Universidad Autónoma de Baja California

Facultad de Ingenieria, Arquitectura y Diseño

Maestría y Doctorado en Ciencias e Ingeniería

Comunicación de Robots Móviles.

Tesis

que para cubrir parcialmente los requisitos necesarios para obtener el grado de

Maestro en Ingeniería

Presenta

Luis Fernando Pinedo Lomeli

Ensenada, Baja California, Enero del 2016.

UNIVERSIDAD AUTÓNOMA DE BAJA CALIFORNIA

FACULTAD DE INGENIERÍA, ARQUITECTURA Y DISENO
UNIDAD ENSENADA

Comunicación de robots móviles

TESIS

Que para obtener el grado de maestría en ingeniería presenta:

Luis Fernando Pinedo Lomelí

Aprobada por:

Dra. Rosa Martha López Gutiérrez
Director de tesis

Dra. Liliana Cardoza Avendaño
Miembro del Comité

Dr. Cesar Cruz Hernández
Miembro del Comité

Dr. Humberto Cervantes de Ávila
Miembro del Comité

Dr. Adrian Arellano Delgado
Miembro del Comité

Ensenada Baja California, México. Enero 2016

Resumen de la tesis de Luis Fernando Pinedo Lomeli, presentada como requisito parcial para la obtención del grado de MAESTRO EN INGENIERIA. Ensenada Baja California México, Enero 2016.

COMUNICACIÓN DE ROBOTS MOVILES

Director de Tesis: Dra Rosa Martha López Gutiérrez

Los robots móviles han sido empleados en una gran diversidad de aplicaciones industriales y educativas. Uno de los principales problemas que hay es la planeación y el seguimiento de trayectorias. En este trabajo se aborda el problema del movimiento coordinado de un sistema integrado por múltiples robots móviles.

El problema es trasladar a cada robot desde punto inicial determinado hacia un punto final determinado. Los robots se encuentran en un entorno de trabajo con obstáculos, entonces existe un problema de colisión con obstáculos y puede haber colisión entre los robots. Para resolver el problema, se plantean las trayectorias para cada robot, se asegura la evasión de obstáculos, se evita el choque entre los robots del sistema y cada robot cumpla con un movimiento de sincronía sin importar su punto de partida.

Así que para atacar los problemas antes mencionados, se dará una introducción sobre los robots móviles utilizados y el estudio de cada uno por separado, para la demostración de trayectorias. Para realizar la transmisión de datos entre robots será vía Bluetooth, la cual se analizara la forma de transmisión de datos y la comunicación sea automática.

Finalmente se mostrara cada una de las trayectorias demostrando el comportamiento de los sensores en un robot móvil, además de los movimientos coordinados deseados.

Palabras clave: Robots móviles, comunicación, Bluetooth, movimientos coordinados, trayectorias.

“Bienaventurado el hombre que halla la sabiduría, y que obtiene la inteligencia; porque su ganancia es mejor que la ganancia de la plata, y sus frutos más que el oro fino.”

Proverbios 3:13-14

Agradecimientos

A Dios todopoderoso, creador del cielo y de la tierra, creador de las estrellas y todo el universo, creador de todo lo visible y el no visible, por darme la vida, bendecirme siempre, protegerme y cuidar a mi familia.

A mi madre Sandra Lomeli, que cada momento que se encontró conmigo me dio fuerza, sabiduría, diversión y todo su apoyo incondicional. Donde quiera que te encuentres te amo.

A mis hermanos Gabriela, Richard y Michelle por todo su apoyo, confianza, por todas los altibajos que hemos pasado juntos. “Levántate una y otra vez, hasta que los corderos se conviertan en leones”.

A mi directora de tesis, Dra. Rosa Martha López Gutiérrez por compartir sus conocimientos, por sus consejos y dirección para terminar este trabajo, por su apoyo, paciencia y confianza. Muchas gracias.

A mi profesor y revisor de tesis Dr. Cesar Cruz Hernandez, por brindarme su conocimiento, y, sus consejos, como también su amistad.

A mis amigos y, compañeros que me apoyaron y todas las personas que se subieron a esta travesía conmigo.

Y a CONACYT por el apoyo económico brindado durante el desarrollo de esta tesis

Índice general

Índice de figuras

Índice de tablas

Capítulo 1

Introducción

1.1 Motivación

En los últimos años, la investigación en el campo de sistemas robóticos ha permitido realizar tareas más complejas y con mayor precisión. La robótica ha evolucionado conjuntamente con los avances tocológicos recientes, lo cual permite un desarrollo de muchas aplicaciones en aéreas muy diversas. Esta evolución en la robótica ha alcanzado un nivel muy alto en la actualidad como en la industria, la medicina, la aeronáutica y la agricultura. Pero aún se necesita esperar que los robots lleguen a ser un elemento más en aplicaciones cotidianas del hogar y en las situaciones sociales en general.

