientas de construcción a utilizar así como las dimensiones del robot y la comunicación con el sistema. Estos aspectos fueron considerados de la forma más general posible, ya que el fin del proyecto es la implementación del sistema de control y planificación de trayectorias para robots omnidireccionales, no siendo el aspecto más importante la construcción del robot.

3.2. Estructura principal

En esta subsección se presenta la estructura del robot omnidireccional diseñado para realizar pruebas con el fin de validar las ventajas y desventajas de robots omnidireccionales en general.

Para el diseño de este robot se tuvieron en cuenta todas las estructuras estudiadas, descritas en la subsección 2.1.2, como son los robots de tres ó más ruedas convencionales y especiales. Se consideraron las características del *kit Lego Mindstorms NXT* [16] en lo referente a motores y piezas, con el cual se pueden controlar hasta tres motores. Este *kit* fue utilizado debido a que se encontraba a disposición provisto por el grupo MINA[23], permitiendo abstraerse de la electrónica de la construcción, de forma de enfocarse en el desarrollo del control y el *software*. Al utilizar este *kit*, es posible desarrollar un robot de tres ruedas omnidireccional, permitiendo el estudio de estos robots mediante un prototipo.

Para la estructura principal del robot se utilizaron tres ruedas omnidireccionales con los rodillos dispuestos a 90^o respecto de la rueda principal. Las ventajas de utilizar esta estructura son que posee una mecánica (un único motor por rueda) y control simplificados. Como desventajas, se encuentran que la estructura posee menor estabilidad y capacidad de carga que robots de más ruedas, sin embargo se descartaron al no influir en las pruebas diseñadas.

Estas consideraciones determinaron el diseño del robot (como se muestra en la figura 18), considerándolo suficientemente general para evaluar ventajas y desventajas de los robots omnidireccionales sin perder generalidad. Partiendo de esto, el próximo paso fue evaluar las distintas opciones de ruedas, tema tratado en la sección 3.3.

El resultado de utilizar la estructura y las ruedas descritas se puede apreciar en la figura 19.

Luego de la implementación física se utilizó la aplicación *Lego Digital Designer* [10] para realizar un modelo digital del robot. Esta aplicación fácilmente crea un manual de construcción en HTML[9], el cual se puede ver en el sitio *web* del proyecto [24] así como en el manual de usuario o en el documento de diseño mecánico del robot construído. En la figura 20 se muestran distintas vistas del modelo. Cabe destacar que por las limitaciones del software utilizado, el modelo fue realizado con ruedas convencionales, ya que no existía un modelo de las ruedas omnidireccionales utilizadas para el robot.

[9]Siglas de *HyperText Markup Language* (Lenguaje de Marcas de Hipertexto), es el lenguaje de marcado predominante para la construcción de páginas web. Es utilizado para describir la estructura y el contenido en forma de texto, así como para complementar el texto con objetos tales como imágenes.

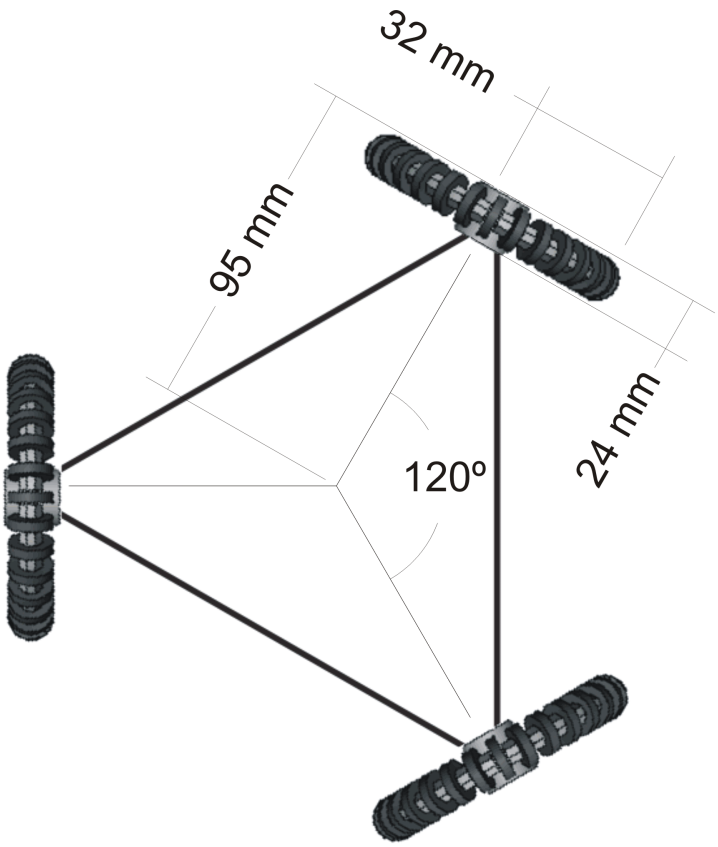

Figura 18: Disposición de las ruedas en el robot.

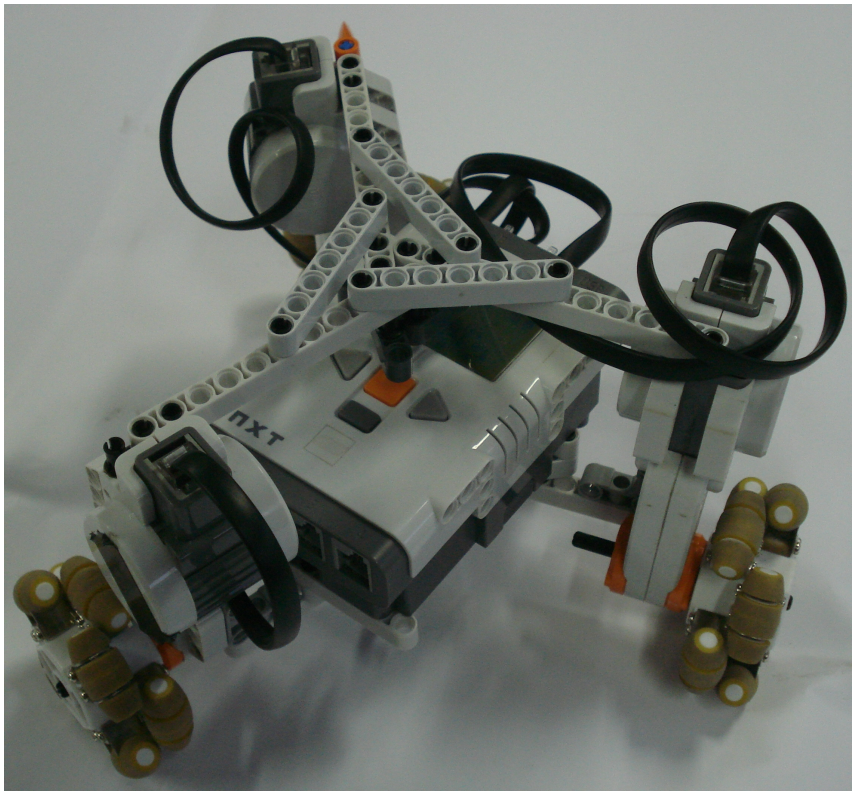

Figura 19: Robot creado con *kit Lego Mindstorms NXT*.

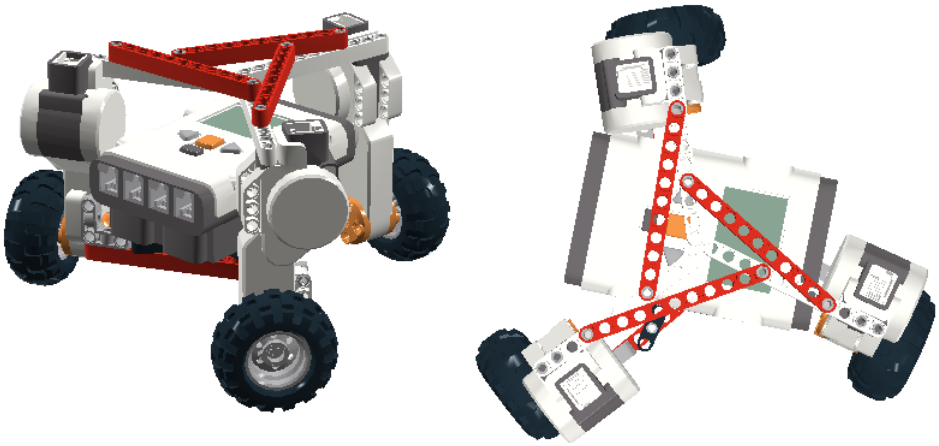

Figura 20: Robot creado con Lego Digital Designer[10]. Vista en perspectiva (izquierda) y vista superior (derecha).

3.3. Elección de ruedas

A continuación, se explica de forma detallada los componentes más importantes para poder realizar los desplazamientos omnidireccionales: las ruedas.

Para la elección de ruedas a utilizar en el robot desarrollado en el proyecto se realizó un relevamiento de las existentes en la *Internet*. Se consideraron aspectos como capacidad de carga, tamaño de la rueda y rodillos, precio y compatibilidad con el resto de los componentes del robot. Como principal, se buscaba la capacidad de realizar movimientos omnidireccionales. Utilizando ruedas especiales se logra la omnidireccionalidad de forma directa además de quitar complejidad al diseño mecánico, por lo que se optó por la utilización de estas ruedas.

Respecto de la configuración, en los robots de tres ruedas, es recomendable utilizar ruedas especiales en las que los rodillos se encuentran a un ángulo de 90^o respecto de la rueda principal. Es posible utilizar ruedas cuyo ángulo entre los rodillos y la rueda sea distinto al anterior, pero esto introduciría una mayor complejidad mecánica en la creación u obtención de las ruedas.

También fue considerada la capacidad de carga de las ruedas así como el precio de compra a través de *Internet* y la compatibilidad con el kit de construcción *Lego Mindstorms NXT*.

Las capacidades de carga de las ruedas disponibles superan ampliamente la necesaria para el robot construído. Para ejemplificar, las ruedas del fabricante *School of Robotics [11]* (ver figura 21) poseen una capacidad de carga de 25kgs. por rueda. Las ruedas del fabricante *Kornylak Corporation* [12] (ver figura 22) poseen una capacidad de carga dependiente del tipo de ruedas, pero varían entre los 34kgs. y los 135kgs. Por lo tanto, esta característico no limitó las ruedas posibles de adquirir.

Sobre los precios de compra de cada tipo de rueda, se encontró que son muy dispares de acuerdo a cada fabricante. Para evaluar el precio total de cada rueda, se considera la cantidad necesaria para asegurar que los rodillos siempre posean un punto de contacto con la superficie para cada rueda. Para las ruedas del fabricante *School of Robotics*, alcanza con adquirir tres ruedas dobles para el robot, cada una a un costo de US$11 [11]. Considerando al fabricante *Kornylak Corporation*, de igual forma, se deben adquirir tres ruedas dobles (ver figura 22) . Sin embargo, el precio de cada rueda simple, considerando las de calidad más baja, es de US$14,67 ya que se construyen de forma de poseer una mayor capacidad de carga que las del fabricante anterior [12]. A pesar de que el precio es superior a las ruedas del primer fabricante, el adquirir estas ruedas podría justificarse para una futura reutilización en un robot de mayor tamaño y peso.

Por último, la característica más importante al momento de elegir las ruedas utilizadas en el robot es la compatibilidad con el *kit* de construcción *Lego Mindstorms NXT,* siendo ésta la herramienta que se utiliza para construir el robot. De las ruedas estudiadas, solamente las del fabricante *School of Robotics* son compatibles de forma nativa, por lo tanto se utilizaron éstas en el robot construído[11]. Estas ruedas además brindan dos características adicionales: los rodillos son recubiertos en goma lo que permite una mayor fricción con la superficie de contacto y que los rodillos son suficientemente anchos como para no perder la capacidad omnidireccional en una superficie rugosa o blanda (por ejemplo: una moquete). Con el fin de ejemplificar lo anterior, si los rodillos no tuviesen el radio suficiente, al utilizarlos en una moquete se notaría que la rueda principal entraría en contacto con la superficie, perdiendo la capacidad omnidireccional.

Otra consideración fue el tamaño de las ruedas ya que se pretendió que el tamaño del robot no superara los 18 centímetros de diámetro por considerarse éste un tamaño apropiado para participar en futuras competencias. Así mismo, el peso de las ruedas, de 52 gramos no representa un peso significativo respecto del total del robot.

La tabla 2 compara las ruedas consideradas al momento de adquirir las ruedas.

Fabricante	Capacidad de carga por rueda	Precio por rueda	Compatibilidad con kit Lego Mindstorm NXT
School of Robotics	25 kg	34 - 135 kg	Si
Kornylak Corporation	US$ 11	US$ 14,67	No

Cuadro 2: Tabla comparativa de las características de las ruedas consideradas.

Las especificaciones de las ruedas adquiridas a *School of Robotics* se pueden ver en la tabla 3.

Característica	*Detalle*
Dimensiones	64mm. de diámetro, 24mm. de ancho.
Peso	52gr.
Capacidad de carga	25 kg. por rueda.
Material	Goma de alta fricción, marco de plástico, ejes y marcos de acero inoxidable.
Compatibilidad	Directa con el *kit Lego Minstorm NXT*. Adaptable a otros kits.
Fabricante	School of Robotics [11]
Modelo	Holonomic Wheel - Código *1220*
Precio	U$S 11 más envío
País de Origen	Singapur

Cuadro 3: Tabla detallando las características de las ruedas a utilizar.

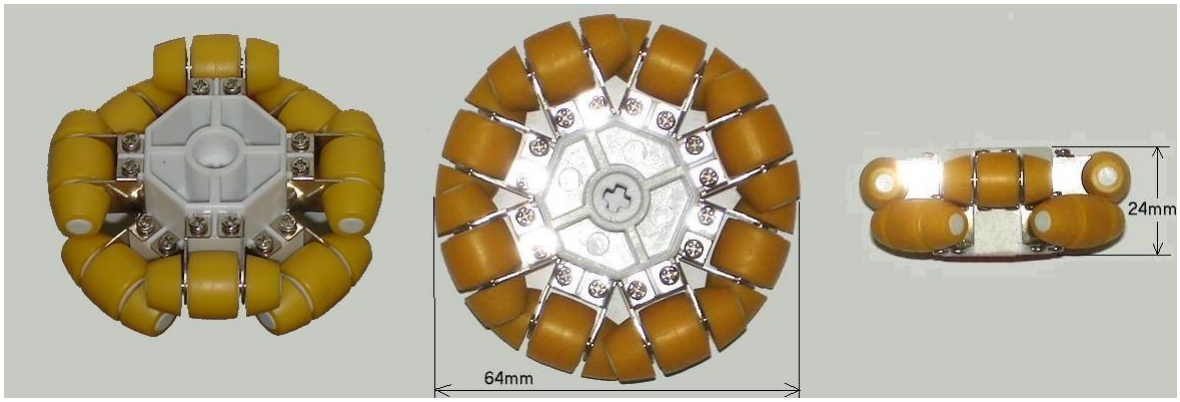

Figura 21: Ruedas omnidireccionales utilizadas para la construcción del robot. Vista de perspectiva (izquierda), vista lateral (centro) y vista superior (derecha).[11]

En la figura 21 se pueden ver las ruedas adquiridas.

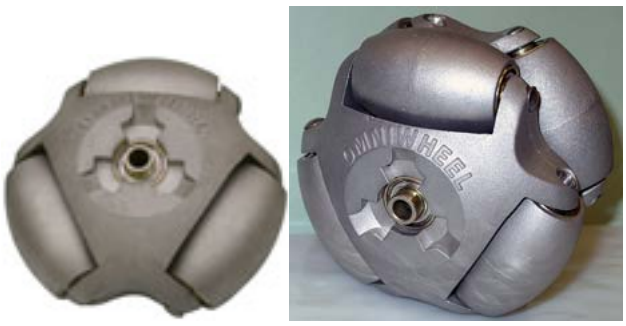

Figura 22: Ruedas de la empresa Kornylak. A la izquierda una rueda simple, a la derecha, aplicación de estas ruedas[12].

3.4. Control del Robot

En esta sección se describe el control básico necesario para el robot desarrollado en la sección 3.2. La primer parte de esta sección describe los cálculos matemáticos necesarios para controlar el robot con el comportamiento deseado y finalmente se detallan valores determinados de forma empírica para controlar los motores del *kit Lego Mindstorms NXT*[16].

Los cálculos para el control básico del robot fueron vistos en la sección 2 de este mismo documento. Para el control básico del robot el objetivo es obtener las velocidades angulares de las ruedas (vector $\begin{pmatrix} \dot{\phi}_1 & \dot{\phi}_2 & \dot{\phi}_3 \end{pmatrix}^T$ en radianes/segundo) a partir de la rotación actual del robot (θ) y la velocidad deseada en coordenadas cartesianas (vector $\begin{pmatrix} \dot{x} & \dot{y} & \dot{\theta} \end{pmatrix}^T$, con $\dot{x}$ e $\dot{y}$ en metros/segundo y $\dot{\theta}$ en radianes/segundo).

El vector buscado es el determinado por las ecuaciones 25.

$$\begin{pmatrix} \dot{\phi}_1 \\ \dot{\phi}_2 \\ \dot{\phi}_3 \end{pmatrix} = \frac{1}{r} \begin{pmatrix} -\sin(\theta) & \cos(\theta) & L \\ -\sin(\frac{\pi}{3} - \theta) & -\cos(\frac{\pi}{3} - \theta) & L \\ \sin(\frac{\pi}{3} + \theta) & -\cos(\frac{\pi}{3} + \theta) & L \end{pmatrix} \begin{pmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \end{pmatrix} \tag{25}$$

Donde r es el radio de las ruedas, L es la distancia desde el centro del robot a las ruedas, y θ es el ángulo entre el eje X y la primer rueda.

Finalmente, se debe conocer que valores se deben enviar a los motores para que tomen la velocidad angular determinada por la ecuación 25. Los motores del *kit Lego Mindstorms NXT*[16] reciben el parámetro de velocidad como un valor (potencia) entre -100 y 100. El método para obtener el factor de transformación (constante $K_{rotToPower}$[10]) entre velocidad angular de ruedas a potencia de motores fue el siguiente.

Se fijó una potencia del motor en P_k, luego se tomó el tiempo (t_k) que toma una rueda en dar N vueltas. Este tiempo fue muestreado una cantidad de veces significativa y finalmente se promediaron los valores tomados de t_k.

Luego de determinar estos parámetros empíricamente, la potencia P a enviar a un motor para que rote a una velocidad angular $\dot{\phi}$ está determinada por:

$$P = K_{rotToPower}\dot{\phi} \tag{26}$$

donde $K_{rotToPower}$ es la constante determinada por

$$K_{rotToPower} = \frac{P_k.t_k}{2.\pi.N}$$

Las constantes P_k, t_k y N son las detalladas anteriormente.

En el caso de los motores utilizados del *kit Lego Mindstorms NXT* [16], el resultado de esta constante fue

$$K_{rotToPower} = 5{,}61$$

Luego de determinar esta constante, se comprobó empíricamente que la constante vale para un muestreo de los posibles valores de P, confirmando la hipótesis planteada en la ecuación 26, esto es, la potencia P a determinar en los motores es lineal a la velocidad angular deseada.

[10] No se encontraron referencias a esta constante en la documentación del *kit* de construcción.

Cabe destacar que éste valor se cumple en condiciones de carga cero; esto es con los motores levantados. Cuando los motores están cargados, esta linealidad se mantiene hasta una potencia del 60 % debido al control de potencia[11] que realizan los motores; superando ésta, la velocidad se mantiene constante[25].

[11] Los motores del *kit Lego Mindstorms NXT* poseen un algoritmo de control cerrado que utiliza la retroalimentación de los *encoders* de los motores para ajustar la potencia efectiva enviada a éstos, de forma de proveer velocidades consistentes. El algoritmo de control continuamente regula la potencia enviada a las motores para mantener la velocidad solicitada.

4. Solución *EasyRobots*

En esta sección, se introducen los aspectos que describen los objetivos y las soluciones propuestas para el control de robots omnidireccionales a través de la aplicación desarrollada, *EasyRobots*. Además, se describe la arquitectura de la solución de forma general a fin de que el lector comprenda las interacciones internas más importantes de la aplicación desarrollada.

4.1. Desafíos y motivación

En la actualidad, para poder desarrollar un sistema robótico académico se debe contar con un ambiente de prueba adecuado, resolviendo entre otros, los siguientes desafíos:

- Sensado: se deben posicionar sensores en el robot o fuera de éste.
- Actuado: construcción mecánica del robot con sus motores.
- Modelado del entorno.

Desarrollar estas tareas puede insumir mucho tiempo y recursos, sin embargo pueden realizarse utilizando por ejemplo sistemas de visión artificial como *Ergo* o *Doraemon* (para resolver el sensado) y *kits* de construcción *Lego Mindstorms NXT* (para la construcción de los robots).

Luego de montado el ambiente robótico, existe otro desafío de igual o mayor importancia, como es el desarrollo de los comportamientos de los robots para llevar a cabo los objetivos planteados (por ejemplo la planificación de trayectorias para un robot jugador de fútbol robótico). Los desafíos más importantes a desarrollar en este ámbito son:

- Comunicación con sistemas de sensado
- Comunicación con sistemas de actuación
- Comportamientos del robot

En la figura 23 se representa un paradigma robótico mostrando como se relacionan estas componentes (sensado, actuado y planificación).

