DIVISIÓN DE ESTUDIOS DE POSGRADO E INVESTIGACIÓN

"DISEÑO Y DESARROLLO DE UNA PLATAFORMA MULTI-ROBOT, BASADO EN LA TECNOLOGÍA LEGO MINDSTORMS"

TESIS

PARA OBTENER EL GRADO DE

MAESTRO EN CIENCIAS EN CIENCIAS DE LA COMPUTACIÓN

PRESENTA:

ISC. JOSÉ MANUEL ROMERO ORTA

DIRECTOR DE TESIS:

DR. ARTURO HERNÁNDEZ RAMÍREZ

CO-DIRECTOR DE TESIS:

DR. JOSÉ GABRIEL RAMÍREZ TORRES

CD. MADERO, TAMPS., MEX. NOVIEMBRE 2011.

Resumen

La robótica cooperativa móvil en los últimos años ha sido una valiosa herramienta para el desarrollo de plataformas tecnológicas utilizando robots de navegación autónoma. En la actualidad existen un gran número de aplicaciones donde los vehículos autónomos son empleados con éxito: exploración espacial, manejo de explosivos, agricultura, transporte automático en áreas urbanas, sistemas de seguridad, así como exploración en sitios de riesgo para el ser humano.

Para obtener los resultados deseados en cuanto a las acciones de los robots, es importante resolver el problema de la localización y la planeación de movimientos. Para que los robots puedan realizar ciertas tareas a través de sus movimientos es necesario que se conozca su espacio de trabajo, su ubicación y orientación, y la ubicación de los obstáculos para poder generar una trayectoria libre de colisiones.

El objetivo principal de este trabajo es resolver el problema de localización utilizando técnicas de visión por computadora que permitan saber dónde están los robots y poder corregir las trayectorias que estos deben de seguir. Se utilizan los algoritmos Camshift y Contornos de las librerías OpenCv, para obtener las ubicaciones de los robots, el objeto meta y los obstáculos dentro del ambiente conocido. Con estos datos se genera una matriz de ocupación con la cual el algoritmo de Dijkstra proporciona la trayectoria a seguir por cada uno de los robots Lego Mindstorms para llegar a una meta libre de colisiones.

En concreto se propone una arquitectura distribuida de control basada en agentes para la navegación autónoma de vehículos en interiores y especialmente orientada a la localización por visión.

Summary

Cooperative mobile robotics in recent years has been a valuable tool for the development of technology platforms for autonomous navigation using robots. At present there are a large number of applications where autonomous vehicles are used with success: space exploration, handling of explosives, agriculture, auto transport in urban areas, security systems and exploration risk sites for humans.

To get the desired results in terms of the actions of the robots, it is important to solve the problem of localization and planning of movements. If robots can perform certain tasks through their movements need to be aware of your work space, location and orientation, and location of the obstacles to generate a collision-free path.

The main objective of this paper is to solve the location problem using computer vision techniques that allow us to know where the robots and paths to correct these should follow. Algorithms are used Camshift and Contours of the OpenCV libraries to obtain the locations of the robots, the target object and obstacles within the familiar environment. With these data, generates an array of employment to which the Dijkstra algorithm provides the path to follow for each of the Lego Mindstorms robots to achieve a collision-free goal. Specifically, it proposes a distributed control architecture based on agents for autonomous navigation of vehicles and especially indoor localization oriented vision.

Tabla de contenido

Índice de Figuras

Índice de Tablas

1

Introducción

En los últimos años el enfoque de la investigación hacia la robótica cooperativa móvil, ha sido cada vez mayor. Grupos de robots móviles se crean con el objetivo de estudiar cuestiones como la competencia para la obtención de recursos, el origen de la cooperación y la organización del grupo, el aprendizaje de uno mismo y movimiento coordinado. En la actualidad existen un gran número de aplicaciones donde los vehículos autónomos son empleados con éxito: exploración espacial, manejo de explosivos, agricultura, transporte automático en áreas urbanas, sistemas de seguridad, así como exploración en sitios de riesgo para el ser humano.

Un robot móvil que opera en el mundo físico debe ser consciente de su entorno. Una gran parte de este es la conciencia de saber en qué lugar del ambiente está y donde ha estado. La localización de un robot móvil en un entorno conocido es uno de los problemas fundamentales de la robótica móvil, junto con la construcción automática de mapas del entorno. Se han propuesto un gran número de modelos, enfoques y técnicas para resolver ambos problemas. Muchas de estas técnicas presentan soluciones que sólo son útiles en situaciones muy específicas. En los últimos años ha surgido un gran interés en el desarrollo de algoritmos para estimar la localización del robot a partir de los datos percibidos por sus sensores. En el contexto de los robots móviles, el problema general de la localización puede ser formulado dado un modelo del entorno, como una descripción geométrica, un mapa topológico o una rejilla de ocupación. La tarea seria estimar la localización del robot en el modelo basándose únicamente en observaciones efectuadas por el robot. Dichas

observaciones suelen consistir en información de odometría acerca de los movimientos realizados por el robot y en información de distancias a los obstáculos más cercanos obtenidas mediante sensores de alcance (sonar, láser) [Gallardo 1999]. Las observaciones también pueden consistir en el procesamiento de imágenes adquiridas a través de una o varias cámaras, utilizadas como elemento de sensado para cerrar el lazo de control [García 2008], las cuales presentan alternativas muy interesantes para resolver el problema de la localización.

En este trabajo se presenta un sistema de navegación autónomo basado en visión por computadora, para un escenario pequeño. Se propone el uso de la tecnología LEGO Mindstorms NXT que incluye un sensor de ultrasonido y conexión inalámbrica bluetooth, con el propósito de que dicha arquitectura móvil cumpla con el propósito de encontrar la ruta más corta libre de colisiones, desde un inicio hasta una meta.

El objetivo principal de este trabajo es el desarrollo de una plataforma multi-robot, utilizando dos robots Lego; para resolver el problema de la localización se utilizan técnicas de visión por computadora basada en los algoritmos Camshift y Contornos de las librerías de OpenCV.

1.1 Antecedentes de Robótica

La palabra Robot fue utilizada por primera vez en el año de 1921 por el escritor checo Karel Capek (1890-1938) en su obra Rossum´s Universal Robot (R.U.R) presentada en el teatro nacional de Praga [Barrientos, 2007]. Su origen es la palabra eslava robota, que se refiere al trabajo realizado de manera forzada. En esta obra los robots de R.U.R eran máquinas androides fabricadas a partir de la fórmula obtenida por un brillante científico llamado Rossum. Estos robots servían a sus jefes humanos desarrollando todos los trabajos físicos, hasta que finalmente se rebelan contra estos, destruyendo toda la vida humana, a excepción de uno de sus creadores, con la frustrada esperanza de que les enseñe a reproducirse. Pero fue el escritor americano de origen ruso Isaac Asimov (1920-1992) quien dio mayor impulso a la palabra robot. En 1945 publicó en la revista Galaxy Science Fiction, en la que define por primera vez las tres leyes de la robótica [Barrientos, 2007].

1. Un robot no puede perjudicar a un ser humano, ni con su inacción permitir que un ser humano sufra daño.
2. Un robot a de obedecer las órdenes recibidas de un ser humano, excepto si tales órdenes entran en conflicto con la primera ley.
3. Un robot debe de proteger su propia existencia mientras tal protección no entre en conflicto con la primera o segunda ley.

Actualmente, la mayoría de la sociedad asocia la palabra robot con cualquier máquina hecha por el hombre que desarrolle el trabajo y tareas pertenecientes a los humanos.

Para la Asociación Japonesa de Robótica Industrial (JIRA), los robots son dispositivos capaces de moverse de modo flexible análogo al que poseen los organismos vivos, con o sin funciones intelectuales, permitiendo operaciones en respuesta a las órdenes humanas.

Para el Instituto de Robótica de América (RIA), un robot industrial es un manipulador multifuncional y reprogramable diseñado para desplazar materiales, componentes, herramientas o dispositivos especializados por medio de movimientos programados variables con el fin de realizar tareas diversas. Esta última definición es la que más aceptación ha tenido en el ambiente de la robótica.

1.2 Robótica Móvil

Con el propósito de ampliar el espacio de trabajo de un robot es necesario aumentar la movilidad del mismo, por lo tanto se tiene la necesidad de utilizar la robótica móvil. En [Ramírez, 2000] se hace mención de que la característica más remarcable de un robot móvil es evidentemente su medio de locomoción.

1.2.1 Locomoción

Un robot móvil necesita mecanismos de locomoción que le permitan moverse sin límites a través de su medio ambiente. Pero hay una gran variedad de posibles formas de moverse, por lo que el enfoque de selección de un robot de locomoción es un aspecto importante del

diseño del robot móvil. En el laboratorio, hay robots de investigación que pueden caminar, saltar, correr, deslizarse, patinar, nadar, volar, y, por supuesto, correr. La mayoría de estos mecanismos de locomoción han sido inspirados en sus equivalentes biológicos [Siegwart 2004]. Hay sin embargo una acepción, la rueda, que es una invención humana que logra una eficiencia extremadamente alta en la tierra plana. Este mecanismo no es completamente ajeno a los sistemas biológicos, la forma de caminar con dos piernas, se puede aproximar por un polígono móvil con lados iguales en longitud a la duración del paso. A medida que el paso disminuye, el polígono se aproxima a un círculo o rueda (Figura 1.1). Pero la naturaleza no desarrolló completamente una rotación activamente potente, la cual posee la tecnología de la locomoción de ruedas.

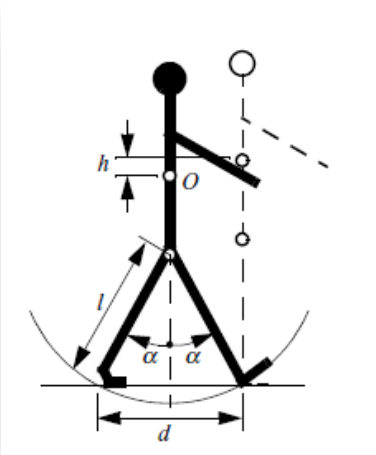


Figura 1.1 Sistema de caminar de un humano. Tomado de [Siegwart 2004].

La eficiencia de la locomoción de ruedas depende en gran medida de las cualidades ambientales, especialmente de la dureza y lo plano del suelo. Mientras que la eficiencia de la locomoción de las piernas depende de la masa de la pierna y la masa corporal.

1.2.2 Cinemática

La cinemática es el estudio más básico de cómo se comportan los sistemas mecánicos. En robótica móvil, tenemos que comprender el comportamiento mecánico del robot, tanto para

el diseño de robots móviles adecuados para las tareas y de entender cómo crear software de control de una instancia de hardware de robot móvil [Siegwart 2004].

Por supuesto, los robots móviles no son los primeros sistemas complejos mecánicos. Los robots manipuladores han sido objeto de estudio profundo durante más de treinta años y son mucho más complejas. El área de trabajo de un robot manipulador es crucial porque define la gama de posibles posiciones que se puede lograr por su efector final en relación con su soporte para el medio ambiente. El área de trabajo de un robot móvil es igualmente importante, ya que define la gama de posibles planteamientos que el robot móvil puede conseguir en su entorno.

1.2.3 Clasificación de Robots Móviles

En [Pruski, 1996] se clasifican los robots móviles en cuatro grupos distintos, de acuerdo al medio de locomoción que utilizan:

1. **Robot con ruedas.** Los robots con ruedas se podría decir que son los más populares ya que son más sencillos y fáciles de construir, además la carga que puede trasportar es relativamente mayor. Su principal operación es llevada a cabo en sitios industriales o ambientes interiores, aunque existe igualmente su aplicación en ambientes exteriores, como en la exploración espacial. La mayoría de los robots de este tipo presentan las restricciones no holonómicas que limitan el movimiento instantáneo que el robot puede realizar. Estas restricciones tienen como consecuencias el aumentar la complejidad del problema de planificación de trayectorias y de su control.
2. **Robot con orugas.** Cuando el terreno es accidentado, las ruedas pierden la eficiencia de la locomoción. Lo cual limita la capacidad de evolución del robot móvil equipado con este tipo de sistemas de locomoción. En estas condiciones, las orugas son más interesantes debido a que estas permiten aumentar la adherencia al terreno y atravesar por los obstáculos más importantes. Este tipo de robots presenta igualmente las restricciones no holonómicas.

3. **Robots con patas**. Los robots con patas permiten desplazamientos más eficientes en situaciones donde el terreno es todavía más incierto (rugoso, con obstáculos, con desniveles, etc.), además de ofrecer un control de estabilidad más completo. Por el contrario, la concepción y el control de una máquina con patas es muy compleja y su velocidad de evolución es generalmente reducida.
4. **Otros medios de locomoción.** Es el grupo de todos los robots móviles que utilizan un sistema de locomoción diferente de los tres anteriores. Por ejemplo, los robots móviles que se desplazan por arrastre, los robots submarinos, los robots para exploración espacial, robots voladores, etc. Las aplicaciones de este tipo de robots son muy especializadas y su arquitectura es en general específica a su aplicación.

1.3 Sistema Multi-robot

Un sistema multi-robot puede definirse como un conjunto de robots situados en un entorno común [Cruz, 2004]. Esta definición, tan genérica, engloba una gran diversidad de sistemas, que pueden presentar importantes diferencias entre sí. Algunas de las características arquitectónicas diferentes que deben ser tomados en cuenta para crear o analizar cualquier sistema multi-robot se describe a continuación [Leal, 2009].

1.3.1 Estructura Organizacional

La estructura organizativa del grupo de robots pueden ser centralizada o descentralizada, o una combinación de los dos, dependiendo de la forma en que el sistema sea controlado.

Centralizada. Este es un sistema que depende por completo de una unidad central (ya sea una unidad presente en el grupo o un centro de mando lejos de la posición actual del grupo) que estará al mando de todo el grupo para alcanzar la meta.

Descentralizado. Es un sistema que carece de control central, cada agente tiene el mismo nivel de "Autoridad" dentro del grupo. Este tipo podría ser considerado más difícil, ya que cada unidad deberá disponer de la potencia computacional para analizar la situación, procesar los datos y tomar decisiones.

Híbrido. Es una mezcla de ambos esquemas, un claro ejemplo es el tipo de arquitectura donde hay un control central desde el exterior, y cuando se presenta un problema o situación que justifique al grupo a tomar medidas, una unidad dirigente surgirá y coordinará al resto para llegar a la meta.

1.3.2 Tipos de unidades

Las unidades en el grupo, tanto físicamente como en términos de potencia de cálculo, pueden ser homogéneas o heterogéneas. Esta decisión influirá en gran medida de la complejidad de las unidades en la manera de percibir unas a otras y, finalmente, cómo se logra su interacción.

Homogéneo. Cuando todas las unidades son exactamente iguales, tienen la gran ventaja de que cada unidad conoce las dimensiones, capacidades y limitaciones del resto de las unidades en el grupo, y se puede saber qué esperar de ellos, así como predecir la posición de la unidad y la configuración en un momento dado. Este es el tipo más frecuente de la unidad cuando se trata de unidades modulares reconfigurables.

Heterogéneo. Las unidades en el grupo podrían tener diferentes capacidades y construcción física. Esto aumentará la complejidad de la interacción entre las unidades, pero también ofrece la ventaja de contar con unidades especializadas para algunas tareas.

1.4 Problemas de la Robótica Móvil en General

La localización de un robot móvil en un entorno conocido es uno de los problemas fundamentales de la robótica móvil, junto con la construcción automática de mapas del entorno. El robot debe ser capaz de contestar en cada momento a las siguientes tres preguntas: ¿dónde estoy? ¿Hacia dónde debo moverme? ¿Cómo llegaré hasta allá? De estas tres preguntas surgen tres problemas fundamentales de la navegación: el problema de la localización, el problema de la planificación de trayectorias y el problema de la utilización de mapas del entorno [Gallardo, 1999]. El proceso de posicionamiento y la generación

simultanea de un mapa del entorno es formalmente conocido como SLAM por las siglas en inglés (Simultaneous localization and mapping).

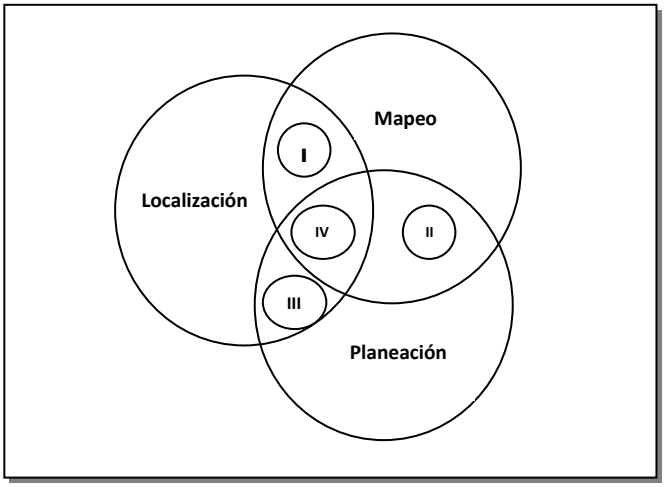


Figura 1.2. El campo de la exploración robótica y sus regiones de integración. Tomado de [Makarenko,2002]

Localización y mapeo simultaneo, SLAM, es una técnica usada en robots y vehículos autónomos para construir un mapa dentro de un ambiente desconocido, aun cuando es controlado en muchos casos, manteniendo registro de su posición en todo momento. Las soluciones existentes alcanzan diversos grados de la integración entre las tareas de localización, mapeo y planeación. La figura 1.2 ilustra el campo de la exploración Robótica, con cuatro regiones. La Región I representa gráficamente la integración de la localización y mapeo, SLAM. La Región II representa la integración del control de la planeación y el mapeo ejemplificada por prácticamente todas las estrategias de exploración. La Región III, que integra la localización y planeación es el campo de la navegación activa y la gestión del sensor. La plena integración de los tres componentes en la Región IV se refiere a la exploración ya integrada [Makarenko, 2002].

Los modelos y métodos para resolver los problemas de localización y mapeado deben tratarse con ciertas limitaciones y restricciones prácticas del funcionamiento de los robots móviles. Algunas de ellas son las siguientes [Gallardo 1999]:

Localidad de los sensores. El rango de percepción de la mayoría de sensores (ultrasonidos, láser, cámaras) está limitado a una zona pequeña alrededor del robot. Para adquirir información global, el robot debe explorar activamente su entorno.

Ruido en los sensores. Las observaciones realizadas por los sensores son normalmente ruidosas, y la distribución estadística de este ruido no suele ser sencilla de modelar.

Ruido en la posición. Los movimientos del robot no suelen ser exactos, produciéndose los denominados errores de odometría. Estos errores son, además, acumulativos con la distancia recorrida. Por ejemplo, pequeños errores en la rotación del robot pueden tener efectos importantes en la estimación de los movimientos de traslación y en la determinación de la posición final del robot.

Entornos complejos y dinámicos. Los espacios de trabajo en los que evoluciona el robot suelen ser complejos y dinámicos, haciendo prácticamente imposible mantener modelos consistentes de los mismos

Necesidad de tiempo real. Los requisitos de la aplicación (control de un robot móvil) obligan a procesar la información en un tiempo real. Esto limita la complejidad del procesamiento realizado por los métodos de localización, así como los modelos del entorno.

1.5 Descripción del problema

Este trabajo de investigación se enfoca en resolver el problema de localización utilizando técnicas de visión por computadora que permitan saber dónde están los robots y poder generar a cada uno de ellos una trayectoria que deben de seguir para alcanzar una meta (figura 1.3). Para ello, se utilizan dos robots Lego Mindstorms NXT tipo carro con tres ruedas, los cuales deben ser capaces de salir de un punto de inicio y llegar a un punto meta, siguiendo una trayectoria libre de obstáculos entre estos dos puntos. Dicha trayectoria es generada por el Algoritmo de Dijkstra, el cual procesa una matriz de ocupación llenada a través del procesamiento de una imagen proporcionada por una cámara web montada en el techo del ambiente.

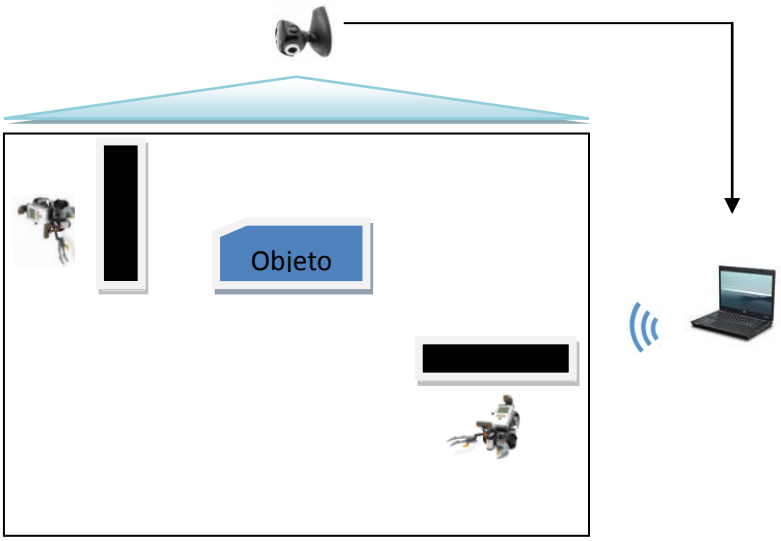


Figura 1.3. Representación esquemática del sistema de control

El procesamiento de la imagen consiste en localizar (a través de marcas) a los robots, utilizando los algoritmos Camshift y Contornos de las librerías de OpenCv. Estos proporcionan la ubicación de cada robot así como su orientación en el ambiente. Por otro lado, el procesamiento de la imagen proporciona también la ubicación de los obstáculos y del objeto meta.

1.6 Justificación del problema

En la actualidad la robótica móvil es uno de los temas que más ha suscitado interés dentro de la comunidad científica en el área de la inteligencia artificial, en lo que respecta al diseño de algoritmos de navegación, control, trazado de trayectorias y planificación de tareas en escenarios estructurados y no estructurados. Aunque en la actualidad se han propuesto diferentes métodos y técnicas para resolver la planificación de movimientos de robots móviles, encontrar mejores metodologías de solución posibilita su aplicación para resolver problemas de la vida real. En el caso del problema Box-Pushing, algunas de sus aplicaciones son mover cajas voluminosas o contenedores dentro de un almacén, rescate de personas, transportar materiales peligrosos, etc. Utilizar sistemas multi-robot para realizar estas tareas y otras que facilitan en gran medida la vida del ser humano, minimizaría costos económicos, evitaría riesgos y sobre todo vidas humanas.

1.7 Objetivos

Los objetivos, tanto general como los específicos de este trabajo de investigación, están enmarcados dentro de la plataforma de robótica móvil Lego Mindstorms NXT.

1.7.1 Objetivo general

Resolver el problema de localización utilizando técnicas de visión por computadora que permitan saber dónde están los robots y poder corregir las trayectorias que estos deben de seguir.

1.7.2 Objetivos específicos

- Desarrollar un procedimiento para establecer la comunicación entre la computadora y el sistema multi-robot a través de bluetooth .
- Determinar la forma y el color de las marcas visuales artificiales.
- Utilizar los algoritmos Camshift y Contornos de las librerías de OpenCv para identificar los robots dentro del ambiente.
- Utilizar técnicas de visión basadas en las librerías de OpenCv para obtener la matriz de ocupación.
- Utilizar el algoritmo de Dijkstra para la planeación de las trayectorias de cada uno de los robots en base a la matriz de ocupación.
- Desarrollar un procedimiento para controlar los movimientos de los robots en base al modelo cinemático propuesto en [Ojha 2009] y [Benedettelli 2006].

1.8 Alcances y limitaciones

1.8.1 Alcances

- Este trabajo es realizado a nivel físico con Robots Lego Mindstorms NXT.

- Utiliza los algoritmos Camshift y Contornos de las librerías de OpenCV para la detección de los robots a través de marcas artificiales colocadas a cada uno de ellos.
- Se propone una comunicación a través de Bluetooth entre la PC de Control y el Robot Lego.
- Utiliza el Algoritmo de Dijkstra para obtener la trayectoria que cada robot debe seguir para alcanzar la meta
- Utiliza una memoria de navegación en un archivo XML.

1.8.2 Limitaciones

- Se considera un ambiente estructurado y estático.
- La superficie de navegación debe ser plana.
- La carga de las baterías de los robots debe ser superior a los 7500 mA.
- La representación de los obstáculos y el ambiente se modelan en dos dimensiones.
- Los obstáculos son considerados como polígonos convexos.

1.9 Organización de la Tesis

El presente documento se encuentra organizado de la siguiente forma:

Los Capitulo 2 y 3 están dentro del marco conceptual y se aborda la teoría de agentes, la cual sirve como base para el desarrollo de una arquitectura de control híbrida basada en agentes para abordar la problemática planteada en este trabajo.

Capítulo 4 Estado del Arte. En esta sección del documento se hace una recopilación de los trabajos relacionados con la navegación de sistemas multi-robot basados en la tecnología Lego Mindstorms utilizando diferentes técnicas de visión. Por otro lado también se describen los trabajos referentes a la teoría de agentes que fueron tomados como marco de referencia para el desarrollo de este trabajo.

Capítulo 5 Metodología de Solución. En este capítulo se describe la metodología de solución para resolver el objetivo principal de este trabajo que es el de desarrollar una plataforma multi-robot, utilizando robots Lego, y técnicas de visión por computadora basada en los algoritmos Camshift y Contornos de las librerías de OpenCV.

Capítulo 6 Experimentación y Resultados. En esta sección se describen los experimentos y resultados obtenidos en la ejecución de los algoritmos desarrollados en este trabajo.

Capitulo 7 Conclusiones y Trabajos Futuros. Finalmente, se presentan las conclusiones del trabajo y los posibles trabajos que puedan surgir a partir de este.

2

Agentes y Sistemas Multi-Agentes

La historia de la computación ha estado marcada por cinco tendencias importantes [Wooldridge, 2002]:

- La ubicuidad
- La interconexión
- La inteligencia
- La delegación y
- La orientación humana.

Estas tendencias plantean importantes retos para los desarrolladores de software. Con respecto a la ubicuidad y la interconexión, aun no se sabe que técnica utilizar para desarrollar sistemas para explotar la energía del procesador en todas partes. Los modelos actuales de desarrollo de software han demostrado ser totalmente inadecuados, aun cuando se trata de un número relativamente pequeño de procesadores.

Las tendencias a la delegación y a la inteligencia son cada vez mayores e implican la necesidad de construir sistemas informáticos que puedan actuar con eficacia. Estos, a su vez, implican dos capacidades. La primera es la capacidad de los sistemas para operar de forma independiente, sin la intervención directa del ser humano. El segundo es la necesidad

de los sistemas informáticos para poder actuar de tal manera que representen intereses particulares, mientras interactúan con los seres humanos o sistemas.

La tendencia hacia la interconexión y la distribución tiene en la ciencia computacional un desafío clave. Mucha de la energía del campo intelectual a lo largo de las últimas tres décadas se ha dirigido hacia el desarrollo de herramientas de software y mecanismos que permitan construir sistemas distribuidos con mayor facilidad y fiabilidad.

Como se dijo anteriormente, los retos a los que se enfrentan los desarrolladores de aplicaciones y sistemas de información son cada vez más complejos. Los agentes son un nuevo paradigma para el desarrollo de aplicaciones software y actualmente son objeto de gran interés por parte de varios campos de la informática y de la Inteligencia Artificial (IA) [Jennings y Wooldridge, 1998]. Un agente puede verse como una evolución del concepto de objeto software, perfeccionada gracias a la influencia de la inteligencia artificial, que permite incorporar características como la racionalidad, la inteligencia, la autonomía o el aprendizaje. Al igual que los humanos, los agentes deben tener habilidades sociales y ser capaces de realizar trabajos o resolver problemas de forma distribuida. Se habla entonces de un sistema multiagente, en el que los agentes cooperan e interactúan para conseguir los objetivos finales del sistema.

2.1 ¿Qué es un Agente?