En la actualidad, los robots en su mayoría son excesivamente caros y su función está muy especializada, debido a ello, los estudios actuales se centran en conseguir que sea más general y económico, de tal manera que puedan ser accesibles a todas las personas que quieran resolver algún problema en sus tareas cotidianas.

Debido a los avances tecnológicos, construir un robot móvil es posible, esto se viene realizando en universidades y centros de investigación. En CENIDET se encuentra un robot móvil guiado automáticamente [3], También, existen la posibilidad de adquirir un robot de tipo educativo hecho por compañía que elabora juguetes y productos de entrenamiento, como LEGO, MECANO, SONY, etc.

Los robots múltiples están cada vez más presentes en la robótica industrial y de servicios, en particular cuando se requiere de capacidades de movimiento flexible y se requiere realizar tareas con mayor complejidad que las que solo robot pueda realizar; la productividad puede ser mejorada reduciendo los tiempos para realizar ciertas tareas.

En los últimos años muchos esfuerzos de investigación se han dirigido hacia el control de robots móviles. El interés en este campo está muy bien justificado por las diversas ventajas que estos sistemas presentan respecto a los robots móviles individuales y están bien apoyados por las mejoras en las tecnologías que permiten la interacción y la integración entre múltiples sistemas.

Además de investigaciones entre múltiples robots móviles que unas de sus aplicaciones son: limpieza de residuos tóxicos, transportación y manipulación de objetos, exploración y vigilancia, y tareas de búsqueda y rescate.

En este trabajo se estudia el caso de robots móviles, bajo cierto caso particular que es la sincronización de robots móviles. La relevancia de las aplicaciones para los sistemas robóticos móviles, está en resolver el problema de movimiento de cada robot del sistema.

1.2 Planteamiento del problema

La inquietud por seguir en una línea de investigación que es la sincronización entre robots móviles, los cuales pueden ser utilizados para formar trabajos colectivos, estudios de comportamientos de la naturaleza, entre otras aplicaciones. A desarrollado problemas como la comunicaciones entre robots móviles en topología maestro esclavo, utilizando sensores de distintos tipos para evitar colisiones entre ellos y colisiones con algún obstáculo que se encuentre en el espacio de trabajo de los robots. Ahora bien, para los robots puedan llevar a cabo dicha tarea, presentan con problemas: como planear el movimiento de cada robot, la comunicación de cada robot sea lo más eficiente sin pérdida de información, las condiciones de evitar obstáculos y evitar colisiones entre robots.

Entonces el problema a resolver es: lograr una comunicación entre robots móviles que tengan un movimiento coordinado mediante la planeación de las trayectorias, que cada robot móvil evite colisiones entre ellos y obstáculos.

1.3 Objetivos

1.3.1 Objetivo General

El principal objetivo de este trabajo consiste en llegar a conocer en profundidad las opciones que ofrece la sincronización entre sistemas y llevado a cabo en forma práctica en robots móviles. Además de conocer las aplicaciones en colaboración de robots.

1.3.2 Objetivos específicos.

A nivel más detallado, los objetivos que han motivado la presente investigación son los siguientes:

- Investigación de distintos tipos de robots móviles.
- Estudio de alcances y limitaciones del Robot Móvil LEGO EV3
- Funcionamiento de Sensores y actuadores del Robot LEGO EV3
- Estudio de las opciones de intercomunicaciones entre dispositivos
- Analizar intercomunicación entre Robots Móvil LEGO EV3
- Estudio de funcionamiento de interfaz EV3 echo por LABview
- Estudio de programación JAVA para LEGO EV3
- Desarrollo se trayectorias con un robots LEGO EV3
- Desarrollo de trayectorias con dos robots móviles LEGO EV3 con topología maestro-esclavo
- Desarrollo de trayectorias de sincronización entre dos robots Móviles LEGO EV3

1.4 Metas

Con el desarrollo de esta investigación en el aria de programación, electrónica y robótica se va contribuir en el desarrollo de aplicaciones tecnológicas como educativas.

Investigación

- Se investigó documentación sobre los robots móviles y clasificaciones de ellos
- La información de robots móviles se centralizara en los robots móviles LEGO EV3, como configuración, herramientas, programación, entre otros.
- Se estudió modos de comunicaciones entre dispositivos.
- Diseño de trayectorias para validar el estudio de los robots móviles.
- Se obtuvo la eficiencia de la configuración del robot móvil que se utilizara para validar distintas pruebas.
- Al examinar los distintos modos de comunicación entre dispositivos, se seleccionó el adecuado.
- Con el estudio de los robots móviles LEGO se seleccionó el mejor método para programarlos.

Practicas

- Se desarrolló una caracterización del robot móvil LEGO para encontrar los alcances y limitación del mismo.
- Con la caracterización se encontró la forma de operación y así diseñar trayectorias para el control del robot.
- Trayectoria de evasión de objetos con un solo robot móvil.
- Trayectoria de forma de sinusoidal con dos robots móviles en topología maestro – esclavo
- Trayectoria en forma de ocho con dos robots móviles en topología maestro – esclavo
- Trayectorita de formación entre dos robots móviles LEGO empezando en distintos puntos del plano en topología maestro-esclavo.