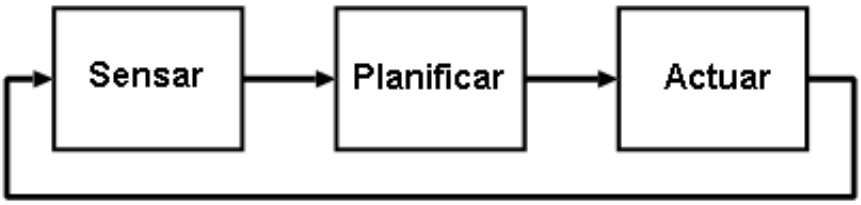

Figura 23: Vista general del paradigma jerárquico de la robótica. [8]

Para el desarrollo de cada una de componentes, son imprescindibles vastos conocimientos de programación, algoritmia, física, entre otros, ya que deben resolverse, desafíos como son las comunicaciones por red, en caso de que los comportamientos se evalúen externamente al robot, con los sitemas de actuación y sensado, y la implementación de los comportamientos. Estos desafíos dificultan la aproximación a la robótica a técnicos jóvenes con escasa formación en el tema, desmotivando el interés en el área.

El sistema *EasyRobots*, desarrollado en este proyecto de grado, intenta resumir la experiencia teórico-práctica sobre el control y construcción de robots omnidireccionales, encapsulándolo en un entorno que simplifica el desarollo y la resolución de los desafíos más comunes al desarrollar un sistema robótico. En *EasyRobots* se define un *marco de trabajo*, en el cual el usuario o desarrollador pueden construir y ejecutar sistemas robóticos. Para el desarrollo de estos sistemas se utilizan bloques predefinidos, los cuales resuelven problemas puntuales y específicos, como son:

- Interpretación y procesamiento de los datos sensados.

- Control y comunicación hacia los robots.
- Modelado de diversos tipos de robots.
- Comportamientos en alto nivel.

Los usuarios de este sistema pueden desarrollar sistemas robóticos tan solo manipulando archivos XML. Estos archivos contienen toda la información relevante a los sistemas de visión artificial, actuación, morfología de los robots y comportamientos, pudiendo ser editados mediante editores gráficos. Mediante el uso de este sistema jóvenes investigadores podrán adquirir conceptos de robótica generando mayor interés en esta área, entre otras como pueden ser programación y matemática. Además, usuarios con sólidos conocimientos de programación pueden utilizar el sistema para generar sistemas más escalables mediante la reutilización de componentes.

En *EasyRobots* actualmente pueden desarrollarse sistemas robóticos para robots móviles impulsados por motores de revolución utilizando sistemas de visión artificial basados en posicionamiento, sin embargo se diseñó de forma extensible para poder incluir otros tipos de elementos (robots, actuadores, sensores y comportamientos).

4.2. Descripción de la aplicación EasyRobots

Objetos de la aplicación y relaciones entre éstos Para la creación de sistemas robóticos, EasyRobots dispone de diferentes objetos funcionales. Estos son:

- *Entidades*: objetos físicos, incluyendo obstáculos y pelotas, además de:
  - Robots móviles omnidireccionales de tres o más ruedas.
  - Robots móviles de dos ruedas.
- *Sensores*: permiten obtener las posiciones globales de las diferentes entidades de la aplicación. Se incluyen *Sistemas de Visión Artificial Doraemon* y predicción de posiciones, pudiéndose incluir en el futuro otros tipos como son sensores basados en el reconocimiento de marcas de terreno y radiofaros.
- *Algoritmos de Fusión de Sensores*: modelan las técnicas de fusión de sensores que fueron explicadas en la sección 2.2. Se incluye un algoritmo de promedio ponderado de las mediciones de los sensores y otro selector de sensores, pudiéndose incluir otros por ejemplo basados en aproximaciones estadísticas.
- *Actuadores de revolución*: permiten realizar cambios en el mundo físico. Se incluyen actuadores de revolución de *Lego Mindstorms NXT* [16] y se pueden incluir en el futuro actuadores prismáticos para modelar, por ejemplo, pateadores o montacargas.
- *Comportamientos de las entidades controladas*: permiten controlar uno o más robots de forma simultánea. Se incluye el comportamiento reactivo *Campos potenciales,* otra basada en el seguimiento de puntos y comportamientos programables en el lenguaje *Java*. Otros tipos, como son comportamientos basados en *Grafos de Voronoi* o *Grafos de visibilidad*, pueden ser incluidos

En un sistema robótico, estos componentes necesitan relacionarse entre sí. En *EasyRobots* se determinaron y modelaron las siguientes relaciones (representadas en la figura 24):

- Relación *sensores - sensores lógicos*: En una sistema robótico, se debe modelar el estado del mundo físico (disposición de los objetos y robots). Para este fin, *EasyRobots* cuenta con sensores lógicos que contienen la información relevante al mundo físico. Estos sensores lógicos transforman las mediciones del mundo físico, provistas por los sensores físicos a los *Sistemas de Sensado*, a datos internos del sistema.
- Relación *sensores lógicos - entidades*: Cada entidad en del sistema puede requerir conocer su posición, por lo que se deben asociar a sensores lógicos. Además, los sensores lógicos podrían necesitar la velocidad de las entidades, por ejemplo, aquellos sensores que predicen la posición futura de una entidad a partir de su posición y su velocidad.
- Relación *entidades - comportamientos*: La ejecución del comportamiento de un robot requiere conocer su posición y velocidad, generándose la necesidad de esta relación. Además, luego de determinar el nuevo comportamiento de los robots, se debe comunicar las nuevas velocidades a éstos.

- Relación *entidades* - *actuadores*: Luego de determinar el próximo comportamiento de una entidad, se debe reflejar un cambio en el mundo físico. Las nuevas velocidades de los robots deben ser enviadas a los Sistemas de Actuación. En esta versión del sistema, se desarrolló[12] un *Sistema de Actuación* específico para la comunicación con los robots *Lego Mindstorms NXT* [16].

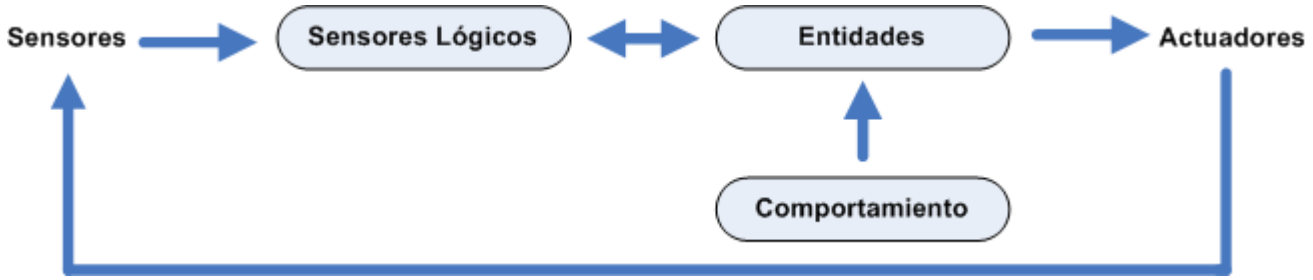

Figura 24: Relaciones entre componentes de *EasyRobots*.

Cabe destacar que las relaciones antes descritas no siempre deben de estar presentes. Por ejemplo, se debe poder modelar una entidad que no posea un actuador asociado; esto puede producirse en aplicaciones de prueba donde solamente se necesite conocer las velocidades asignadas a cada entidad, así como robots no controlados por el sistema.

La figura 25 ejemplifica una implantación de la aplicación. Tal como se puede observar, se cuenta con un *Sistema de Visión Artificial* que recibe datos de una cámara de visión global y las envía a la aplicación *EasyRobots*. Tras determinar los comportamientos, las velocidades a asignar a cada rueda de cada robot son enviadas al *Sistema de Actuación* presente. Este sistema finalmente envía las velocidades a los robots.

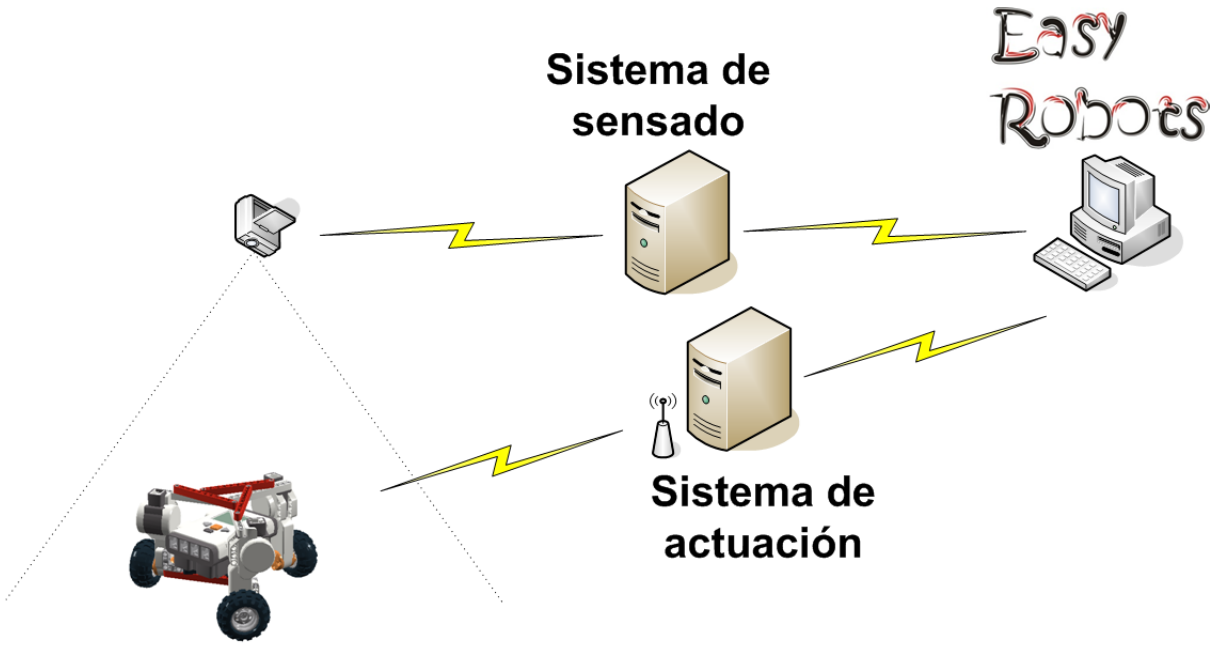

Figura 25: Vista de implantación de la aplicación.

Interacción del usuario con la aplicación La interacción del usuario con la aplicación en su versión actual es limitada. El usuario puede interactuar con la aplicación para cargar sistemas robóticos y activar o desactivar

[12]Si bien se disponía de un servidor para controlar robots mediante *Bluetooth* en el grupo MINA, éste debió adaptarse a fin de poder enviar velocidades para controlar tres ruedas en simultáneo (desde EasyRobots hasta el servidor de control). Estas velocidades son enviadas por el servidor secuencialmente a cada rueda a través de *Bluetooth*.

la ejecución de los comportamientos definidos dentro de dicha aplicación. La interacción se produce mediante la *interfaz de línea de comandos*[13] con ayuda de un menú que describe las acciones posibles.

Para cargar una aplicación, el usuario indica la ubicación del archivo de definición y el sistema carga a memoria la aplicación contenida en dicho archivo. Durante la carga, se crean las comunicaciones entre los sensores lógicos y físicos así como entre los actuadores y los robots físicos. A pesar de que las comunicaciones son creadas, no se observa actividad en los robots físicos ya que todos los robots se inicializan sin comportamiento. Para dar movilidad a un robot, se debe activar alguno de los comportamientos definidos que controle al robot. Para activar un comportamiento, el usuario ingresa el identificador deseado. En caso de no conocer los identificadores de los comportamientos, el usuario puede listar los presentes en la aplicación mediante una opción del menú. Análogamente es posible desactivar un comportamiento activo.

Como se explicó anteriormente, para cargar un sistema robótico el usuario debe indicar la ruta del archivo de definición del sistema. El formato del archivo de definición puede variar, sin embargo por defecto se desarrolló un formato de archivo XML para cargar los sistemas. En la sección 4.4 se explica el formato general de estos archivos. Para conocer en mayor detalle el formato de estos archivos referirse a la subsección 4.3.1.

Integración con aplicaciones externas Es importante hacer notar que en la aplicación está previsto, y se permite, la integración con otras aplicaciones. Por ejemplo, se pueden crear sensores lógicos que establezcan comunicación con una aplicación que retorne como resultado posiciones de las entidades, y así poder crear casos de prueba con datos iniciales conocidos. De igual forma, es posible crear actuadores que se comuniquen con aplicaciones que reciban velocidades de las entidades (actuadores físicos o simuladores) y despliegue de forma visual el desplazamiento de cada una de las entidades. Conociendo los protocolos de comunicación con las aplicaciones externas, se pueden desarrollar nuevos sensores lógicos y/o actuadores que implementen dichas comunicaciones, logrando así la integración.

La figura26 ejemplifica lo anterior.

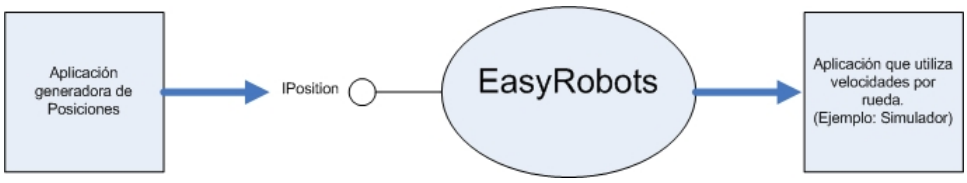

Figura 26: Integración con aplicaciones externas

Actualmente el sistema puede interactuar con la aplicación *Doraemon* [17] para obtener el posicionamiento de las entidades y con robots *Lego Mindstorms NXT* [16] para enviar velocidades a robots físicos, aunque otros sensores y actuadores pueden ser utilizados.

4.3. Funcionalidad del sistema *EasyRobots*

A continuación se detallan las funcionalidades del sistema. Se distinguen funcionalidades visibles para el usuario e internas a la aplicación.

4.3.1. Funcionalidades para el usuario

Crear un sistema robótico Utilizando un archivo *XML* se le permite al usuario definir un sistema robótico. Como se describe en la sección 4.7, si el usuario desea utilizar otro formato de archivo debe desarrollar el cargador adecuado para dicho formato. La figura 27 muestra un sistema robótico definido completamente. Los componentes utilizados son abarcados en mayor detalle en la sección 4.7.

[13] Programa informático que actúa como interfaz de usuario para comunicar al usuario con el sistema al cual pertenezca mediante una ventana que espera órdenes escritas por el usuario haciendo uso del tecládo, las interpreta y las entrega al sistema para su ejecución.

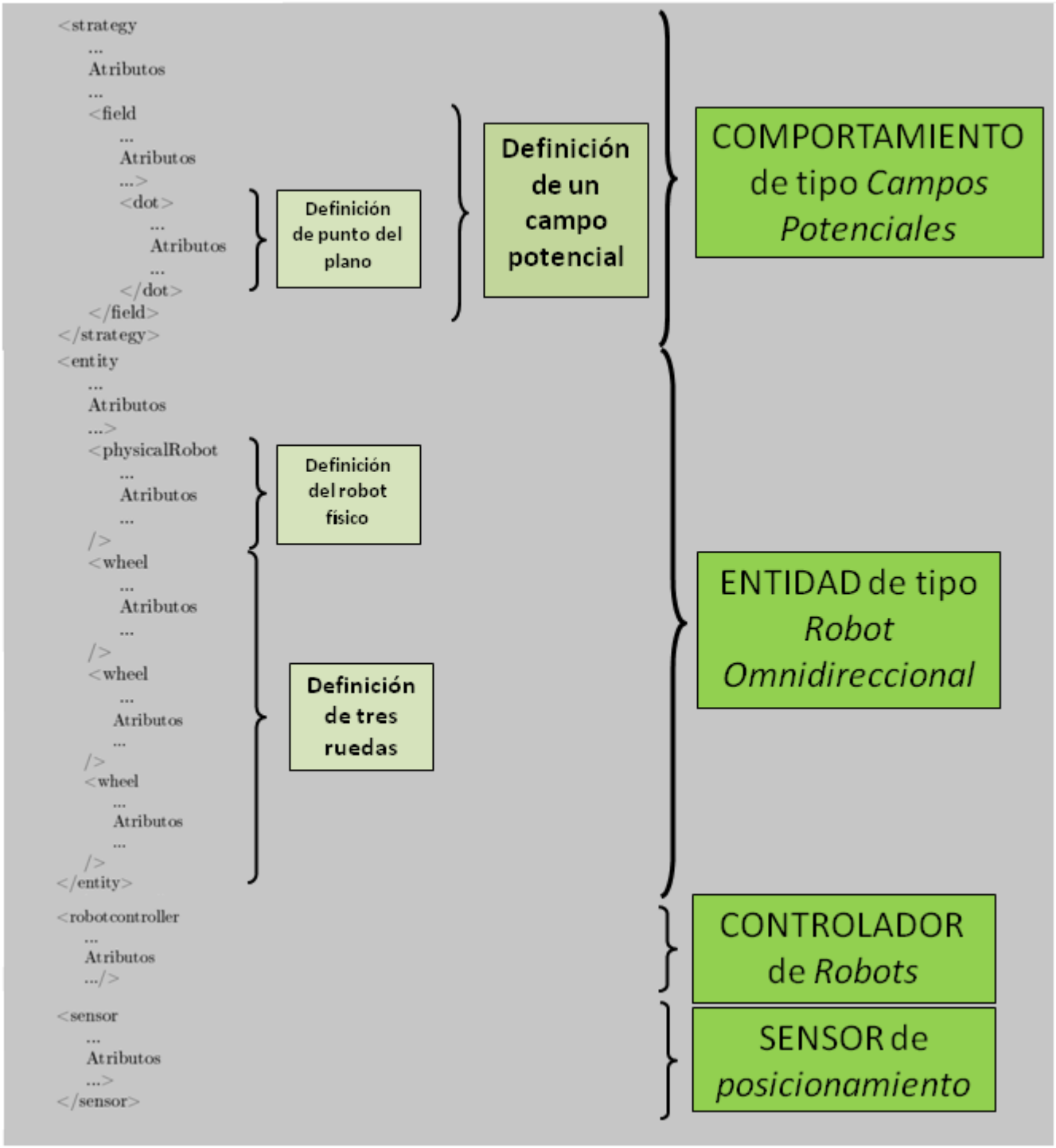

Figura 27: Archivo de definicion de un sistema robótico.

Para comprender los detalles de la definición del sistema en el archivo *XML*, referirse al documento de Descripción de la Arquitectura [26] o al Manual de Uso [27].

A partir del ejemplo anterior, se puede observar que el usuario cuenta con la posibilidad de cargar cualquier componente del sistema robótico.

Activar un comportamiento Luego de cargar un sistema, el mismo se encuentra activo. Sin embargo, aún no se ejecutan los comportamientos de las entidades que determinan las velocidades de los robots. Para comenzar a controlar los robots, el usuario debe activar un comportamiento. Luego de realizar esta operación, los robots

cuyo control está determinado por el comportamiento activo comenzarán a moverse. Es posible definir más de un comportamiento para un mismo robot; sin embargo puede encontrarse, como máximo, un comportamiento activo por robot.

Desactivar un comportamiento Esta función es la contraparte de la activación de un comportamiento. Para desactivar un comportamiento activo, el usuario debe indicar su identificador. Cabe notar que los robots controlados por el comportamiento desactivado recibirán el mensaje de detención cuando ocurra la desactivación.

Listar comportamientos Esta función despliega en la consola del usuario la lista de identificadores de los comportamientos presentes en el sistema robótico, cualquiera sea el estado de éstos (activos o inactivos).

4.3.2. Funcionalidades internas del sistema

Obtener posiciones de las entidades Permite que el sistema robótico reciba las posiciones de las entidades físicas dentro del *marco de trabajo*. Las posiciones son recibidas desde los Sistemas de Sensado, por ejemplo el sistema de visión artificial Doraemon [17], para ser procesadas por la aplicación. Esta funcionalidad permite que el sistema mantenga el estado de las entidades físicas actualizadas. Es posible utilizar varios Sistemas de Sensado en simultáneo, de forma de obtener señales de diversos sensores.

Fusionar posiciones de las entidades Esta funcionalidad se encuentra ligada a *"Obtener posiciones de las entidades"* anteriormente descrita. Esta función cubre la fusión de datos sensoriales obtenidos, también denominada *Sensor Fusion*. La fusión de datos sensoriales, así como sus ventajas y desventajas se encuentran explicadas detalladamente en la sección 2.2. El sistema permite utilizar diversos algoritmos de fusión de sensores en simultáneo, a fin de modelar distintas situaciones del entorno, como pueden ser robots utilizando distintos tipos de sensores.

Evaluación de comportamientos Evalúa los comportamientos definidos en el sistema. A partir de las posiciones y velocidades de las entidades, se calculan las velocidades a enviar a los robots. Las velocidades determinadas corresponden a un sistema de coordenadas global al sistema. El sistema permite ejecutar diversos comportamientos en simultáneo, cada uno controlando un conjunto de robots. Esto permite, por ejemplo, controlar dos equipos contrarios de fútbol de robots de forma simultánea.