Lamentablemente, no existe una definición universalmente aceptada del término agente, y de hecho existe mucho debate y controversia sobre este tema. Sin embargo una definición es importante, porque de lo contrario, existe el peligro de que el término pierda todo su sentido. Una de las definiciones más aceptadas es la siguiente [Wooldridge y Jennings, 1995]:

An agent is a computer system that is situated in some environment, and that is capable of autonomous action in this environment in order to meet its design objectives. [1]

[1] ***Un agente es un sistema informático que se encuentra en un entorno, y que es capaz de actuar con autonomía en este ámbito con el fin de cumplir con sus objetivos de diseño.***

La definición implica que hay una frontera bien definida y una interfaz concreta entre el agente y su entorno. El aspecto clave de la definición es la autonomía, que se refiere en un principio que los agentes pueden operar por su cuenta, sin la necesidad de un guía humano. Un agente autónomo tiene el control sobre sus propias acciones y estado interno, es decir, un agente puede decidir si desea realizar una acción solicitada. La definición sitúa a un agente en un entorno particular, en que el agente puede "sentir y accionar". Esto lleva al comportamiento de respuesta. Además, la definición implica que los agentes pretenden alcanzar soluciones a problemas con límites bien definidos, para alcanzar un propósito específico, es decir, con objetivos específicos a lograr, y que exhiben un comportamiento activo flexible y proactivo. Para [Wooldridge y Jennings, 1995] el término agente disfruta de las siguientes propiedades:

- **Autonomía**: Los agentes operan sin la intervención directa de los humanos u otros, y tienen alguna clase de control sobre sus acciones y estados internos.
- **Habilidades sociales**: Los agentes interaccionan con otros agentes (y posiblemente con humanos) por medio de algún lenguaje de comunicación de agentes.
- **Reactividad**: Los agentes perciben su entorno, (que puede ser el mundo físico, un usuario vía un interfaz gráfico, una colección de otros agentes, Internet, o quizás una combinación de todos ellos), y responden en un periodo determinado de tiempo a los cambios que en él se producen.
- **Proactividad**: Los agentes no se limitan a actuar en respuesta a su entorno, sino que son capaces de manifestar comportamientos dirigidos por metas tomando iniciativas propias.

La figura 2.1 proporciona una visión abstracta de un agente. En este diagrama, podemos ver la salida generada por la acción del agente con el fin de afectar a su entorno. En la mayoría de los dominios de complejidad razonable, un agente no tendrá un control completo sobre su medio ambiente. Tendrá en el mejor de los casos un control parcial, ya que puede influir en él. Desde el punto de vista del agente, esto significa que la misma acción realizada dos veces en circunstancias aparentemente idénticas parecen tener efectos completamente diferentes, y, en particular, puede no tener el efecto deseado. Por ello, los agentes en el más trivial de los ambientes deben estar preparados para la posibilidad del fracaso. Podemos

resumir esta situación formalmente diciendo que los ambientes son, en general, no deterministas.

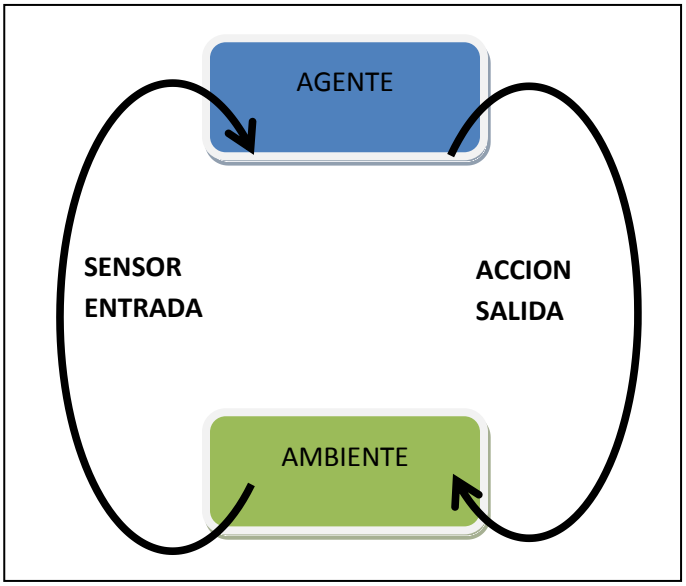


Figura 2.1. Un agente en su entorno. El agente lleva la información sensorial del medio ambiente, y produce como salida las acciones que la afectan. La interacción es usualmente una constante, que no termina.

Normalmente, un agente tendrá un repertorio de acciones de que disponer. Este conjunto de posibles acciones representa la capacidad de los agentes para modificar sus entornos. Pero no todas las acciones se pueden realizar en todas las situaciones. Por ejemplo, la acción "comprar un avión" fallará si los fondos disponibles son insuficientes. Las acciones por lo tanto tienen condiciones previas asociadas con ellos, que definen las situaciones en las que se pueden aplicar. El problema clave que enfrentan los agentes es el de decidir cuál de sus acciones se debe realizar con el fin de satisfacer mejor sus objetivos de diseño.

2.2 Clasificación de Agentes

Así como existen diferentes definiciones para el concepto de agente, también sucede lo mismo para definir una clasificación de los diversos tipos de agentes. Algunos autores [Russel y Norvig, 1995], [Nwana, 1996], han identificado clases de agentes estableciendo criterios que se basan en características comunes de los agentes o de los entornos de

aplicación. En [Russell y Norving, 1995] la clasificación se basa en el tipo de programa utilizado para implementar las funcionalidades del agente, estableciendo el estado intermedio entre percepciones y acciones. El criterio que utilizan para diferenciar los tipos de agentes es el tipo de programa utilizado para implementar el agente. De esta forma distinguen entre:

- Agentes de reflejo simple, agentes que reaccionan ante estímulos, proporcionando una respuesta y sin capacidad de memoria.
- Agentes reflejo con estado interno (Reflex Agent with State) que incorporan un estado de memoria interno.
- Agentes basados en metas (Goal Based Agent) que incorporan metas que dotan a este tipo de agente de capacidades deliberativas.
- Agentes basados en utilidad (Utility Based Agents) que incorporan el concepto de utilidad. Una función que a cada estado le asocia su grado de utilidad y que permite la toma de decisiones racionales. Esto, cuando satisfacer algunas metas implica un conflicto o cuando el agente puede desear obtener varias metas y no existe la certeza de lograr ninguna de ellas.

Por otro lado en [Nwana, 1996], basa la clasificación en varias dimensiones:

- Respecto a su movilidad, los agentes se dividen en móviles o estáticos, según su capacidad o no de desplazarse en una red.
- Un agente puede clasificarse como deliberativo o reactivo. Los primeros derivan del paradigma del pensamiento deliberativo: los agentes poseen un modelo simbólico interno de razonamiento y se ocupan de la planificación y la negociación con el objetivo de conseguir coordinarse con otros agentes. Los agentes reactivos, por el contrario, no poseen ningún modelo simbólico interno de su entorno y actúan empleando un comportamiento del tipo estímulo respuesta, respondiendo al estado actual del entorno en que está contenido.
- Los agentes pueden clasificarse según los atributos primarios que deberían mostrar. Nwana distingue tres principales: autonomía, aprendizaje y cooperación. De este modo existen los agentes autónomos, de aprendizaje y cooperativos. Las combinaciones dos a dos de estas características originan los agentes colaborativos (cooperación y autonomía), colaborativos de aprendizaje (aprendizaje y

cooperación) y de interfaz (aprendizaje y autonomía). Finalmente, la combinación de los tres elementos da como resultado los agentes inteligentes. Esta distribución se muestra en la Figura 2.2.

- En algunas ocasiones, los agentes pueden ser clasificados por el papel que desempeñan. Por ejemplo, los agentes de información o los agentes de Internet. Estos tipos de agentes se dedican a la búsqueda y procesamiento de información en una red, como en el caso de Internet.
- Por último, Nwana habla de agentes híbridos para referirse a los que combinan dos o más de las categorías anteriores.

En conclusión podemos decir que las clasificaciones mencionadas se basan en los atributos de los agentes como criterio principal, así como en la necesidad de diferenciar aquellos agentes que son simplemente reactivos de aquellos que incorporan algún mecanismo de razonamiento. Existen una serie atributos que pueden servir de referencia, como son la autonomía, la proactividad, el aprendizaje, y la cooperación. Además se menciona la existencia de agentes híbridos que flexibilizan las clasificaciones y permiten la integración de más de un criterio de clasificación.

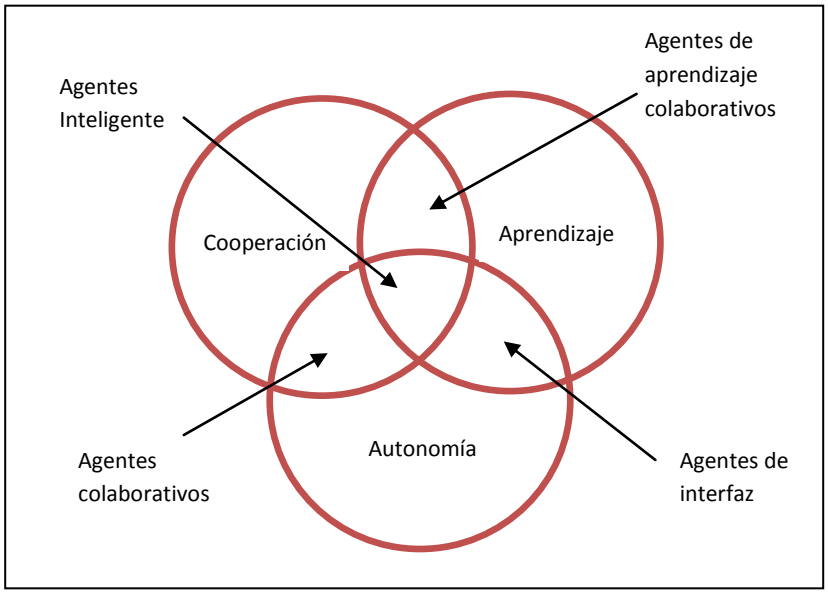


Figura 2.2 Clasificación de los agentes según su característica primaria [Nwana, 1996].

2.3 Arquitecturas de Agentes

Existe una gran variedad de arquitecturas, una primera clasificación de las arquitecturas se puede basar en el acceso de las capas hacia los sensores y actuadores. Si todas las capas tienen acceso a los sensores y actuadores es una arquitectura horizontal, si sólo la capa más baja tiene acceso a los sensores y/o actuadores, es una arquitectura vertical. También se pueden clasificar las arquitecturas según el tipo de procesamiento empleado en deliberativas, reactivas e híbridas.

2.3.1. Arquitectura deliberativa

Son aquellas arquitecturas que utilizan modelos de representación simbólica del conocimiento. Suelen estar basadas en la teoría clásica de planificación. Estos agentes parten de un estado inicial y son capaces de generar planes para alcanzar sus objetivos [Maes, 1989]. Estas arquitecturas son las heredadas de las investigaciones en Inteligencia Artificial clásica y fueron las primeras en seguir en el campo de la robótica [Brooks, 1991]. Enfocan el problema de la generación de comportamiento en un robot como una tarea secuencial que exige una representación explícita del mundo para tomar decisiones basándose en esta representación y siguiendo las directrices marcadas por un planificador. Plantean el problema como una secuencia de estrategias de: percepción del entorno, construcción de un modelo del mundo, planificación basada en ese modelo y actuación.

2.3.2. Arquitectura reactiva

Los numerosos problemas que lleva asociados utilizar una representación simbólica del conocimiento han conducido al estudio de modelos más efectivos de representación del conocimiento [Bonasso et al., 1995]. Las arquitecturas reactivas se caracterizan por no tener como elemento central de razonamiento un modelo simbólico y por no utilizar razonamiento simbólico complejo [Keith et al., 1996]. Un ejemplo típico de estas arquitecturas es la propuesta de Roodney Brooks, conocida como arquitectura de subsunción [Brooks, 1991]. Esta arquitectura se basa en las hipótesis de que la inteligencia

es una propiedad emergente de ciertos sistemas complejos y de que ello permite generar comportamientos inteligentes sin necesidad de construir un modelo simbólico. Las arquitecturas de subsunción manejan jerarquías de tareas que definen un comportamiento. Suelen estar organizadas en jerarquías de capas, de menor a mayor nivel de abstracción (Brooks, 1991).

2.3.3. Arquitectura híbrida

Las arquitecturas deliberativas y las reactivas representan dos extremos opuestos del problema de una arquitectura para la organización del conocimiento y control. Las deliberativas ponen su énfasis en la parte racional de la resolución del problema, insistiendo en una representación completa y por lo tanto difícil de manejar y actualizar; en los métodos generales de resolución de problemas y en la planificación para tomar la decisión más correcta. Por el contrario, los enfoques reactivos se centran en la capacidad de reacción frente a cambios en el entorno, insistiendo en que el robot está inmerso en un mundo concreto, que sus respuestas deben ser rápidas, sencillas y que es la interacción del sistema con el entorno la que guía el comportamiento observable, sin necesidad ni de memoria ni de representación ni planificación. Entre ambos extremos se sitúan las arquitecturas híbridas que pretenden incorporar, con diferentes grados, tanto capacidades de deliberación como de reacción a los robots móviles, a fin de que puedan desenvolverse de modo autónomo en un mundo dinámico y además que incorporen mecanismos de planificación y representación a medio o largo plazo.

2.4 Sistema Multi-Agentes

La idea de un sistema multiagente (SMA) es muy simple. Un agente es un sistema informático que es capaz de actuar independientemente en nombre de su usuario o propietario. En otras palabras, un agente puede descubrir por sí mismo lo que necesita para satisfacer sus objetivos de diseño, en lugar de que se le diga de forma explícita qué hacer en cada momento. Un SMA es aquel que consiste en una serie de agentes, que interactúan

entre sí, por lo general mediante el intercambio de mensajes a través de algún tipo de infraestructura de la red informática [Wooldridge, 2002]. En la mayoría de los casos, los agentes en un sistema multiagente se representan o actúan en nombre de usuarios o propietarios con objetivos y motivaciones muy diferentes. Con el fin de interactuar con éxito, en un SMA hay que distinguir cuatro conceptos, que aunque estén muy relacionados entre sí, hacen referencia a características diferentes de un SMA. Éstos son: comunicación, coordinación, cooperación y negociación [Rodríguez, 2010].

Comunicación. Es la habilidad de los agentes para comunicarse entre sí, esto es, intercambiar información y conocimiento de forma comprensible. Permite a los agentes obtener el conocimiento necesario para decidir la secuencia de acciones que debe ejecutar en función de sus objetivos.

Coordinación. Se define como un conjunto de acciones suplementarias que pueden realizarse en un entorno multi-agente para alcanzar un objetivo y que un agente, con los mismos objetivos, no podría alcanzar por sí solo.

Cooperación y negociación. La cooperación es el mecanismo por el cual los agentes, que trabajan juntos para lograr un objetivo común, definen una estrategia para alcanzar este objetivo. Por otro lado, la negociación permite alcanzar decisiones de coordinación conjuntas mediante la comunicación explícita.

La necesidad de desarrollar aplicaciones complejas compuestas de multitud de subsistemas que interaccionan entre sí obliga a distribuir la inteligencia entre diversos agentes y a construir SMA. Estos permiten la gestión inteligente de un sistema complejo, coordinando los distintos subsistemas que lo componen e integrando los objetivos particulares de cada subsistema en un objetivo común.

3

Arquitectura de Control Híbrida Basada en Agentes

Un robot puede ser considerado como un sistema complejo dotado de distintos tipos de habilidades: perceptivas, motoras, de procesamiento y de comunicación con el exterior. En el campo de la Robótica e Inteligencia Artificial, la organización de estos conocimientos y habilidades en la obtención de un objetivo, se conoce como **arquitectura de control** del robot. Cualquier arquitectura de control tiene que organizar las habilidades de un robot a fin de dar una respuesta coherente y robusta a la obtención de un conjunto de objetivos ante un tipo de entorno. Sin embargo no existe la arquitectura de control ideal, sino diferentes diseños o aproximaciones a la solución de un determinado tipo de problemas [Murphy, 2000]. Para esto se deben de considerar una serie de restricciones para poder abordar una arquitectura para la generación de comportamiento autónomo. Acorde a las características mencionadas en la sección 2.3, este trabajo propone una arquitectura hibrida basada en agentes como modelo apropiado para cumplir los objetivos propuestos de navegación. El modelo que se propone tiene sus bases en una arquitectura denominada AGRO-AMARA

[García, 2004]. Esta a su vez, tiene sus principios en los trabajos realizados en [García-Alegre et al., 1992, 1995a, 1995b, 1998, 1999] materializados en una arquitectura inicial denominada AMARA.

3.1 Arquitectura de Control Propuesta

El diseño de los agentes de comportamiento adecuado, permite resolver el problema de la navegación global. Logrando que el robot navegue con la mínima intervención humana desde una posición inicial a un destino objetivo, siguiendo el mejor camino posible y evitando chocar con obstáculos. En esta sección describiremos a detalle cada uno de los agentes implicados en la navegación del robot de la arquitectura propuesta (figura 3.1).

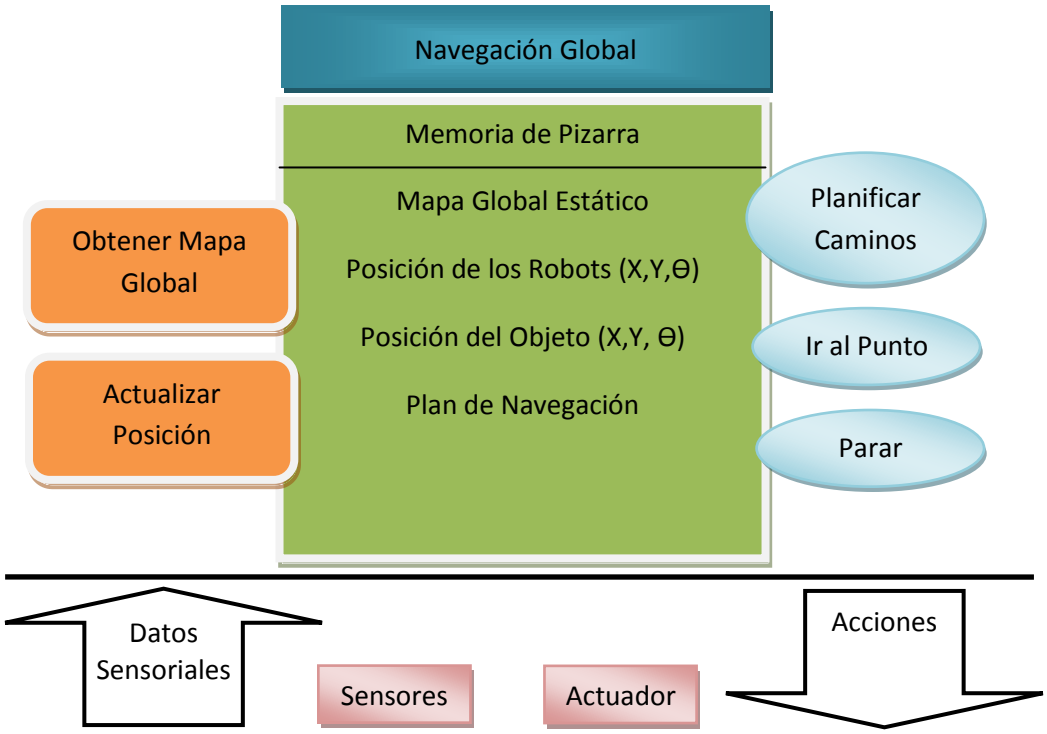


Figura 3.1 Arquitectura de agentes para la navegación global. Las elipses de color azul, situadas a la derecha de la figura corresponden a los agentes de actuación. Los rectángulos con esquinas redondeadas de color naranja, situados a la izquierda son los agentes perceptivos. En el centro de la imagen en color verde, se muestra el contenido de la pizarra.

Primeramente se describe la función y restricciones de los agentes perceptivos que son los que van a activar a determinados agentes de actuación. La exposición de los agentes de esta

arquitectura, empieza con los de nivel inferior, estos son los que están más cercanos a los actuadores y poseen una representación temporal instantánea o de corto alcance. Finalmente se expondrán los de mayor alcance y mayor grado de abstracción y representación.

3.1.1 Agentes preceptivos de la arquitectura de navegación global

Los agentes perceptivos que conforman esta arquitectura son: **Actualizar Posición** y **Obtener Mapa Global**. A continuación se describen cada uno de ellos.

3.1.1.1 Agente Actualizar Posición.

El principal objetivo de este agente es la obtención y actualización de la posición del robot en todo momento. Esta tarea es importante para poder llevar a cabo la tarea de navegación autónoma del robot. El agente Actualizar Posición recibe información relativa a su localización a través de una cámara web, utilizada como sensor. Tiene como única entrada una señal de activación y como salida la posición estimada de los robots (X_R,Y_R, Θ_R) y del objeto meta (X_O,Y_O, Θ_O). En la figura 3.2 se muestra el esquema del agente Actualizar Posición.

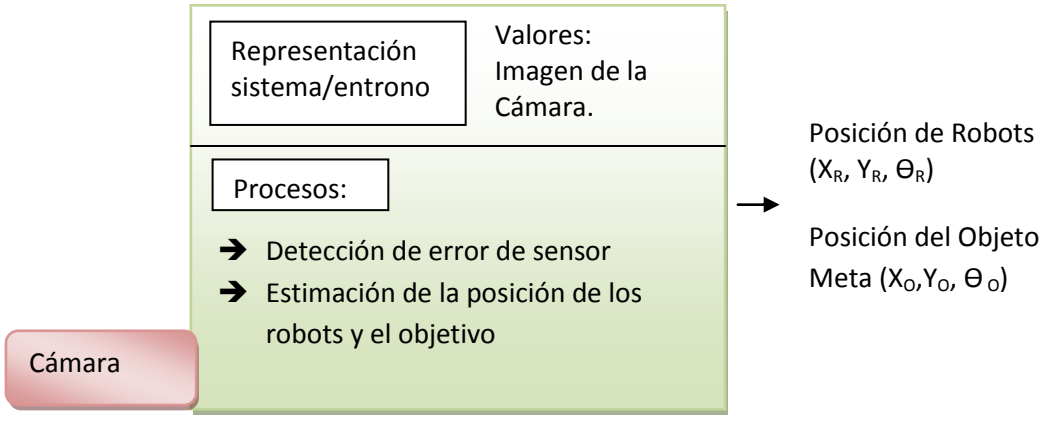


Figura 3.2 Esquema de entradas y salidas, representación y procesos en el agente Actualizar Posición.

Dentro de los procesos que el agente Actualizar Posición tiene, está el de detectar si el sensor funciona correctamente, es decir, si la cámara está encendida o no pudo recuperar la imagen deseada.

Por otro lado está el proceso de la estimación de la posición de los robots y objeto final. Este se adquiere mediante una serie de algoritmos para el procesamiento de imágenes basados en las librerías de OpenCv de Intel [Bradsky, 2008]. Uno de estos algoritmos es el encargado de obtener un histograma de colores para representar a cada uno de los robots, así como también para el objeto meta. Esta parte corresponde a la calibración del sensor de visión.

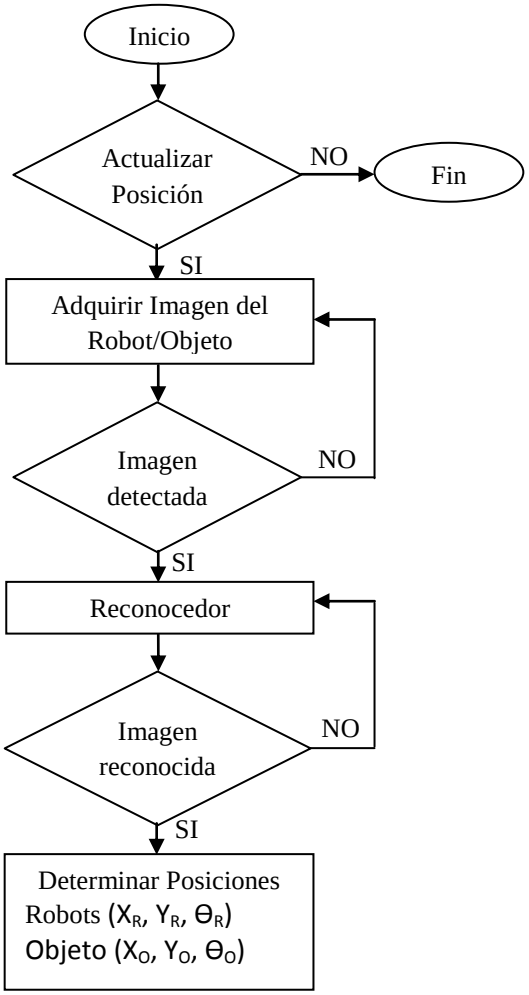


Figura 3.3 Diagrama de flujo de información del agente Actualizar Posición

Una vez obtenidos los histogramas de cada uno de los elementos antes mencionados, otro algoritmo se encarga de hacer la búsqueda de estos elementos para obtener su posición y orientación. Este algoritmo es CamShift [Bradsky, 2008], que utiliza la información de color, pero en lugar de depender de un solo color, hace un seguimiento de una combinación de colores y crea un histograma de color para representar el objeto. Por otro lado utilizamos el algoritmo de Contornos también de las librerías de OpenCv para obtener la posición y orientación de los robots.

La figura 3.3 muestra el diagrama de flujo de información en el agente Actualizar Posición, junto con el pseudocódigo del anexo A, describen a detalle el funcionamiento de este agente perceptivo.

3.1.1.2 Agente Obtener Mapa Global.

La utilización de mapas basados en rejilla de ocupación es común en robótica, dado que simplifican mucho el problema de la navegación al dividir el ambiente en celdas. La generación y actualización del agente Mapa Global de rejilla se obtiene al cargar una imagen del ambiente a través de la cámara web, y de la estimación de la posición de los robots y el objeto final proporcionada por el agente Actualizar Posición. La imagen del ambiente es convertida en un mapa de probabilidades, donde cada rejilla, blanca o negra, tiene un valor que determina si se trata de una zona de tránsito o no (Figura 3.4).

Figura 3.4 Imagen global del ambiente representado en un mapa de rejilla. La primera corresponde al mapa global original y la segunda al mapa global binarizado.

El agente Obtener Mapa Global dispone como única entrada la señal de activación y como salida la actualización del mapa global en forma de rejilla de ocupación. En la figura 3.5 se muestra el esquema del agente Obtener Mapa Global.

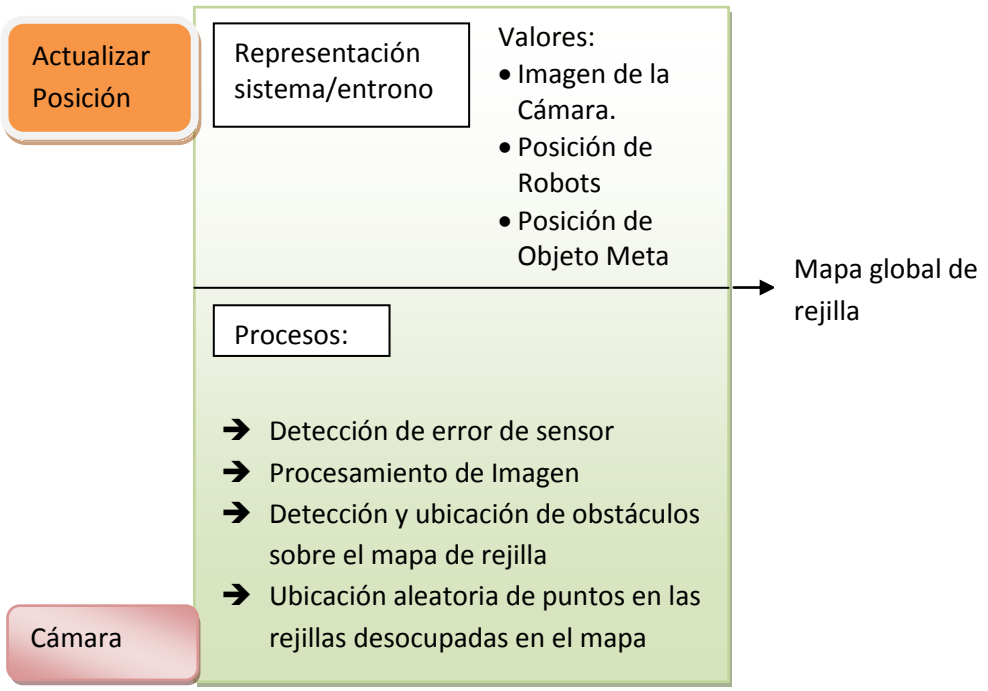


Figura 3.5 Esquema de entradas y salidas, representación y procesos en el agente Obtener Mapa Global.

Dentro de los procesos que tiene el agente Actualizar Posición, está el de detectar si el sensor funciona correctamente, es decir, si la cámara está encendida, o no pudo recuperar la imagen deseada.

El procesamiento de la imagen, básicamente consiste en una serie de algoritmos [Ibarra, 2009] construidos con las librerías de OpenCv de Intel [Bradsky, 2008]. Primeramente se captura el cuadro de imagen del ambiente global, se convierte a escala de grises, se mejora el contraste de la imagen y se elimina el ruido, y posteriormente se binariza la imagen para diferenciar los espacios desocupados de los ocupados. Por último el mapa obtenido divide el ambiente en un rectángulo reticulado de 210 x 153 (cm), y celdillas de 42 x 30.6 (cm), donde cada una de estas tiene un valor asociado de ocupación. Las dimensiones del mapa están determinadas en base a la resolución de la cámara web de 640 x 480 pixeles. El tamaño de la celdilla se ha determinado en base a las medidas del robot, para que este pueda realizar en determinado momento un giro de 360° sobre su eje, sin salir de la celdilla.

Antes de seguir con el proceso siguiente, se ubican en el mapa global de rejilla los valores obtenido del agente Actualizar Posición. Estos valores se identifican de la siguiente manera: 1 para el Robot y 2 para el objeto meta. Una vez realizado lo anterior, se ejecuta el proceso de detección y ubicación de los obstáculos dentro del mapa global de rejilla proporcionado por el proceso anterior.

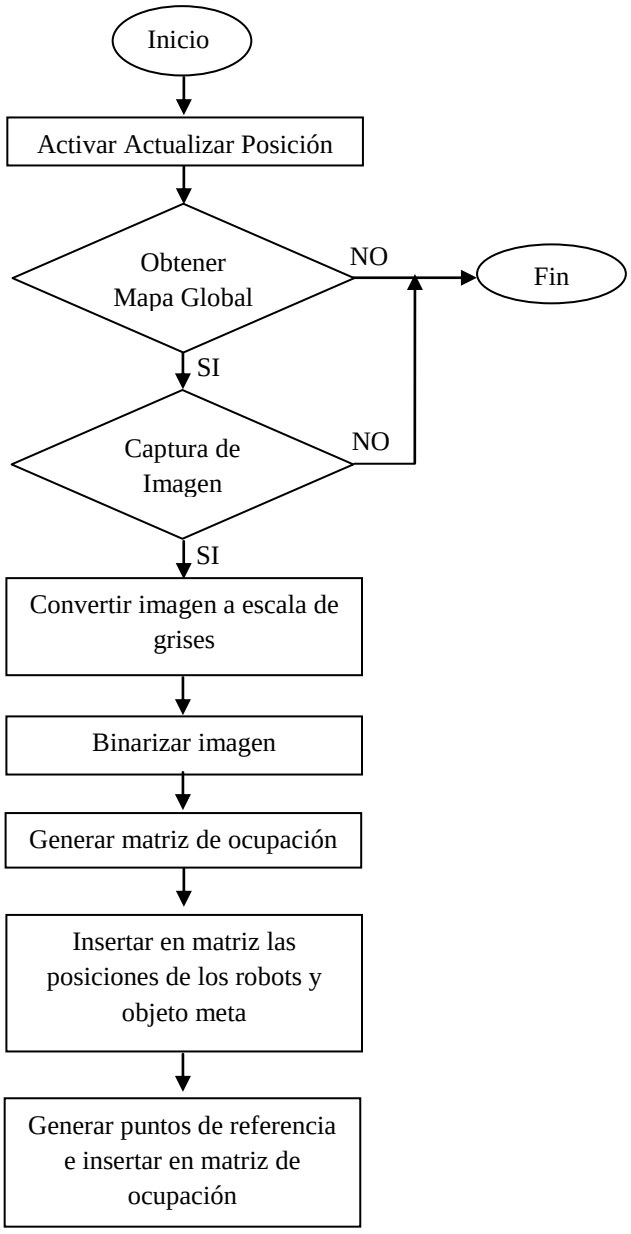


Figura 3.6 Diagrama de flujo de información del agente Obtener Mapa Global.

Se han definido dos valores para indicar si la celdilla está ocupada o desocupada, estos valores son 0 y -1 respectivamente. Para determinar el valor de la celdilla, se hace un barrido pixel por pixel de cada una de las celdillas que forman el mapa global y se lleva un conteo del número de pixeles obscuros. Si el número de pixeles obscuros es mayor o igual a las tres cuartas partes del total de pixeles de la celdilla, entonces ponemos el valor de -1 a esa celdilla indicando que está ocupada. En caso contrario ponemos el valor de 0 indicando que esta celdilla está desocupada. Estos valores podrán almacenarse en la determinada celdilla, siempre y cuando el valor de esta no sea 1 o 2, porque estos valores corresponden a los del robot y obstáculo respectivamente.

Por último, ya teniendo el mapa global de rejilla con la posición de los robots y el objeto meta, así como la ubicación de los obstáculos, se generan una serie de números aleatorios. Estos números servirán como puntos de referencia, para trazar la trayectoria de cada uno de los robots. El total de números aleatorios generados, será en función del total de celdillas menos la suma de los obstáculos, robots y objeto meta.

La figura 3.6 muestra el diagrama de flujo de información en el agente Obtener Mapa Global, junto con el pseudocódigo del anexo A, describen a detalle el funcionamiento de este agente perceptivo.

3.1.2 Agentes de Actuación de la arquitectura de navegación global

En este apartado se presentan los agentes de actuación diseñados hacia un planificador dedicado a la optimización de la trayectoria de navegación. Todos ellos siguen el mismo patrón presentado en la figura 3.1. Las entradas, salidas, representación interna del mundo en la que basa sus procesos de razonamiento para tomar decisiones y dirigirlas hacia otros agentes o se registran en la memoria global compartida del sistema, así como los procesos que dan lugar a su comportamiento.

Para el caso de este trabajo, la memoria global compartida se hará utilizando un archivo cargado en memoria con estructura XML. A continuación se describen cada uno de ellos.

3.1.2.1 Agente Parar

Este agente constituye uno de los bloques básico que soporta el comportamiento observable de la navegación. Garantiza una navegación segura desde un inicio de la trayectoria. Su objetivo es parar al robot cuando presente un peligro de choque con un obstáculo o con otro robot. El Agente Parar tiene como entradas las lecturas sensoriales del sensor de ultrasonido montado en la parte delantera del robot. En la figura 3.7 se muestra el esquema del agente Parar.

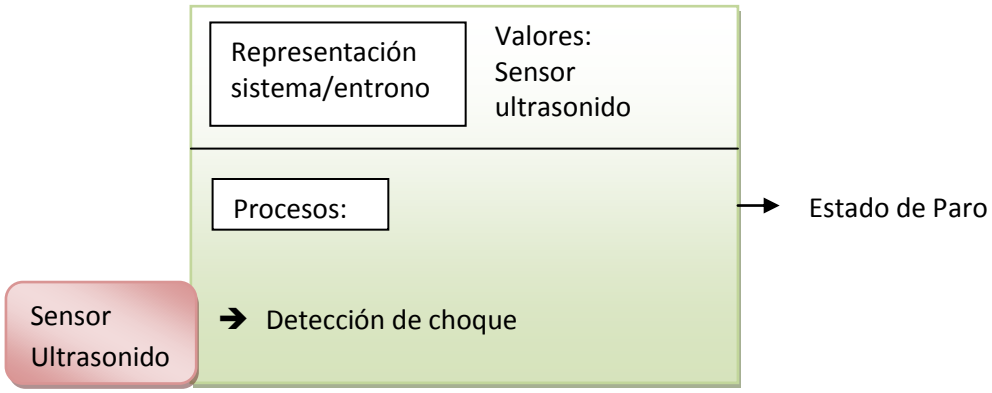


Figura 3.7 Esquema de entradas y salidas, representación y procesos en el agente Parar.

Parar es un agente de actuación, por lo tanto cuando este es activado ejecuta un procedimiento que para inmediatamente al robot.

La toma de decisión es puramente reactiva y se basa principalmente en la medición que hace el sensor de ultrasonido al momento que el robot está avanzando. Si el sensor detecta un objeto a 10 cm del robot entonces el robot se para y evita que este haga colisión con el objeto detectado.

Si la trayectoria que está cubriendo el robot es hacia el punto objetivo meta, entonces el agente ya no modifica su estado, es decir se queda activo. Si está recorriendo una trayectoria en donde su punto final es diferente al objetivo meta, entonces su estado estará en alerta, hasta que se replanifique una nueva ruta.

El diagrama de flujo de la figura 3.8 muestra el modo de operación de este agente.

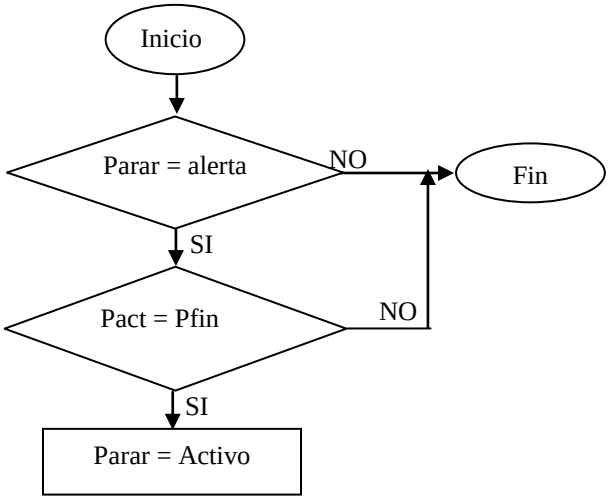


Figura 3.8 Diagrama de flujo de información del agente Parar.

3.1.2.2 Agente Planificar Caminos

La planificación de caminos es uno de los aspectos críticos de la navegación. El objetivo de la navegación global de propósito general implica el desplazamiento de un robot autónomo desde un punto inicial a un punto final siguiendo una trayectoria.

El agente Planificar Caminos elabora un plan de navegación que permite llegar con eficiencia al destino final. Para esto debe de conocer previamente la posición del robot (punto de origen), la posición del objeto meta (punto final) y el mapa del entorno donde se encuentran representados los obstáculos estáticos y los puntos de referencia por donde puede pasar el robot (matriz de ocupación).

Para obtener la posición del robot y del objeto meta, el agente Planificar Caminos debe de activar al agente Actualizar Posición, el cual es el encargado de elaborar esta precepción. De la misma forma, deberá de activar al agente Obtener Mapa Global, para obtener la matriz de ocupación, que representa el mapa del entorno global.

El agente Planificar Caminos tiene entonces como única entrada, una señal de replanificación, que es activada cuando el agente Parar se activa. Como única salida tendrá un plan de navegación que es un conjunto de puntos que representan una posición (x,y) que conectan el punto inicial con el punto final.

Estas trayectorias se almacenan en la pizarra (archivo XML) para que sean usadas por otros agentes de actuación. En la figura 3.9 se muestra el esquema del agente Planificar Caminos.

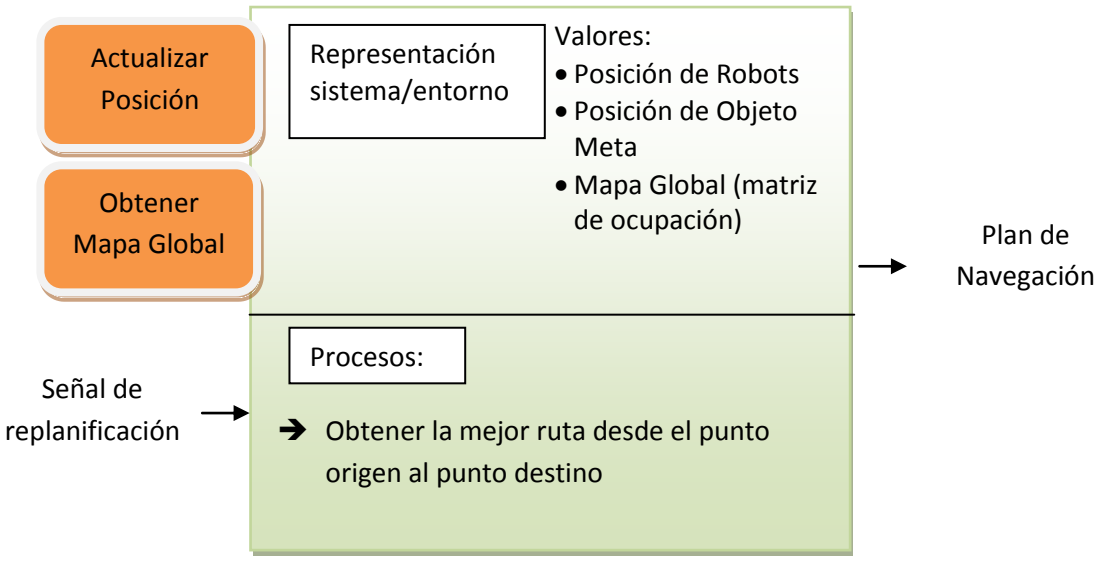


Figura 3.9 Esquema de entradas y salidas, representación y procesos en el agente Planificar Caminos.

El proceso de este agente es el de obtener la mejor ruta desde un punto origen a un punto destino. Este proceso se basa en el algoritmo de Dijkstra, el cual toma en cuenta el costo de movimiento que se tiene entre los nodos.

Al tratarse de un robot no-holonómico, el camino más corto no siempre resulta ser el más adecuado si el vehículo tiene que realizar varios giros. Esto no es de consideración en este trabajo, por lo que está fuera del alcance del mismo y concretamos solo obtener la mejor ruta posible desde un origen hasta un destino.

El proceso de planificación de caminos se ejecuta una sola vez, solo se replanifica cuando el agente Ir al Punto hace activar al agente Parar y este a su vez activa una bandera de replanificación.

El diagrama de flujo de la figura 3.10 muestra el modo de operación de este agente.

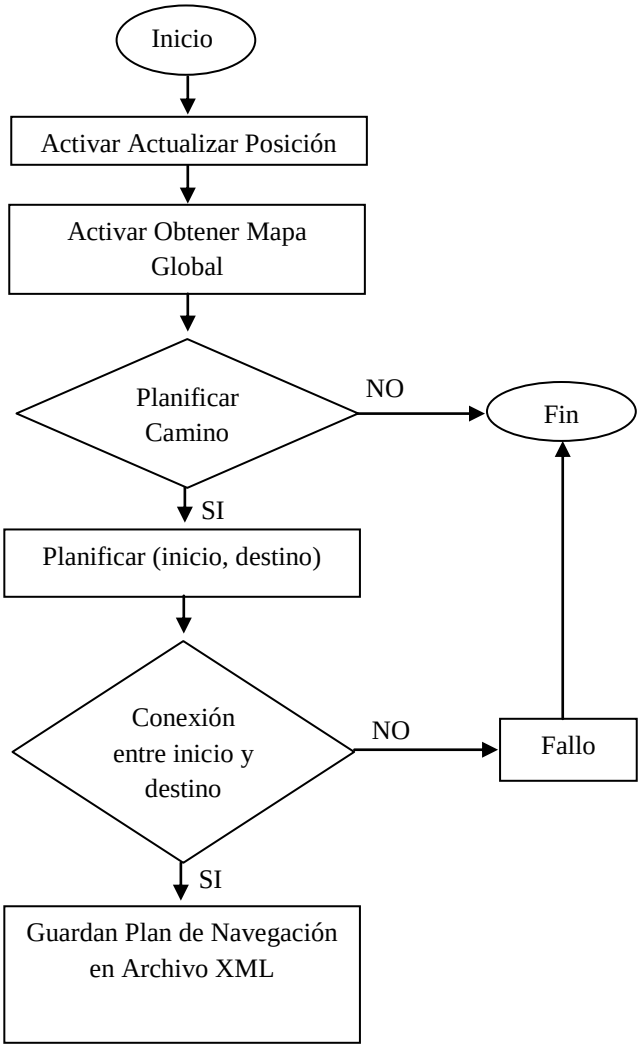


Figura 3.10 Diagrama de flujo de información del agente Planificar Caminos.

3.1.2.3 Agente Ir al Punto

El objetivo del agente **Ir al Punto** es dirigir al robot desde una posición inicial expresada como (X_R, Y_R, Θ_R) hasta una posición destino (X_O, Y_O, Θ_O) con una orientación determinada. Para ello **Ir al Punto** utiliza la habilidad de los agentes **Parar** y **Planificar Caminos**, activándolos y modulándolos adecuadamente de acuerdo con el entorno.

Para que el robot navegue satisfactoriamente, el agente Ir al Punto tiene que realizar las siguientes tareas:

- Seleccionar un subobjetivo o posición a la que tiene que llegar

- Analizar si es necesario realizar un nuevo plan de navegación
- Informar si ha conseguido llegar a la posición requerida o no y finalizar el proceso.

La figura 3.11 describe el esquema de entradas, salidas, representación del entorno y procesos del agente Ir al Punto. Las entradas serian básicamente las coordenadas absolutas de los puntos a recorrer. Estos puntos son los del plan de navegación proporcionado por el agente Planificar Caminos y almacenados en el archivo XML (memoria del sistema). Las salidas son las siguientes: 1) la señal de replanificación que indica si es o no necesario elaborar un nuevo plan de navegación y constituye a su vez una entrada del agente **Planificar Caminos**, 2) la señal de objetivo conseguido, 3) la señal de fallo en la ejecución, y 4) las señales de activación de los agentes implicados: **Parar**, **Planificar Caminos** y **Actualizar Posición**.

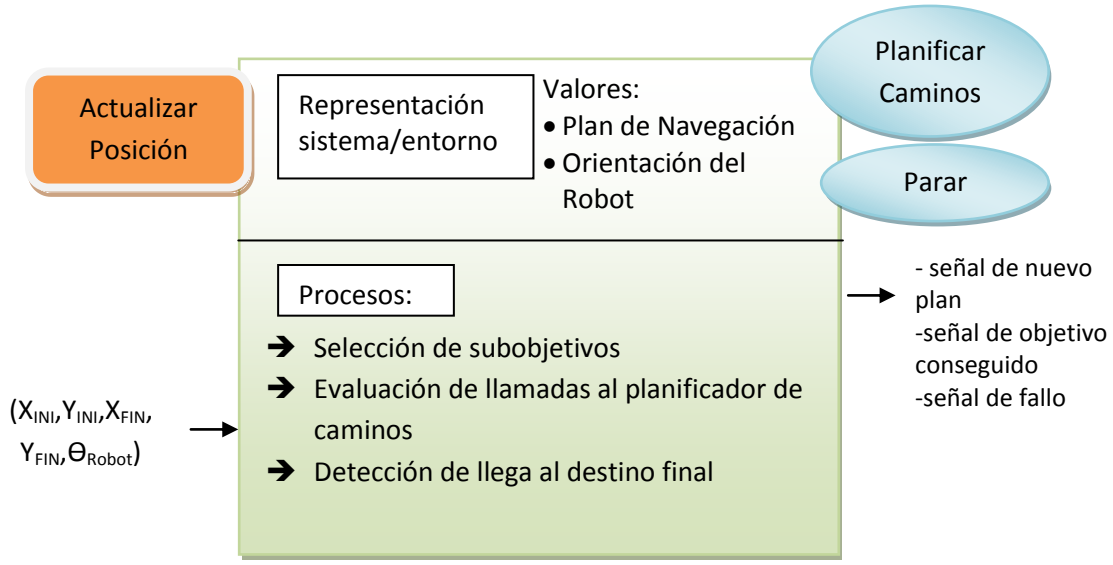


Figura 3.11 Esquema de entradas y salidas, representación y procesos en el agente Ir al Punto.

El agente **Ir a Punto** al activarse envía, a su vez, señales de activación a los agentes de actuación **Parar**, y **Planificador de Caminos** y al agente perceptivo **Actualizar Posición.** Después inicia su ciclo de ejecución con un tiempo de iteración de 6 (s.). En cada iteración comprueba en primer lugar si se ha llegado al destino final, si es así, finaliza el proceso satisfactoriamente con una señal. Si aún no ha llegado al destino final, comprueba si se ha elaborado un nuevo plan de navegación, y de ser afirmativo actualiza el subobjetivo. Si

continua con el plan de navegación anterior, comprueba si el robot ha llegado al subobjetivo en curso, y si es así, establece un nuevo subobjetivo. En caso contrario analiza si el robot se ha perdido, si hace falta replanificar o bien si existe algún fallo.

El diagrama de flujo de información del agente **Ir al Punto** se muestra en la figura 3.12. En esta se muestran las entradas, salidas, representación y procesos de este agente.

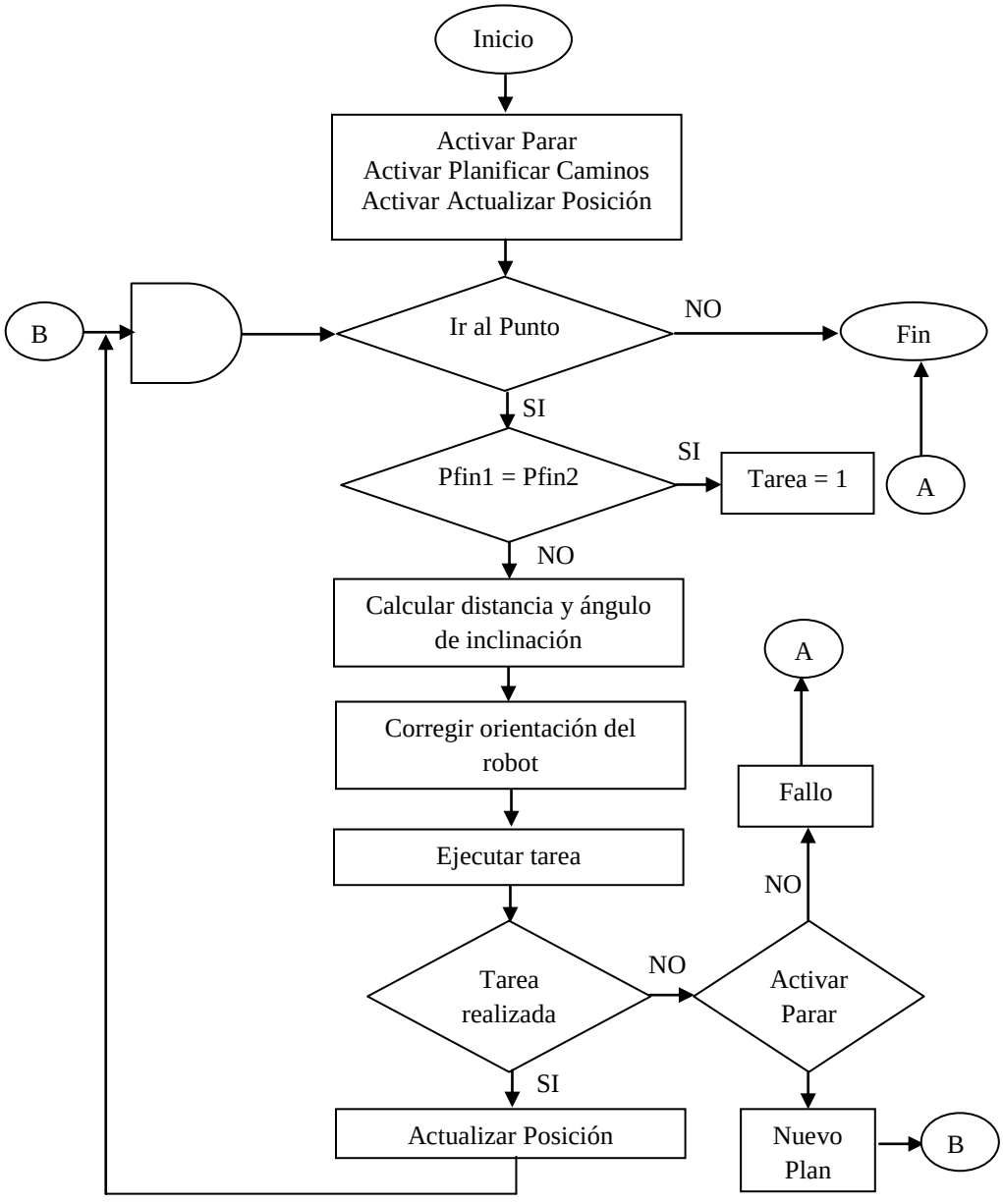


Figura 3.12 Diagrama de flujo de información del agente Ir al Punto.

4

Estado del Arte

4.1 Trabajos relacionados

A continuación se mencionan los trabajos que fueron revisados para conocer el enfoque que diferentes autores aplican al problema de localización y navegación de robots autónomos utilizando la Tecnología Lego Mindstorms. Así como aquellos trabajos que fueron tomados como base teórica respecto a la teoría de agentes.

4.1.1 Reconfigurable Multi Robot Society based en LEGO Mindstorms (Sociedad Multi Robot Reconfigurable basada en Lego Mindstorms).

Este trabajo fue desarrollado por David Leal Martínez en la Universidad Tecnológica de Helsinki. Su investigación describe la creación y el éxito de una sociedad de múltiples robots reconfigurables basados en los sistemas de LEGO Mindstorms NXT (Leal 2009), así como la descripción de dos prototipos que se crearon en el camino y no tuvieron éxito.
Aunque surgieron algunos problemas y el sistema no podía ser configurado con gran precisión como se deseaba, se obtuvieron buenos resultados. Entre los resultados importantes de esta investigación destaca que la creación de prototipos con Lego

Mindstorms es una forma rápida de prototipos mecánicos con algoritmos básicos de prueba con estabilidad y viabilidad. Por otro lado se demostró que el conjunto de Lego Mindstorms NXT no es totalmente dependiente de las limitaciones y especificaciones de Lego. Puede ser actualizado, es decir, se le pueden agregar más sensores y motores utilizando una tarjeta de expansión. El algoritmo de asignación de líder fue aplicado con éxito en LabVIEW, el cual podría ser la base de un algoritmo más complejo de asignación para el líder.

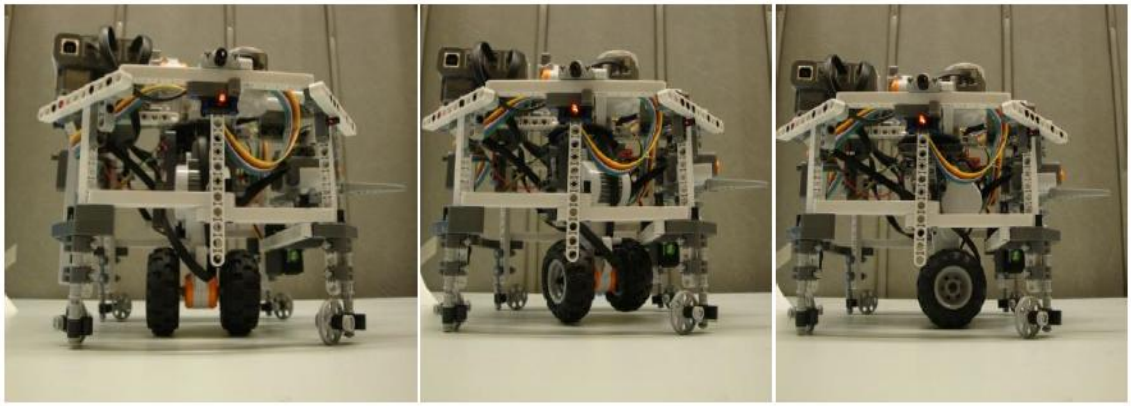


Figura 4.1. Prototipo elaborado con tecnología Lego Mindstorms. Se muestra el sistema de giro que lleva a cabo al levantar el sistema

La parte importante del prototipo desarrollado en este trabajo es el sistema de movimiento. Mientras el sistema de movimiento está en su posición más alta, las ruedas están unidas al movimiento del actuador y no están en contacto con el suelo.
De este modo el peso del vehículo esta soportado por las cuatro ruedas sueltas de las cuatro esquinas. Ahora el sistema de movimiento es dirigido por la rotación de la pieza de rotación principal. En la figura 4.1 se muestra como hace el movimiento de rotación de 90º antes de bajarlo.

4.1.2 Squadre di Robot Mobili basati su Tecnologia Lego Mindstorms (Equipo de Robots Móviles basados en la Tecnología Lego Mindstorms)

En este trabajo de investigación fue desarrollado por Daniele Benedetelli, en la Faculta de Ingeniería, de la Universidad de Siena, Italia. En este trabajo se describe el proceso de diseño y desarrollo de un equipo de robots móviles construidos con Lego Mindstorms.

El equipo pone a prueba una estrategia destinada a lograr el movimiento colectivo con una configuración específica de sistema multi-robot. Los robots deben colocarse en un círculo y girar en torno a una referencia virtual (figura 4.2), tratando de mantener una distancia constante entre ellos (Benedetelli 2006). La ley de control, está diseñada para una arquitectura descentralizada, y aplicada a un equipo que supervisa el robot con una cámara para controlar la distancia.

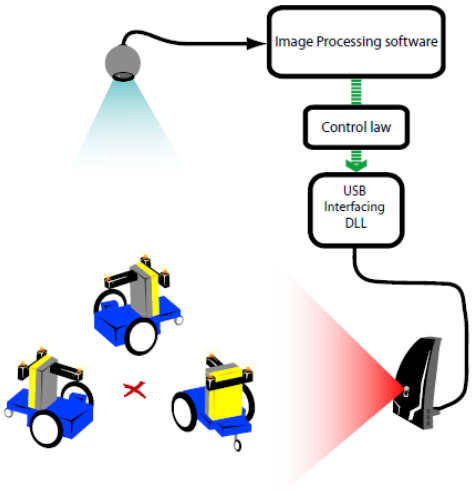


Figura 4.2. Sistema de supervisión

La aportación de este trabajo es el modelo cinemático. La generación del movimiento del robot está basada en un control en bucle cerrado. Se va variando la velocidad lineal y angular en función de la estimación de posición, que se va calculando a intervalos de tiempo establecidos. El robot móvil se modela como un monociclo (figura 4.3).