Asignar velocidades a los robots lógicos Asigna nuevas velocidades a los robots lógicos luego de evaluar los comportamientos definidos para cada uno.

Conversión de velocidades lineales a velocidades angulares Permite el envío de velocidades a los robots físicos. Para poder realizar esa tarea, primero se realiza la conversión de velocidades deseadas a velocidades angulares de las ruedas (en radianes por segundo) considerando la estructura del robot. Finalmente, estas velocidades son enviadas al controlador del robot, el cual comunica la nueva velocidad al *Sistema de Actuación*.

4.4. Arquitectura de la Solución

Luego de explicar los aspectos generales de la aplicación, se explica en mayor detalle los elementos que integran la estructura interna de *EasyRobots*.

La arquitectura de la aplicación está compuesta por una variedad de componentes que interactúan entre si a fin de lograr el control de diversos tipos de robots. El diseño se centró en la extensibilidad de la aplicación de forma de permitir modelar diferentes tipos de robots, como son de dos ruedas u omnidireccionales de tres o más ruedas, sensores del entorno, comportamientos y actuadores sobre los robots.

La arquitectura está distribuída en capas (ver figura 28), siguiendo el patrón de diseño *Model View Controller* [28]. Este patrón está compuesto por tres capas de abstracción de las funciones del sistema, a fin de independizar los componentes más importantes entre sí: la capa *View* que abstrae la interfaz de interacción del usuario con la aplicación, la capa *Controller* que modela la comunicación de la interfaz de usuario a la lógica del sistema, mientras que la capa *Model* contiene la lógica de la aplicación. Los componentes que se explican en esta sección se encuentran distribuídos en estas capas de acuerdo a las funcionalidades que cumplen en la aplicación.

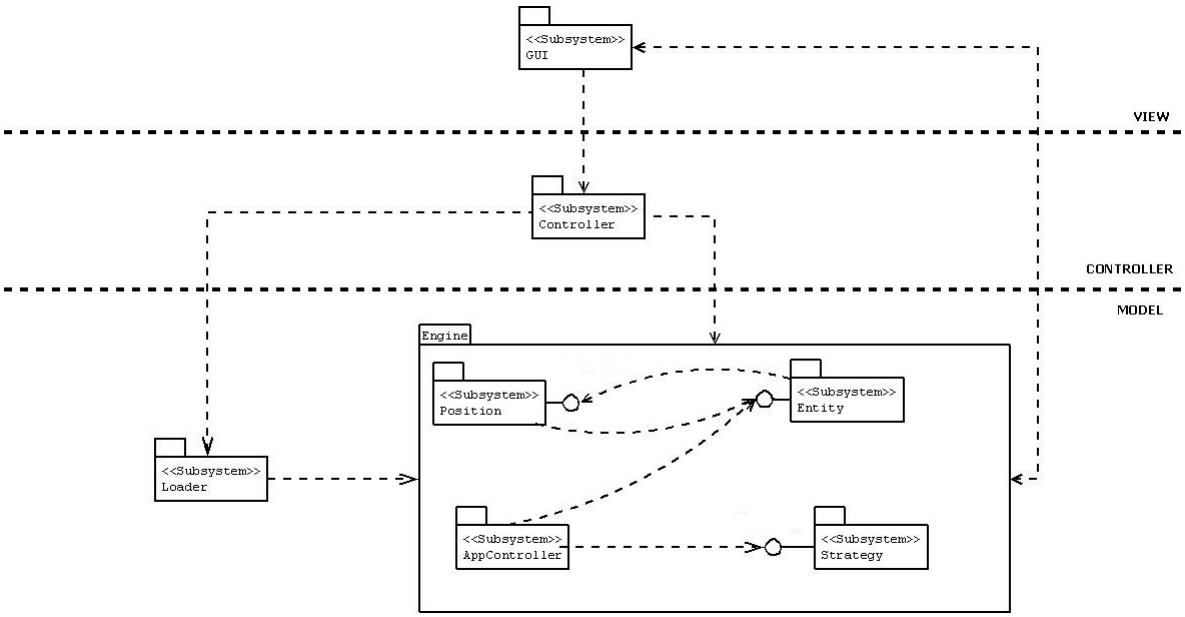

Figura 28: Arquitectura del sistema

A continuación se describen en mayor detalle los diferentes componentes que integran la aplicación indicando la capa a la que pertenecen.

4.4.1. Subsistema GUI (capa *View*)

Representa la interfaz de usuario desde la cual se interactúa con la aplicación. El usuario puede activar un comportamiento, cargar un nuevo sistema robótico, así como desplegar información del estado de los sistemas mediante notificaciones recibidas de éstos. Para recibir notificaciones de cambios de los subsistemas *Position* y *Entity*, la interfaz se debe registrar en estos subsistemas.

4.4.2. Subsistema Controller (capa *Controller*)

Se encarga de comunicar la interfaz gráfica con la capa lógica del sistema. Su objetivo es el de aislar la capa *View* de las operaciones del sistema, al ser la componente que direcciona las solicitudes a la capa lógica a través de las interfaces definidas. Para ello, recibe pedidos desde la interfaz de usuario, los procesa e invoca a las operaciones correspondientes de la capa lógica.

4.4.3. Subsistema Loader (capa *Model*)

Controla la carga de los sistemas robóticos a partir de un archivo de definición. A partir de éste, el subsistema crea las instancias necesarias para la posterior ejecución del sistema robótico. En el archivo pueden quedar tipos de entidades sin definir, por ejemplo: comportamientos, si únicamente se quiere registrar la posición de objetos, o controladores de robots físicos en caso de que no se desee interactuar con los robots. Sin embargo, los componentes que se encuentren especificados en el archivo deben asegurar seguir el esquema definido. Esto significa que es responsabilidad del usuario definir el sistema robótico correctamente.

4.4.4. Subsistema Position (capa *Model*)

Este subsistema modela el manejo de las distintas fuentes de datos de posicionamiento de robots y entidades. En éste, se cargan los distintos sensores lógicos que reciben datos de los *Sistemas de Sensado*. Además, existen sensores lógicos compuestos que implementan algoritmos de fusión de sensores. Cuando se recibe una nueva posición de una entidad, se le notifican los cambios para que los comportamientos se evalúen utilizando la posición actualizada. Cada *sensor lógico* se ejecuta en un hilo de ejecución propio, en el cual períodicamente se evalúa cambios con los *Sistemas de Sensado*.

Cabe destacar que los sensores lógicos no necesariamente tienen que estar asociados a sensores físicos, sino que pueden recibir datos de una aplicación externa que genera posiciones lógicas, por ejemplo un simulador.

A partir de las posiciones recibidas, existe la posibilidad de ejecutar algoritmos de fusión de sensores sobre un conjunto de sensores lógicos. También se brinda la posibilidad de definir *Sensores de Predicción*.

La figura 29 ejemplifica lo explicado anteriormente. En ésta, se puede observar que se cuenta con dos *Sistemas de Sensado* (A y B). Los datos sensados por A y B actualizan la posición de las entidades F y H utilizando los sensores lógicos C y D.

Se puede observar además un componente de predicción (I), que predice la nueva posición de la entidad (G) utilizando su velocidad. El resultado de la predicción se fusiona con los sensores C y D a través del componente de fusión de sensores E. La posición de la entidad G se actualiza con el resultado de esta fusión.

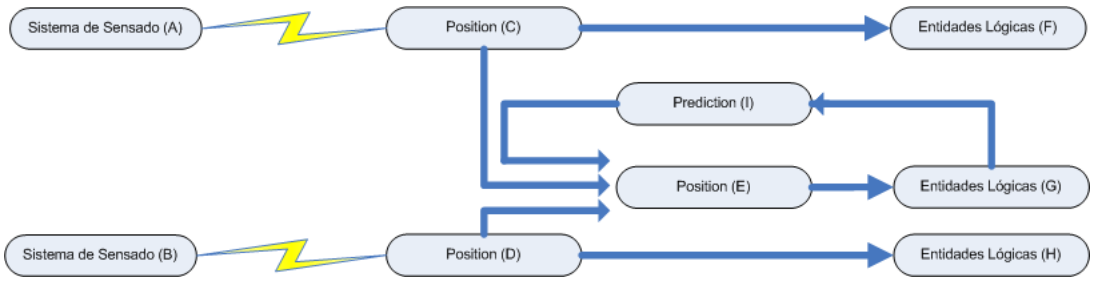

Figura 29: Subsistema Position de un sistema robótico de ejemplo.

4.4.5. Subsistema Entity (capa *Model*)

Este subsistema modela los robots (controlados por el sistema o no) y entidades (objetos como puden ser pelotas y obstáculos), además de controlar la comunicación con los robots físicos. Es el encargado de mantener el modelo (posiciones, velocidades, estructuras) de los robots y entidades.

Para modelar las entidades presentes en el sistema, se necesitan definir los siguientes atributos generales (cada tipo de robot específico puede requerir atributos adicionales):

1. Velocidad angular máxima de los motores: Esta restricción puede requerirse por limitaciones de los motores utilizados o si simplemente se desea limitar la velocidad de las ruedas. Es utilizada al momento de obtener las velocidades de cada rueda a partir de la velocidad lineal del robot, pudiendo ser utilizada para escalar las velocidades angulares deseadas.

2. Velocidad lineal máxima del robot: Este dato puede ser requerido para determinar el comportamiento de los robots, por ejemplo para limitar las velocidades a asignar a un robot o incluso para predecir intersecciones entre robots y objetos móviles.

3. Comunicación con el robot físico: Para que se puedan enviar velocidades al robot se requiere un modelo de comunicación con el mismo, cuya información se determina en el archivo de configuración. Se requieren, entre otros, parámetros tales como dirección de red y puerto, para que el establecimiento de la conexión con los *Sistemas de Actuación* sea exitoso.

Además de dichos datos, se pueden requerir datos adicionales. Por ejemplo, para modelar un robot omnidireccional con ruedas *Mecanum* o especiales, se debe brindar información de la disposición de las ruedas (un robot de este tipo puede tener tres o más ruedas), determinando así la estructura del robot. Estos datos son necesarios internamente para poder convertir velocidades del sistema de ejes cartesianos a velocidades angulares de las ruedas. Para ello, se cuenta con la componente *Rueda* que contiene los datos necesarios para definir la posición de la rueda y sus rodillos respecto del centro del robot. En la figura 30 se pueden observar los datos necesarios para describir la posición de cada rueda.

Para modelar entidades físicas que no reciben velocidades, se cuenta con la componente *Entidades Generales*. Éstas pueden ser utilizadas para modelar entidades de las cuales únicamente se reciben posiciones a través del subsistema *Position*, para ser utilizadas en la evaluación de los comportamientos presentes en el sistema. Dentro de esta categoría pueden modelarse objetos como una pelota u obstáculos en área de trabajo.

Como fue mencionado anteriormente, el subsistema convierte la velocidad lineal a velocidades angulares de las ruedas. La utilidad de esta funcionalidad es la siguiente: las ejecuciones de los comportamientos activos presentes en el *Subsistema AppController* retornan las velocidades en coordenadas cartesianas y el subsistema *Entity* las convierte a velocidades angulares de las ruedas.

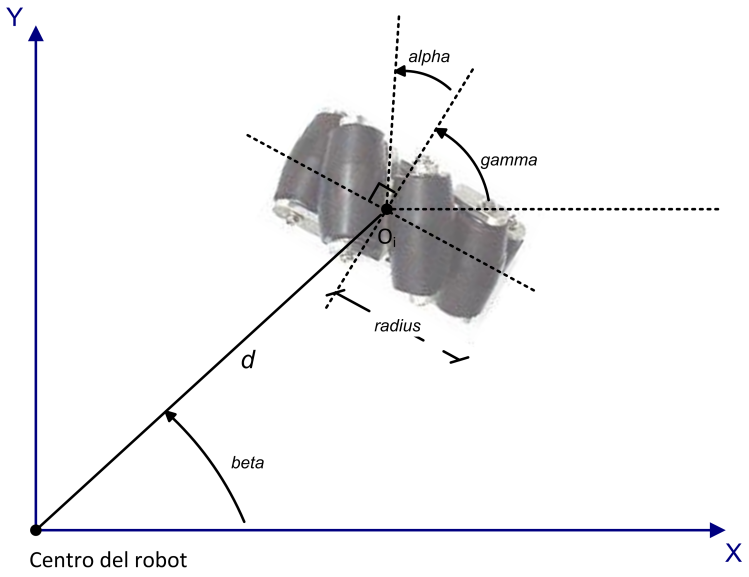

Figura 30: Variables necesarias para definir la posición de una rueda.

La ejecución de este subsistema ocurre por eventos y cíclicamente, distinguiéndose entre dos casos:

- Por eventos: provenientes del subsistema *Position* o del subsistema *AppController*. El subsistema *Position* notifica sobre el cambio de posición de las entidades, para que esta tome sus nuevas posiciones. De la misma manera, el subsistema *AppController* consulta las posiciones a las entidades referenciadas por un comportamiento, evalúa el comportamiento y finalmente notifica las nuevas velocidades a los robots que controla.
- Cíclicamente, la componente que se comunica con los *Sistemas de Actuados* consulta y asigna las velocidades de ruedas del robot.

La figura 31 ejemplifica este subsistema en un sistema robótico. El subsistema *Position* notifica las nuevas posiciones al *Entity*, el subsistema *AppController* consulta estas posiciones y asigna nuevas velocidades luego de evaluar los comportamientos. Finalmente el subsistema *Entity* envía las nuevas velocidades al robot fisico a través del *Componente de Robot Físico*, que se comunica con el *Sistema de Actuación* (p. ej. mediante *Bluetooth*).

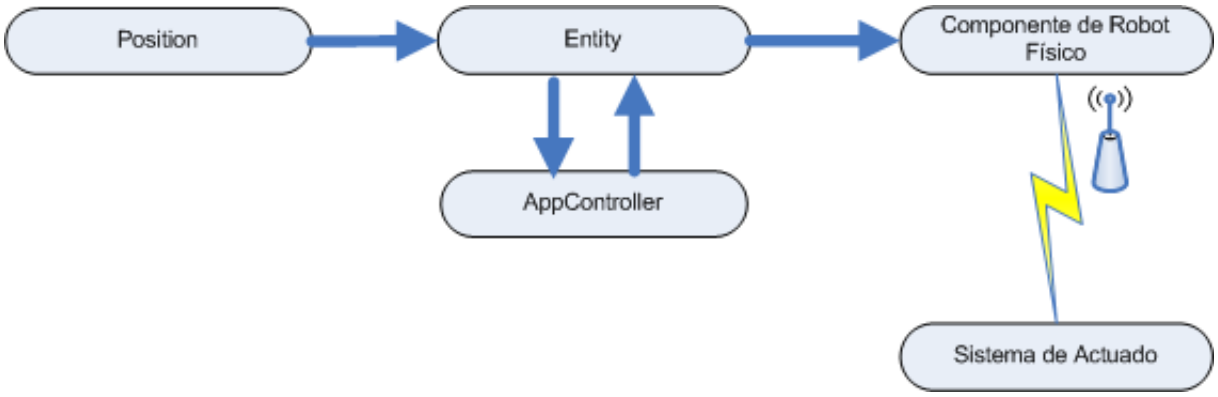

Figura 31: Subsistema Entity de un sistema robótico de ejemplo.

4.4.6. Subsistema AppController (capa *Model*)

Este subsistema mantiene la ejecución de los comportamientos que se encuentren activos en el sistema robótico.

La ejecución del subsistema se realiza periódicamente, en un hilo de ejecución para cada comportamiento activo. El subsistema solicita las posiciones a las entidades que requiere el comportamiento a evaluar. Tras su evaluación, el subsistema asigna las velocidades obtenidas de la evaluación a los robots controlados por la misma.

4.4.7. Subsistema Strategy (capa *Model*)

Modela el conjunto de comportamientos que pueden encontrarse activos o inactivos en el sistema robótico.

Los componentes que representan a los comportamientos poseen referencias a las entidades cuyas posiciones son necesarias para su evaluación. De igual forma, poseen referencias a los robots a los cuales controla. Estos datos son necesarios para que el subsistema *AppController* que evalúa los comportamientos, solicite las posiciones a las entidades correctas y asigne las velocidades resultado a los robots correctos.

Los tipos de comportamiento que el usuario puede utilizar en la versión actual de la aplicación son (para comprender en mayor detalle el diseño interno de estos comportamientos consultar el documento de Descripción de Arquitectura del Sistema [26]):

4.5. Interacción entre subsistemas

En el sistema *EasyRobots* las funcionalidades determinan principalmente tres interacciones entre los subsistemas detallados..

- *Inicialización*: Cuando el usuario desea cargar un sistema robótico para su posterior ejecución, el sistema carga el archivo de definición creando los sensores lógicos, entidades y comportamientos. Finalmente, se inicializa la comunicación con los *Sistemas de Sensado*, así como la comunicación con los *Sistemas de Actuación*. Luego de llevar a cabo la inicialización, se comienzan a registrar las nuevas posiciones de las entidades y los robots controlados permanecen inmóviles.

- *Activacion de un comportamiento*: Se lleva a cabo cuando el usuario selecciona un comportamiento a activar. Luego de esta etapa, el o los comportamientos seleccionados pasan a estar en ejecución y se comienzan a enviar nuevas velocidades a los robots.

- *Ejecución*: Cuando el sistema está inicializado y algún comportamiento se encuentra activo, los sensores lógicos actualizan el modelo con nuevas posiciones, se evalúan los comportamientos activos y se envían las velocidades a los robots.

4.6. Decisiones de Diseño

En esta sección se explica las decisiones más importantes tomadas a la hora de determinar el diseño de la aplicación.

4.6.1. Mensajes inter-sistemas

La comunicación de los *Sistemas de Sensado* y los *Sistemas de Actuación* con la aplicación *EasyRobots* es epecífica a cada implementación de estos sistemas. Como ya fue especificado, en la versión de la aplicación que se implementó, se modeló el *Sistema de Visión Artificial Doraemon* y el *Sistema de Actuación* que permite interactuar con robots *Lego Mindstorms NXT*.

El *Sistema de Visión Artificial Doraemon* envía mensajes mediante Broadcast[14] en código ASCII[15] a la aplicación mediante el protocolo *UDP*[16], requiriendo que la aplicación conozca la estructura del mensaje a fin de obtener los datos relevantes incluidos en él.

El cuadro 4 ejemplifica un mensaje recibido del servidor *Doraemon*. La primer línea del mensaje contiene la cantidad de objetos configurados en el servidor, el número de serie del mensaje y el tiempo transcurrido entre el envío del último mensaje y el anterior. La siguiente línea contiene las coordenadas de la posición de la cámara $(x_{cam}, y_{cam}, z_{cam})$. Las líneas restantes poseen la información de los objetos capturados. Dichas líneas incluyen el identificador del tipo de objeto (robot, pelota, entre otros), el identificador del objeto, indicación si el objeto fue encontrado en el último paquete, la posición del objeto (en coordenadas cartesianas en tres dimensiones(x, y, z)), la orientación del objeto en radianes (entre $-\pi$ y π) y la velocidad del objeto $(\dot{x}, \dot{y})$.

Para crear la conexión con dicho sistema se requiere conocer el puerto a través del cual se reciben los mensajes y el largo máximo posible de un mensaje.

[14] Modo de transmisión de información mediante paquetes que son recibidos por todos los dispositivos presentes en una red.

[15] Acrónimo inglés de *American Standard Code for Information Interchange* (Código Estadounidense Estándar para el Intercambio de Información), es un código de siete bits utilizado para representar caracteres mediante cadena de bits.

[16] Protocolo de comunicación de la capa Transporte del Modelo OSI basado en el intercambio de datagramas. Permite el envío de datagramas a través de la red sin que se establezca una conexión previa al envío.

```
7 6188 0.000290976 ; #defined objects, frame#, time diff
1605.82 -708.394 1321.44 ; x, y, z coordinates of camera
2 spot1 Found 1232.5 416.374 0 0 0 0 ; object information
2 spot2 Found 1559.22 417.359 0 0 0 0
2 spot3 Found 1260.55 812.189 0 0 0 0
2 spot4 Found 902.726 1002.43 0 0 0 0
2 spot5 Found 746.045 735.631 0 0 0 0
1 ball1 Found 1677.99 1205.55 50 0 -2.75769 1.19908
0 car54 Found 1783.53 873.531 100 2.63944 1.47684 -6.49056
```

Cuadro 4: Ejemplo de mensaje recibido del cliente Doraemon [17].

Los controladores de las entidades lógicas, para interactuar con los robots *Lego Mindstorms NXT*, envían mensajes en código *ASCII* también mediante el protocolo *UDP*. Para que el mensaje sea correctamente interpretado por el *Sistema de Actuación* que es la componente que recibe el mensaje, debe poseer una estructura válida.

El cuadro 5 ejemplifica un mensaje enviado por la aplicación al *Sistema de Actuación*. El mensaje está compuesto por el identificador de la tarjeta física de red del robot, un comando (en este caso, *set_output_state*, indicando que se desea asignar velocidades a los motores), y las velocidades para cada motor (éstas están definidas entre los valores -100 y 100, ver 3). Estos valores son separados por el carácter ",".

```
00:16:53:01:4E:3B,set_output_state_3,60,-30,40
```

Cuadro 5: Ejemplo de mensaje enviado por la aplicación al *Sistema de Actuación*.