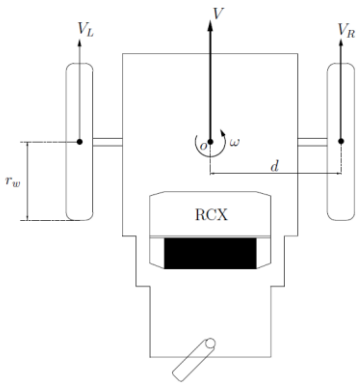


Figura 4.3. Vehículo Esquematizado

Considera el siguiente modelo de monociclo plano:

$$\dot{x}(t) = v\ \cos\theta(t)$$
$$\dot{y}(t) = v\ \text{sen}\,\theta(t) \qquad (3.1)$$
$$\dot{\theta}(t) = u(t)$$

El estado del robot es descrito por las variables x,y, θ, que representan la posición y orientación del vehículo en un sistema de referencia adecuado. Las entradas del sistema son la velocidad tangencial V(t) y la velocidad angular w(t).

Con referencia a la figura 3.3, w_R y w_L son la velocidad angular de la rueda derecha e izquierda respectivamente; d es la longitud del eje de transmisión de robot y r_w el radio de las ruedas motrices, se tiene:

$$V_R = W_R * R_w$$
$$V_L = W_L * R_w \qquad (3.2)$$

La relación entre la velocidad del vehículo y la velocidad de la rueda es la siguiente:

$$V = \frac{V_R + V_L}{2}$$
$$w = \frac{V_R - V_L}{2d} \qquad (3.3)$$

La primera se encuentra aplicado al principio de superposición, considerando por separado las contribuciones de las ruedas.

$$V/_{V_{L=0}} = \frac{V_R}{2}$$

$$V/_{V_{R=0}} = \frac{V_L}{2}$$

Donde $V = \frac{V_R}{2} + \frac{V_L}{2} = \frac{V_R + V_L}{2}$

La segunda se obtiene restando a la velocidad de una rueda la velocidad del vehículo y dividiendo entre la longitud del eje.

$$V_R - \frac{V_R + V_L}{2} = \frac{V_R + V_L}{2}$$

$$\frac{V_R - V_L}{2} = w * d$$

Donde $$w = \frac{V_R - V_L}{2d}$$

Sustituyendo (3.2) en (3.3) y resolviendo respecto a w_R y w_L, se tiene

$$w_L = \frac{V + d * w}{r_w}$$

$$w_L = \frac{V - d * w}{r_w} \qquad (3.3)$$

Esto muestra como se calcula la velocidad angular de cada rueda.

Los cálculos anteriores descritos fueron fundamentales para hacer los movimientos giratorios de este trabajo de investigación.

4.1.3 Navegación autónoma de un robot guiado por visión con operaciones básicas de localización y mapeo en un ambiente controlado.

Este trabajo fue publicado por Mariana N. Ibarra B., Juan M. Ramírez C., Alejandro Díaz M., Jorge Martínez C., Rogerio Enríquez C., e Irma J. García E. del Instituto Nacional de Astrofísica, Óptica y Electrónica (INAOE), Puebla, México.

El objetivo de este trabajo está orientado a la navegación autónoma, utilizando un sistema de visión por computadora y control difuso (Ibarra 2009). El sistema desarrollado permite también la experimentación con técnicas de localización y mapeo simultaneo básicas de un robot Lego NXT, que podrían ejecutarse completamente en la computadora transfiriendo vía bluetooth las señales de control para el desplazamiento del robot.

La arquitectura móvil se encuentra equipada con una cámara web y un sensor de ultrasonido para la percepción del ambiente y de los obstáculos presentes en el mismo (Fig 4.4).

Figura 3.4 Representación esquemática del sistema.

Un aspecto central del sistema es la búsqueda de la ruta más cercana en un mapa de localización tipo rejilla. El algoritmo de navegación implementado en este trabajo es Dijkstra, y fundamenta su utilización por diversas razones. En primer lugar, las numerosas e importantes aplicaciones existentes del mismo hacen de este uno de los más populares en la ciencia computacional, aumentando así la importancia de su aprendizaje; y por otra parte, la facilidad de su aplicación, ya que requiere un bajo nivel de procesamiento.

4.1.4 Programación de LEGO MindStorms bajo GNU/Linux

Este trabajo fue publicado por Vicente Matellán Olivera, Jesús M. González B., Pedro de las Heras Quiroz y José Centeno G., de la Universidad Rey Juan Carlos, Departamento de Ciencias Experimentales e Ingeniería, Madrid, España,

Esta investigación se centra en presentar la programación mediante software libre de los robots LEGO Mindstorms. Comienza dando un repaso a la situación emergente de la robótica comercial, centrándose especialmente en el papel de los LEGO Mindstorms [Matellán 2002], después aborda la programación bajo GNU/Linux, presentando el lenguaje NQC y el sistema operativo LegOS haciendo especial énfasis en las herramientas relacionadas utilizables bajo GNU/Linux. NQC es una versión reducida de C que permite el desarrollo rápido de programas mientras que LegOS es un sistema operativo completo que permite la programación en lenguajes tradicionales como C o C++. Además, presenta algunas herramientas para GNU/Linux relacionadas con LegOS como simuladores o compiladores web.

4.1.5 Robot Arena: Infrastructure for Applications Involving Spatial Augmented Reality and Robots (Robot Arena: Infraestructura para aplicaciones en realidad especial aumentada y robots)

Este trabajo fue desarrollado por Daniel Calife, Alexandre Tomoyose, Diego Spinola, João Bernardes, Romero Tori, de la Universiad Politecnica de Sao Pablo, Brasil. En esta investigación se describe una infraestructura de realidad aumentada compuesta por una cámara, un proyector, el software para el seguimiento, controlar y aumentar la realidad, y un robot real basado en la tecnología Lego Mindstomrs (figura 4.5).

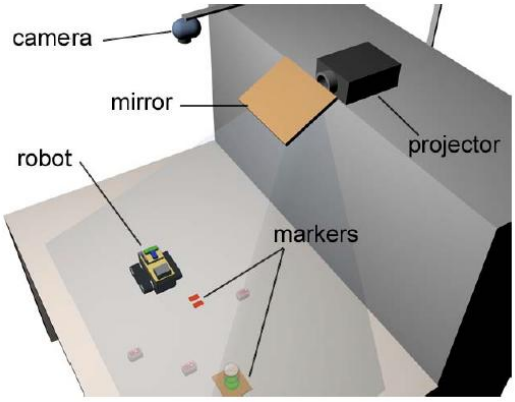


Figura 4.5 Infraestructura del Robot Arena

Este espacio aumentado del medio ambiente se destina a ser utilizado como banco de pruebas [Calife 2007] para experimentos en aplicaciones interactivas de realidad aumentada, como innovadora de juegos, aplicaciones educativas y la telepresencia.
La parte real de la infraestructura del Robot Arena consta de un robot armado con componentes de Lego Mindstorms y controlada a través de una conexión inalámbrica. Durante la construcción de este robot, surgieron algunas cuestiones importantes, como su control y comunicación, y el registro y seguimiento.

4.1.6 Modelo Adaptativo para Organizaciones Virtuales de Agentes.

Este trabajo fue desarrollado por Sara Rodríguez González de la Universidad de Salamanca, España. Presenta bases teóricas principalmente en la teoría de agentes y de las organizaciones virtuales, así como un marco para el desarrollo de organizaciones de agentes. Hace una revisión del estado del arte de los sistemas multi-agente desde un punto de vista organizacional junto con algunas áreas relacionadas entre las que destaca la coordinación y planificación de agentes. Estas tecnologías proporcionan una infraestructura robusta que permite a los agentes comunicarse y coordinarse de forma distribuida y obtener información del contexto de forma automática [Rodríguez, 2010]. Por otro lado propone un modelo integrador en el que una arquitectura de agentes puede ejecutar acciones basándose en un modelo de planificación social en una organización de agentes. La base del modelo es el razonamiento basado en casos, es decir, apoyándose en experiencias similares ocurridas en el pasado, en contextos similares, la organización actúa. El modelo ha sido utilizado para crear un entorno inteligente en un escenario real y se han evaluado positivamente sus capacidades para la planificación de una organización de agentes.

4.1.7 Navegación Autónoma de Robots en Agricultura: Un Modelo de Agentes

Este trabajo fué desarrollado por Lía García Pérez en la Facultad de Ciencias Físicas de la Universidad Complutense de Madrid, España. En esta investigación se presenta el diseño y desarrollo de una arquitectura de control y organización del conocimiento para la navegación autónoma de vehículos agrícolas en entornos dinámicos, denominada AGRO-AMARA. Esta es una arquitectura híbrida basada en agentes organizados jerárquicamente en lo que se refiere a la reutilización de habilidades; diseñada específicamente para vehículos y labores agrícolas. En esta arquitectura los agentes comparten información sobre la base de dos mecanismos, paso de mensajes y memoria compartida, en este caso concreto con la estructura de una memoria de pizarra. El diseño de arquitectura sigue una filosofía conservadora y oportunista en todas sus facetas que busca lograr los principios básicos de modularidad, reusabilidad de procesos, y de facilidad de escalado e implantación en diferentes plataformas [García, 2004]. El principal aporte de la arquitectura AGRO-

AMARA para el desarrollo de este trabajo es la definición dos tipos de agentes en base al tipo procesamiento: agentes perceptivos y agentes de actuación.

Los agentes perceptivos están enfocados a la elaboración y al mantenimiento de los estímulos requeridos en las tareas de control. La percepción está orientada a la acción, mientras que los recursos y los mecanismos de control perceptivo vienen determinados por el comportamiento.

Los agentes de actuación tienen como objetivo encaminar la acción a seguir, utilizando las percepciones elaboradas por los agentes perceptivos y el soporte proporcionado por una representación del sistema y el mundo. Las percepciones se almacenan en una memoria compartida de tipo pizarra [Hayes-Roth, 1985]. Este modelo, empleado habitualmente en arquitecturas de control [Matellán y Borrajo, 2001, Konolige et al., 1997], permite almacenar estructuras con información con grados de abstracción diferente. Así permite que cada agente trabaje con el tipo de representación que le resulte más adecuado. La utilización de una representación de tipo pizarra, permite desacoplar los niveles perceptivos de los de actuación. Esto proporciona a la arquitectura mayor flexibilidad, ya que permite a los agentes de actuación de alto nivel tomar percepciones de bajo nivel si así es requerido.

Los agentes en AGRO-AMARA, constan de tres aspectos fundamentales: un conjunto de procesos de cómputo y de comunicación que definen la competencia de cada agente, la representación del sistema y del entorno con distinto grado de resolución, y el conjunto de entradas y salidas asociadas a cada agente

5

Metodología de Solución

La representación del ambiente y la localización de un robot móvil en un entorno conocido es uno de los problemas fundamentales de la robótica móvil. El robot debe ser capaz de contestar en cada momento a las siguientes tres preguntas: ¿dónde estoy? ¿Hacia dónde debo moverme? y ¿Cómo llegaré hasta allá? De estas tres preguntas surgen tres problemas fundamentales de la navegación: el problema de la localización, el problema de la planificación de trayectorias y el problema de la utilización de esquemas de actuación.

La finalidad de los métodos de localización y mapeo simultáneos (SLAM) es que un robot pueda construir de forma incremental un mapa de su medio de navegación [Durrant-Whyte, 2006]. Mientras utiliza este mapa estima la trayectoria realizada por el robot al recorrer el ambiente desconocido sin ninguna información previa. De esa forma el robot tendrá la información necesaria para realizar tareas como planificar caminos, generar trayectorias, evitar obstáculos, entre otras cosas.

La problemática de este trabajo de investigación está inmersa dentro de la línea de investigación del SLAM, aunque para este trabajo el sistema ejecuta movimientos coordinados en un medio conocido y estructurado.

5.1 Infraestructura Robótica

Este trabajo propone una plataforma robótica basada en la tecnología Lego Mindstorms NXT. Lego Mindstorms fué uno de los resultados de la fructífera colaboración entre Lego y el MIT (Massachusetts Institute of Technology). Esta asociación se emplea como ejemplo de relación entre la industria y la investigación académica que resulta muy beneficiosa para ambos socios [Mindell 2000]. Lego financiaría investigaciones del grupo de epistemología y aprendizaje del MIT sobre cómo aprenden los niños y a cambio obtendría nuevas ideas para sus productos, que podría lanzar al mercado sin tener que pagar regalías al MIT. El fruto de esta colaboración fue el desarrollo del MIT Programmable Brick (Ladrillo programable).

La principal característica de esta tecnología es su flexibilidad, puesto que gracias a sus sensores y actuadores, así como a la unidad de control programable, se pueden desarrollar una gran variedad de proyectos y actividades [Valera 2007].

Los componentes más importantes de esta tecnología son la unidad de control, los sensores electrónicos y los actuadores (Figura 5.1). La unidad de control está basada en su potente microcontrolador ARM7 de 32 bits, que incluye 256 Kb de memoria Flash y 64 Kb de memoria RAM. Para las comunicaciones, el NXT está equipado con un puerto USB 2.0 y con un dispositivo inalámbrico Bluetooth clase II, V2.0. Así mismo, el NXT dispone de 4 entradas (una de ellas incluye una expansión IEC 61158 Type 4/EN 50 170 para usos futuros) y 3 salidas analógicas. Para las entradas existen cuatro tipos de sensores electrónicos: de contacto, de luz, de sonido y de distancia. En este trabajo solo se considera el sensor de distancia.

Figura 5.1. Componentes del Lego Mindstorms NXT

El sensor de distancia (ultrasónico) proporciona al robot un sentido de visión. El sensor emite un sonido y mide el tiempo que la señal tarda en regresar, para luego calcular la distancia, a la cual se encuentra el objeto u obstáculo. Es el mismo principio utilizado por los murciélagos y el sonar de las naves, tiene una rango de 0 a 255 cm con una precisión de +/- 3 cm. El ultimo componente de Lego y el más importante son los actuadores. Estos son motores de corriente continua que incorporan sensores de rotación con una precisión de +/-1 grado. Para el movimiento de un modelo motorizado el firmware (el sistema operativo interno del NXT), dispone de un sofisticado algoritmo PID, el cual permite que el modelo se desplace con precisión.

5.1.1 Diseño y Construcción del Robot LEGO NXT tipo vehículo

Una vez determinadas las especificaciones de funcionalidad de los componentes electrónicos se procedió a la construcción del Robot (Anexo A). Se ensamblaron cada uno de sus componentes con las piezas de sujeción y dándole la ubicación adecuada a cada componente electrónico de acuerdo a sus especificaciones y a las necesidades del proyecto. Como se puede apreciar en la figura 5.2, esta base de robot (chasis) utiliza dos motores de accionamiento y una rueda giratoria de ricino. Esta rueda se utiliza para que fácilmente pueda acoplarse en cualquiera tipo de piso, alfombras o suelos duros. Tiene un tercer motor montado debajo en el centro con algunos puntos de anclaje para las cosas que se pueden adjuntar más adelante. En la parte frontal está colocado el sensor de distancia, el cual se utiliza para detectar objetos en su trayecto. Este chasis es para uso robusto.

Figura 5.2 Robot tipo vehículo.

5.1.2 Control de Movimiento de los Robots.

El robot funciona con tres ruedas, dos de ellas acopladas a un servomotor cada una, y otra libre en la parte trasera con la única intención de equilibrar el peso. Este modelo de robot tiene una cinemática diferencial (Figura 5.3), la cual consiste en dar movimiento independiente a cada una de las dos ruedas tractoras. De manera que combinando las velocidades de una y otra, y el sentido de giro, se consigue dar velocidad angular al robot.

La generación del movimiento del robot está basada en un control en bucle cerrado, donde se va variando la velocidad lineal y angular en función de la estimación de posición, que se va calculando a intervalos de tiempo establecidos.

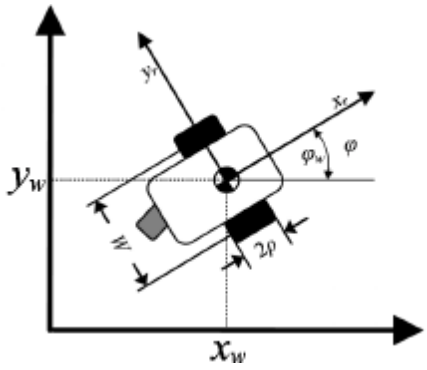


Figura 5.3 Unidad diferencial, marco de referencia y parámetros [Ojha 2009].

La generación del movimiento del robot está basada en un control en bucle cerrado, donde se va variando la velocidad lineal y angular en función de la estimación de posición, que se va calculando a intervalos de tiempo establecidos.

La dinámica de la unidad puede ser descrita utilizando el siguiente modelo cinemático [Ojha 2009]:

$$\begin{bmatrix} V_{ref} \\ W_{ref} \end{bmatrix} = \begin{bmatrix} \frac{\rho}{2} & \frac{\rho}{2} \\ \frac{\rho}{w} & -\frac{\rho}{w} \end{bmatrix} \begin{bmatrix} w_r \\ w_l \end{bmatrix} \quad (1)$$

donde V_{ref} es la velocidad lineal de la unidad, W_{ref} es la velocidad angular de la unidad, ρ es el radio de las ruedas y w es la distancia entre las ruedas motrices. Las velocidades angulares de la rueda derecha y la rueda izquierda están representadas por w_r y w_l respectivamente.

Resolviendo (1) es posible calcular w_r y wl aplicando la inversa de la matriz (que es constante y no singular, por lo que solo se calcula una vez). Como se trata de un giro puro, w_r y w_l son de signo contrario pero con el mismo valor absoluto.

$$w_r = \frac{w * w_{ref}}{2\rho} \quad (2)$$

$$w_l = -\frac{w * w_{ref}}{2\rho} \quad (3)$$

Para este trabajo de investigación es necesario encontrar la constante de tiempo que debemos aplicar para obtener el giro de acuerdo a los grados que se indiquen. Para ello tenemos que el incremento en giro está dado por:

$$\Delta_{\emptyset} = \mathrm{w_{ref}} * \mathrm{t} \quad (4)$$

Despejando obtenemos la velocidad angular:

$$\mathrm{w_{ref}} = \frac{\Delta_{\emptyset}}{\mathrm{t}} \tag{5}$$

Sustituyendo en (2) obtenemos:

$$w_r = \frac{w \ * \ \Delta_{\emptyset}}{2\rho t} \tag{6}$$

Para obtener el tiempo que debemos aplicar a cada motor, despejamos ***t*** de (6):

$$t = \frac{w \ * \ \Delta_{\emptyset}}{2\rho w_r} \tag{7}$$

5.1.3 Entorno de Programación.

Una característica que tienen en común muchos robots, es que poseen recursos computacionales limitados, tal es el caso del procesamiento y el almacenamiento [Valera, 2005]. Para tratar de evitar estos problemas en este trabajo proponemos un entorno alternativo. La filosofía de trabajo cambia puesto que se utiliza una computadora como unidad de control y no el NXT. Dada la complejidad de las operaciones a realizar en este trabajo, el software se construyó de forma modular. De tal manera que se puedan reutilizar funciones de alto nivel, simplemente llamando a las funciones correspondientes de bajo nivel según el modelo cinemático y los parámetros del robot utilizado. En la figura 5.4 puede verse de manera esquemática los entornos de desarrollo y lenguajes utilizados.

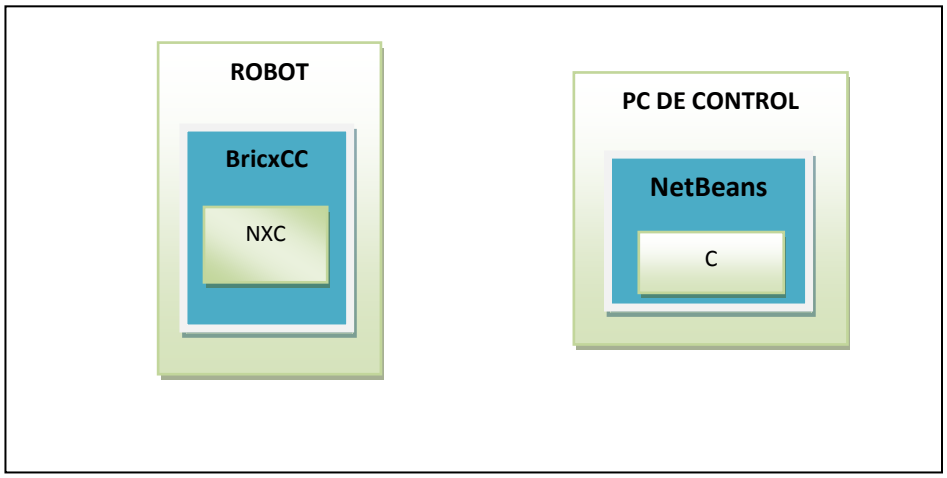


Figura 5.4 Arquitectura del software de desarrollo

Para este tipo de entorno propuesto se tienen dos plataformas de trabajo. Una de estas plataformas, está montada sobre el sistema Operativo Windows 7 (Robot), la cual se utiliza para programar las actividades que realizaran los robots. Estos programas están realizados en un lenguaje de alto nivel basado en C llamado NXC y el entorno de desarrollo BricxCC, el cual es muy completo y cómodo de utilizar. Una vez que los programas son desarrollados estos se transfieren al robot a través del puerto USB o Bluetooth. Por otro lado tenemos la parte importante de esta plataforma de desarrollo, la PC de Control. Esta tiene como base el sistema operativo Linux y como entorno de desarrollo NetBeans.

Debido a la complejidad de los algoritmos de control que se realizan en este proyecto y al procesamiento de los algoritmos de visión artificial, la parte medular de la programación de este trabajo de investigación está desarrollada en lenguaje C.

5.2 Comunicación entre la computadora y los robots

El módulo de comunicaciones permite que una PC de Control pueda enviar órdenes al robot, así como obtener las lecturas de sus sensores y su posición. Sirviéndose de estos datos, la aplicación en la PC de Control mostrará de manera visual el mapa del entorno con el robot y los valores de posición en tiempo real. Además permitirá de forma fácil enviar órdenes de tareas y objetivos al robot.

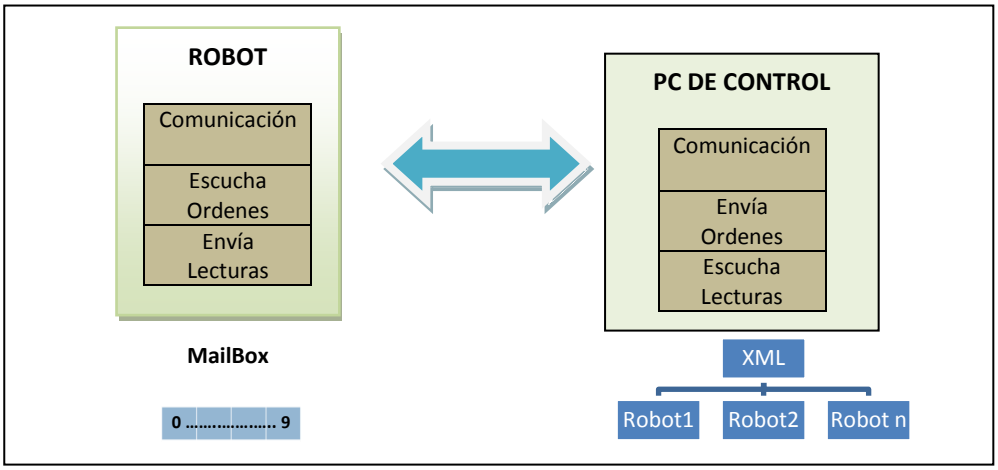


Figura 5.5 Esquema de comunicación.

En la figura 5.5 se describe como el robot guarda sus mensajes en un repositorio de memoria llamado *mailbox*. De este repositorio pasan a un árbol de instrucciones XML de donde se controlan todas las tareas que deben ser ejecutadas por los robots. Los *mailbox* son simplemente buffers en donde se almacenan los mensajes. El sistema operativo del ladrillo NXT tiene 10 *mailbox*.

El mensaje que la PC de Control envía al robot tiene la estructura mostrada en la figura 5.6. Estos mensajes se encuentran almacenados en la estructura de árbol XML, de donde son obtenidos por la PC de Control y enviados al Robot. Una vez que el robot lee el mensaje en su *mailbox* de entrada, este responde en un *mailbox* de salida de donde la PC de Control lo rescata.

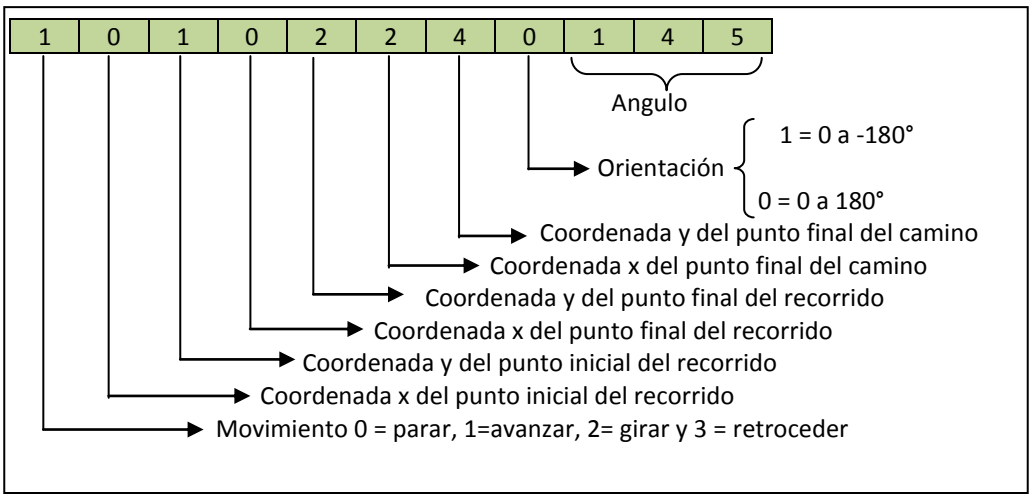


Figura 5.6 Estructura del mensaje enviado por la PC de Control al Robot.

La respuesta que el robot envía a la PC de Control es solo un carácter, regresa un 1 si la tarea fue realizada satisfactoriamente y 0 si ocurrió un error. Como anteriormente mencionamos esta respuesta es almacenada en un *mailbox* de salida diferente al de entrada.

El algoritmo 1 muestra una descripción general del envío y recepción de mensajes entre la PC de Control y los robots.

Algoritmo 1. Comunicación

Algoritmo de comunicación en PC de Control

1: **abrir** comunicación con el robot a través de bluetooth.
2: **conectar** el robot (hilo)
3: **leer** tareas de archivo XML
4: **mientras** haya tareas **hacer**
5: **ejecutar** programa en robot
6: **escribe** mensaje al robot en *mailbox* 3
7: **esperar** respuesta
8: **leer** respuesta desde *mailbox* 2 (hilo)
9: **si** respuesta es 0 **entonces**
10: **regresa** error
11: **fin si**
12: **fin mientras**
13: **finaliza** conexión (hilo)

Algoritmo de comunicación en Robot

1: **recibe** el mensaje en mailbox 3
2: **extrae** datos del mensaje
3: **ejecuta** tarea
4: **si** hay error al ejecutar tarea **entonces**
5: mensaje = 0
6: **sino**
7: mensaje = 1
8: **fin si**
8: **escribe** respuesta en mailbox 2

5.3 Identificación de los robots en el ambiente

En este trabajo se identifican a los robots a través de marcas artificiales. Estas generalmente son patrones que uno puede diseñar y poner intencionalmente en el ambiente para facilitar su identificación y localización (Figura 5.7). Son obtenidas a través de visión ya que poseen características invariantes en cuanto a escala, orientación y parcialmente a la iluminación (SIFT) [Lowe, 2004].

Las marcas artificiales se han utilizado con éxito en la literatura como un medio tanto para dotar como para proporcionar información adicional a un robot y facilitar los proceso de localización así como el desarrollo de otro tipo de tareas [Muñoz, 2006].

Figura 5.7. Marcas visuales artificiales para la identificación de los Robots.

Para poder identificar a los robots y al objeto meta dentro del ambiente estructurado, fue necesario hacer un entrenamiento previo para el reconocimiento de las marcas. Es decir,

primero se grabaron los histogramas de cada una de las marcas que identifican a los robots y al objeto meta (Algoritmo 2).