Luego de creado el mensaje, éste se envía mediante una conexión creada previamente. Para la creación de dicha conexión, se requiere la dirección de red y el puerto del *Sistema de Actuación*.

Las comunicaciones presentadas anteriormente son específicas para los *Sistemas de Sensado* y *Actuación* utilizados. Al tratarse de un *marco de trabajo*, la aplicación brinda la posibilidad de desarrollar modelos de comunicación con otros sistemas. Asimismo, como se explicó en la sección 4.2, un tipo de sistema externo puede ser un simulador, por lo que la comunicación en este caso está prevista y es posible modelarla.

4.6.2. Intefaz de Usuario

La interfaz de usuario se realiza por línea de comandos. A nivel de aplicación, dicha interfaz debe comunicarse con la capa lógica del sistema. Asimismo puede obtener información de los subsistemas que forman parte del modelo de la aplicación a fin de notificar al usuario nuevos cambios en el sistema robótico. La capa de modelo del sistema brinda interfaces para solicitar datos que se deseen representar, como por ejemplo posiciones y velocidades de los robots.

4.6.3. Manejo de Errores

El manejo de errores críticos se realiza mediante excepciones. Se poseen dos tipos de excepciones:

1. *Aquellas que se producen ante una solicitud del usuario.* Este caso se produce por ejemplo ante la solicitud de cargar un sistema. Las excepciones pueden provenir de distintas fuentes:

   a) Del subsistema *Loader*, por ejemplo al pretender cargar un archivo XML mal formado o inexistente.

   b) Del subsistema *Position*, por ejemplo, al establecer la comunicación con un *Sistema de Sensado* con dirección de red mal especificada.

   c) Del subsistema *Entity*, por ejemplo, al establecer la conexión con un *Sistema de Actuación* inexistente o incorrecto.

   En estos casos, las excepciones son retornadas hasta la interfaz de usuario para que el usuario pueda solucionar el problema.

2. *Aquellas que se producen internamente en la aplicación.* Este caso corresponde a las excepciones que ocurren internamente en el sistema durante la ejecución de un sistema, por ejemplo al enviar velocidades a un robot. Como dichas excepciones no son referentes a una invocación del usuario, se registran en el *Log*[17] del sistema para su posterior estudio. A su vez, el sistema intenta recuperarse del error o continuar sin esa funcionalidad.

Por lo explicado previamente, si un desarrollador desea extender el sistema, por ejemplo, programando otros mecanismos de carga de sistemas o de comunicación con los *Sistemas de Sensado* y *Actuación*, debe considerar que se cuenta con la opción de lanzar excepciones que son visibles en la interfaz de usuario. Se recomienda que en estos casos se cree un tipo de excepción específico de forma de poder identificar fácilmente el origen de las fallas.

4.6.4. Manejo de Registros de *Log*

Para registrar eventos importantes de la aplicacion, como por ejemplo cambios en velocidades de un robot luego de evaluar un comportamiento, se cuenta con el registro de *log* del sistema.

El *log* del sistema se implementó utilizando la librería *Log4j* [18]. En esta librería se definen distintos niveles de registro. A saber, de mayor a menor nivel de importancia:

- *FATAL*: Error fatal y detenimiento de la ejecución de la aplicación.
- *ERROR*: Errores que no detienen la ejecución de la aplicación. La aplicación maneja el error y continúa la ejecución.
- *WARN*: Advertencia sobre cierta anomalía de la ejecución de la aplicación.
- *INFO*: Mensaje de información que se encuentra en la aplicación.
- *DEBUG*: Mensajes de depuración de la aplicación. No es recomendable activar este estado cuando la aplicación se encuentre implantada.
- *TRACE*: Similar a Debug, pero con mayor nivel de detalle.

Asimismo, este sistema de registro puede direccionar la salida a distintos destinos. Entre ellos, se pueden dirigir a la consola del usuario, a un archivo en el sistema de archivos u otros medios.

Para utilizar el sistema de registro, es necesario determinar el modo en como se realiza el registro. En la aplicación implementada se cuenta con el archivo *log4j.xml*. El contenido de dicho archivo se observa en el cuadro 6.

En este archivo se especifica los posibles destinos donde se graban los eventos (*Appender)* y cual de estos destinos se utiliza (especificado con la etiqueta *root*), especificando además del menor nivel de registro.

Se pueden configurar diversos destinos, por ejemplo la consola de usuario (indicado por la clase *ConsoleAppender)* o un archivo de texto (indicado por la clase *FileAppender*).

En la versión actual de la aplicación, los registros se envían a un archivo llamado *System.log*, con un nivel minimo de *DEBUG*. Cada nuevo registro se concatena a los anteriores. El formato de cada uno es el siguiente: *[nombre thread], tiempo en el que se registró el evento, nivel de importancia del registro y mensaje del registro,* separados por el carácter Tabulador a fin de facilitar la importación de los datos en aplicaciones de planilla de cálculo.

Para mayor nivel de detalle sobre la utilización de ésta librería y sus opciones, ver el sitio *web* de *Logging Services* de *Apache* [29].

4.7. Implementación actual y Extensibilidad del Sistema

En esta subsección se explica la implementación actual del sistema y se dan pautas de la extensibilidad de cada subsistema exponiendo posibles mejoras.

[17] Registro datos o información sobre los eventos que ocurren en un dispositivo o sistema en particular.

[18] Biblioteca de código abierto desarrollada en el lenguaje de computación *Java* por la fundación *Apache Software* que permite a los desarrolladores de *software* elegir la salida y el nivel de granularidad de los mensajes en tiempo de ejecución.Ver [29].

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE log4j:configuration SYSTEM "log4j.dtd">
<log4j:configuration xmlns:log4j="http://jakarta.apache.org/log4j/">
    <appender name="file" class="org.apache.log4j.FileAppender">
        <param name="File" value="System.log"/>
        <param name="Append" value="false"/>
        <layout class="org.apache.log4j.PatternLayout">
            <param name="ConversionPattern" value="[%t]\t %d{HH:mm:ss,SSS}\t %p\t %m %n"/>
        </layout>
    </appender>
    <appender name="console" class="org.apache.log4j.ConsoleAppender">
        <param name="Target" value="System.out"/>
        <layout class="org.apache.log4j.PatternLayout">
            <param name="ConversionPattern" value="[%t]\t %d{HH:mm:ss,SSS}\t %p\t %m %n"/>
        </layout>
    </appender>
    <root>
        <priority value="DEBUG" />
        <appender-ref ref="file" />
    </root>
</log4j:configuration>
```

Cuadro 6: Contenido del archivo log4j.xml

4.7.1. Subsistema GUI

Permite que un desarrollador implemente diversas interfaces, no imponiendo restricciones sobre la implementación. Sin embargo, se debe considerar que la interfaz debe procesar excepciones provenientes de los subsistemas de la capa lógica, tal como fue explicado en la sección 4.6.3. Además, este subsistema es el encargado de direccionar los pedidos del usuario a través del subsistema *Controller* por lo que se deben utilizar las interfaces que éste provee.

Asimismo, la interfaz puede suscribirse a los eventos de los subsistemas *Position* y *Entity* para recibir y desplegar cambios en los estados de dichos subsistemas. Este subsistema es el que brinda mayor flexibilidad al desarrollador permitiendo implementar cualquier interfaz de usuario en lenguaje *Java*.

A modo de ejemplo, en la aplicación desarrollada en este proyecto, se creó una interfaz de usuario por *línea de comando* para realizar las opereraciones descritas en la sección 4.3.1.

4.7.2. Subsistema Loader

Este subsistema le permite implementar nuevos cargadores de sistemas robóticos. Los nuevos cargadores deben inicializar todas las entidades modeladas en el sistema.

El cargador debe conocer como cargar cada componente del modelo (sensores, comportamientos y robots). Por este motivo, un nuevo cargador debe ser extensible, considerando la posibilidad de que surjan nuevos componentes.

A modo de ejemplo, en la aplicación desarrollada se creó un cargador de archivos en formato *XML*, llamado *XMLLoader*. Éste posee subcargadores que inicializan los distintos componentes. En caso de implementarse un nuevo componente del sistema, se debe implementar únicamente su subcargador.

4.7.3. Subsistema Position

Permite crear nuevos sensores lógicos para comunicarse con *Sistemas de Sensado*. Es posible extender distintas categorías de sensores lógicos, a notar:

Sensor global corresponde a sensores que procesan *Sistemas de Sensado* que devuelven posiciones relativas a un eje de coordenadas fijo.

Fusión de sensores sensores lógicos que implementan algoritmos de fusión de sensores utilizando como entrada datos de sensores lógicos internos al sistema.

Sensor abstracto corresponde a sensores no contemplados en las categorías anteriores.

A modo de ejemplo, en la aplicación desarrollada se crearon cuatro tipos de sensores lógicos:

1. *DoraemonClient*: Sensor de posicionamiento global que procesa los datos de un *Sistema de Visión Artificial Doraemon*.

2. *AverageSensorFusion*: Fusión de datos sensoriales mediante un algoritmo de promedio ponderado. La ponderación se realiza en base a la confianza asociada a los sensores.

3. *FashionSensorFusion*: Fusión de datos sensoriales en la cual se selecciona la medición del sensor más confiable disponible en el momento de la fusión.

4. *PredictionSensor*: Predice posiciones a partir de la última velocidad y posición del robot (posición obtenida de otro sensor lógico).

A fin de profundizar sobre la fusión de sensores, se explican las características de los algoritmos específicos desarrollados. Para poder implementar cada uno de estos, se desarrolló el modelo general de fusión de sensores detallado en la sección 2.2, mediante un diseño que permite la composición de sensores genéricos, facilitando la implementación de nuevos algoritmos de fusión de sensores. Los nuevos algoritmos que se desarrollen, deben tomar como entrada la última posición y confianza de los sensores a fusionar, y luego retornar una nueva posición, resultado de evaluar el algoritmo de fusión.

El uso de este modelo general para fusión de sensores permitiría, por ejemplo, tomar un sistema robótico ya existente y sin mayor costo que el de implementar un algoritmo de fusión de sensores adecuado para la solución, mejorar el sensado para obtener un mayor rendimiento de la solución.

Fusión de sensores de promedio ponderado (*Average Sensor Fusion*) El cuadro 7 contiene el pseudocódigo del algoritmo de fusión a través del promedio ponderado de las mediciones (*AverageSensorFusion*).

```
———————— Comienzo del pseudocódigo————————————
———————— Datos cargados: ————————————
s0,s1, ...., sn-1 - Sensores a fusionar

———————— Variables utilizadas en el pseudocódigo: ————————————
newPos - Posición a retornar al finalizar el algoritmo
sumConfidence - Suma de las confianzas de todos los sensores fusionados
sensorPosition - Posición sensada por el sensor

———————— Pseudocodigo: ————————————

newPos = {0,0,0}
sumConfidence = 0
//Integro cada dato sensado a la suma de posiciones
para cada (sensor si){
    sumConfidence += si -> getConfidence()
    sensorPosition = si -> getPosition()
    sensorPosition *= si -> getConfidence()
    newPos += sensorPosition
}
newPos /= sumConfidence
return newPos
———————— Fin del pseudocódigo ————————————
```

Cuadro 7: Pseudocódigo del algoritmo de fusión de sensores de promedio ponderado.

Este algoritmo puede ser utilizado, por ejemplo, para promediar medidas de sensores redundantes, como pueden ser dos sistemas de visión artificial *Doraemon* o *Ergo*, de forma de disminuir el error en las mediciones realizadas.

Fusión de sensores de selección (*Fashion Sensor Fusion*) Este tipo de fusión de sensores corresponde al descrito en la sección 2.2.3. Puede ser utilizado, por ejemplo, de forma de implementar un sistema de visión híbrido, utilizando en todo momento el sensor que mejor se adapte a cada situación. Como ejemplo de esta aplicación, podría utilizarse el sistema de visión *Doraemon* en las zonas iluminadas, alternando a un sensor de predicción, cuando el anterior no esté disponible (debido a zonas de oscuridad o no alcanzadas por el lente de la cámara).

4.7.4. Subsistema Strategy

Permite agregar nuevos tipos de comportamientos al sistema para modelar sistemas robóticos diferentes a los implementados en la versión actual, por ejemplo comportamientos que incluyan técnicas de aprendizaje automático. Cada nuevo comportamiento que se desarrolle, deberá calcular y retornar las velocidades lineales de los robots controlados.

A modo de ejemplo, en la aplicación desarrollada se incluyen tres tipos de comportamientos:

1. *Potential Fields*: Comportamiento del tipo *Campos potenciales*. En éste se definen campos atractores y repulsores. El comportamiento del robot se ve definido por la suma de estos campos en su posición actual. Las posiciones de los campos pueden ser definidas estáticamente o dependientes de las entidades.

2. *Waypoints*: Comportamiento de puntos de seguimiento. En este comportamiento el robot recorre las posiciones de la lista de puntos determinados por el usuario. La lista puede ser un circuito cerrado, por lo que se pueden determinar la cantidad de vueltas que se desea recorrer. Además, las posiciones pueden ser estáticas o dinámicas dependientes de las posiciones de las entidades.

3. *Comportamiento Java*: Permite que el usuario defina completamente un comportamiento propio mediante un programa *Java*, respetando una interfaz definida. Se indica la ubicación del programa con el comportamiento implementado, el cual se compila en tiempo de ejecución y se registra en el sistema.

Para profundizar sobre estos comportamientos especificos, a continuación se detallan los componentes principales.

Campos Potenciales (Potential Fields) El concepto de Campos Potenciales puede pensarse como una partícula cargada navegando a través de un campo magnético a causa de las fuerzas que influyen sobre ésta. En el área de la robótica, este concepto puede utilizarse para determinar el comportamiento de un robot que intenta alcanzar un objetivo evitando chocar con obstáculos. En particular, se determinan múltiples objetos del entorno del robot con tareas y funciones asociadas, se representa cada uno con un campo potencial y se combinan los campos potenciales para determinar el comportamiento del robot.

Cada campo potencial se entiende como un vector de movimiento que debe tener el robot. Los campos pueden ser atractores o repulsores. Los primeros causan que el robot sea atraído al punto generador del campo mientras que los segundos tienen el efecto opuesto. La figura 32 permite observar el campo potencial generado por un atractor (izquierda) y un repulsor (derecha) . En esta figura, los vectores representan el campo potencial evaluado en cada punto, aunque se debe considerar que cada campo se calcula solamente en la posición donde se encuentre el robot y no en toda la superficie, para evitar realizar cálculos innecesarios.

Los campos pueden entenderse como funciones atómicas, que combinadas permiten modelar ambientes complejos donde se encuentran objetos con diferentes tareas asignadas. Un claro ejemplo, representado en la figura 33 (izquierda), es el de alcanzar un objetivo evadiendo un obstáculo que se interpone en el camino. El aplicar los campos combinados del ejemplo al robot, determina el comportamiento (campo potencial) que se representa en la figura 33 (derecha).

Además de los campos circulares, existen otros como pueden ser los determinados por una recta, en la dirección que se desee respecto de ésta. La figura 34 muestra una recta que genera un campo perpendicular a su dirección.

Finalmente, en la figura 35 se observa un sistema robótico modelado con *Campos Potenciales* en la que un robot debe alcanzar un objetivo evadiendo las paredes del entorno y los obstáculos presentes.

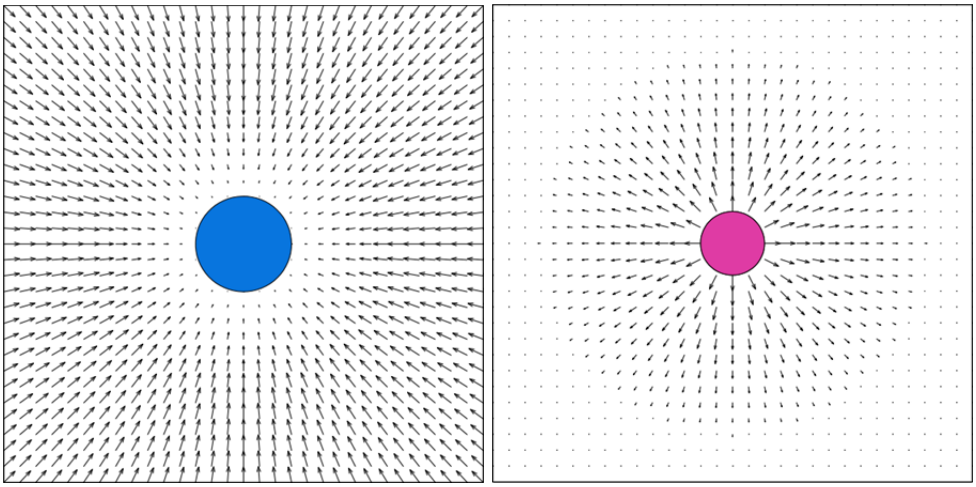

Figura 32: Campos potenciales básicos. (izq.) Campo potencial atractor, (der.) Campo potencial repulsor.[13]

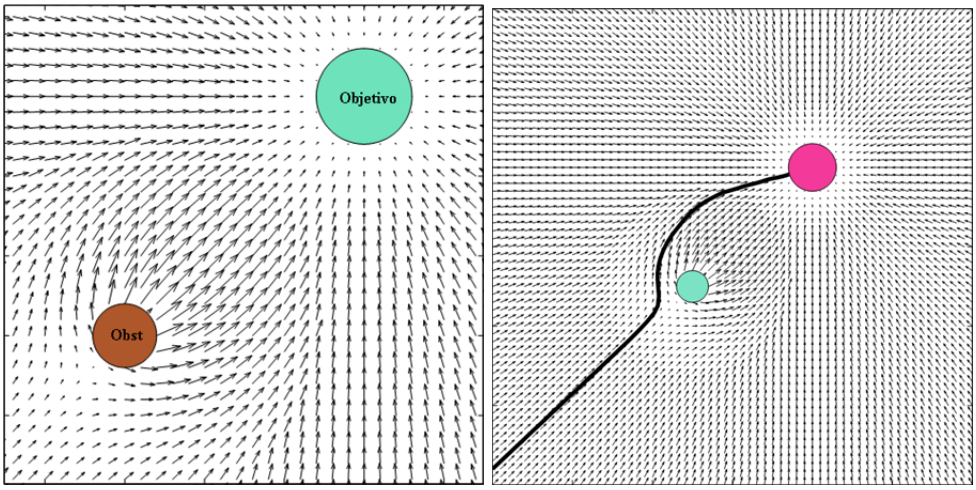

Figura 33: (izq.)Campos potenciales combinados y (der.) Comportamiento del robot determinado por los campos[13].

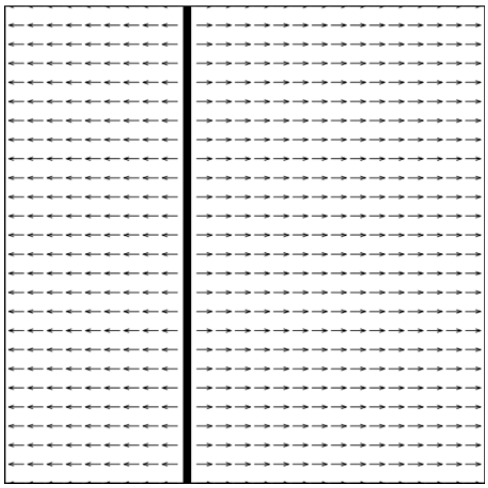

Figura 34: Campo potencial generado por una recta[13].

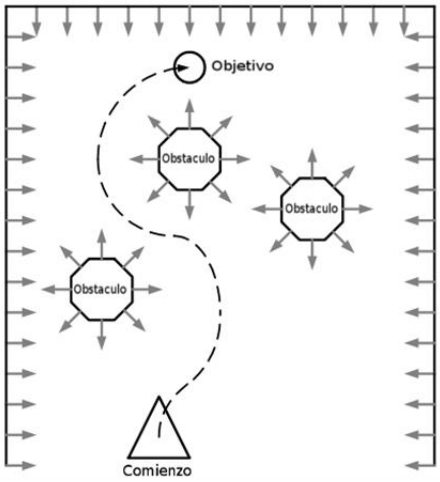

Figura 35: Sistema robótico de ejemplo, modelado con *Campos Potenciales*.

WaypointsStrategy El cuadro 8 contiene el pseudocódigo del algoritmo que lo implementa. Cabe destacar que cuando un robot es controlado por este comportamiento, éste recorre una secuencia de puntos definidos de forma estática o variable, por ejemplo dependiente de la posición de otros objetos sobre la superficie del campo simulado. Como se representa en el pseudocódigo, el robot recorre la lista de *waypoints*. Cuando se detecta que se está a menos de la distancia máxima aceptada del *waypoint* objetivo, se comienza el desplazamiento al siguiente *waypoint* a la velocidad indicada por la variable *speed*. Luego de cada medición, se ajusta la velocidad considerando el campo potencial generado por el objetivo. Este proceso continúa de acuerdo al valor de las variables *loop* y *laps*. Si el valor de *loop* es verdadero, el recorrido se realiza hasta que la ejecución del comportamiento se desactive. Si el valor es *falso*, entonces se recorre la secuencia de puntos tantas veces como la variable *laps* lo indique.

4.7.5. Subsistema Entity

Para comprender la extensibilidad de este subsistema, se divide la explicación en dos partes: extensibilidad de entidades lógicas y comunicación con los robot.

Modelado de entidades lógicas La extensibilidad permite incorporar nuevas categorías de entidades y robots.

En la categoría de entidades genéricas, se modelan entidades que no son controlables desde la aplicación, como son pelotas, robots oponentes y obstáculos.

Al incluir nuevos tipos de robots lógicos, se debe implementar la conversión de velocidades lineales a velocidades angulares, para que éste pueda participar en los comportamientos definidos en el sistema. Para determinar la velocidad angular se debe conocer la estructura del robot que se utiliza (disposición y orientación de las ruedas respecto del centro del robot), quedando esta tarea a cargo del desarrollador.

En el sistema se desarrollaron varias categorías de robots lógicos, a notar:

Robots omnidireccionales Comprende cualquier tipo de robot móvil omnidireccional de tres o más ruedas (mecanum o universales). Para ello, se implementó un algoritmo de control genérico, como se detalla en la sección 2.1.3 (Robots de K ruedas). Estos robots se modelan utilizando ruedas omnidireccionales genéricas como se explica en la subsección 4.4.5.

Robots de dos ruedas Modela los robots no omnidireccionales de dos ruedas.

Controladores de robots físicos Todo robot que desee desplazarse debe contener un controlador asociado para lograr la comunicación con el *Sistema de Actuación*. Para ello, se debe tener un modelo de la comunicación con el robot físico. Si se desea desarrollar un nuevo tipo de comunicación, se deberá implementar el controlador de robots físicos. Éste conoce e implementa el protocolo de comunicación con el *Sistema de Actuación*. Este controlador debe almacenar la configuración necesaria para esta comunicación: dirección de red, puerto, entre otros.

A modo de ejemplo, en la aplicación desarrollada en el proyecto se extendió cada uno de estos componentes:

```
————————- Comienzo del pseudocódigo————————
———————— Datos cargados: ————————
p0,p1,p2,...,pn-1 - Waypoints
epsilon - Distancia (en metros) cuando se considera que se alcanzó un waypoint
loop - Variable booleana que indica si se debe realizar un ciclo de recorrido por siempre
laps - Vueltas
speed - Velocidad (en metros por segundo)

———————— Variables utilizadas en el pseudocódigo: ————————
waypointIterator - Iterador de waypoints (se inicializa con el valor 0)
n - Cantidad de waypoints
p - Colección de puntos

———————— Pseudocodigo: ————————

ret={0,0,0}
//verifico si continúa ejecutándose
if (laps>0 || loop){
    if (distancia(posRobot, p[waypointIterator])
        //Llegue a un waypoint
        waypointIterator++
    if (waypointIterator == n )
    //termine una vuelta
        if (!loop)
            laps–
    //verifico nuevamente que no haya terminado las vueltas
    if (laps>0 || loop){
        campo->setCenter(p[waypointIterator])
        ret = campo->getPower()
    }
}
return ret
———————— Fin del pseudocódigo ————————
```

Cuadro 8: Pseudocódigo del comportamiento de tipo *Waypoints*

1. Se implementó un robot *Lego Mindstorms NXT*. Éste contiene la dirección física del robot (dirección *MAC*[19] del receptor *Bluetooth*).

2. Se desarrolló un controlador de robots físicos que permite la comunicación con robots *Lego Mindstorms NXT* (se especifica la dirección de red, puerto y formato del mensaje a enviar a cada robot).

4.8. Sistemas robóticos de ejemplo

4.8.1. Ejemplo 1 - Robot repartidor

En este primer sistema (su definición en un archivo XML se encuentra en el Apéndice A), representado en la figura 36, se cuenta con un robot omnidireccional de tres ruedas y un objetivo a ser alcanzado. Para alcanzarlo, se deben evitar áreas del campo de juego (obstáculos y paredes).

Para modelar el sistema, se creó un comportamiento del tipo *Campos Potenciales* compuesto por los siguientes campos:

1. Objetivo a ser alcanzado: Campo circular atractor.

2. Obstáculos a evitar: Campos circulares repulsores y un semiplano repulsor.

[19] Acrónimo inglés de *Media Access Control* (Control de Acceso al Medio). Identificador de 48 *bits* (6 octetos) que corresponde de forma única a un dispositivo de red.

El campo atractor debe ser de una magnitud suficiente como para lograr que el robot se dirija al objetivo a pesar de la presencia de campos repulsores. La magnitud de los campos repulsores debe ser, próximo a los campos, suficiente como para evitar que el robot alcance su centro. Lo mismo ocurre con el campo semiplano, para evitar que el robot abandone el escenario.

Un sistema similar al modelado, en el uso diario de los robots puede ocurrir en el caso de un robot repartidor que debe alcanzar un objetivo (campo marcado en color verde en la figura) en un ambiente cerrado, ante la presencia de columnas (campos circulares marcados en rojo en la figura) y paredes (campo semi-plano marcado en rojo).

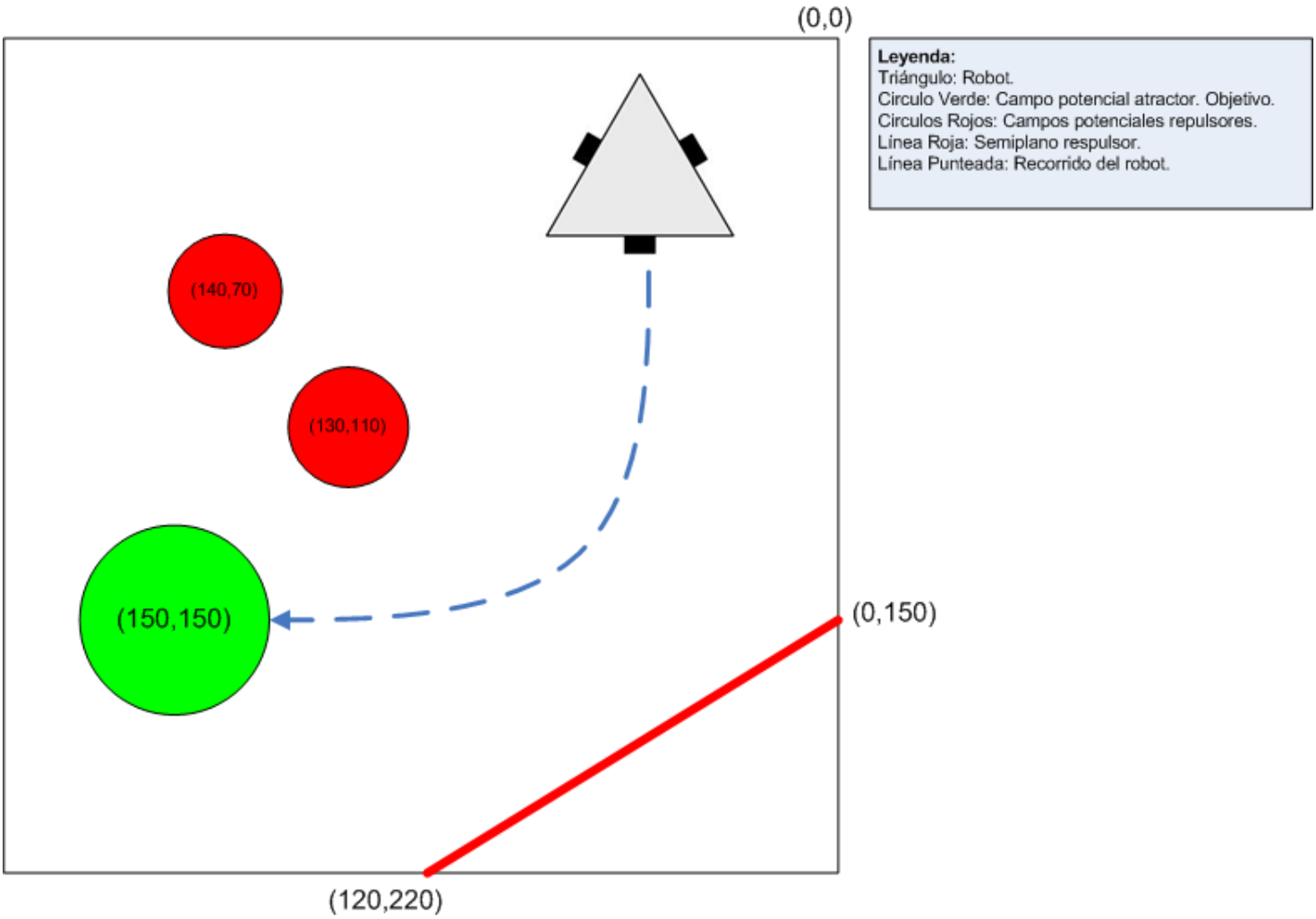

Figura 36: Robot repartidor buscando el destino.

La posición del robot es determinada por un algoritmo de fusión de sensores que realiza un promedio ponderado de las mediciones registradas por un sensor de posición global (cámara de video, utilizando el *Sistema de Visión Artificial Doraemon*) y un sensor predictor.

4.8.2. Ejemplo 2 - Circuito dentado

En la figura 37 se representa un robot omnidireccional de tres ruedas y un comportamiento de seguimiento de puntos compuesto por seis puntos.

Este sistema, cuya definición se encuentra en el Apéndice A, es desarrollado a fin de comprobar la factibilidad de que un robot omnidireccional pueda realizar movimientos omnidireccionales, principalmente en cambios bruscos de dirección.

La posición del robot es registrada mediante un sensor de posición global (cámara, utilizando el *Sistema de Visión Artificial Doraemon*).

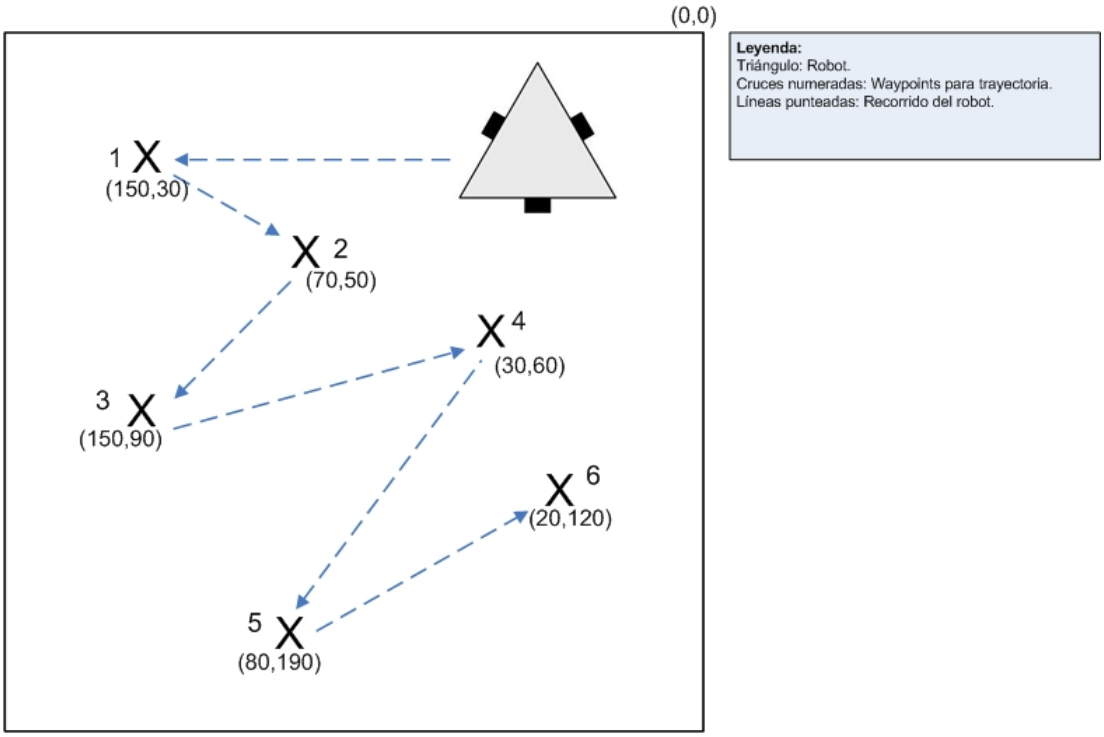

Figura 37: Comportamiento *Waypoints* para un circuito dentado.

5. Evaluación Global

5.1. Introducción

Esta sección presenta los experimentos realizados para verificar las distintas áreas de estudio y desarrollo del proyecto. Se explican los experimentos realizados y el cometido de cada uno de éstos y finalmente se evalúan los resultados obtenidos. En particular, se explican los experimentos realizados sobre el sistema *EasyRobots* a partir del modelado de un robot omnidireccional de tres ruedas y utilizando diferentes sensores.

A partir de la ejecución y evaluación de los experimentos, se pretende identificar los aspectos del sistema *EasyRobots*, del robot omnidireccional construído y del modelado del mundo a mejorar en trabajos a futuro. Considerando lo anterior, se explica el resultado de los experimentos realizados, a fin de poder identificar las fortalezas y debilidades del sistema y las ideas desarrolladas.

En cada uno de los experimentos, se explican los resultados considerando dos áreas principales:

- Aspectos del sistema *EasyRobots*: En este punto se consideran los resultados de la ejecución del sistema, sus debilidades y las mejoras posibles para obtener mejores resultados.
- Aspectos del robot omnidireccional: En este punto se consideran los resultados obtenidos sobre el robot construído y los aspectos a mejorar sobre éste.

5.2. Experimentos

5.2.1. Limitantes tecnológicas

Los experimentos realizados tienen como fin comprobar el funcionamiento del sistema *EasyRobots* y la omnidireccionalidad del robot construído.

La construcción del robot, realizada utilizando el *kit* de construcción *Lego Mindstorms NXT*, supone la existencia de errores debido a los componentes físicos utilizados (motores, ruedas y su deslizamiento sobre la superficie, baterías utilizadas, entre otros). De igual forma, la comunicación con el robot implica que se utilicen medios de comunicación variados, cada uno con ruidos en sus señales, pérdida de paquetes, encolamiento de paquetes, entre otros problemas. El sensado de la posición del robot, mediante el uso de *Sistema de Visión Artificial Doraemon*, posee un error sobre las mediciones registradas. Según se explica en [17], las mediciones registradas por este sistema contienen un error de hasta 1 centímetro en la superficie sensada. Este error también puede variar de acuerdo a la iluminación del ambiente de prueba.

En menor medida, el sistema *EasyRobots* posee errores debido al sistema de redondeo del lenguaje Java cuando se utilizan números reales.

Por tratarse de experimentos en ambientes no deterministas, en los que se incluyen componentes físicos (el robot construído y los *Sistemas de Sensado* y *Actuación*), las condiciones iniciales de cada uno no son reproducibles, dificultando la comparación entre éstos. Esto implica que para poder comparar la utilización de diversos sensores, todos deben sensar la misma ejecución de un experimentos repetidas veces a fin de comparar posteriormente los datos sensados de forma estadística. Si bien se pueden estudiar los resultados de cada ejecución, no se pueden comparar en detalle respecto de otras ejecuciones.

Para el estudio posterior a la ejecución de los experimentos, se registraron en el *log* del sistema las posiciones del robot durante toda la ejecución. En particular, se registraron las posiciones sensadas por los sensores presentes, a saber: sistema *Doraemon*, un sensor de predicción y un algoritmo de fusión de sensores mediante la selección del sensor más confiable en cada medición. Cabe remarcar que estos sensores generan una cantidad de datos discretos, por lo que no se obtiene un muestreo continuo de la trayectoria del robot.

5.2.2. Criterios de Evaluación

A fin de poder evaluar los resultados de la ejecución, se debieron establecer criterios de cada aspecto general en estudio. Es decir, se debieron establecer criterios sobre el robot construído y sobre el sistema *EasyRobots*.

A nivel del robot construído, los criterios fueron:

- Omnidireccionalidad del robot construído: este punto implicó comprobar que el robot pudiese realizar movimientos omnidireccionales. Todos los experimentos ejecutados implican la realización de trayectorias con cambios de dirección, por lo que este criterio se puede evaluar con cada uno de los experimentos. En particular se evalúa que ante cambios de dirección pronunciados en la trayectoria, no se registre al robot en la misma posición durante más de un período de sensado. Es decir que el robot debe realizar un cambio de

dirección sin rotar para posicionarse en la dirección del desplazamiento; demostrando que el robot posee un desplazamiento continuo.

- Trayectorias correctas: implica comprobar que las trayectorias que realice el robot sean correctas. Se deben evaluar las realizadas por el robot respecto las solicitadas por la aplicación, en particular las desviaciones máximas de las trayectorias.

A nivel del sistema *EasyRobots* y el modelado del robot en el sistema, se consideraron los siguientes criterios:

- Funcionamiento de la fusión de sensores: trata sobre el funcionamiento del algoritmo de fusión de sensores utilizado, en particular el de selección de la medida provista por el sensor más confiable del conjunto de sensores disponibles. Se debe comprobar que el muestreo de posiciones de la fusión genere trayectoria continuas en todo momento, incluso cuando el sensor más confiable no se encuentre disponible.

5.3. Ambientes de Prueba

Previo a presentar los diversos experimentos realizados y sus resultados, se describe el robot omnidireccional utilizado, los sensores y otras características que componen los sistemas robóticos utilizados.

5.3.1. Componentes definidos

Robot omnidireccional El robot construído está compuesto por tres ruedas omnidireccionales, cuyos rodillos se encuentran a 90^o respecto del eje de la rueda. La disposición de éstas es la representada en la sección 3. La ubicación del robot al comienzo de cada experimento no es definida.

Como ya fue explicado en la sección 5.2.1, se utilizó el *kit* de construcción *Lego NXT Mindstorms* y los componentes que éste brinda, p. ej. un bloque de control, motores, piezas de construcción, etc.

La comunicación con el robot se realizó haciendo uso de un servidor de comunicación con robots mediante *Bluetooth*. Éste fue provisto por el grupo MINA[23] de la Facultad de Ingeniería.

El bloque del archivo XML para definir el robot utilizado se representa en el cuadro 9. Para el significado de cada atributo, referirse al documento de Descripción de Arquitectura de Software[26].

Sensores Los sensores utilizados son:

1. Cámara: Se contó con una cámara de video genérica dispuesta de forma de registrar las posiciones de los objetos sobre cualquier punto de la superficie de prueba. La comunicación con ésta se logró haciendo uso del *Sistema de Visión Artificial Doraemon*[17] provisto por el grupo MINA.

2. Predicción: Se utilizó un sensor lógico de predicción de posiciones a partir de la posición de un robot y la velocidad deseada (la enviada al robot) en un instante dado.

3. Fusión de Sensores: Se implementó un algoritmo de fusión de sensores que utiliza la medición del sensor más confiable en el instante que se realiza la fusión.

Los bloques *XML* para la definición de estos sensores no son ejemplificados ya que sus atributos representan datos internos a la aplicación, por lo que no aportan a la comprensión de la estructura de la aplicación utilizada. Respecto al bloque de fusión de sensores, cabe destacar que el sensor más confiable es la cámara seguido del sensor de predicción.

Comportamientos El comportamiento utilizado fue *WaypointsStrategy*, definiendo distintas trayectorias dependiendo de los experimentos y los criterios definidos.

Para comprender la definición interna de éste, referirse al documento Descripción de Arquitectura de Software[26]. La definición de los puntos que definen la trayectoria se realizó de forma que todos los desplazamientos se realicen en sentido anti-horario.

Cabe destacar que en los sistemas robóticos definidos, se considera que el robot alcanzó el *waypoint* buscado si se encuentra a menos de 10 centímetros de distancia de éste. Esta distancia fue escogida de forma de contemplar errores en las mediciones del sistema de visión (debido a problemas de iluminación o calibración de parches); además, con esta medida se asegura que el robot llegará a entrar en contacto con el punto, ya que el radio del robot es de 9 centímetros (tomando en cuenta además los errores del sistema de visión y la inercia que lleva el robot debido a su velocidad).