Algoritmo 2. Guardar Histograma

1: Se **captura** la imagen
2: **Crear** estructura de datos para la visualización de la imagen, objeto seleccionado e Histograma
3: La imagen capturada se convierte al espacio de color **HSV**
4: **Si** hay objeto seleccionado
5: **Verificar** que cada pixel de la imagen HSV se encuentre dentro de los rangos y se almacena en una máscara booleana
6: **Copia** el canal H de la imagen HSV en **hue**
7: **Selecciona** la región de interés en la imagen
8: **Selecciona** la misma región de interés de la mascara
9: **Calcula** el histograma de la imagen hue, lo guarda en hist de acuerdo a la mascara boolena
10: **Se obtiene** el valor máximo del histograma
11: **Para** cada bin
12: **Se dibuja un rectángulo** de color diferente para cada canal RGB de la imagen del histograma
13: **Fin para**
14: **Fin si**
15: **Se guarda** el histograma

Posteriormente se llevó a cabo la búsqueda de los histogramas en el ambiente utilizando el algoritmo Camshift y adaptándolo a las necesidades del proyecto y lograr así cumplir este objetivo (Algoritmo 3).

Algoritmo 3. Reconocimiento de Marcas

1: **Recuperar** Histograma guardado en **hist**
2: Se **captura** la imagen en **frame**
3: **Establecer** rectángulo de atención sobre el frame (**track_window**)

4: **Crear** una imagen del tamaño del frame en **backproject**
5: Recoge los valores de los canales seleccionados (**hue**) de la imagen de entrada (**hist**) y lo almacena en **backproject**
6. Se llama a la función **cvCamShift**(backproject, track_window , TermCriteria (CV_TERMCRIT_EPS | CV_TERMCRIT_ITER, 10,1))

Para obtener la posición y la orientación de los robots, fue necesario implementar un algoritmo que reconociera los contornos de la figura. Este método consiste en utilizar la ventana del objetivo (track_window) y aplicar un filtro (cvFindContours) para obtener los contornos en la imagen como una lista de secuencias de vértices. Se "aplanan" los contornos en aquellas secuencias con más de 5 vértices para aproximarlos a polígonos, y se analizan solo aquellos polígonos que quedan con 5 vértices. Se buscan las líneas con ángulo interior menor que corresponde a la punta de la flecha y se busca un punto opuesto sobre el polígono para determinar su inclinación y orientación (Algoritmo 4).

Algoritmo 4. Posición y orientación de los robots

1: **Dibuja** un rectángulo sobre el objetivo en la imagen original
2: **Obtiene** el número de **contornos**
3: **mientras** haya contornos hacer
4: **si** la figura tiene más de 5 vértices
5: **si** es polígono
6: se **repintan** los vértices
7: **fin si**
8: **se determina la punta de la flecha** como el vértice de menor ángulo
9: **se determina el punto intermedio** del extremo posterior a la flecha
10: se determinan los puntos **x y**
11: se calcula la **pendiente**
12: se calcula el **ángulo de inclinación**
13: **sino** “no construyó el polígono”
14: **fin si**
15: **siguiente contorno**
16: **fin mientras**

5.4. Construcción del mapa delimitando el entorno de cada objeto.

No es posible establecer una trayectoria, si no se tiene conocimiento previo del espacio libre por el cual los robots se puedan desplazar y de las posiciones en donde puede encontrarse en colisión con otros objetos.

Una forma de adquirir esta información es la construcción del espacio de configuraciones (C-space) $\boldsymbol{C}$. El C-space $\boldsymbol{C}$ es el conjunto de todas las posibles posiciones $\boldsymbol{q}$ que puede ocupar un objeto $\boldsymbol{A}$ en un medio ambiente $\boldsymbol{W}$ con un conjunto de obstáculos $\boldsymbol{O}$. Aquellas configuraciones $\boldsymbol{q}$ donde $\boldsymbol{A}$ entra en colisiones con $\boldsymbol{O}$ se conocen como espacio de configuraciones obstáculo $\boldsymbol{C_{obs}}$, el complemento de $\boldsymbol{C_{obs}}$ es el espacio de configuraciones libre $\boldsymbol{C_{libre}}$, y son las configuraciones $\boldsymbol{q}$ donde $\boldsymbol{A}$ está libre de colisiones.

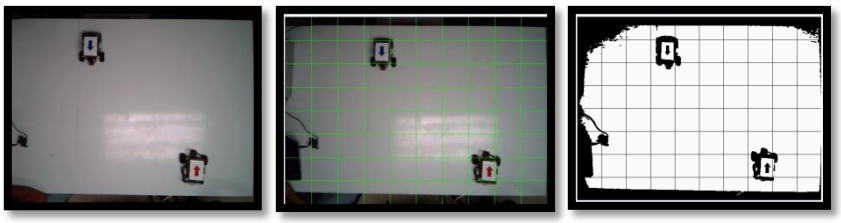
El ambiente $\boldsymbol{W}$ que se maneja en este trabajo es un ambiente plano $\boldsymbol{W} = \boldsymbol{R}^2$ con obstáculos $\boldsymbol{O}$ fijos. El objeto $\boldsymbol{A}$ y los obstáculos $\boldsymbol{O}$ están representados por las coordenadas de los vértices que los delimitan. Para determinar cuáles son las configuraciones de espacio libres, se utilizaron algoritmos basados en el procesamiento de imágenes utilizando las librerías de OpenCv de Intel.

Los obstáculos son identificados a través del mapa de localización tipo rejilla. Se hace un barrido de cada uno de los recuadros del mapa y en cada uno de ellos se hace un conteo de los pixeles obscuros. Si el total de este conteo es mayor a tres cuartas partes del total de pixeles del recuadro, entonces determinamos que esa posición está ocupada. En el Algoritmo 5 se presenta una descripción del proceso [Ibarra, 2009] y la Figura 5.8 los resultados obtenidos.

Algoritmo 5. Construcción del ambiente

1: **Captura** de un cuadro de la **imagen**
2: **Segmentación** del ambiente en una rejilla
3: **Conversión** a escala de **grises**
4: **Mejora** del contraste de la imagen y **eliminación** del ruido
5: **Binarización** de la imagen
6: **Para** cada rejilla

```
7:      Para cada pixel de la rejilla
8:          si el pixel es negro
9:              aumento el contador de ocupado
10:         fin si
11:     si ocupado es mayor o igual a las ¾ partes del total de los pixeles de la rejilla
12:         mapa[rejilla] = -1 // significa ocupado
13:     sino
14:         mapa[rejilla] = 0 // significa desocupado
13:     fin si
14: fin para
```



(a) (b) (c)

Figura 5.8 Procesamiento de la Imagen. (a) Captura del ambiente real. (b) Segmentación del ambiente en una rejilla. (c) Conversión a escala de grises, mejora de la imagen y binarización de la imagen.

5.5 Planeación de la trayectoria

La planificación de tareas, básicamente son las acciones que un robot realizará para ir desde un punto inicial a un punto final, evitando los obstáculos que halla en el ambiente. Para esto utilizamos el algoritmo de Dijkstra, dado que este, requiere un bajo nivel de procesamiento. El ambiente donde se ejecuta el algoritmo es una matriz de cuadros. Donde cada cuadro representa una posición que puede ser ocupada por el robot.

Antes de ejecutar el algoritmo ya se conoce el punto inicial (robot) y el punto final (el objeto), así como también el número y la posición de los obstáculos. Se genera un determinado número de nodos con los espacios vacios en la matriz de cuadros con los cuales el algoritmo dará la ruta más óptima para cada robot y con estas encontrar el objeto.

La información obtenida se almacena en un archivo XML cuya estructura se puede observar en la figura 5.9 y un ejemplo de esta estructura con información real en la figura 5.10. Es importante mencionar que este sistema se limita a trabajar sobre un ambiente de dos dimensiones (x,y).

```
<?xml version="1.0" encoding="utf-8"?>
<ambiente>
        <Robot_1>
                <Itinerario><Iti1 tarea="NULL" status="0" />
                </Itinerario>
                <Actividad tarea="NULL" status="NULL" />
                <Posicion xini="NULL" yini="NULL" xfin="NULL" yfin="2" />
        </Robot_1>
        <Robot_2>
                <Itinerario><Iti1 tarea="NULL" status="0" />
                </Itinerario>
                <Actividad tarea="NULL" status="NULL" />
                <Posicion xini="NULL" yini="NULL" xfin="NULL" yfin="NULL" />
        </Robot_2>
</ambiente>
```

Figura 5.9. Estructura del Archivo XML

```
<ambiente>
        −<Robot_1>
                −<Itinerario>
                        <Iti1 tarea="1414224-62" status="0"/>
                        <Iti2 tarea="14232240" status="0"/>
                        <Iti3 tarea="1323324-89" status="0"/>
                        <Iti4 tarea="13334240" status="0"/>
                        <Iti5 tarea="13424240" status="0"/>
                </Itinerario>
                <Actividad tarea="13424240" status="0"/>
                <Posicion xini="4" yini="1" xfin="2" yfin="4"/>
         </Robot_1>
        −<Robot_2>
                −<Itinerario>
                        <Iti1 tarea="1010224-63" status="0"/>
                        <Iti2 tarea="10212240" status="0"/>
                        <Iti3 tarea="112132489" status="0"/>
                        <Iti4 tarea="11314240" status="0"/>
                        <Iti5 tarea="11424240" status="0"/>
                </Itinerario>
                <Actividad tarea="11424240" status="0"/>
                <Posicion xini="0" yini="1" xfin="2" yfin="4"/>
        </Robot_2>
</ambiente>
```

Figura 5.10 Archivo XML

6

Experimentación y Resultados

En esta sección se presentan una serie de experimentos que permitieron corroborar la funcionalidad de la plataforma desarrollada. Primeramente se presentan las pruebas que se llevaron a cabo para que el robot ejecutara los giros de acuerdo a los grados requeridos. Posteriormente se realizaron pruebas para reconocer las marcas que identificarían a cada robot y por último se presentan los aspectos relevantes sobre el diseño de las pruebas completas así como de los resultados obtenidos.

6.1. Pruebas y calibración de servomotores.

Como anteriormente se mencionó, el robot construido para este trabajo funciona con tres ruedas, dos de ellas acopladas a un servomotor cada una, y otra libre en la parte de atrás con la única intención de equilibrar el peso. Este modelo de robot tiene una cinemática diferencial, la cual consiste en dar movimiento independiente a cada una de las dos ruedas tractoras, de manera que combinando las velocidades de una y otra, y el sentido de giro, se consigue dar velocidad angular al robot [Benedetelli, 2006;Ohja, 2009]. La experimentación consiste en probar una serie de valores para obtener la velocidad angular

de la rueda de acuerdo a una potencia conocida y con ello calibrar los motores para su correcto funcionamiento. Para este experimento se hicieron cinco ejecuciones del programa mostrado en la figura 6.1 para cada ángulo propuesto.

```
task  giro()
{          int x;
           float w = 11.5;     // distancia entre ruedas
           float d = 5.6;      // diámetro de las ruedas
           float ct = 8.85;   // constante determinada por experimentación
           float pi = 3.1416;
           int gr = 180;       // grados del giro
           int pot = 50;       // potencia de los motores
           float wr = pot/ct;
           float t=((w*((gr*pi)/180))/(d*wr))*1000; // calculo de tiempo
           string tiempo = NumToStr(t);
           TextOut(0, LCD_LINE5,tiempo);
           Wait(2000);
           OnFwd(OUT_A,pot);
           OnRev(OUT_B,pot);
           Wait(t);
           Off(OUT_AB);  }
```

Figura 6.1 Función para determinar el ángulo de giro del robot.

En la figura 6.2 se muestra la superficie donde se llevaron a cabo las pruebas para evaluar la eficacia de la función mostrada en la figura 6.1.

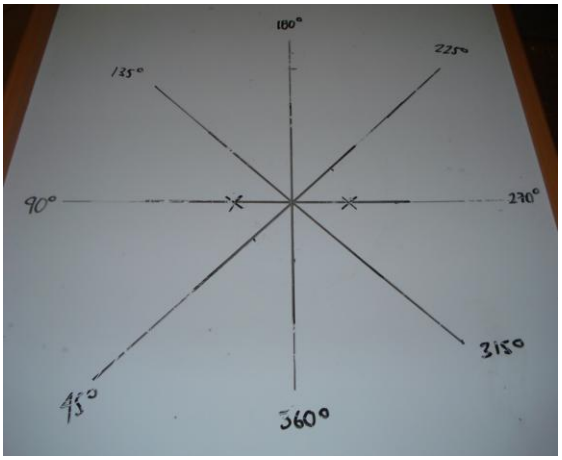


Figura 6.2 Superficie para experimentación para probar la función del ángulo de giro.

En la tabla 6.1 se muestran los valores de la constante promedio para cada uno de los ángulos utilizados en la experimentación, así como el valor de la constante promedio general.

Tabla 6.1.
Valor de constante para el giro

Ángulo	Constante
45°	10.5
90°	10.5
135°	10.6
180°	10.6
225°	10.4
270°	10.4
315°	10.4
360°	10.6
Promedio	**10.5**

Los resultados de la estimación del giro (Tabla 6.2) están dentro de lo esperado, teniendo en cuenta la precisión de los motores utilizados. Se llevaron a cabo 5 pruebas para cada uno de los ángulos con el valor promedio de la constante resultante, para las cuales en algunos casos generó un error promedio de +- 2°.

Tabla 6.2
Experimentación de giros con constante estimada y función de Giro

Angulo/Ejecución	1	2	3	4	5
	Error en el ángulo				
45°	0	0	0	0	1
90°	0	0	1	0	0
135°	-2	-2	0	-1	-2
180°	0	0	1	1	0
225°	-1	-1	0	0	-2
270°	0	2	1	2	2
315°	2	2	2	2	1
360°	1	1	0	0	1

Esta experimentación se llevo a cabo en un robot con llantas planas sobre una superficie lisa y con un nivel de carga de batería no menor a 8000 mA.

6.2. Experimentación con el sistema de visión

Antes de realizar pruebas completas con todo el sistema, se decidió realizar pruebas individuales de reconocimiento de patrones por parte del sistema de visión.

Como se mencionó en la sección 6.3 en este trabajo, se utilizan marcas artificiales para la identificación del robot, así como también se menciona la utilización del algoritmo Camshift para identificar dichas marcas. Por ello se llevaron a cabo pruebas para obtener la combinación de colores que proporcionaran mejores resultados. La tabla 6.3 muestra esta combinación de colores. Para obtener los resultados de la tabla 6.3 se corrió la función *"Seguimiento"* mostrada en los anexos de este trabajo.

Tabla 6.3
Experimentación con colores en marcas artificiales

Color de Fondo	Color del Centro	Tiempo de reconocimiento	Observaciones
Blanco	Amarillo	Indefinido	
Amarillo	Blanco	Indefinido	
Azul	Amarillo	0:20 s	inestable con obstáculos
Amarillo	Azul	Indefinido	
Café	Blanco	Indefinido	
Rojo	Amarillo	0:06 s	
Amarillo	Rojo	0:20 s	Inestable
Azul cielo	Amarillo	0:15 s	Inestable
Verde	Amarillo	0:04 s	
Rosa	Verde	Indefinido	

Esta combinación de colores se da principalmente por el color de fondo que tenemos en el ambiente, el cual es un color grisáceo. Tomando en cuenta el espectro de colores la combinación de colores rojo – amarillo y verde-amarillo, se encuentran cerca uno del otro

en el espectro y alejado del color gris. Con esto se comprueba el porqué se da esta combinación de colores en el ambiente.

Por otro lado el centro de estas marcas es un polígono de cinco lados montado sobre una superficie de color diferente, esto con el objeto de crear un histograma diferente para cada robot. Esta figura poligonal será procesada a través del Algoritmo Contornos con la finalidad de obtener la posición y orientación del robot.

En la figura 6.3 se muestra la forma en que el robot es detectado en el ambiente a través del sistema de visión desarrollado para este proyecto.

Figura 6.3 Identificación del robot por medio de la marca artificial.

Otra de las características importantes de las marcas, además de la combinación de colores, es el tamaño de estas. Entre mayor sea el tamaño de la marca mayor será la distancia desde la cual pueda ser reconocida por el sistema. Para este trabajo tenemos ubicada la cámara a 245 cm del piso, de forma perpendicular, por lo que podemos detectar marcas de hasta 15x15 cm. Para este caso y por necesidades del sistema, el tamaño de la marca es de 21.5x14 cm. Esto con el propósito de cubrir totalmente al robot para que la figura central de esta marca al ser procesada, no se confunda con las partes que conforman la estructura del robot.

6.3. Descripción de los experimentos completos del sistema

El conjunto de pruebas realizadas en este trabajo de investigación está constituido en un ambiente poligonal convexo, en el cual los elementos de inicio y meta tienen diferente posición para cada experimento. Se realizan pruebas con diferentes ambientes para un sistema de dos robots para observar los movimientos de cada uno de ellos de acuerdo a la ruta trazada por el sistema.

6.3.1 Ambientes

Los ambientes utilizados tienen una dimensión de 190x155 cm., este a su vez está dividido en rejilla de 38x31 cm., haciendo un total de 25 de estas. El número de obstáculos es variado y sus dimensiones son de tamaño uniforme de tal manera que cubran más de tres cuartas partes de la rejilla para que pueda ser identificado como obstáculo en el sistema.

6.3.2 Ambiente de implementación

El Hardware utilizado para la experimentación fue el siguiente:

- Robots Lego Mindstorms NXT 2.0
- Desktop, procesador Intel Pentium 4, 2.4 GHz, RAM 2Gb, DD 80Gb.
- Laptop Lenovo ThinkPad , procesador Intel Core 2 Duo, 1.3 GHz, RAM 3 Gb, DD 320Gb.

El Software necesario para realizar las experimentaciones fue el siguiente:
Desktop:

- Sistema Operativo Windows XP SP2
- IDE Brixc Command Center 3.3
- Lenguaje de programacion NXC

Laptop:

- Sistema operativo Ubuntu 10.10
- IDE NetBeans 9.0
- Lenguaje C

6.3.3 Experimentos y resultados

Las experimentaciones se llevaron a cabo en 5 distintos ambientes, ejecutándose 4 veces cada uno. Los resultados obtenidos en las ejecuciones es si los robots han llegado a la meta y el tiempo en que lo hicieron. Se debe de tomar en cuenta que aun teniendo las delimitaciones y restricciones del robot real y del ambiente en el cual se llevan a cabo estas pruebas, se garantiza el desempeño de este sistema de una manera aceptable.

6.3.3.1 Experimento 1.

Objetivo

Dado dos robots Lego Mindstorms NXT en el ambiente mostrado en la figura 6.4, realizar una navegación de la posición donde se encuentran los robots a un objeto que representa la meta utilizando el sistema desarrollado en este trabajo.

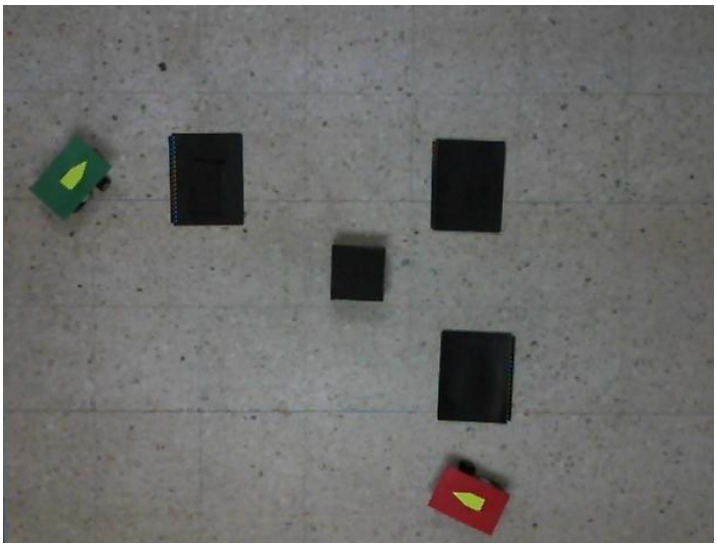


Figura 6.4 Mapa de ambiente para experimento 1.

Procedimiento

Utilizar el sistema desarrollado en este trabajo para obtener la posición y la orientación en el ambiente que tienen los robots al iniciar el proceso, localizar los obstáculos y localizar dentro del mapa la posición del objeto meta. Posteriormente el sistema trazará una

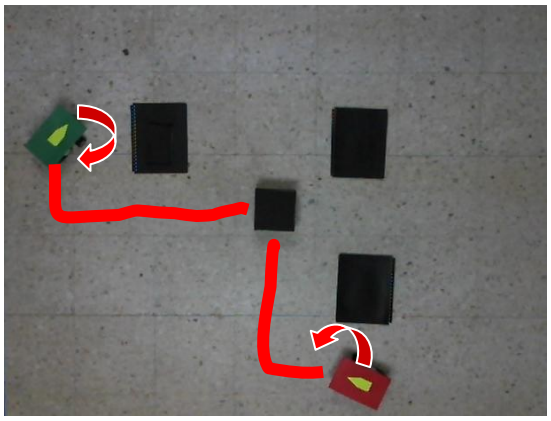
trayectoria para cada robot con la cual debe de navegar y llegar al objeto meta. Se ejecutan 4 pruebas con este ambiente, se registra el tiempo de cada trayectoria y se indica si el robot logro su objetivo en cada prueba.

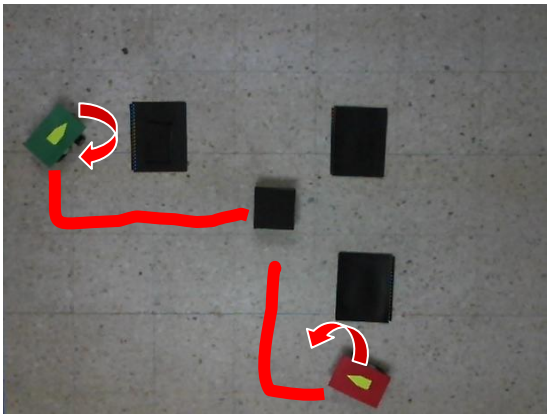
Resultados

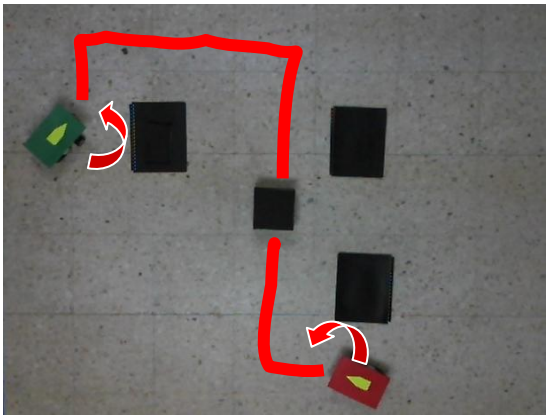
En la tabla 6.4 se puede ver los valores arrojados para esta prueba y en figura 6.5 se pueden observar las trayectorias recorridas por los robots.

Tabla 6.4
Resultados del Experimento 1.

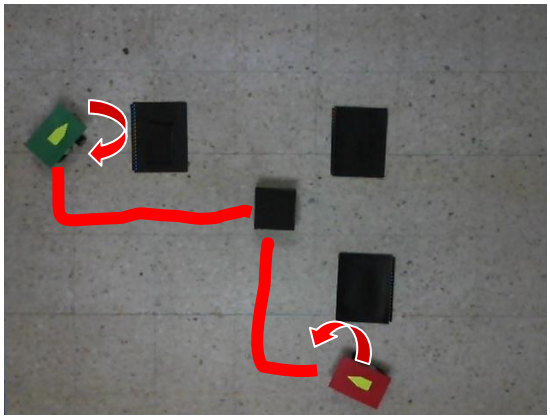
No. Trayectoria	Puntos Iniciales (x,y,Θ)	Punto Meta (x,y)	¿Meta Alcanzada?	Tiempo
1	R1 → 0,1,-44 R2 → 3,4,-149	2,2	SI	1:10
2	R1 → 0,1,-44 R2 → 3,4,-146	2,2	SI	1:12
3	R1 → 0,1,-44 R2 → 3,4,-149	2,2	SI	1:25
4	R1 → 0,1,-45 R2 → 3,4,-149	2,2	SI	1:13


Prueba 1


Prueba 2



Prueba 3



Prueba 4

Figura 6.5 Trayectorias para cada prueba del Experimento 1.

Análisis de resultados

La tabla 6.4 muestra un resultado exitoso en todas las pruebas del Experimento 1. En la Figura 6.5 podemos observar la trayectoria que cada uno de los robots sigue y podemos ver que en una de ellas el robot 1 no sigue la ruta que había estado realizando, lo cual hace que recorra otros puntos y se tarde un poco más. El tiempo promedio de ejecución que tarda el sistema para cumplir con el procedimiento es de 1:15 minutos.

6.3.3.2 Experimento 2

Objetivo

Dado dos robots Lego Mindstorms NXT en el ambiente mostrado en la figura 6.4, realizar una navegación de la posición donde se encuentran los robots a un objeto que representa la meta utilizando el sistema desarrollado en este trabajo.

Procedimiento

Utilizar el sistema desarrollado en este trabajo para obtener la posición y la orientación en el ambiente que tienen los robots al iniciar el proceso, localizar los obstáculos y localizar dentro del mapa la posición del objeto meta. Posteriormente el sistema trazará una trayectoria para cada robot con la cual debe de navegar y llegar al objeto meta. Se ejecutan

4 pruebas con este ambiente, se registra el tiempo de cada trayectoria y se indica si el robot logro su objetivo en cada prueba.

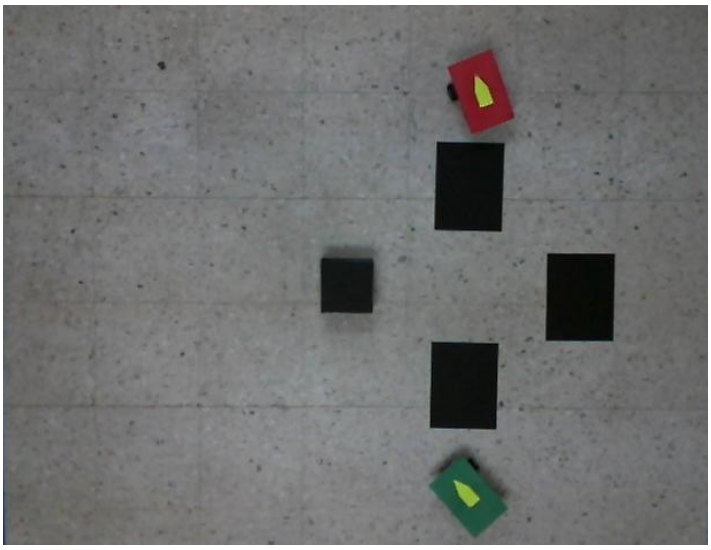


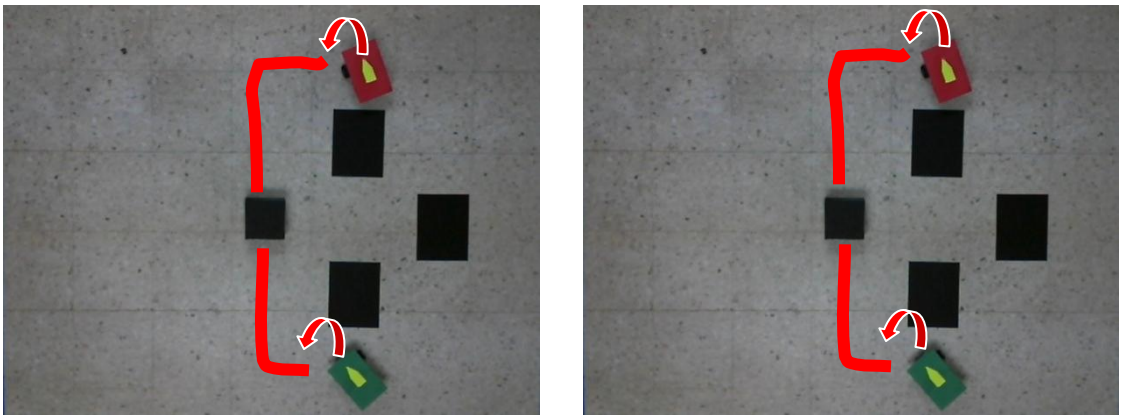
Figura 6.6 Mapa de ambiente para experimento 2.