```
<entity
    xsi:type="OmnidirectionalRobot"
    id="idRobot"
    positionId="sensor1"
    maxwheelrotation="15">
    <physicalRobot
        xsi:type="NXTRobot"
        physicalAddress="00:16:53:01:4E:3B"
        physicalControllerId="nxtController"/>
    <!- Primera rueda ->
    <wheel
        alpha="1.57079633"
        beta="0.0"
        d="0.095"
        gamma="0.0"
        radius="0.032"
    />
    <!- Segunda rueda ->
    <wheel
        alpha="1.57079633"
        beta="2.0943951"
        d="0.095"
        gamma="2.0943951"
        radius="0.032"
    />
    <!- Tercera rueda ->
    <wheel
        alpha="1.57079633"
        beta="4.1887902"
        d="0.095"
        gamma="4.1887902"
        radius="0.032"
    />
</entity>
```

Cuadro 9: Bloque de XML para definir el robot utilizado.

En todos los experimentos, se indicó que la velocidad máxima del robot al desplazarse debía ser 0.3 metros por segundo. Se decidió utilizar esta velocidad, ya que al utilizar valores mayores (0.5 m/s o mayor) se notó que el robot al llegar al destino continuaba avanzando debido a la inercia de su movimiento; al utilizar velocidades menores se notó que el robot tardaba en comenzar a moverse y lo hacía de forma pausada. La velocidad elegida permite al robot desplazarse realizando movimientos suaves y continuos.

5.4. Experimentos realizados

5.4.1. Recorrido cuadrado utilizando Doraemon

En este experimento se requiere que el robot realice una trayectoria cuadrada diez veces. Se busca observar el desempeño haciendo uso de diversos sensores. Para determinar la trayectoria, se utiliza el comportamiento *Waypoints* y se crean cuatro puntos para modelar el cuadrado a recorrer. Considerando los criterios presentados en la sección 5.2.2, se debe evaluar que el robot realice una trayectoria continua ante cambios de dirección y que su trayectoria sea similar a la trayectoria ideal. Para el sensado, se utiliza solamente la cámara de visión global en conjunto con el *Sistema de Visión Artificial Doraemon*.

Se comenzó el experimento utilizando como sensor al sistema *Doraemon*. En la figura 38 se puede observar el resultado de la ejecución. Como característica principal de esta ejecución, se remarca que la distancia máxima entre el robot y la trayectoria ideal fue de 6,7 centrímetros, por lo que a pesar de realizar varias vueltas, el robot se mantiene próximo de la trayectoria ideal.

La imagen 39 muestra la distancia del robot a la trayectoria ideal a lo largo de todo el recorrido. Como se puede ver, el promedio de la distancia a la trayectoria ideal es de 1,8 centímetros. Además, su desviación estándar[20] es de 1,3 centímetros. Esto permite concluir que el error en el recorrido se mantiene acotado y próximo al promedio.

Finalmente, la imagen 40 muestra la orientación del robot a lo largo del recorrido, la cual permanece acotada en el rango de ángulos$(-31°, -4°)$. Esto muestra empíricamente la capacidad omnidireccional del robot, ya que es capaz de recorrer una trayectoria sin variar su orientación, por ser el control de su orientación independiente de la velocidad lineal. Esta pequeña variación se debe a fuerzas externas (rozamiento en las ruedas), a retardos al asignar la velocidad en las ruedas (los comandos para las ruedas se envían secuencialmente para cada rueda a través de *Bluetooth*) así como a la construcción mecánica del robot que presenta desviaciones del modelo teórico utilizado (distribución de ruedas y diferencias entre rendimiento de los motores).

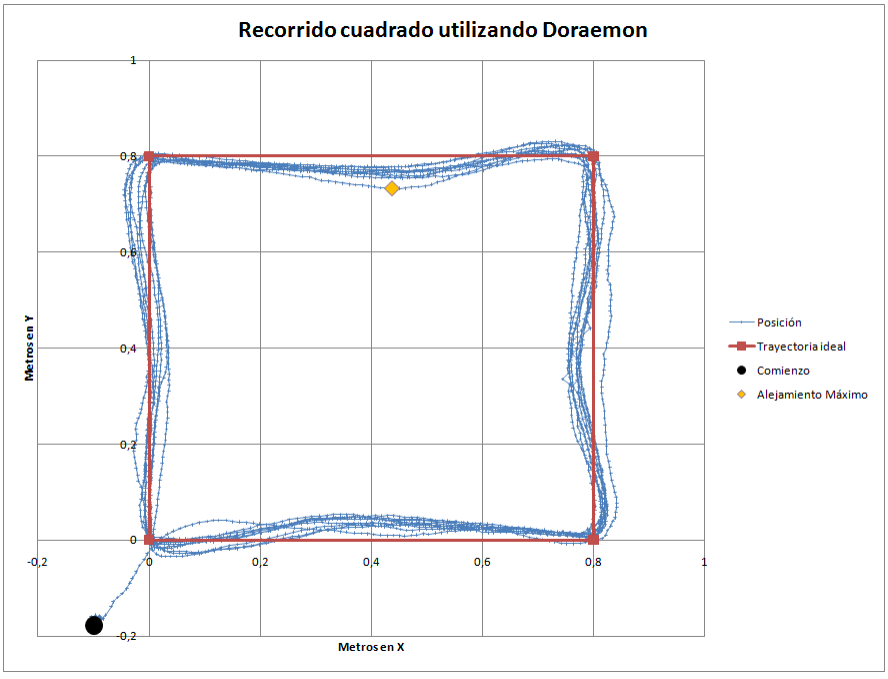

Figura 38: Resultado del recorrido cuadrado utilizando *Doraemon*.

[20] La desviación estándar es una medida del grado de dispersión de los datos con respecto al valor promedio. Dicho de otra manera, la desviación estándar es simplemente el "promedio" o variación esperada con respecto a la media aritmética.

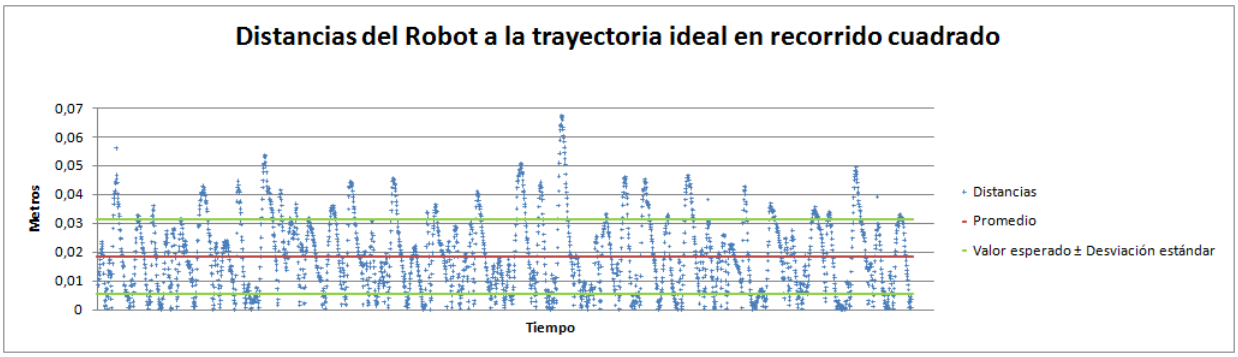

Figura 39: Distancias del robot omnidireccional a la trayectoria cuadrada ideal.

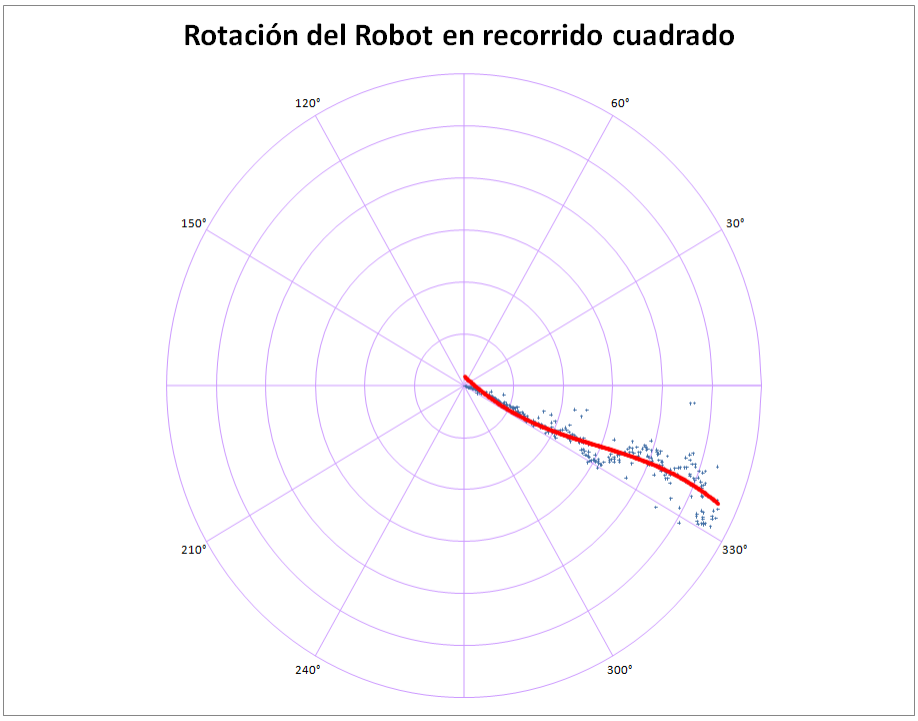

Figura 40: Rotación del robot omnidireccional a lo largo de la trayectoria cuadrada.

5.4.2. Recorrido cuadrado utilizando Predicción

Este experimento es similar al primero, teniendo como únicas diferencias que en este caso se utiliza un sensor predictor y se realizan cinco vueltas. En la figura 41 se observa el resultado de la ejecución. A diferencia de la ejecución de la prueba utilizando *Doraemon*, se observa como la trayectoria se aleja de la ideal. Durante la primer vuelta, marcada en color verde, el robot se mantuvo próximo a la trayectoria ideal. Sin embargo, a medida que se realizaban las siguientes vueltas, la trayectoria se distancia cada vez en mayor medida de la trayectoria ideal. De acuerdo al estudio posterior, se determa que el robot se aleja 32,8 centímetros de la trayectoria ideal al cabo de la quinta y última vuelta, marcada en color azul en la figura. Se puede notar que su posición real varía de la ideal por las desviaciones causadas por componentes externas como son los cambios en las velocidades de las ruedas y el rozamiento con la superficie; por otro lado, al no recibir una retro-alimentación de su posición real, el alejamiento es cada vez mayor. Es decir, que cuando se predice pero no se observa, el robot no puede corregir su trayectoria, alejándose paulatinamente de la trayectoria ideal.

En la figura 42 puede verse la orientación del robot a lo largo de la trayectoria. A partir de esta gráfica puede concluirse que la trayectoria mantiene aproximadamente la forma ideal, pero a medida que el robot varía su orientación, al no corregir las velocidades, la trayectoria se ve rotada.

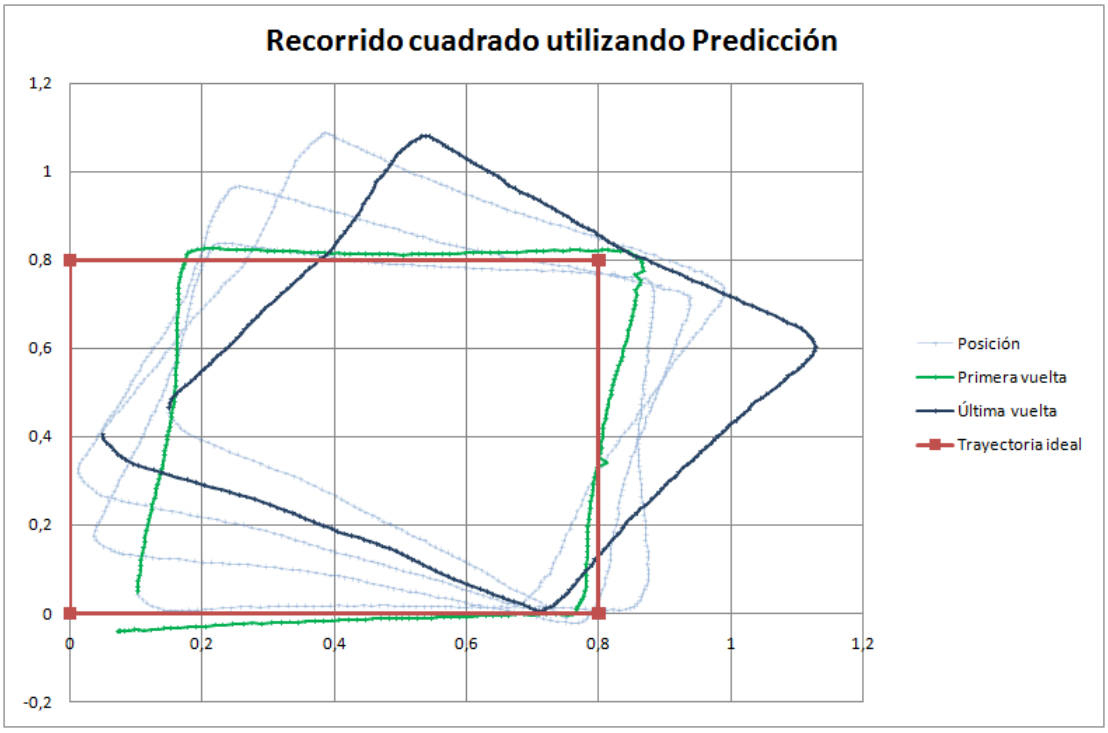

Figura 41: Resultado del recorrido cuadrado utilizando Predicción.

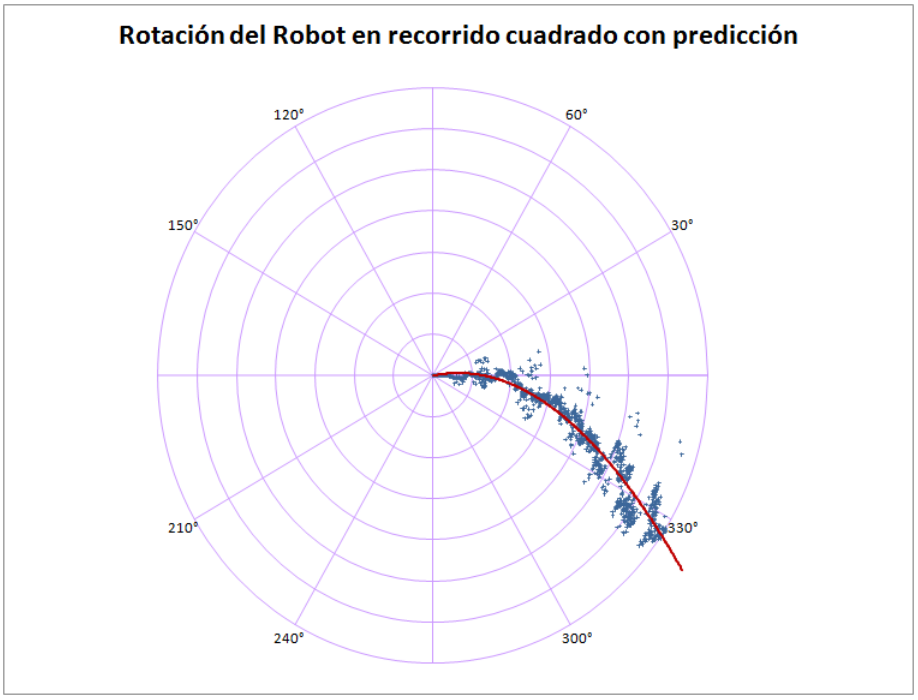

Figura 42: Rotación del robot a lo largo de la trayectoria cuadrada con predicción.

5.4.3. Recorrido cuadrado utilizando Fusión de Sensores

Este experimento, similar a los anteriores, consiste en realizar la misma trayectoria cuadrada, realizando diez vueltas, y utilizando fusión de sensores de tipo selector (ver sección 2.2.3). En la figura 43 se puede observar el resultado de su ejecución. Como ya fue explicado al comienzo de la sección, la fusión considera al sensor *Doraemon*

y a un sensor de predicción. El sensor *Doraemon* es definido como el más confiable de ambos. Se simuló que el sensor *Doraemon* no fuera capaz de registrar posiciones en la parte superior de la trayectoria, p. ej. por la existencia de escasa luz. Se puede observar que cuando el *Doraemon* no sensa más posiciones (punto A en la gráfica), el sensor predictor comienza a registrar la posición. Para ello, la primera predicción es realizada a partir de la última posición sensada por el *Doraemon*. A medida que la predicción avanza, la trayectoria se aleja de la trayectoria ideal hasta el momento (punto B en la gráfica) en el que el *Doraemon* comienza a registrar la posición nuevamente y corrige la trayectoria. La mayor distancia entre la trayectoria realizada respecto de la ideal es de 17,1 centímetros y se produce justamente cuando se utiliza al sensor predictor.

La imagen 44 muestra la distancia del robot a la trayectoria ideal a lo largo de todo el recorrido. Se puede observar que el promedio de la distancia a la trayectoria ideal es de 2,7 centímetros y su desviación estándar es de 2.9 centímetros.

En caso de no utilizarse este tipo de fusión de sensores, utilizando únicamente *Doraemon*, cuando el robot alcanzara la zona de oscuridad, se podrían seguir dos estrategias: mantenerse estático hasta que la cámara comience a captarlo, o continuar avanzando. Ninguna de estas estrategias permitirían al robot llegar a su destino, ya que la cámara no capta su posición en la zona de oscuridad.

Como conclusión, este tipo de fusión de datos sensoriales muestra un correcto funcionamiento, en los que existen zonas de oscuridad o zonas de oclusión entre objetos y robots. Este tipo de fusión de sensores podría ser reforzado mediante técnicas de aprendizaje para corregir el funcionamiento del algoritmo de predicción mientras el *Doraemon* se encuentra disponible (por ejemplo ajustando los parámetros del modelo del robot o considerando las mediciones del Doraemon para corregir las posiciones predecidas), y utilizar predicción en zonas de oscuras o de oclusión.

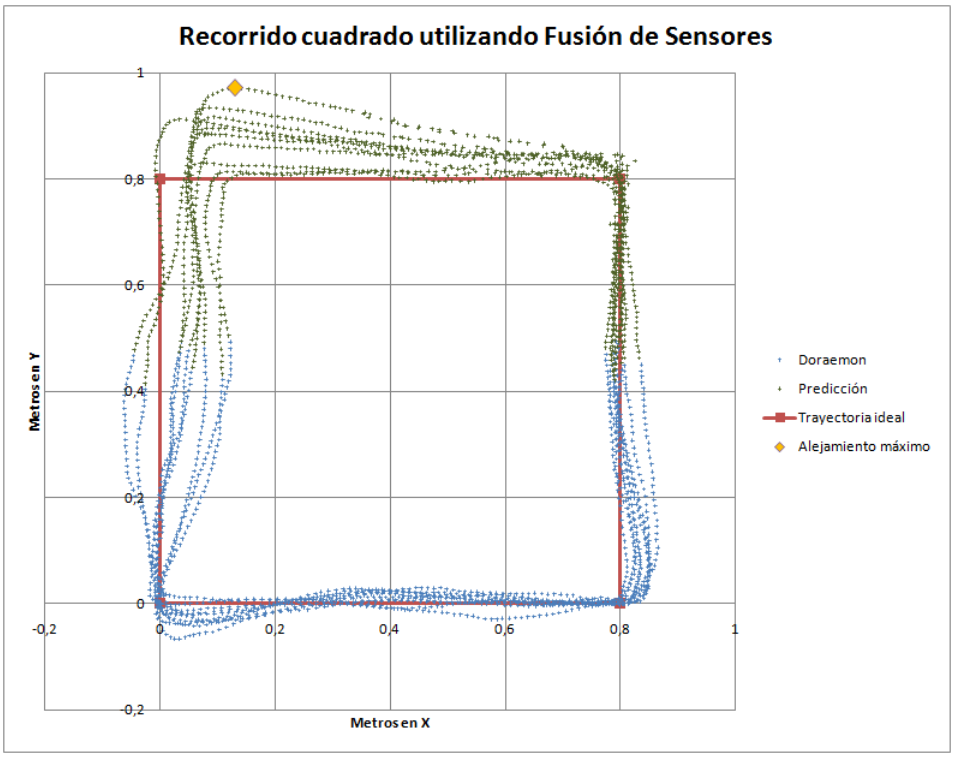

Figura 43: Resultado del recorrido cuadrado utilizando Fusión de sensores.

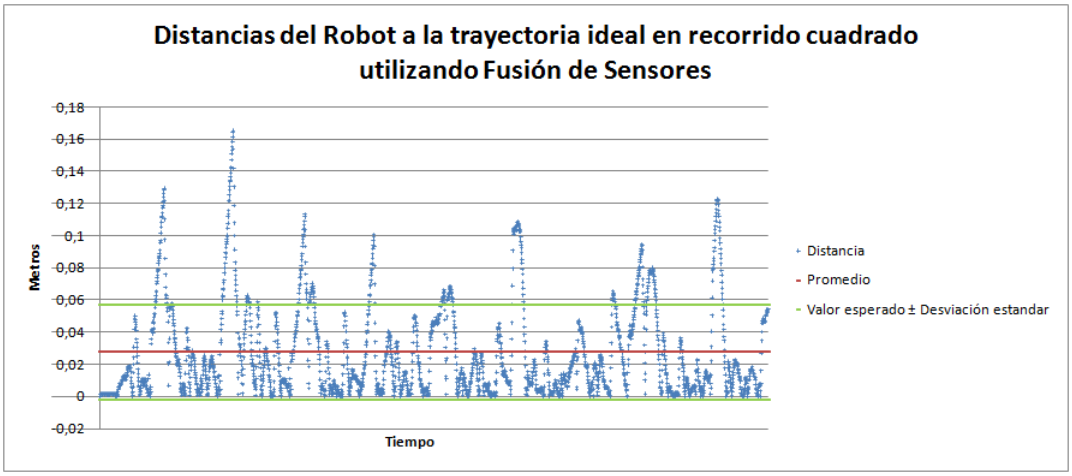

Figura 44: Distancias del robot omnidireccional a la trayectoria cuadrada ideal utilizando Fusión de Sensores.

5.4.4. Recorrido dentado utilizando Doraemon

En este experimento el robot realiza una trayectoria dentada determinada por una serie de puntos, durante tres vueltas. El sensado es similar al del primer experimento, utilizándose solamente una cámara global y el *Sistema de Visión Artificial Doraemon* para recibir los datos. Esta prueba tiene como finalidad analizar el comportamiento omnidireccional del robot construído así como verificar que se puedan definir trayectorias complejas. La figura 45 muestra el resultado de la prueba. Tal como se observa, el robot realiza una trayectoria próxima a la ideal ya que se obtuvo una desviación máxima de 6,3 centímetros. La ejecución de este recorrido tardó 66,3 segundos.

La imagen 46 muestra la distancia del robot a la trayectoria ideal a lo largo de todo el recorrido. Como se puede ver, el promedio de la distancia a la trayectoria ideal es de 1,1 centímetros. Además, su desviación estándar es de 1,1 centímetros.

Finalmente, la imagen 47 muestra la orientación del robot a lo largo del recorrido, la cual permanece acotada en el rango de ángulos$(-50°, 1°)$. Puede verse que el ángulo varía en mayor medida respecto al recorrido cuadrado utilizando *Doraemon*, debido a que esta trayectoria tiene más cambios de dirección, acentuándose el efecto del envío secuencial de las velocidades a las ruedas y la fricción de los rodillos sobre la superficie. Cabe destacar que la orientación del robot no se ajusta durante la trayectoria.

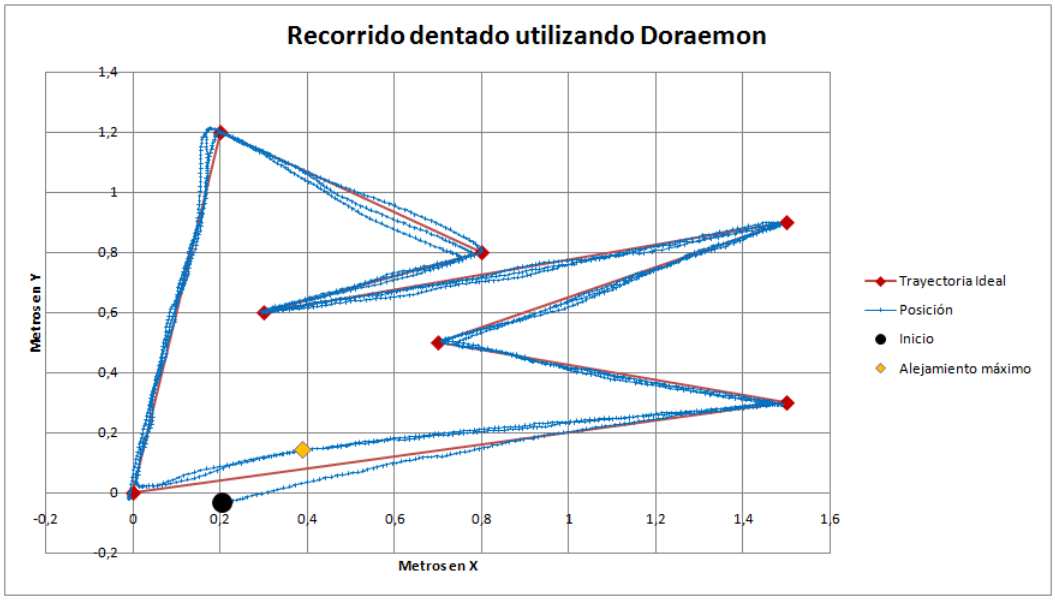

Figura 45: Resultado de la trayectoria dentada utilizando Doraemon.

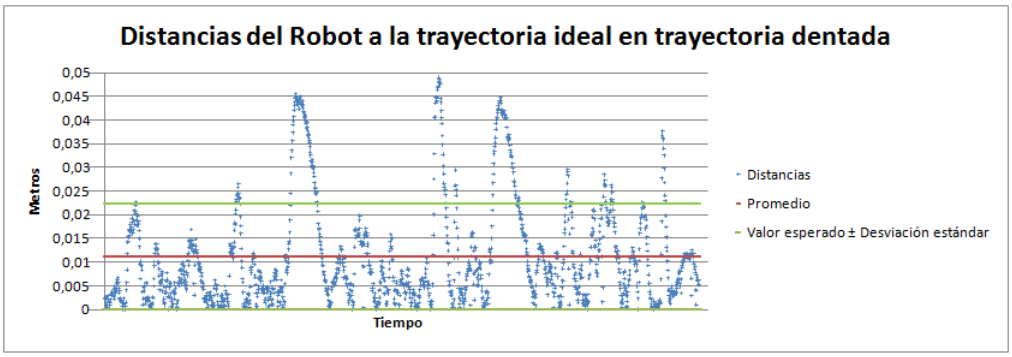

Figura 46: Distancias del robot omnidireccional a la trayectoria dentada ideal.

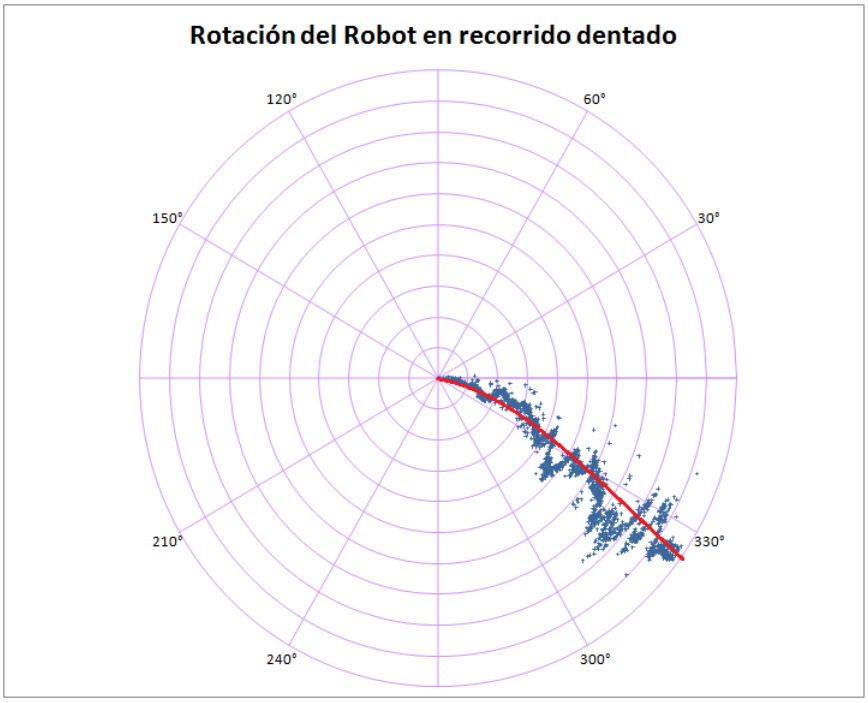

Figura 47: Rotación del robot omnidireccional a lo largo de la trayectoria dentada.

5.4.5. Recorrido dentado de un robot de dos ruedas utilizando Doraemon

Como extensión al cuarto experimento, se ejecutó la misma trayectoria del experimento anterior pero haciendo uso de un robot de dos ruedas capaz de desplazarse en ambos sentidos perpendiculares al eje de las ruedas. La estrategia de navegación implementada para este robot continuamente adapta la orientación hacia la dirección deseada, a fín de que el robot se encuentre en constante movimiento.

En la figura 48 se presenta el resultado. Se puede observar como su trayectoria no se ajusta a la ideal tal como en el caso de utilizarse una robot omnidireccional. El desajuste se debe a que se requiere que el robot esté en constante desplazamiento y por lo tanto debe adaptarse a la nueva dirección solicitada. En este caso, la desviación máxima de la trayectoria realizada respecto de la ideal fue de 10,9 centímetros, aunque sería aún mayor de no utilizarse la marcha atrás.

La imagen 49 muestra la distancia del robot a la trayectoria ideal a lo largo de todo el recorrido. Como se puede ver, el promedio de la distancia a la trayectoria ideal es de 3,2 centímetros. Además, su desviación estándar es de 2,4 centímetros.

La ejecución de la trayectoria insumió 80,6 segundos a diferencia de los 66,3 segundos que tardó la ejecución de la misma trayectoria para el robot omnidireccional. Esta diferencia es explicada por la no-omnidireccionalidad del robot de dos ruedas que debe ajustar su orientación mientras se desplaza.

Finalmente, la imagen 50 muestra la orientación del robot a lo largo del recorrido. Como puede observarse, el robot debe rotar para lograr realizar la trayectoria, debido a que no es omnidireccional. Esto demuestra que el desplazamiento lineal del robot de dos ruedas es dependiente de su orientación.

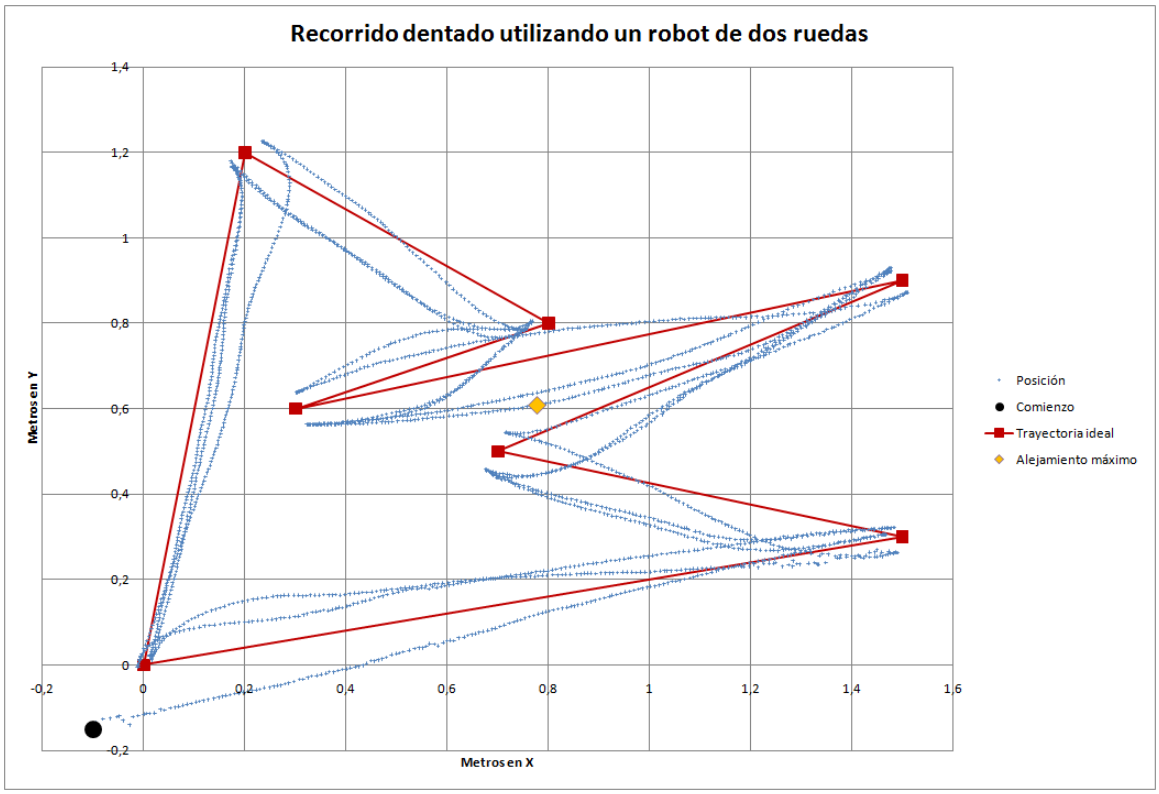

Figura 48: Resultado de la trayectoria dentada utilizando un robot de dos ruedas.

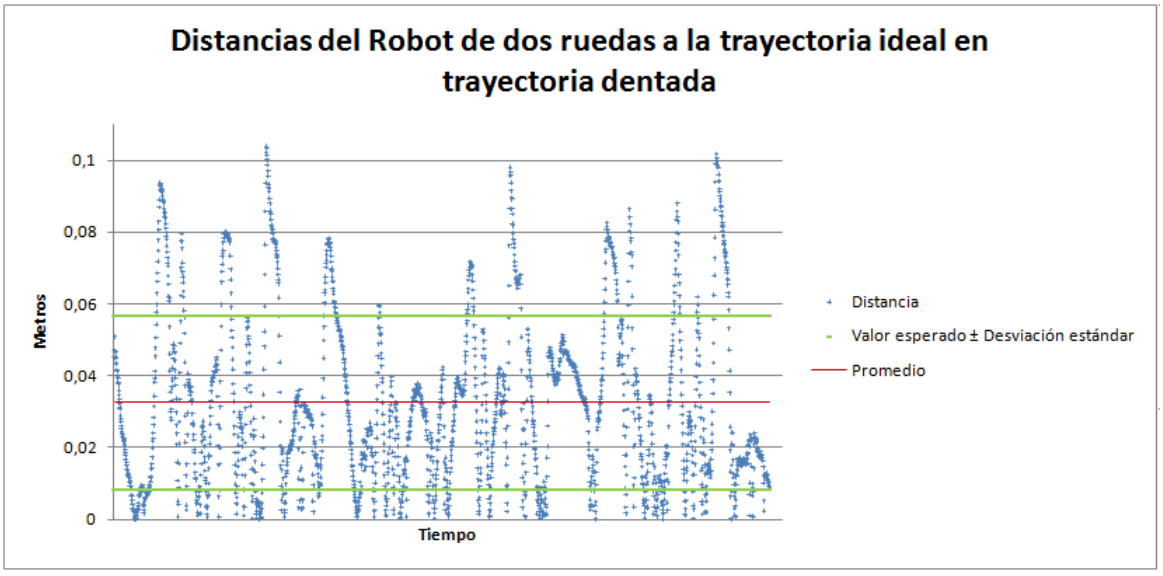

Figura 49: Distancias del robot de ruedas a la trayectoria cuadrada ideal.

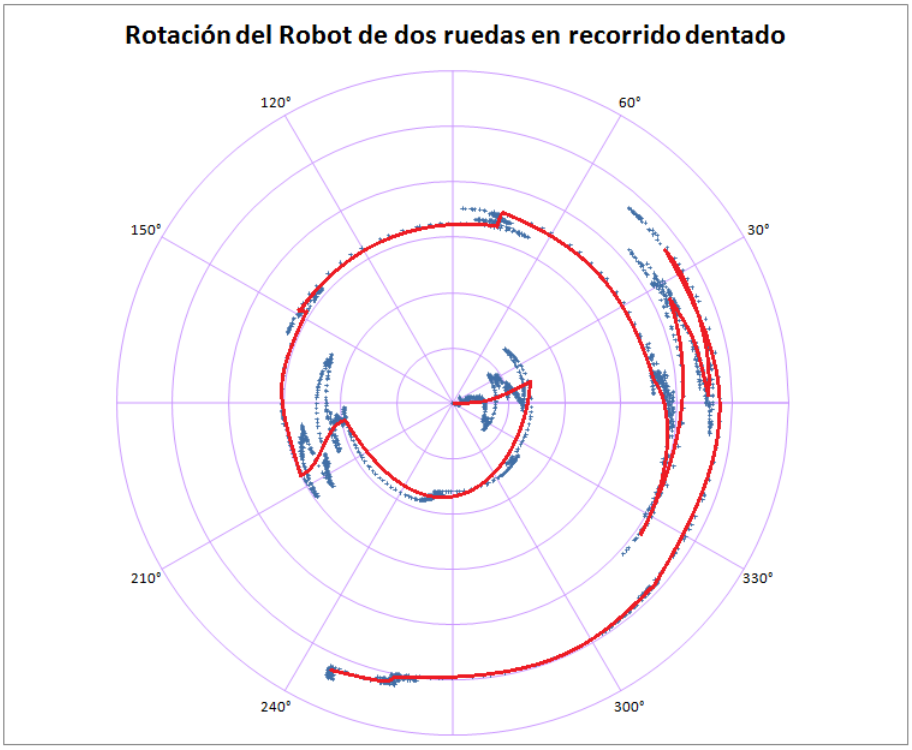

Figura 50: Rotación del robot de dos ruedas a lo largo de la trayectoria dentada.

6. Conclusiones

El desarrollo de este proyecto permitió investigar nuevos aspectos de la robótica como son los robots omnidireccionales y fusión de sensores, así como desarrollar una aplicación para control, posicionamiento (beneficiándose de las ventajas de la fusión de sensores) y planificación de comportamientos de robots móviles.

Se construyó un robot omnidireccional con la mínima cantidad de ruedas posibles, con todos los beneficios de omnidireccionalidad buscados. Su construcción permitió realizar pruebas que mostraran la posibilidad de realizar movimientos omnidireccionales . Permitió además estudiar los distintos tipos de ruedas existentes, así como las diversas estructuras posibles a utilizar pudiendo discriminar entre las distintas aplicaciones de éstas. Se obtuvo una mayor comprensión de los cálculos matemáticos necesarios para derivar las ecuaciones de control de los robots omnidireccionales en general. Se detectaron buenos resultados en el control del robot y el seguimiento de trayectorias definidas, incluyendo trayectorias que requerían cambios bruscos de dirección comprobando la superioridad, en estos escenarios, de estos robots frente a los convencionales.

Las características anteriores convierten a los robots omnidireccionales en una atractiva opción para utilizarlos en un equipo de fútbol de robots por su gran movilidad e independencia entre la velocidad lineal y angular de éstos.

En lo que respecta a la fusión de sensores, la investigación de las técnicas comúnmente utilizadas permitió comprender el concepto de fusión de datos sensoriales y distintos modelos, a fin de implementar motores para representar del mundo. Se realizó una revisión bibliográfica de los distintos modelos para esta implementación y finalmente se consideró una de estas opciones para el desarrollo de un módulo de fusión de sensores a ser utilizado en el sistema *EasyRobots*. Este módulo se diseño de forma que fuera extensible para incorporar nuevas estrategias de fusión, planteadas como trabajos a futuro. Para comprobar su extensibilidad se implementaron dos estrategias de fusión de sensores: uno que selecciona la medición realizada por el sensor más confiable de los sensores disponibles y otro que promedia, de forma ponderada según la confianza, los valores medidos en los distintos sensores.

Se comprobó que el algoritmo de fusión de sensores por promedios ponderados no era adecuado para la fusión entre *Doraemon* y predicción debido a que *Doraemon* es suficientemente confiable. Este algoritmo es recomendable para fusionar sensores similares, por ejemplo dos sistemas *Doraemon* cubriendo la misma área pero con distintas condiciones de luz. Esto llevó a desarrollar un algoritmo de fusión de sensores selector demostrando la validez del modelo de fusión desarrollado y la facilidad de extender el sistema. Se comprobó que este último algoritmo es recomendable en sistemas donde los sensores tienen distinta confianza dependendiendo de la situación.

Se implementó *EasyRobots,* para el control de sistemas multi-robots que incorporara los conocimientos adquiridos a lo largo del proyecto. Fue diseñado para poder generar sistemas robóticos integrando componentes prediseñados, mediante la definición de archivos de configuración en formato XML, abstrayendo tres componentes básicos de esos sistemas, a notar: sensado, planificación y actuación en el mundo físico. De la definición de esta herramienta surgen dos perfiles posibles para los agentes involucrados con la herramienta: *desarrollador* y *usuario*. El perfil de *desarrollador* puede diseñar y programar nuevos componentes a ser utilizados en los sistemas robóticos por los usuarios. Los *usuarios* pueden definir sistemas robóticos utilizando los componentes definidos, editando su configuración en archivos XML.

La herramienta permite modelar diversos componentes del mundo físico, como son robots, sensores, comportamientos y sistemas de comunicación con dichos robots. Para el modelado de robots, se crearon los componentes específicos necesarios para incluir cualquier robot móvil omnidireccional de tres o más ruedas omnidireccionales y robots de dos ruedas no omnidireccionales, incluyendo los sistemas de comunicación con éstos, así como el modelado de otras entidades, dinámicas o estáticas, como pueden ser pelotas, robots oponentes u obstáculos fijos presentes en el escenario.

Dentro de los sensores que se pueden utilizar, se encuentran las herramientas de visión global (*Doraemon* [17]) y predicción de nuevas posiciones basada en los comandos enviados a las ruedas, simulando el comportamiento de codificadores ópticos presentes en los motores de algunos robots. Sobre estos sensores, se permite implementar diversos algoritmos de fusión de datos sensoriales basados en promedios ponderados o selectores de sensores.

De igual forma, se pueden utilizar tres tipos de comportamientos: Campos potenciales, seguimiento de marcas y comportamientos genéricos definidos en lenguaje *Java*. Los campos potenciales permiten definir comportamientos complejos a partir de primitivas que pueden ser atractores o repulsores; el comportamiento de seguimientos de marcas, o *Waypoints*, permite definir circuitos que el robot debe recorrer; la utilización de comportamientos definidos en lenguaje Java permite que usuarios con conocimientos de programacion y conocimientos en robótica definan comportamientos complejos.

El diseño del sistema lo hace extensible, de forma de interactuar con otras aplicaciones y agregar nuevas funcionalidades. Esto permite interactuar con otros *Sistemas de Sensado*, como pueden ser otros tipos de sensores (ultrasonido, contacto, infrarrojo), simuladores robóticos o sistemas de posicionamiento global (GPS); permite además

agregar *Sistemas de Actuación* de forma de controlar nuevos tipos de robots físicos o lógicos (p. ej., simuladores).

6.1. Extensiones y trabajo a futuro

En esta subsección se detallan los trabajos a futuro para mejorar la aplicación desarrollada ampliando la gama de funcionalidades ya brindadas. Asimismo se sugieren posibles extensiones y modificaciones de los subsistemas que puedan ser utilizadas en futuras aplicaciones.

Transformación de sistemas de coordenadas de *Sistemas de Sensado*

Actualmente, los datos recibidos de cada *Sistema de Sensado* son directamente introducidos al modelo. Esto implica que los distintos *Sistemas de Sensado* deben utilizar el mismo sistema de coordenadas y unidades que el de la aplicación.

Se sugiere implementar una etapa de transformación de los datos recibidos desde cada *Sistema de Sensado* (expresados en coordenadas y unidades internas de ese sistema) al sistema de coordenadas utilizado internamente a la aplicación. Esta situación se puede observar en la figura 51.

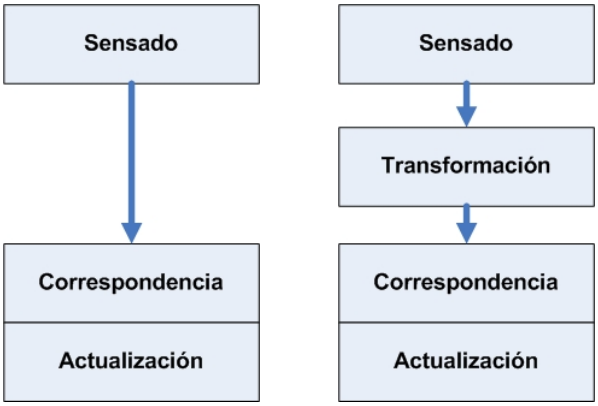

Figura 51: Etapas actuales (izq.) y futuras mejoras posibles (der.) de las etapas del sensado de datos e ingreso de éstos a la aplicación.

Por ejemplo, para el *Sistema de Visión Artificial Doraemon* se puede transformar el sistema de coordenadas mediante una transformación lineal que puede definirse en el archivo de definición del sistema robótico. Dicha transormación varía luego de cada calibración del Doraemon, por lo que no puede ser especificada de forma definitiva. Actualmente se debe corresponder el sistema de coordenadas de Doraemon con el del sistema robótico para un correcto funcionamiento.

Nuevos algoritmos de fusión de sensores

Se propone como mejora la implementación de otros algoritmos de fusión de sensores. Actualmente la aplicación cuenta con dos algoritmos; uno que realiza un promedio ponderado de los datos recibidos de dichos sensores y otro que selecciona la medida más confiable de los sensores disponibles. Se recomienda ver la sección 2.2 y el documento *Posicionamiento y Sensor Data Fusion* [30] para profundizar en los principios a tener en cuenta al realizar un algoritmo de fusión de sensores así como las técnicas implementables. Una opción en este sentido sería implementar algoritmos basados en *Filtros de Kalman* [21].

Nuevos Sistemas de Sensado

Actualmente la aplicación cuenta con la comunicación desde un único *Sistema de Sensado* externo. Dicha comunicación permite recibir datos del *Sistema de Visión Artificial Doraemon,* el cual registra el mundo físico a través de una cámara de video. Como ya fue explicado en la sección 4.7.3, se pueden desarrollar fácilmente nuevos componentes que permitan comunicarse con otros *Sistemas de Sensado*. Esto implica que antes de desarrollar dichos componentes se deba conocer el medio de comunicación y el formato de los mensajes que se reciben del sistema.

[21] Algoritmo que sirve para estimar el estado de un sistema basado en una serie de medidas con ruido[31].

Como ejemplo, se sugiere implementar la comunicación con el Sistema *ERGO*[22] [17].

Mejora en cálculo de confianza de *Doraemon*

Actualmente, la confianza asociada a las mediciones de este sistema no varía durante la ejecución. Se propone que la confianza sea variable entre mediciones, esto es, la confianza disminuye tendiendo a cero mientras no se reciban nuevos datos. Una vez que se recibe una nueva medición, la confianza se reestablece al máximo definido.

Soporte para más tipos de robots

La aplicación soporta el modelado de robots móviles omnidireccionales de tres o más ruedas. El diseño de la aplicación, de acuerdo a la sección 4.7.5, permite que se incorporen implementaciones de otros tipos de robots, incluso otros tipos de no omnidireccionales, por ejemplo aquellos con dirección tipo *Ackerman*[23].

Cabe destacar que si el robot a desarrollar no es omnidireccional, el sistema prevé esto permitiendo al desarrollador retornar la velocidad lineal factible del robot, al momento de solicitar las velocidades angulares de las ruedas.

Otra mejora que se sugiere es el soporte para nuevos *Sistemas de Actuación*. La utilización de nuevos robots construídos con diferentes motores y bloques de control diferentes a los *Lego Mindstorms NXT* implica que el medio de comunicación y el formato de mensajes pueda cambiar respecto a los ya implementados. Por esta razón, se deben crear nuevos componentes que se comuniquen con los robots.

Interfaz gráfica de usuario y para la creación de sistemas robóticos

Actualmente la creación de sistemas robóticos se realiza mediante la creación de archivos en formato *XML*. Para facilitar la creación de sistemas, se sugiere implementar una interfaz gráfica que mediante la utilización de bloques predefinidos permita definir el comportamiento de los robots. Dicha aplicación debe generar archivos en formatos soportados por *EasyRobots*.

Además, se sugiere la implementación de interfaces más amigables para el uso de EasyRobots en lo que respecta a la carga de sistemas robóticos, activación de comportamientos y demás funcionalidades del usuario.

Control de la correctitud del archivo XML

En la versión de la aplicación desarrollada, el control que se realiza sobre el archivo *XML* es únicamente que siga el esquema definido por el archivo de esquema *XSD*[24]. No se realizan otros controles sobre la correctitud del sistema robótico definido, como podría ser el caso de un comportamiento referenciando a un robot no existente en el sistema.

Por ésto, se recomiendan realizar los controles mencionados. Éstos pueden ser internos al sistema, por ejemplo, realizando un control sobre la definición de todas los componentes referenciados en el sistema o inherentes al formato del archivo de definición utilizado. Un ejemplo de este último tipo de control es el uso de las herramientas *ID* e *IDRef* [32] que provee *XML*, que permiten referenciar desde una componente a otras definidas en mismo archivo, y su correcto uso se verifica automáticamente al validar el archivo.

Mejoras en la definición de comportamientos

Comportamiento anidados Actualmente la definición de comportamientos no permite utilizar resultados de otros comportamientos. El anidamiento podría ser útil en sistemas robóticos extensos, en los cuales se requiera reutilizar comportamientos definidos, por ejemplo aquellos comunes en competencias, como el caso de evitar choques con los bordes de una cancha de fútbol de robots. Existen ejemplos de distintos modelos para anidamiento de comportamientos como pueden ser aquellos basados la *Arquitectura de Subsunción de Brooks*[25] o en máquinas de estados.

Se sugiere extender *EasyRobots* permitiendo el uso de comportamientos anidados y compuestos.

[22] Versión mejorada de su predecesor *Doraemon*, mitigando ciertas debilidades, a notar: *Doraemon* maneja de forma precaria la oclusión entre objetos y su correcto funcionamiento depende significativamente de una buena iluminación.

[23] Sistema de dirección utilizado en los vehículos de transporte convencionales (automóviles, colectivos).

[24] Lenguaje de esquema utilizado para describir la estructura y las restricciones de los contenidos de los documentos XML de una forma muy precisa, más allá de las normas sintácticas impuestas por el propio lenguaje XML.

[25] Arquitectura basada en niveles de competencia donde cada nivel es un conjunto de comportamientos que cumplen un objetivo. A mayor nivel, se definen comportamientos más específicos.[8]

Comportamientos con datos de posiciones relativas En los comportamientos definidos actualmente, las posiciones utilizadas en la evaluación pueden ser estáticas o iguales a la posición de una entidad, no pudiendo definirse un punto relativo a la posición de una entidad.

Una mejora posible es brindar esta posibilidad, por ejemplo, poder especificar la posición "cinco centímetro a la izquierda del robot A", utilizando fórmulas matemáticas. Se encuentran disponibles librerías que evalúan expresiones matemáticas genéricas, como lo hace por ejemplo la librería *MathEclipse* [33].

Apéndice A

Archivo de definición en XML del Robot repartidor.