Resultados

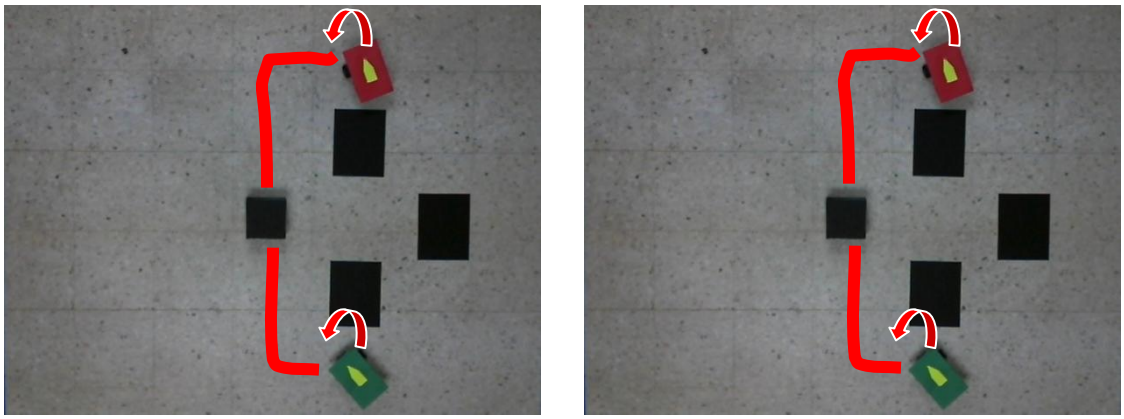
En la tabla 6.5 se puede ver los valores arrojados para esta prueba y en figura 6.7 se pueden observar las trayectorias recorridas por los robots.

Tabla 6.5
Resultados del Experimento 2.

No. Trayectoria	Puntos Iniciales (x,y,Θ)	Punto Meta (x,y)	¿Meta Alcanzada?	Tiempo
1	R1 → 3,4,-130 R2 → 3,0,-110	2,2	SI	1:17
2	R1 → 3,4,-127 R2 → 0,3,-110	2,2	SI	1:17
3	R1 → 3,4,-133 R2 → 3,0,-110	2,2	SI	1:17
4	R1 → 3,4,-130 R2 → 3,0,-110	2,2	SI	1:17



Prueba 1 Prueba 2



Prueba 3 Prueba 4

Figura 6.7 Trayectorias para cada prueba del Experimento 2.

Análisis de resultados

La tabla 6.5 muestra un resultado exitoso en todas las pruebas del Experimento 2. En la Figura 6.7 podemos observar la trayectoria que cada uno de los robots sigue, para este experimento observamos que siempre sigue la misma trayectoria. El tiempo promedio de ejecución que tarda el sistema para cumplir con el procedimiento es de 1:17 minutos.

6.3.3.3 Experimento 3

Objetivo

Dado dos robots Lego Mindstorms NXT en el ambiente mostrado en la figura 6.4, realizar una navegación de la posición donde se encuentran los robots a un objeto que representa la meta utilizando el sistema desarrollado en este trabajo.

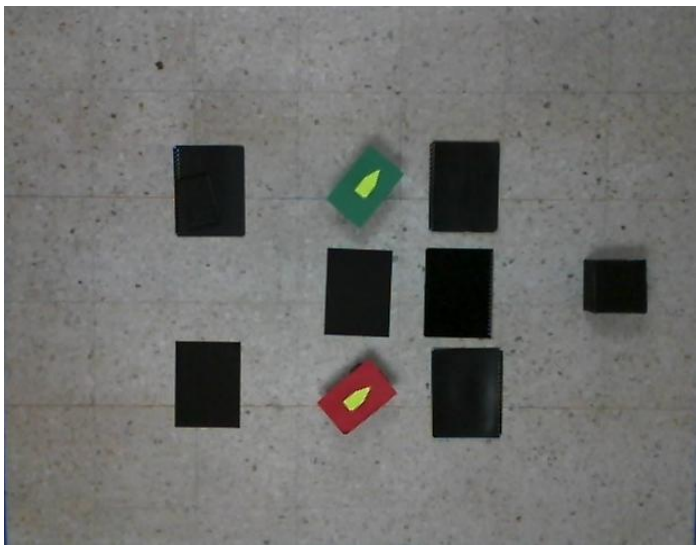


Figura 6.8 Mapa de ambiente para experimento 3.

Procedimiento

Utilizar el sistema desarrollado en este trabajo para obtener la posición y la orientación en el ambiente que tienen los robots al iniciar el proceso, localizar los obstáculos y localizar dentro del mapa la posición del objeto meta. Posteriormente el sistema trazará una trayectoria para cada robot con la cual debe de navegar y llegar al objeto meta. Se ejecutan 4 pruebas con este ambiente, se registra el tiempo de cada trayectoria y se indica si el robot logro su objetivo en cada prueba.

Resultados

En la tabla 6.6 se puede ver los valores arrojados para esta prueba y en figura 5.9 se pueden observar las trayectorias recorridas por los robots.

Tabla 6.6
Resultados del Experimento 3.

No. Trayectoria	Puntos Iniciales (x,y,Θ)	Punto Meta (x,y)	¿Meta Alcanzada?	Tiempo
1	R1 → 2,1,- 47	2,4	SI	2:07

	R2 → 3,2, - 43			
2	R1 → 2,1,- 45 R2 → 3,2, - 43	2,4	SI	2:10
3	R1 → 2,1,- 45 R2 → 3,2, - 43	2,4	SI	2:08
4	R1 → 2,1,- 47 R2 → 3,2, - 45	2,4	SI	2:03

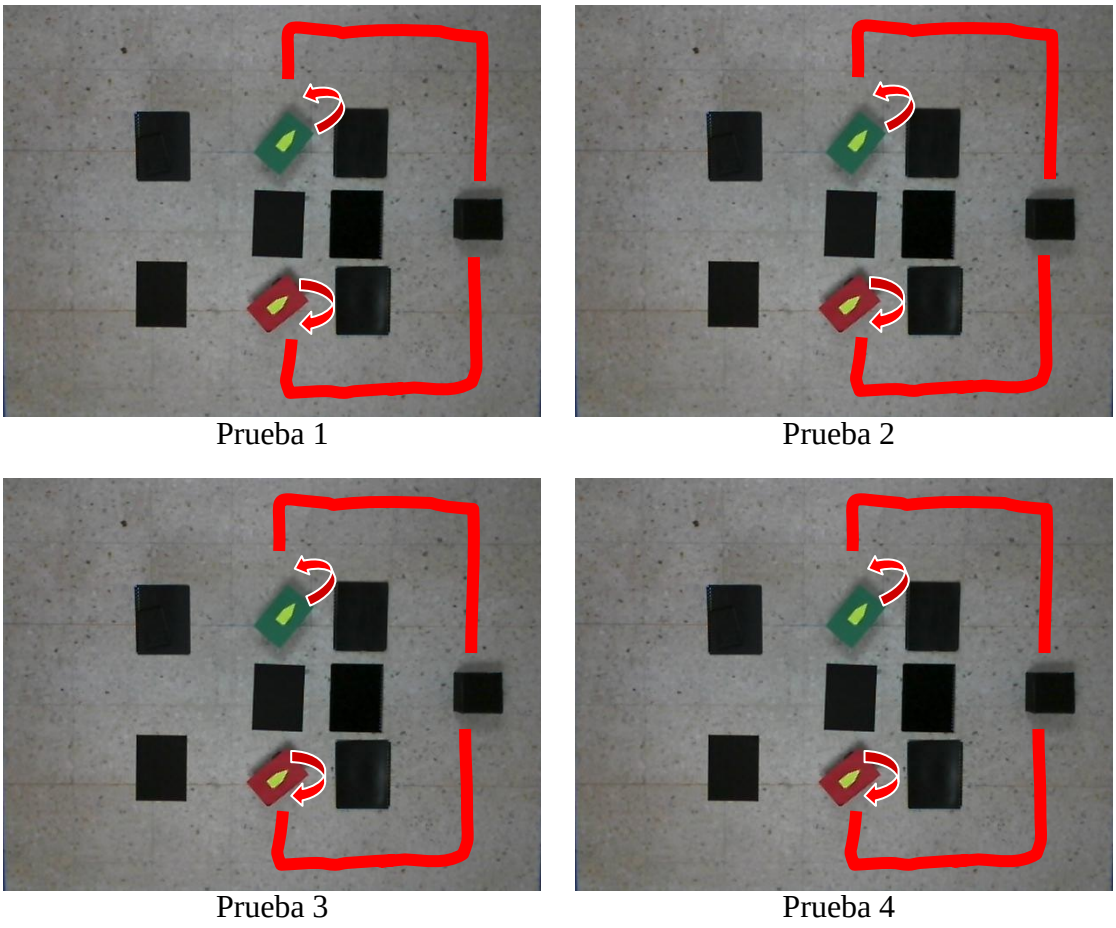


Figura 6.9 Trayectorias para cada prueba del Experimento 3.

Análisis de resultados

La tabla 6.6 muestra un resultado exitoso en todas las pruebas del Experimento 3. En la Figura 6.9 podemos observar la trayectoria que cada uno de los robots sigue, para este experimento observamos que siempre sigue la misma trayectoria. El tiempo promedio de ejecución que tarda el sistema para cumplir con el procedimiento es de 2:07 minutos.

6.3.3.4 Experimento 4

Objetivo

Dado dos robots Lego Mindstorms NXT en el ambiente mostrado en la figura 6.4, realizar una navegación de la posición donde se encuentran los robots a un objeto que representa la meta utilizando el sistema desarrollado en este trabajo.

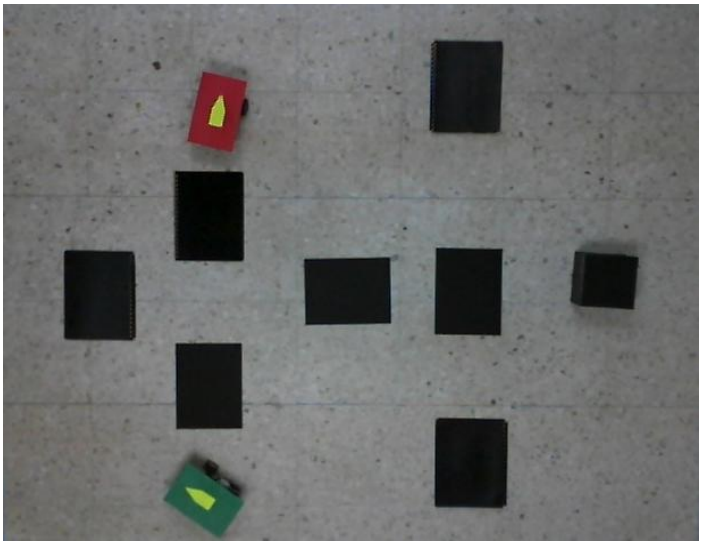


Figura 6.10 Mapa de ambiente para experimento 4.

Procedimiento

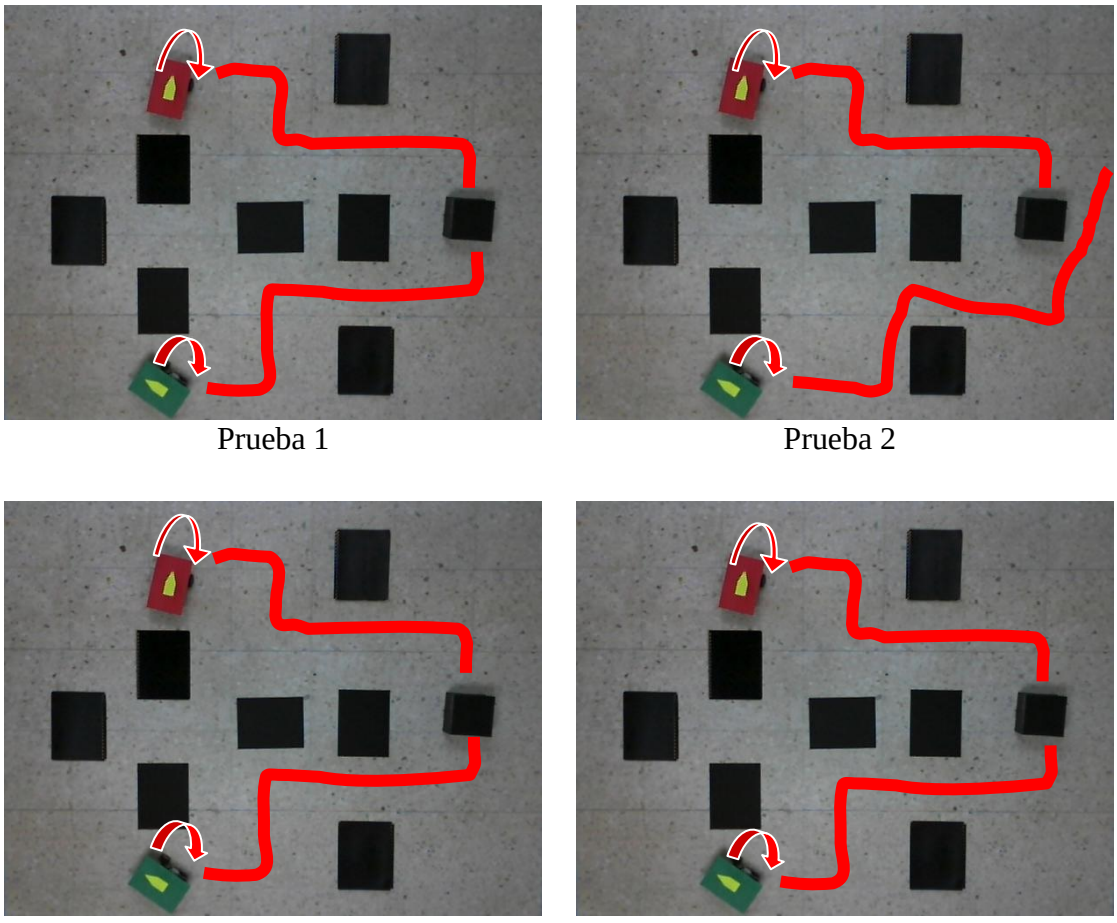
Utilizar el sistema desarrollado en este trabajo para obtener la posición y la orientación en el ambiente que tienen los robots al iniciar el proceso, localizar los obstáculos y localizar dentro del mapa la posición del objeto meta. Posteriormente el sistema trazará una trayectoria para cada robot con la cual debe de navegar y llegar al objeto meta. Se ejecutan 4 pruebas con este ambiente, se registra el tiempo de cada trayectoria y se indica si el robot logro su objetivo en cada prueba.

Resultados

En la tabla 6.7 se puede ver los valores arrojados para esta prueba y en figura 6.11 se pueden observar las trayectorias recorridas por los robots.

Tabla 6.7
Resultados del Experimento 4.

No. Trayectoria	Puntos Iniciales (x,y,Θ)	Punto Meta (x,y)	¿Meta Alcanzada?	Tiempo
1	R1 → 1,4,-142 R2 → 1,0,-67	2,4	SI	2:03
2	R1 → 1,4,-149 R2 → 1,0,-67	2,4	NO	
3	R1 → 1,4,-145 R2 → 1,0,-67	2,4	SI	2:03
4	R1 → 1,4,-142 R2 → 1,0,-70	2,4	SI	2:03



Prueba 1 Prueba 2

Prueba 3 Prueba 4

Figura 6.11 Trayectorias para cada prueba del Experimento 4.

Análisis de resultados

La tabla 6.7 muestra que no todas las pruebas fueron exitosas en Experimento 4. En la Figura 6.11 podemos observar la trayectoria que cada uno de los robots sigue. Para este experimento observamos que en la Prueba 2 el robot perdió la trayectoria, esto fue debido a que en el proceso de Localización la orientación proporcionada fue errónea. El tiempo promedio de ejecución en las pruebas exitosas fue de 2:03 minutos.

6.3.3.5 Experimento 5

Objetivo

Dado dos robots Lego Mindstorms NXT en el ambiente mostrado en la figura 6.4, realizar una navegación de la posición donde se encuentran los robots a un objeto que representa la meta utilizando el sistema desarrollado en este trabajo.

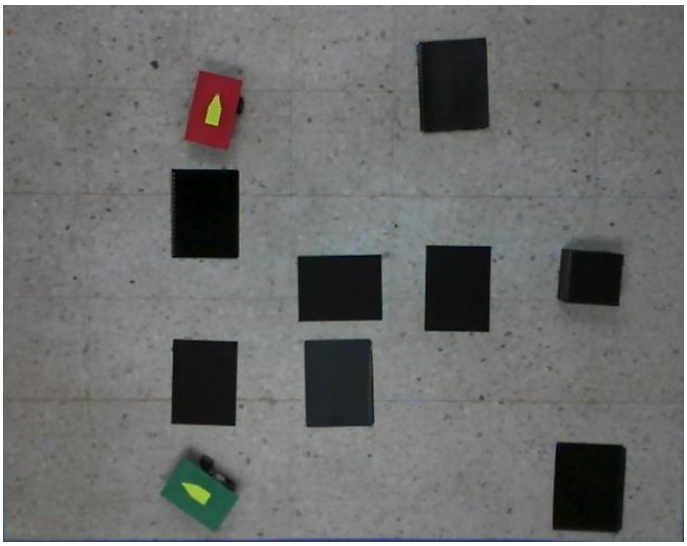


Figura 6.12 Mapa de ambiente para experimento 5.

Procedimiento

Utilizar el sistema desarrollado en este trabajo para obtener la posición y la orientación en el ambiente que tienen los robots al iniciar el proceso, localizar los obstáculos y localizar dentro del mapa la posición del objeto meta. Posteriormente el sistema trazará una

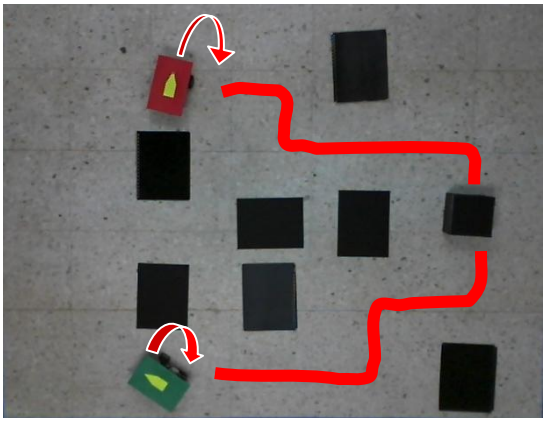
trayectoria para cada robot con la cual debe de navegar y llegar al objeto meta. Se ejecutan 4 pruebas con este ambiente, se registra el tiempo de cada trayectoria y se indica si el robot logro su objetivo en cada prueba.

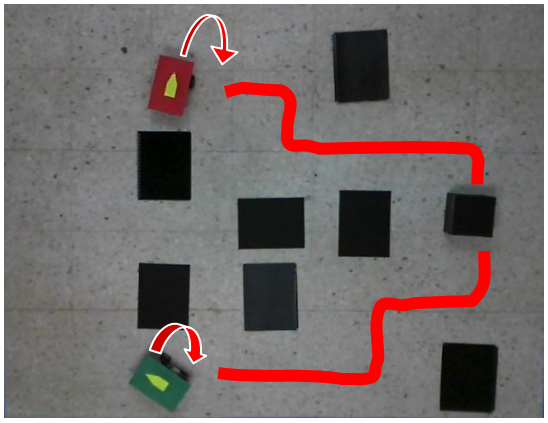
Resultados

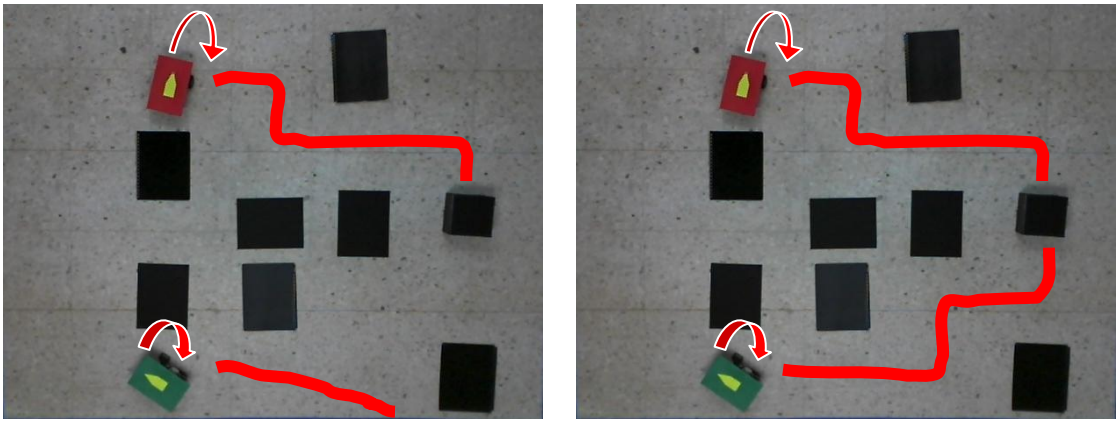
En la tabla 6.8 se puede ver los valores arrojados para esta prueba y en figura 6.13 se pueden observar las trayectorias recorridas por los robots.

Tabla 6.8
Resultados del Experimento 5.

No. Trayectoria	Puntos Iniciales (x,y,Θ)	Punto Meta (x,y)	¿Meta Alcanzada?	Tiempo
1	R1 → 1,4,-142 R2 → 1,0,-67	2,4	SI	1:58
2	R1 → 1,4,-144 R2 → 1,0,-67	2,4	SI	1:58
3	R1 → 1,4,-149 R2 → 1,0,-67	2,4	NO	
4	R1 → 1,4,-142 R2 → 1,0,-70	2,4	SI	2:00


Prueba 1


Prueba 2



Prueba 3 Prueba 4

Figura 6.13 Trayectorias para cada prueba del Experimento 4.

Análisis de resultados

La tabla 6.8 muestra que no todas las pruebas fueron exitosas en el Experimento 5. En la Figura 6.13 podemos observar la trayectoria que cada uno de los robots sigue. Para este experimento observamos que en la Prueba 3 el robot perdió la trayectoria, esto fue debido a que en el proceso de Localización la orientación proporcionada fue errónea. El tiempo promedio de ejecución en las pruebas exitosas fue de 1:58 minutos

7

Conclusiones y Trabajos Futuros

7.1 Conclusiones

Al terminar este trabajo se concluye en lo siguiente:

- De acuerdo al objetivo planteado al inicio de este trabajo, se pudo demostrar experimentalmente que es posible desarrollar un sistema localización y navegación con robots Lego Mindstorms NXT (Sección 6.3).
- El proceso de localización, ha resultado, desde cierto punto, satisfactorio al utilizar las librerías de OpenCv con sus algoritmos Camshift y Contornos (Sección 5.3). Estas técnicas de visión permitieron generar un mapa del entorno (Sección 5.4) y localizar en este, los obstáculos, objeto meta y los robots. Para los robots además de su localización, también proporcionaron la orientación.
- Para el proceso de navegación, el algoritmo de Dijkstra cumple una función importante dentro de la generación de las trayectorias. Este es el encargado de

encontrar el camino más corto entre el punto inicio (robot) y el punto meta (objeto). Obteniendo de esta manera la trayectoria más óptima (Sección 5.5).

- Por otro lado la utilización de la teoría de agentes ha servido como base para que este trabajo se desarrolle bajo una arquitectura de control hibrida basada en agentes (Seción 3.1). En esta arquitectura los agentes comparten información sobre la base de dos mecanismos, paso de mensajes (Sección 5.2) y memoria compartida. Para este caso se utiliza como memoria la estructura de un archivo XML (Sección 5.5).

7.2 Trabajos Futuros

A continuación se enlistan una serie de posibles mejoras:

- Se propone enriquecer los algoritmos de tratamiento de imágenes propuestos en este trabajo para disminuir el error con los cambios de iluminación.
- El error en la localización es acumulativo, por lo tanto, se propone extender el algoritmo para solucionar este problema. Un primer acercamiento es realizar una localización global periódicamente para corregir la orientación del robot en cada punto del recorrido.
- Desarrollar una rutina para verificar el estado de los sensores y tomar decisiones.
- Llevar a la experimentación este trabajo en un ambiente más grande utilizando una cámara de mayor alcance y otro tipo de comunicación.

Anexos

A. Pseudocódigo de los Agente de la Arquitectura Propuesta

- **Agente Actualizar Posición**

```
retorno =  recoverHisto("histograma.xml"));
if (retorno==0) {
        printf("No carga histograma \n ");
        return 1; }
if ((capture = cvCaptureFromCAM(1)) == NULL)    {
        fprintf(stderr, "No se pudo inicializar la captura.\n");
        return -1; }
cvNamedWindow("CamShiftDemo", 1);
for (;;) {
        IplImage* frame = 0;
        frame = cvQueryFrame(capture);
        track_window = cvRect(0, 0, frame->width, frame->height); }
 if (!image) {
        image = cvCreateImage(cvGetSize(frame), IPL_DEPTH_8U, 3);
       image->origin = frame->origin;
       hsv = cvCreateImage(cvGetSize(frame), IPL_DEPTH_8U, 3); // imagen en HSV
        hue = cvCreateImage(cvGetSize(frame), IPL_DEPTH_8U, 1);  // imagen de un canal (hue)
        mask = cvCreateImage(cvGetSize(frame), IPL_DEPTH_8U, 1);
        backproject = cvCreateImage(cvGetSize(frame), IPL_DEPTH_8U, 1);    }
cvCalcBackProject(&hue, backproject, hist);
cvAnd(backproject, mask, backproject, 0);
cvCamShift(backproject, track_window, cvTermCriteria(CV_TERMCRIT_EPS |
                        CV_TERMCRIT_ITER, 10, 1),  &track_comp, &track_box);
track_window = track_comp.rect;
cvCvtColor(backproject, image, CV_GRAY2BGR);
cvEllipseBox(frame, track_box, CV_RGB(255, 0, 0), 3, CV_AA, 0);
robot = cvCreateImage(cvSize( track_window.width,track_window.height), 8, 3);
```

```
if( g_storage==NULL ) {
      g_gray = cvCreateImage( cvGetSize(robot), 8, 1 );
      g_storage = cvCreateMemStorage(0);    }
else {  cvClearMemStorage( g_storage ); }
nc=cvFindContours( g_gray, g_storage, &contours, sizeof(CvContour), CV_RETR_EXTERNAL,
                        CV_CHAIN_APPROX_SIMPLE );
while (contours){
          if(contours->total>5) {
                    poligono = cvApproxPoly(contours, sizeof(CvContour),  g_poligono,
                                                CV_POLY_APPROX_DP,cuenta, 0);
                    if(poligono->total==5){
                              i_punta =indice_menor(m);
                              p_extr.x = (Punto[i1].x+ Punto[i2].x)/2;
                              p_extr.y = (Punto[i1].y+ Punto[i2].y)/2;
                              ang_inc = atan2(Punto[i_punta].y-p_extr.y,Punto[i_punta].x –
                                                  p_extr.x)*180/3.1416;
                              posx = (Punto[i_punta].x+p_extr.x)/2+ekis;
                              posy = (Punto[i_punta].y+p_extr.y)/2+ye; }
          }
        contours = contours->h_next;
}
```

- **Agente Obtener Mapa Global.**

```
IplImage *img = NULL;
CvCapture* capture = cvCaptureFromCAM(1);
if(capture ) {
     cvNamedWindow( "Original", CV_WINDOW_AUTOSIZE );  }
img= cvQueryFrame( capture );
CvSize imgSize;
 imgSize.width = img->width;
imgSize.height = img->height;
y = img->height / m;
x = img->width / n;
for (int h = 1; h <= n; h++)
```

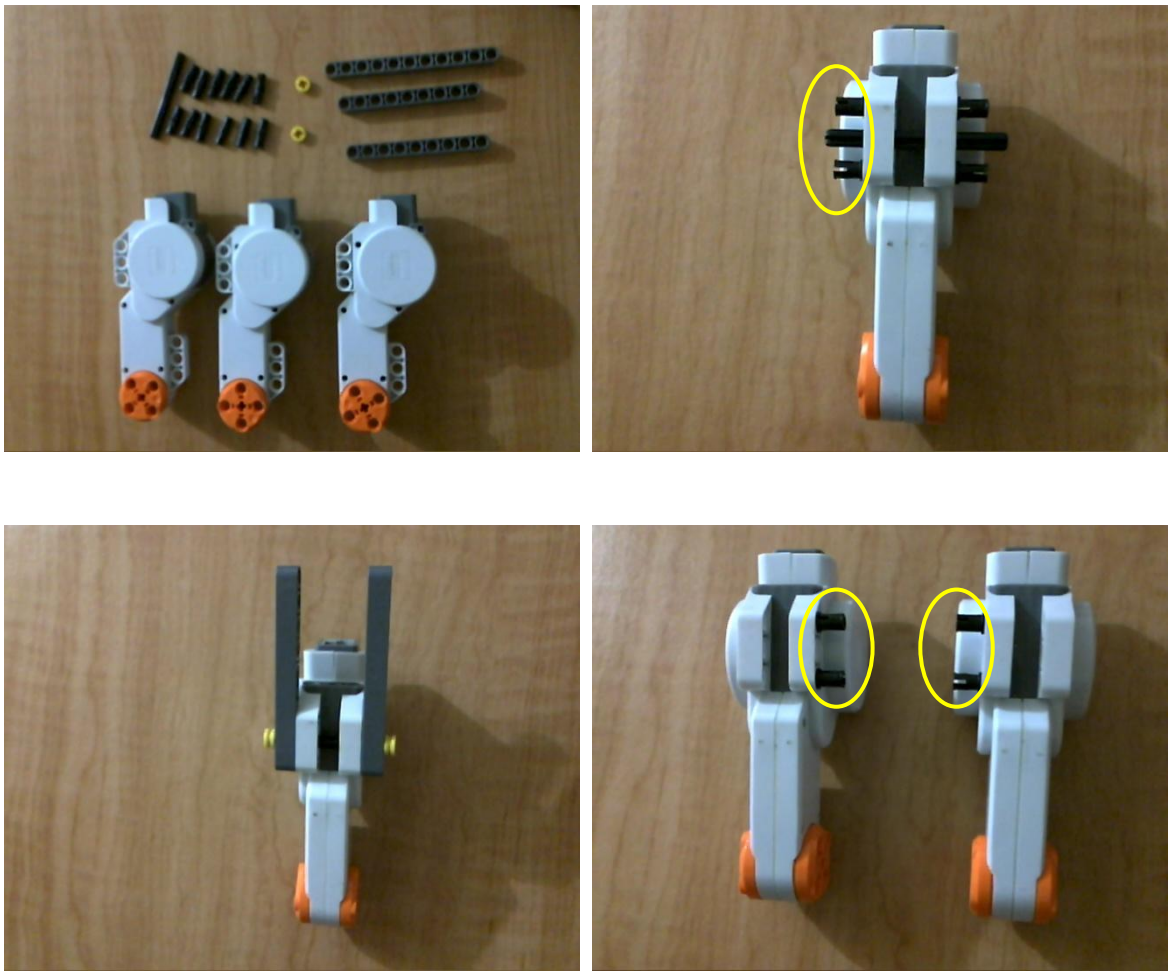
```
        cvLine(img, cvPoint(x*h, 0), cvPoint(x*h, img->height),cvScalar(0,240, 0, 0), 1,8,0);
for (int k = 1; k <= m; k++)
         cvLine(img, cvPoint(0, y * k), cvPoint(img->width, y * k),cvScalar(0,240, 0, 0 ), 1,8,0);
 IplImage* gris = cvCreateImage(cvSize(img->width, img->height),IPL_DEPTH_8U, 1);
 cvCvtColor(img, gris,CV_BGR2GRAY);
binarizada = cvCreateImage(cvSize(img->width, img->height), IPL_DEPTH_8U, 1);
 CvScalar s;
for (int a = 0; a < img->height; a++) {
       for (int b = 0; b < img->width; b++) {
          s = cvGet2D(img, a, b);
          if (s.val[0] >= 30){
             s.val[0] = 250;
             cvSet2D(binarizada, a, b, s);
           }
           else {
             s.val[0] = 0;
             cvSet2D(binarizada, a, b, s);
          }
 }
// Generar Matriz de Ocupación
for (x = 0; x < img->width; x += escalax){
    for (y = 0; y < img->height; y += escalay){
          for (xx = 0; xx < escalax; xx++){
             for (yy = 0; yy < escalay; yy++){
                pixl = cvGet2D(img, y+yy,x+xx);
                if (pixl.val[0] == 250){
                   desocupados++; }
             }
          }
          if (desocupados >= ((escalay * escalax) * 3 )/ 4 {
            if(mapa[y / escalay][x / escalax] == 0){
               mapa[y / escalay][x / escalax] = 0;
            }
          }
          else{
             if(mapa[y / escalay][x / escalax] == 0){
               mapa[y / escalay][x / escalax] = -1;
             }
```

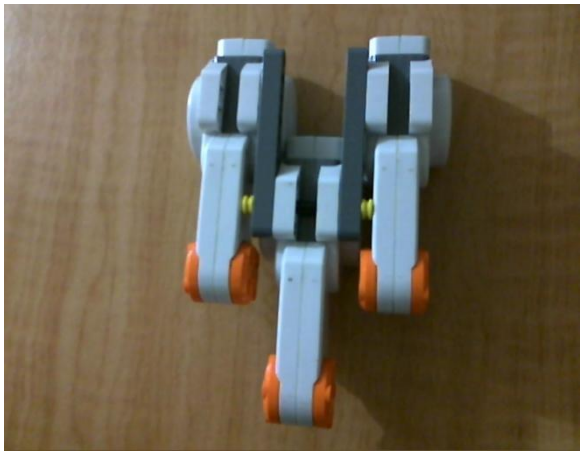
```
          }
            desocupados = 0;
         }
    }

for (int a = 0; a < TOTPUNTOS; a++) {
        randomX = (rand() % m) ;
        randomY = (rand() % n);
        if (randomX < m && randomY < n) {
            if (mapa[randomX][randomY] == 0) {
                mapa[randomX][randomY] = puntos;
                puntos++;
                if(puntos>TOTPUNTOS)
                    a=puntos;
            } else
               a--;  }
        else {
            a--;  }
    }
conectarPuntos();
Dikjstra();
```

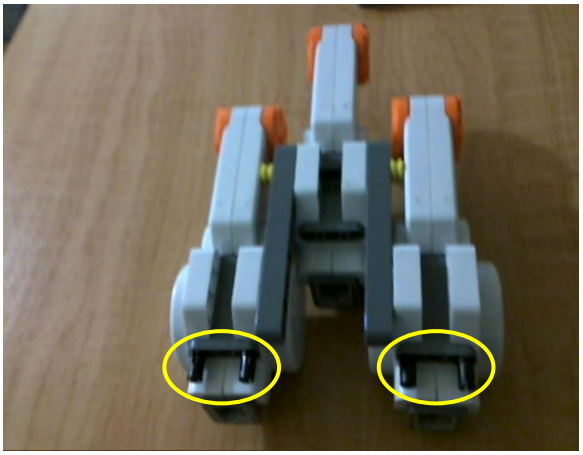
B. Construcción del Modelo Tipo Carro

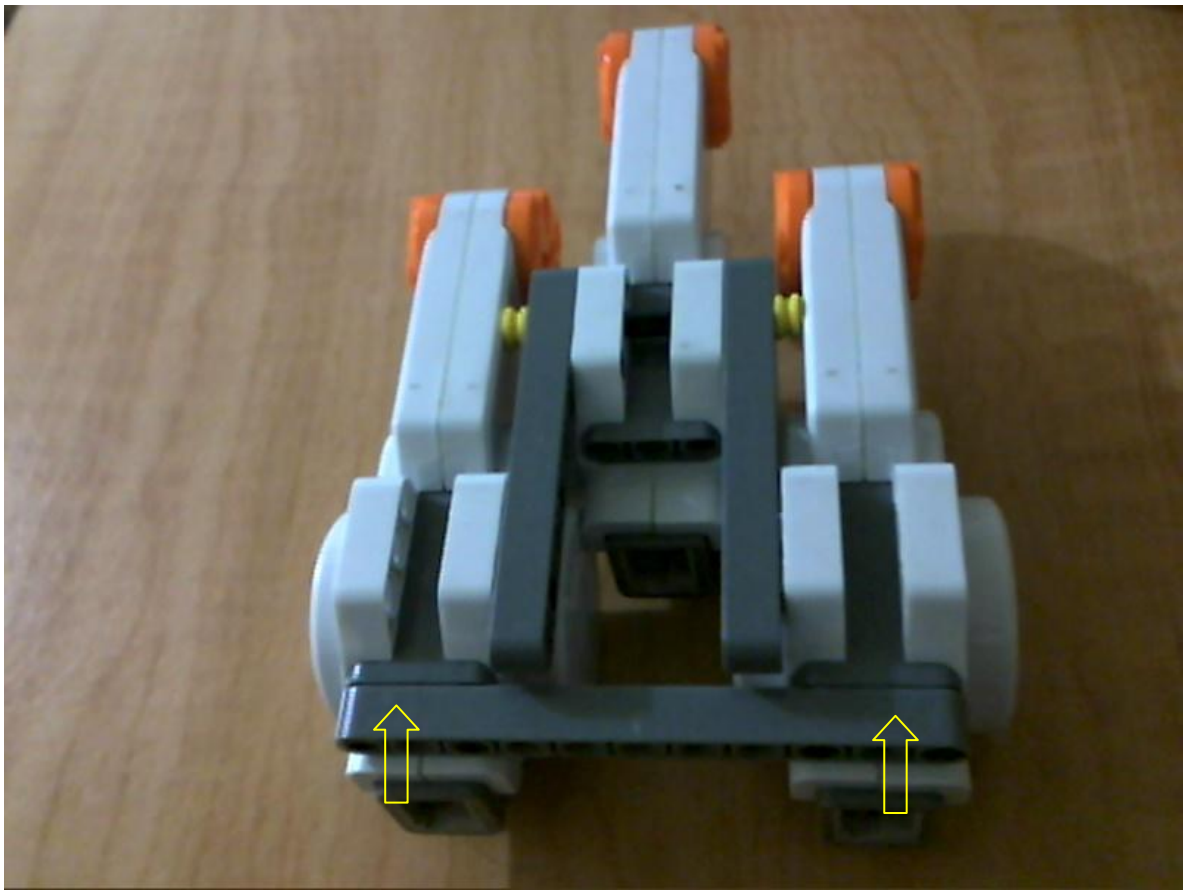
A continuación se ilustra cada uno de los pasos para construir el Robot Lego Mindstorms NXT tipo carro, que cubrió las necesidades del proyecto. Para ello mostraremos por etapas el armado de cada uno de los módulos que lo conforman.

Módulo 1





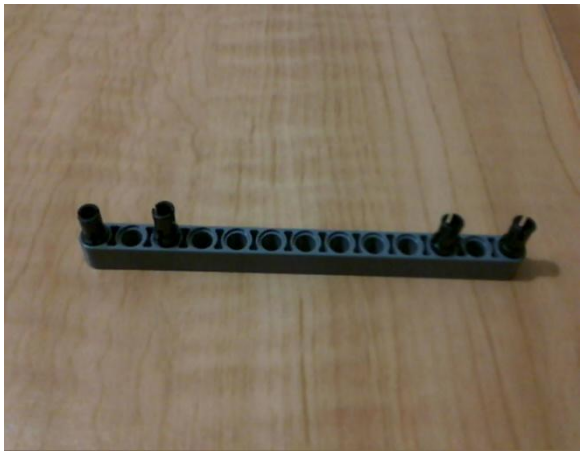


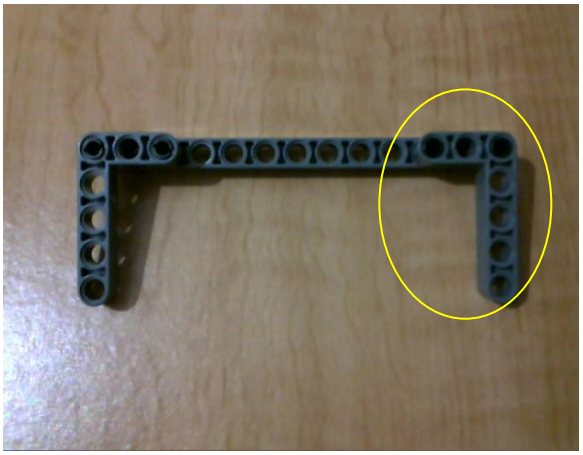
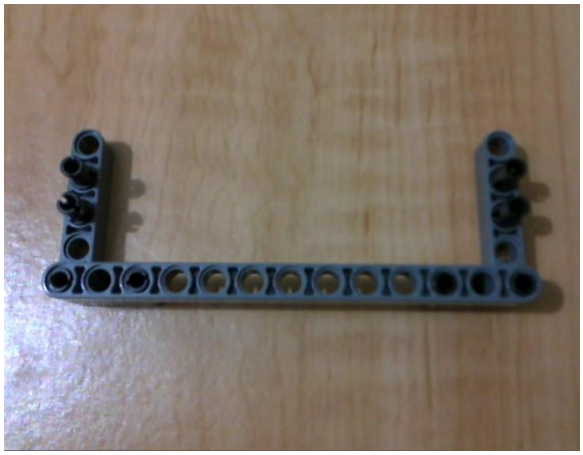
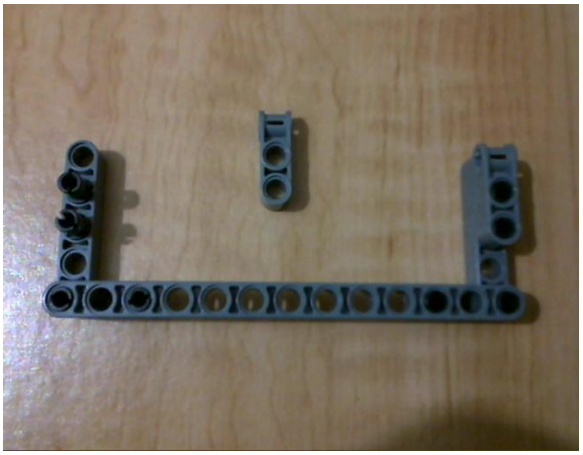
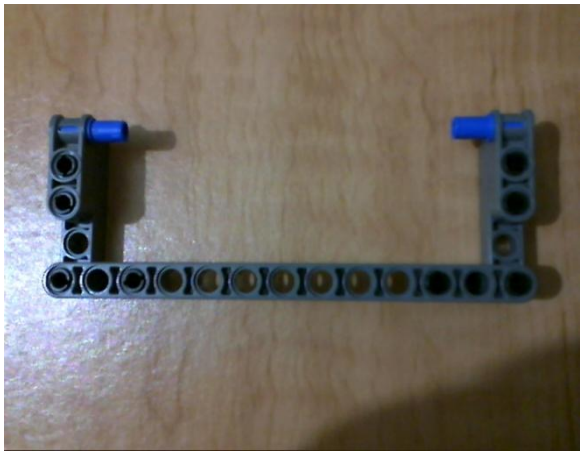
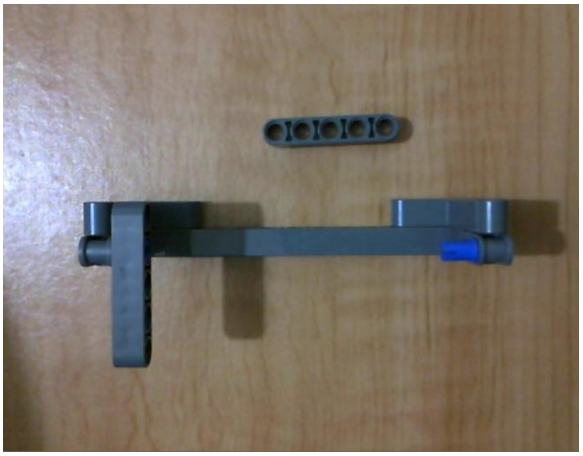


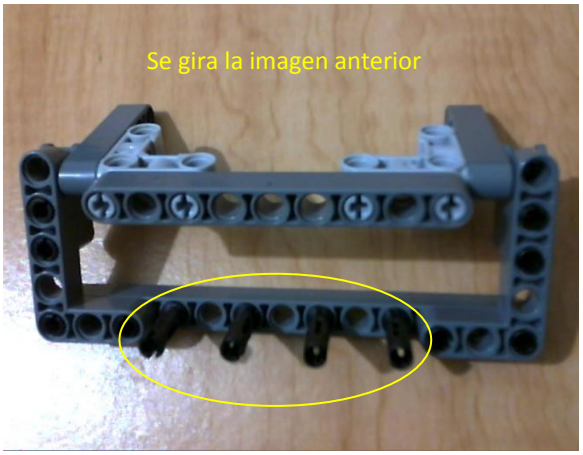
Módulo 2

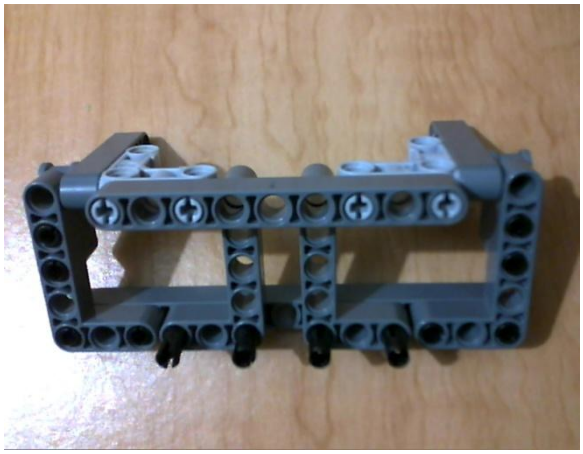
Módulo 3. Este módulo es la unión del módulo 1 con módulo 2