```
<application
    xmlns="http://www.example.org/RobotApp"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://www.example.org/RobotApp RobotApp.xsd ">
    <strategy
    xsi:type="PotentialFieldsStrategy"
    id="1">
        <controlledRobot>idRobot</controlledRobot>
        <field
            xsi:type="FieldCircle"
            maxWidth="5"
            minWidth="0.1"
            powEnd="-15"
            powStart="-15">
            <dot>
                <position>150</position>
                <position>150</position>
            </dot>
        </field>
        <field
            xsi:type="FieldCircle"
            maxWidth="5"
            minWidth="0.1"
            powEnd="-3"
            powStart="-15">
            <dot>
                <position>140</position>
                <position>70</position>
            </dot>
        </field>
        <field
            xsi:type="FieldCircle"
            maxWidth="5"
            minWidth="0.1"
            powEnd="-3"
            powStart="-15">
            <dot>
                <position>130</position>
                <position>110</position>
            </dot>
        </field>
        <field
            xsi:type="FieldSemiPlane"
            angle="90"
            maxWidth="5"
            minWidth="0.1"
            powEnd="-3"
            powStart="-15">
            <dot>
                <position>120</position>
                <position>220</position>
            </dot>
```

```
            <dot>
                <position>0.0</position>
                <position>150</position>
            </dot>
        </field>
    </strategy>
    <entity
        xsi:type="OmnidirectionalRobot"
        id="idRobot"
        positionId="sensor1"
        maxwheelrotation="15">
        <physicalRobot
            xsi:type="NXTRobot"
            physicalAddress="00:16:53:01:4E:3B"
            physicalControllerId="nxtController"/>
        <wheel
            ...
        />
        <wheel
        <wheel
            ...
        />
        <wheel
            ...
        />
    </entity>
    <robotcontroller
        id="nxtController"
        port="1234"
        xsi:type="NXTController"
        ip="192.168.32.29" />
        <sensor
            xsi:type="DoraemonClient"
            confidence="1"
            id="sensorD"
            period="50"
            maxMessageLength="512"
            portServer="6363">
            <idDoraemonToEntity
                string1="OBJ1"
                string2="idRobot" >
            </idDoraemonToEntity>
        </sensor>
        <sensor
            xsi:type="PredictionSensor"
            managedEntityId="idRobot"
            referenceSensorId="sensor1"
            id="sensorP"
            confidence="0.3"
            period="70" />
        <sensor
            xsi:type="AverageSensorFusion"
            confidence="0.0"
            id="SF1"
            managedEntityId="idRobot"
            period="90" >
```

```
            <fusionedSensorId>sensorD</fusionedSensorId>
            <fusionedSensorId>sensorP</fusionedSensorId>
        </sensor>
</application>
```

Archivo de definición en XML del Circuito dentado.

```
    <application
        xmlns="http://www.example.org/RobotApp"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:schemaLocation="http://www.example.org/RobotApp RobotApp.xsd ">
        <strategy
            id="waypointsStrat"
            xsi:type="WaypointsStrategy"
            epsilon="0.1"
            laps="1"
            loop="false"
            speed="0.5">
            <controlledRobot>controlledRobot</controlledRobot>
            <waypoint>
                <position>150</position>
                <position>30</position>
            </waypoint>
            <waypoint>
                <position>70</position>
                <position>50</position>
            </waypoint>
            <waypoint>
                <position>150</position>
                <position>90</position>
            </waypoint>
            <waypoint>
                <position>30</position>
                <position>60</position>
            </waypoint>
            <waypoint>
                <position>80</position>
                <position>190</position>
            </waypoint>
            <waypoint>
                <position>20</position>
                <position>120</position>
            </waypoint>
        </strategy>
        <entity
            xsi:type="OmnidirectionalRobot"
            id="idRobot"
            positionId="sensor1"
            maxwheelrotation="15">
            <physicalRobot
                xsi:type="NXTRobot"
                physicalAddress="00:16:53:01:4E:3B"
                physicalControllerId="nxtController"/>
                <wheel
                    ...
                />
```

```
            <wheel
                ...
            />
            <wheel
                ...
            />
    </entity>
    <robotcontroller
        id="nxtController"
        port="1234"
        xsi:type="NXTController"
        ip="192.168.32.29" />
    <sensor
        xsi:type="DoraemonClient"
        confidence="1"
        id="sensorD"
        period="50"
        maxMessageLength="512"
        portServer="6363">
        <idDoraemonToEntity
            string1="OBJ1"
            string2="idRobot" >
        </idDoraemonToEntity>
    </sensor>
</application>
```

Glosario

Bluetooth: Especificación industrial para Redes Inalámbricas de Área Personal (*WPAN*) que posibilita la transmisión datos entre diferentes dispositivos mediante un enlace por radiofrecuencia.

CAFR: Acrónimo de Campeonato Argentina de Fútbol de Robots.

Cinemática: Área de estudio de las leyes del movimiento de los cuerpos sin tener en cuenta las causas que lo producen limitándose, esencialmente, al estudio de la trayectoria en función del tiempo.

FIRA*:* Acrónimo inglés de *Federation of International Robot-soccer Association*. La FIRA se creó el 5 de junio de 1997 durante el torneo MiroSot97 (*Micro-robot Soccer Tournament*) realizado en Daejeon, Corea. Desde entonces todos los años se organizan dos eventos de nivel internacional: FIRA *Robot World Cup* y FIRA *Robot World Congress*, con el propósito de promover la investigación en sistemas robóticos autónomos entre otros.

FRUTo: Proyecto de grado de la Facultad de Ingeniería en el cuál se desarrolló un equipo de fútbol de robots que se desempeñó en la liga simulada de la FIRA.

Fusión de sensores: Combinación de datos sensoriales derivados de un conjunto dispar de sensores, de forma de obtener información sensorial mejor que la que se podría obtener al utilizar los sensores individualmente. El término *mejor* puede referirse -dependiendo del caso- a más preciso, más completo o en algunos casos generar un dominio de percepción distinto al de los sensores fusionados (por ejemplo, fusionar dos imágenes generando una vista estereoscópica, de forma de obtener información de profundidad).

GSim: Acrónimo inglés de *Geometric Simulator* (Simulador Geométrico). Simulador desarrollado en el marco del proyecto FRUTo.

Holonómico: Capacidad de un robot de realizar cualquier cambio de la dirección de desplazamiento de forma instantanea.

Lego Mindstorms NXT: Conjunto de piezas para el rápido prototipado de robots que posee elementos básicos de las teorías robóticas, como la unión de piezas y la programación de acciones.

Lego Digital Designer: Programa informático que puede ser utilizado para diseñar modelos *Lego*. Permite a los usuarios generar instrucciones de construcción haciendo uso del programa o mediantes páginas *web*.

MAS: Acrónimo inglés de *Multi-Agent System*. Refiere a sistemas donde se cuentan con multiples agentes robóticos.

MINA: El Grupo MINA (Network Management - Articial Intelligence), es un grupo de investigación perteneciente al Instituto de Computación de la Facultad de Ingeniería.

Omnidireccional: Capacidad de un robot de realizar cualquier cambio de la dirección de desplazamiento de forma instantanea.

RoboCup: Originalmente llamada *Robot World Cup Initiative* es una iniciativa internacional de investigación y educación. En ese sentido intenta fomentar la investigación en Inteligencia Articial y robótica proporcionando un problema estándar donde una amplia gama de tecnologías puede ser integrada y examinada, así como promover proyectos de educación integrada.

Sistema de Actuación: Sistema de *software* que permite enviar comandos a actuadores físicos. Posee los datos necesarios para establecer la comunicación con estos actuadores.

Sistema de Sensado: Sistema de *software* que obtiene datos sensoriales del mundo físico, transformándolos en una representación interna. Posee los datos necesarios para establecer la comunicación los sensores y como procesar la información sensada.

Referencias

[1] Mark West and Haruhiko Asada. International conference on robotics and automation. In *Design of a Holonomic Omnidirectional Vehicle*, volume Vol. 3, pages 97–103. Nice, France, May 1992.

[2] Victor Grosu Ioan Doroftei and Veaceslav Spinu. *Bioinspiration and Robotics: Walking and Climbing Robots*, chapter 29 Omnidirectional Mobile Robot - Design and Implementation, page 544. I-Tech Education and Publishing, Technical University of Iasi, Romania, September 2007. ISBN 9783902613158.

[3] A. El-Shenawy, A. Wellenreuther, A.S. Baumgart, and E. Badreddin. Comparing different holonomic mobile robots. In *Proc. ISIC Systems, Man and Cybernetics IEEE International Conference on*, pages 1584–1589, October 2007. ISBN 9781424409914.

[4] Chuntao Leng and Qixin Cao. Velocity analysis of omnidirectional mobile robot and system implementation. In *Proc. 2nd IEEE International Conference on Automation Science and Engineering CASE '06*, pages 81–86, October 2006. ISBN 1424403103.

[5] Robot Store. Robot store hong kong. Retrieved 15 November 2009 http://www.robotstorehk.com/gearboxes/gearboxes.html.

[6] S.L. Dickerson and B.D. Lapin. Control of an omni-directional robotic vehicle with mecanum wheels. In *Proc. Vol.1., NTC '91. National Telesystems Conference*, pages 323–328, March 1991. ISBN 0780300629.

[7] Amir Abdollahi Hooman Aghaebrahimi Samani and Saeed Ziaee Rad Hossein Ostadi. Design and development of a comprehensive omni directional soccer player robot. In *International Journal of Advanced Robotic Systems*, volume 6. IN-TECH, 2004. ISSN 17298806.

[8] Robin R. Murphy. *Introduction to AI Robotics*. MIT Press, Cambridge, MA, USA, November 2000. ISBN 0262133830.

[9] James L. Crowley and Yves Demazeau. Principles and techniques for sensor data fusion. *Signal Process.*, Vol. 32(Num. 1-2):5–27, May 1993. ISSN 0165-1684.

[10] The Lego Group. Lego digital designer. Retrieved July 2009 http://ldd.lego.com/.

[11] School of Robotics. Retrieved on October 2008 http://www.schoolofrobotics.com/.

[12] Kornylak Corporation. Omni wheels for conveyors, multidirectional, powered & skate. Retrieved October 2008 http://www.omniwheel.com/.

[13] Michael A. Goodrich. Retrieved on October 2009 http://borg.cc.gatech.edu/ipr/files/goodrich_potential_fields.pdf.

[14] Darrick Addison IBM. Introduction to robotics technology - robots marching into open source. Retrieved on October 2009 - http://www.ibm.com/developerworks/linux/library/l-rob.html, October 2009.

[15] RoboCup. Robocup world championship. Retrieved 19 August 2009 http://www.robocup.org/.

[16] Mindstorms. Mindstorms nxt kit. Retrieved April 2008 http://mindstorms.lego.com/Products/Default.aspx.

[17] John Anderson and Jacky Baltes. A pragmatic global vision system for educational robotics. In *Robots and Robot Venues: Resources for AI Education*, AAAI Spring Symposium Series, pages 1–6. Stanford, CA, March 2007.

[18] Rafael Sisto and Santiago Martínez. Estado del arte. Instituto de Computación, Facultad de Ingeniería, Universidad de la República, Uruguay, March 2009.

[19] Alexander Gloye Förster Raul Rojas. Holonomic control of a robot with an omni-directional drive. In *Künstliche Intelligenz*, volume 20. BöttcherIT-Verlag, July 2006.

[20] Spenko M. Yu, H. and S. Dubowsky. Omni-directional mobility using active split offset castors. *ASME Journal of Mechanical Design*, Vol. 126(Num. 5):p. 822–829, September 2004.

[21] Jae-Bok Song and Kyung-Seok Byun. Design and control of an omnidirectional mobile robot with steerable omnidirectional wheels. *Pro Literatur Verlag, Germany / ARS, Austria*, Vol. 12:pages 223–240, December 2006. ISBN 386611284X.

[22] J. Gu, M. Meng, A. Cook, and P.X. Liu. Sensor fusion in mobile robot: some perspectives. In *Proc. 4th World Congress on Intelligent Control and Automation*, pages 1194–1199 vol.2, 2002. ISBN 0780372689.

[23] Grupo MINA. Network management - artificial intelligence. Retrieved 18 May 2009 http://www.fing.edu.uy/inco/grupos/mina/index.html.

[24] Rafael Sisto and Santiago Martínez. Página principal del proyecto de grado: Control y comportamiento de robots omnidireccionales. Retrieved on August 2009 http://www.fing.edu.uy/inco/grupos/mina/pGrado/omnidireccionales.html.

[25] Philippe E. Hurbain. Nxt motor internals. Retrieved October 2009 http://philohome.com/nxtmotor/nxtmotor.htm.

[26] Rafael Sisto and Santiago Martínez. Descripción de arquitectura de software. Technical report, Instituto de Computación, Facultad de Ingeniería, Universidad de la República, Uruguay, 2009.

[27] Rafael Sisto and Santiago Martínez. Manual de uso. Technical report, Instituto de Computación, Facultad de Ingeniería, Universidad de la República, Uruguay, 2009.

[28] Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addisson-Wesley, Toronto, Ontario. Canada, November 1994. ISBN 0201633612.

[29] Apache Software Foundation. Logging services. Retrieved August 2009 http://logging.apache.org/log4j/1.2/manual.html.

[30] Rafael Sisto and Santiago Martínez. Posicionamiento y sensor data fusion. Technical report, Instituto de Computación, Facultad de Ingeniería, Universidad de la República, Uruguay, 2009.

[31] G. Welch and G. Bishop. An introduction to the Kalman filter. Technical report, University of North Carolina (Chapel Hill), Chapel Hill, NC, July 2006.

[32] W3C. Extensible markup language (xml). Retrieved 11 August 2009 http://www.w3.org/XML/.

[33] MathEclipse. Matheclipse parser. Retrieved 31 October 2009 http://www.matcheclipse.org.