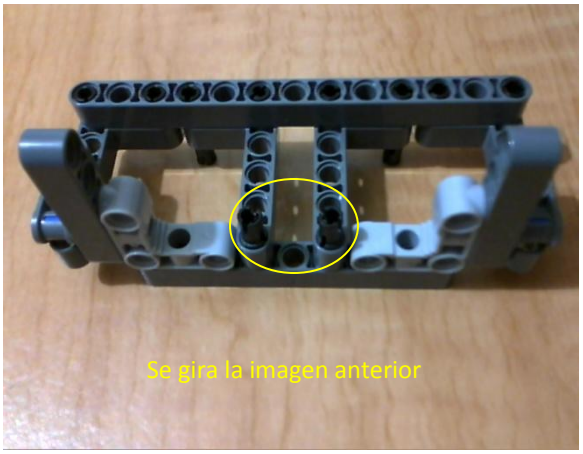
Módulo 4

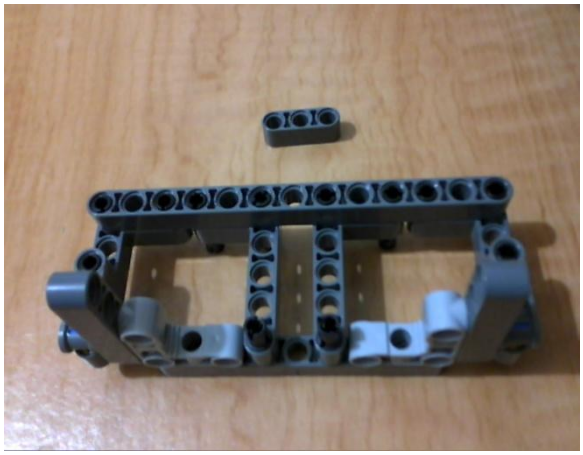


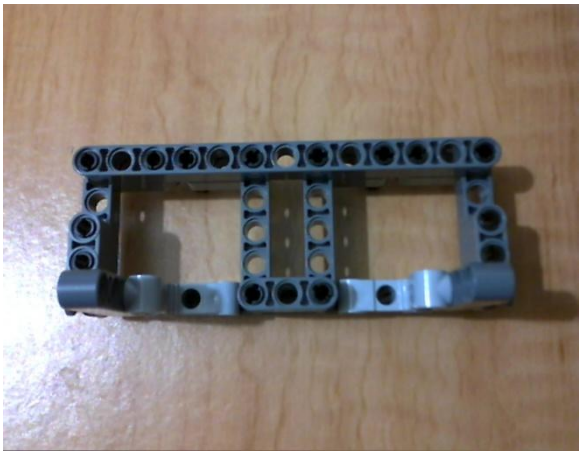






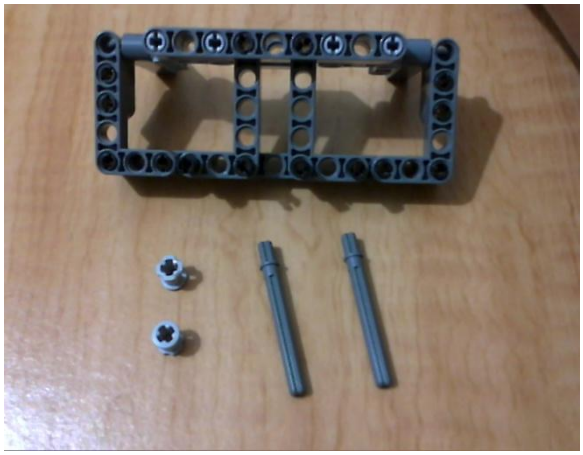


Se gira la imagen anterior

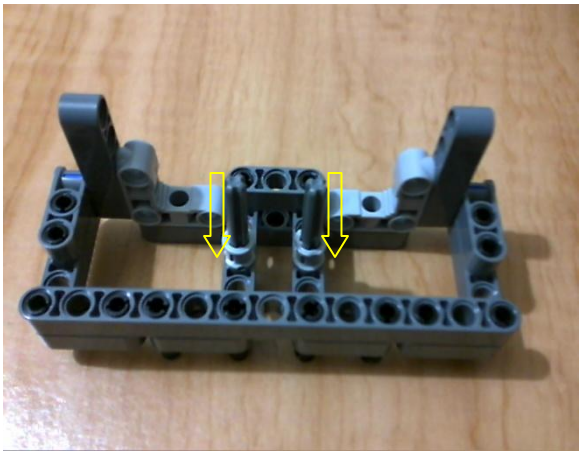


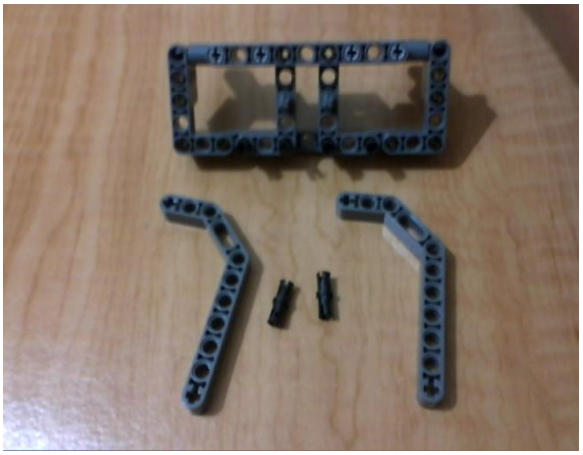

Se gira la imagen anterior

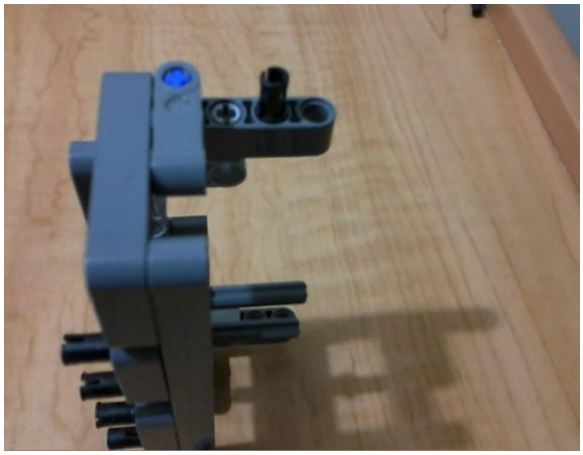


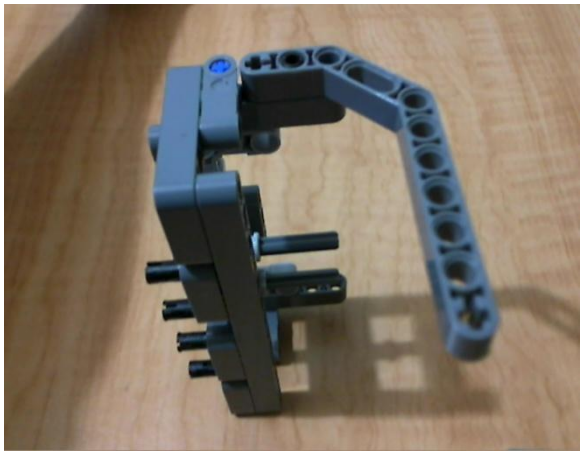


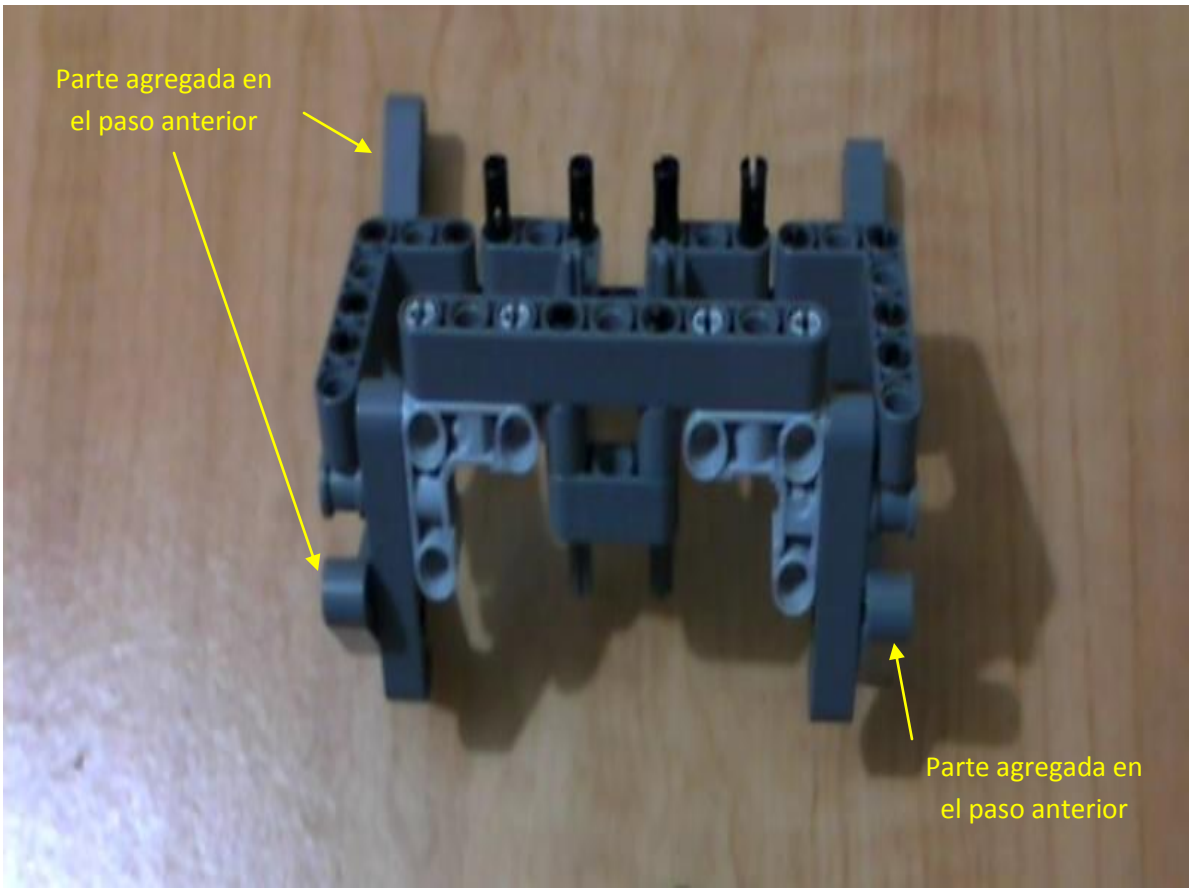






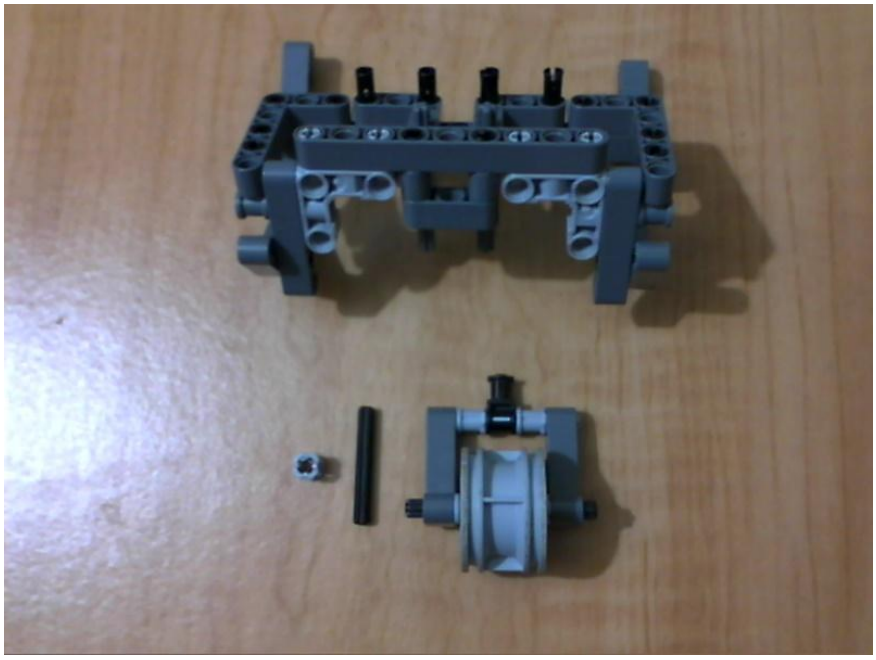


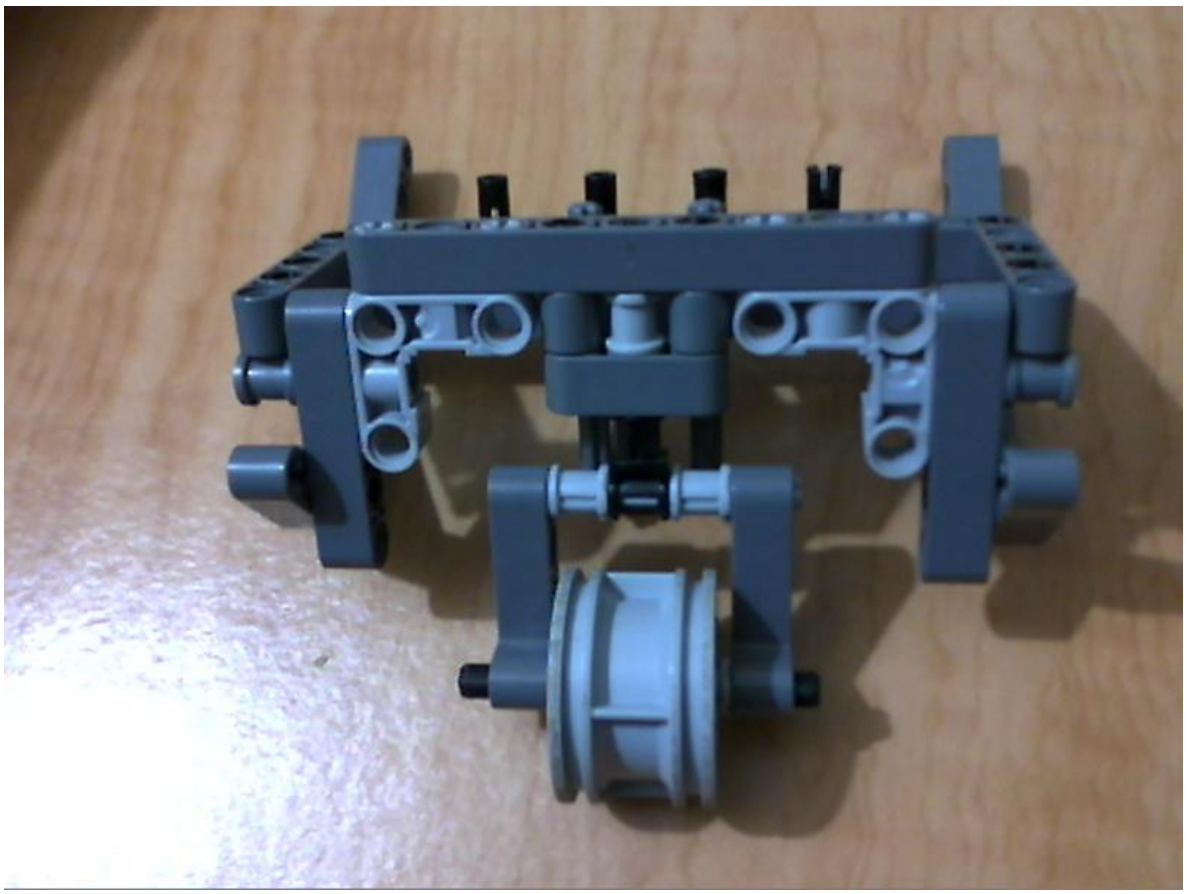



Parte agregada en
el paso anterior
Parte agregada en
el paso anterior

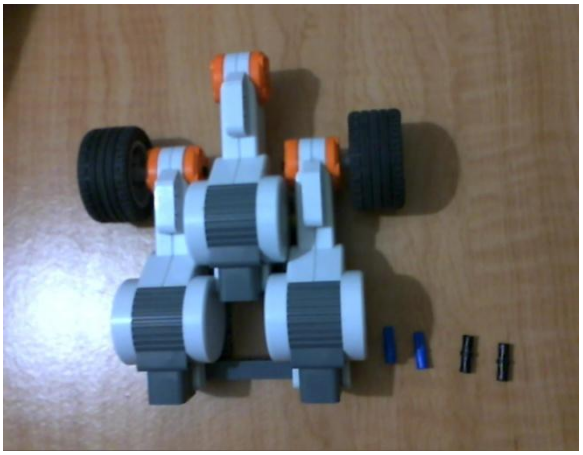
Módulo 5

Módulo 6. Este módulo es la unión del módulo 4 con módulo 5

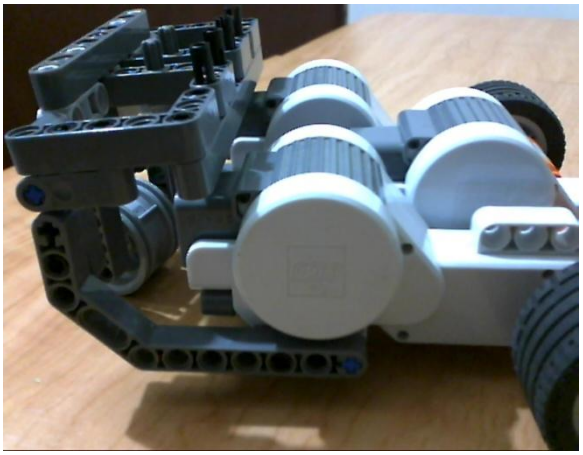
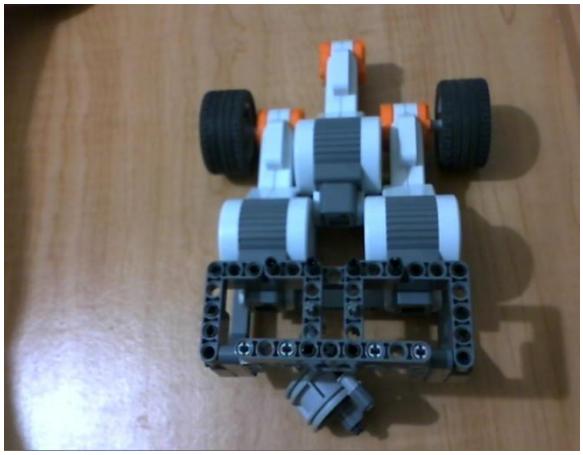




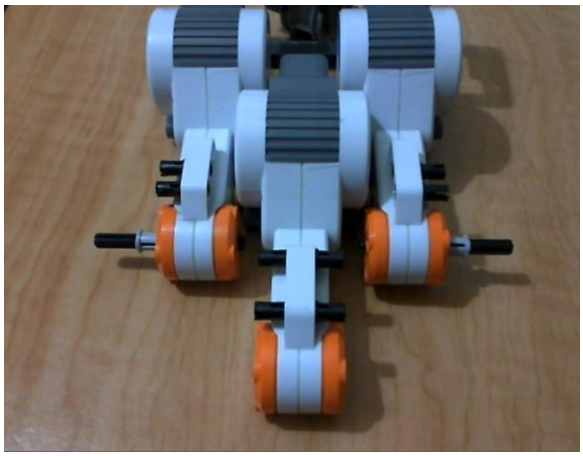
Módulo 7. Complementando el módulo 3.

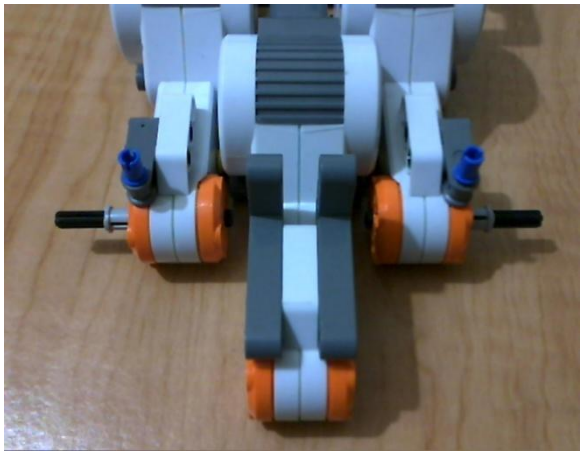



Módulo 8. Este módulo es la unión del módulo 6 con módulo 7.

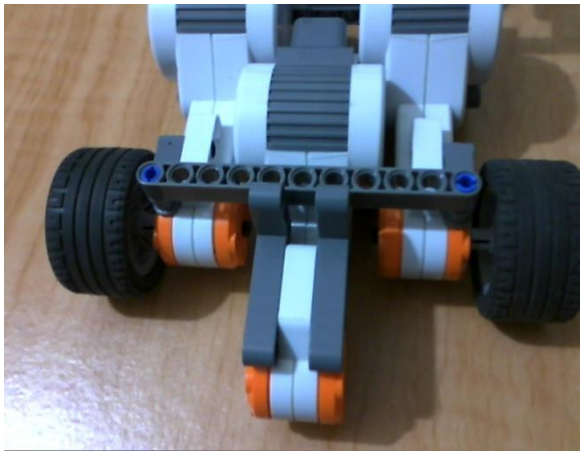



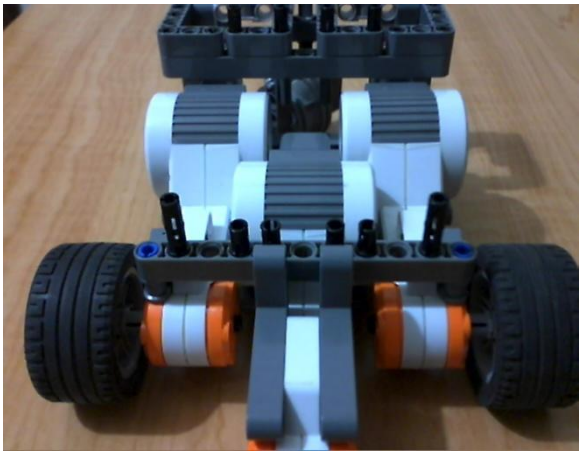
Módulo 9. Complementando el módulo 8. Se le quitan las llantas.

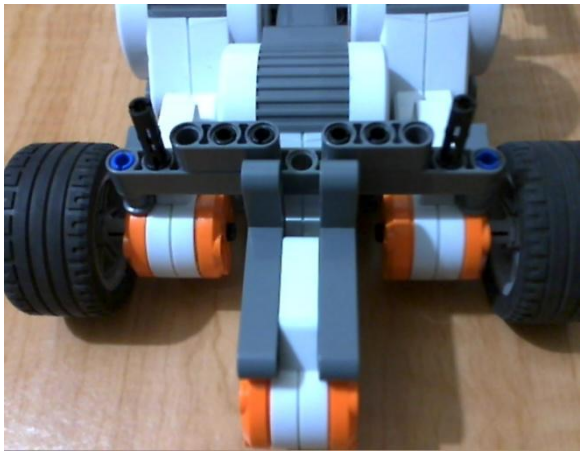





Módulo 10. Complementando el módulo 9.

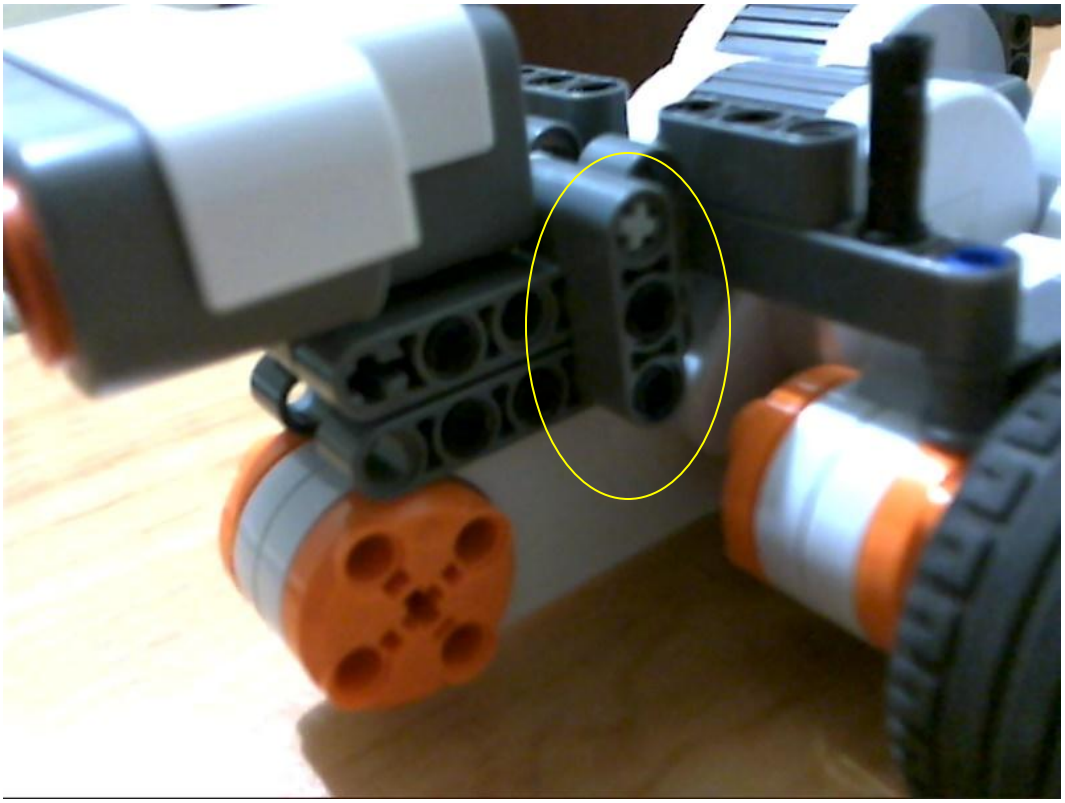


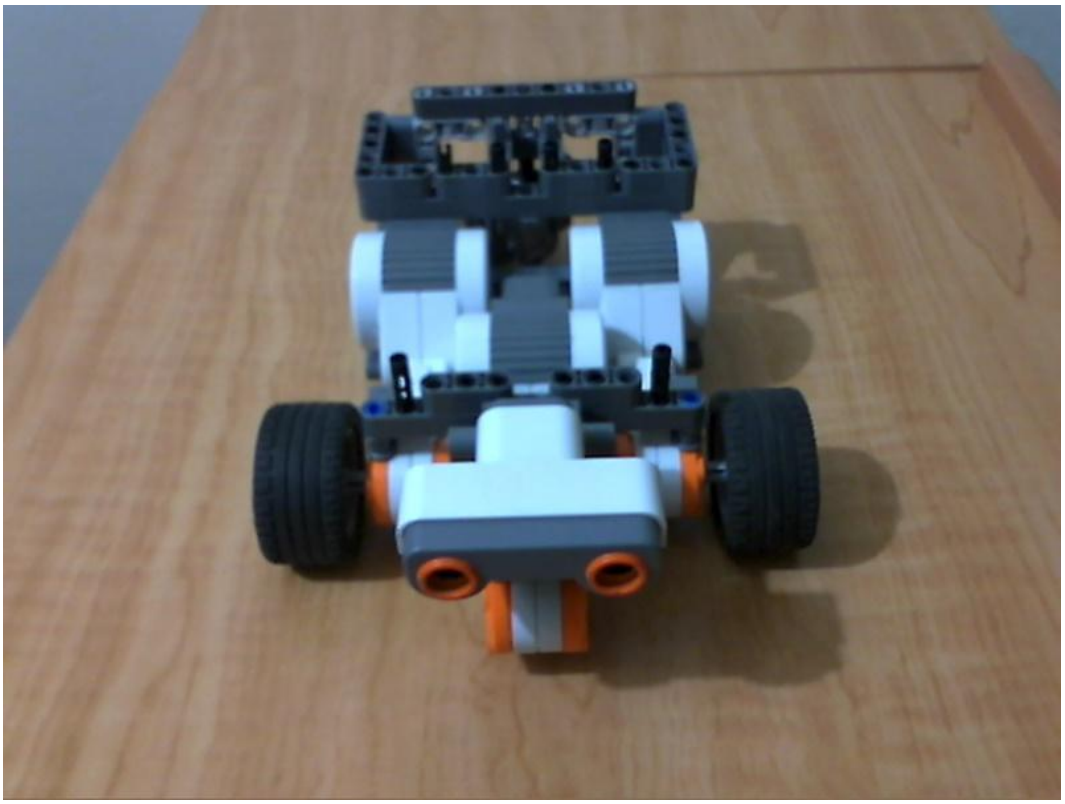




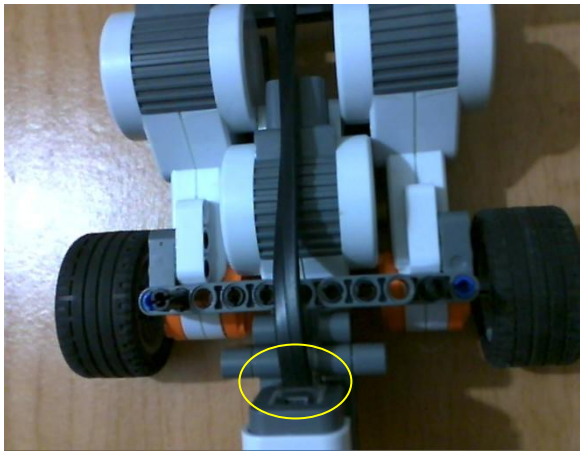
Módulo 11.

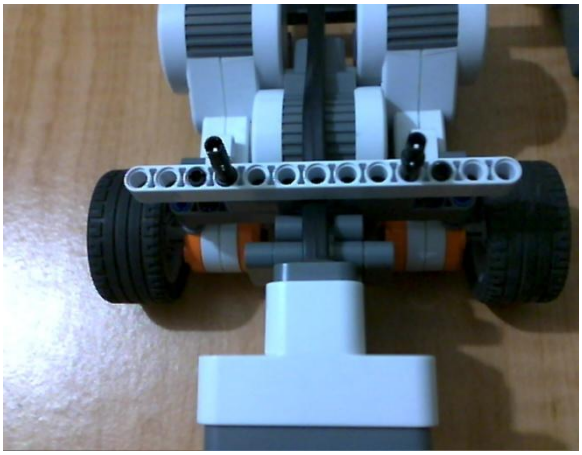
Módulo 12. Complementando el módulo 10 con el módulo 11.



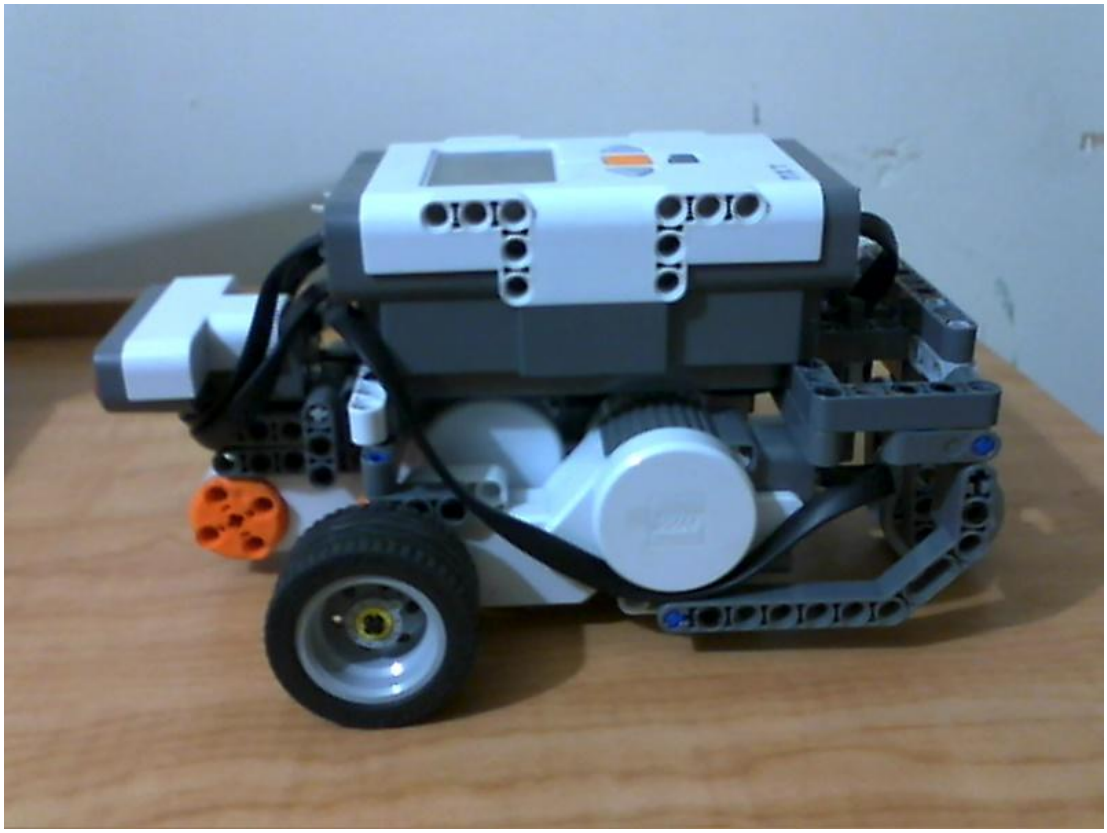


Módulo 13.





Resultado.



Bibliografía

[Barrientos, 2007]. Barrientos A.; Peñil L., "Fundamentos de Robótica", Segunda Edición, Editorial McGraw-Hill. España, 2007.

[Benedettelli 2006]. Daniele Benedettelli "Squadre di robot mobili basati su tecnologia Lego Mindstorms", Universit`a Degli Studi di Siena, Facult`a di Ingegneria, Siena, Italia, 2006.

[Bonasso et al., 1997]. R. P. Bonasso, Firby, R. J., Gat, E., Kortenkamp, D., Miller, D. P., y Slack, M. G. Experiences with an Architecture for Intelligent, Reactive Agents. Journal of Experimental and Theoretical Artificial Intelligence, 9(2/3):237–256. 1997.

[Brooks, 1991]. R. A. Brooks. "New Approaches to Robotics". Science, 235:1227–1232.

[Calife et al., 2007]. D. Calife, A. Tomoyose, D. Spinola, J. Bernardes, R. Tori; "Robot Arena: Infrastructure for Applications Involving Spatial Augmented Reality and Robots", Escola Politécnica da Universidade de São Paulo; Brasil 2007.

[Decker et al., 1996]. K. Decker., A. Pannu, K. Sycara, and M. Williamson; "Designing Behaviors for Information Agents" The Robotics Institute, Carnegie-Mellon University; Pittsburgh, USA, 1996.

[Gallardo 1999]. Domingo Gallardo López "Aplicación del muestreo Bayesiano en robots móviles: estrategias para localización y estimación de mapas del entorno", Departamento de Tecnología Informática y Computación, Universidad de Alicante., Madrid, España 1999.

[García et al., 2008]. I. J. García [(1)], J. M. Ramírez[(1)], M. N. Ibarra [(1)],M. Gómez[(2)] , "Seguimiento Autónomo de la Posición de un Objeto por Visión y Control Neurodifuso en MATLAB", [(1)] Coordinación de Electrónica [(2)]Coordinación de Ciencias Computacionales Instituto Nacional de Astrofísica, Óptica y Electrónica; Tonantzintla, Puebla, México 2008.

[García, 2004]. L. García; "Navegación Autónoma de Robots en Agricultura: Un Modelo de Agentes"; Universidad Complutense de Madrid, España. 2004.

[García-Alegre et al., 1992]. García-Alegre, M. C. y Guinea, D. "Building and arquitecture for a farming robot". En Bio-Robotics 97. International Workshop on Robotics and Automated Machinery for bio-Productions, págs. 255–260, Gandia España. 1992.

[García-Alegre et al., 1995]. García-Alegre, M., P.Bustos, y D.Guinea. "Complex behaviour generation on autonomous robots: A case study". En Proceedings of the IEEE International Conference on Systems, Man and Cybernetics, p´ags. 1729–1734, Vancouver Canadá. 1995.

[García-Alegre et al., 1998]. García-Alegre, M. C. y Recio, F. "Basic Visual and Motor Agents for Increasingly Complex Behavior Generation on a Mobile Robot. Autonomous Robots", 5:1–10. 1998.

[Hayes-Roth, 1985].Hayes-Roth, B.; "A Blackboard Architecture for Control. Artificial Intelligence", 26:251–321. 1985.

[Ibarra et al., 2009]. M. N. Ibarra, J. M. Ramírez, A. Díaz, J. Martínez, R. Enríquez, I. J. García; "Navegación autónoma de un robot guiado por visión con operaciones básicas de localización y mapeo en un ambiente controlado"; Coordinacion de Electronica; Instituto Nacional de Astrofisica, Optica y Electronica; Tonantzintla, Puebla, Mexico. 2009.

[Jennings y Wooldridge 1998]. Jennings, N., Wooldridge, M. (eds.), "Agent Technology:Foundations, Applications and Markets", Springer, 1998.

[Leal 2009]. David Leal Martinez, "Reconfigurable Multi Robot Society based on LEGO Mindstorms", Helsinki University of Technology, Faculty of Electronics, Communications and Automation, Helsinki, Finlandia, 2009.

[Lowe 2004]. D. G. Lowe, "Distinctive Image Features from Scale-Invariant Keypoints"; Computer Science Department University of British Columbia Vancouver, B.C., Canada, 2004.

[Maes, 1989]. P. Maes; "The dynamics of action selection"; In Proceedings of the International Joint Conference on Artificial Intelligence, Detroit. Morgan Kaufmann. 1989.

[Makarenko et al., 2002]. Alexei A. Makarenko, Stefan B. Williams, Frederic Bourgault, Hugh F. Durrant-Whyte, "An Experiment in Integrated Exploration";Universidad de Sydney, Australia, 2002

[Matellán 2002]. Vicente Matellán Olivera, Jesús M. González Barahona, Pedro de las Heras Quiroz, José Centeno González, "Programación de LEGO MindStorms bajo GNU/Linux", Universidad Rey Juan Carlos, Departamento de Ciencias Experimentales e Ingeniería, Madrid, España, 2002.

[Matellán y Borrajo, 2001]. Matellán, V. y Borrajo, D. "ABC^2: an architecture for intelligent autonomous systems". En Journal of Intelligent and Robotic Systems 32: 93–114; Netherlands. 2001.

[Muñoz 2006]. R. Muñoz, "Soft-Computing and Computer Vision Techniques Applied to Autonomous Robot Navigation and Human-Robot Interaction"; Departamento de Ciencias de la Computación e Inteligencia Artificial, Universidad de Granada, España, 2006.

[Murphy, 2000]. R.R. Murphy; Introduction to AI Robotics. Intelligent Robots and Autonomous Agents. MIT Press, 1ª edición.

[Nwana, 1996]. Nwana H. S., "Software Agents: An Overview"; Knowledge Engineering Review, vol. 11 (3), pág 205-244. 1996.

[Ojha 2009]. Ojha U., "Delay Tolerant Behavior Control based Adaptive Bandwidth Allocation for Unmanned Ground Vehicles in a Network Control System", Faculty of North Carolina State University, Raleigh, North Carolina, USA, July 2009.

[Pruski 1996]. A.Pruski, Robotique Mobile – La Planification de trajectoire, Ed. Hermes, 1996.

[Ramírez, 2000]. G. Ramírez, "Contribution a la planification de trajectories sans collition de robots mobiles non holonomes – Approche basée sur le calcul de distance dans I´espace des vitesses", Université de Poitiers, France, 2000.

[Rodríguez, 2010]. S. Rodríguez; "Modelo Adaptativo para Organizaciones Virtuales de Agentes"; Universidad de Salamanca, España. 2010.

[Russel y Norvig, 1995]. Russell S. and Norvig P.,"Artificial Intelligence: A Modern Approach"; Englewood Cliffs, NJ: Prentice-Hall, 1995.

[Shapiro 2000]. Linda Shapiro[1] & George Stockman[2], "Computer Vision", The [1]University of Washington, Seattle Washington, USA; [2]Department of Computer Science, Michigan State University, East Lansing, MI, USA; 2000.

[Siegwart, 2004]. R. Siegwart e Illah R. Nourbakhsh, "Introduction to Autonomous Mobil Robots", Massachusetts Institute of Technology, USA, 2004.

[Sucar, 1999]. L. E. Sucar, "Visión Computacional"; Instituto Nacional de Astrofísica, Óptica y Electrónica; Tonantzintla, Puebla, México. 1999.

[Valera et al., 2007] A. Valera, M. Vallés, Francisco de Borja Ponz, " Plataforma para la realización de trabajos prácticos de mecatrónica basados en robots Legos", Dpto. Ingeniería de Sistemas y Automática, Universidad Politécnica de Valencia, España, 2007.

[Wooldridge y Jennings, 1995]. M. Wooldridge, N. Jennings; "Intelligent agents: theory and practice", The Knowledge Engineering Review, vol. 10(2) pp. 115-152, 1995.

[Wooldridge, 2002]. Michael Wooldridge; "An Introduction to MultiAgent Systems"; Editorial John Wiley & Sons Ltd. USA, 2002.