1.5. Estructura de la memoria

En este punto se presenta la estructura en capítulos de esta memoria. Aparte de este primer capítulo de introducción. El resto de los capítulos se organizan de la siguiente forma:

- El capítulo 2 se presenta el área de trabajo donde se sitúa esta investigación. Incluye las características y aspectos más importantes que describen a estos conceptos.

    - En primer lugar se definen algunos conceptos en base a la robótica, antecedentes tipos de robots indicando sus características de cada uno de ellos.
    - Posteriormente se muestra las características robot móvil que se utilizara en esta investigación, como antecedentes, conceptos

Capítulo 2

Estado Del arte

2.1 Introducción

En el año de 1961 hace su aparición en los E.U.A. el primer robot creado por Joseph Engelberger; inicia su comercialización en 1968 a través de la empresa UNIMATION, y posteriormente los japoneses han llevado su desarrollo para industrias automotriz, plantas ensambladoras con velocidad de dos automóviles por minuto. Varios de los aspectos hacen interesante la industria robotizada:

- No se necesita infraestructura social para instalar una planta robotizada, es decir, medios de transporte, escuelas, restaurantes, etc., ya que los robots no consumen estos servicios.
- Los robots no requieren de representante sindical, ni pierden el tiempo en cuestiones políticas, ni van al baño, ni toman café.
- La calidad que producen es perfecta, o lo que llaman los japonenses calidad total-cero errores, y de ser programados para cometer errores intencionales de manera que el producto parezca hecho a mano.

Muchas personas se preguntan ¿Qué expectativas tenemos en el futuro inmediato, cuando en pocos años será posible amortizar un robot por su equivalente a un año de sueldos y salarios de los trabajadores sustituidos? Pero lo más importantes es mencionar el hecho que la calidad que se obtiene a través de los robots es de cero errores o también llamada "calidad total", independientemente de que la velocidad de producción es impresionante.

En última instancia, el avance de la robótica ha venido a acelerar los procesos deductivos, a abatir costos, a consumir.

En este capítulo, se presenta los antecedentes de los robot, como su clasificación, también muestra algunas definiciones y las características de los robots a utilizar, el cual será el

Robot Lego EV3. Además nos muestra las plataformas existentes para la manipulación de los robots Lego EV3.

2.2 Robótica

Los robots son máquinas en las que se integran componentes mecánicos, eléctricos, electrónicos y de comunicaciones, y dotadas de un sistema informático para su control en tiempo real, percepción del entorno y programación.

La robótica es todavía una disciplina relativamente joven. En efecto, aunque el término robot se acuña en los años veinte del pasado siglo, la robótica industrial nace en los cincuenta y sólo en los setenta comienzan a impartirse cueros de robótica en un gran número de universidades.

En los años ochenta y noventa se han diseñado un gran número de máquinas cuyo objetivo no es sustituir la actividad directa de un trabajador en una cadena de producción. Se trata de realizar tareas en lugares difíciles por el tamaño de los objetos que es necesario manipular. Así, se han abordado aplicaciones en la exploración espacial, actividades subacuáticas, manipulación y transporte de materiales peligrosos, minería, agricultura, construcción, cirugía, etc. La mayoría de esta autonomía llegando a construir robots. El número de estos robots aumentará de forma importante en los próximos años. Otro sector de importancia creciente en las aplicaciones de la robótica es el de los robots de servicios, entre los cuales se incluyen los robots domésticos, robots de ayuda a los discapacitados, y robots asistentes en general. Tampoco hay que olvidar que los humanos siempre han sentido una gran atracción por las máquinas que imitan los estos más visibles de los seres vivos en general y de las personas en partículas.

Cuando este tipo de configuraciones se incluyen elementos dotados de movilidad (robots, vehículos, helicópteros,...), la capacidad de obtener y procesar información, así como la de generar acciones de control, es variable y en gran medida dependiente del tiempo. Es más, los recursos disponibles en cada momento por cada elemento para realizar las actividades que se le asignen, también variables, son limitados. No obstante, las idea de hacer uso de distintos tipos de robots, bien por su tamaño, su velocidad de movimiento, sus sensores, su capacidad de volar, etc. Por eso, es preciso definir una estructura jerárquica que ayude como al control y coordinación de forma que se asegure que las acciones de máxima prioridad.

2.3 Antecedentes

La noción de robot proviene de dos conceptos genéricos, los humanoides y los autómatas. El concepto de humanoide es un concepto ancestral. De hecho, leyendas de Egipto,

Babilonia o sumeria fechadas hace 5000 años, reflejan la imagen de la creación, en la que los dioses dan vida a modelos de arcilla. Una variación es la idea del gólem, asociada a la Praga del siglo XVI, en la que un modelo de arcilla, después de cobrar vida, se vuelve destructivo. El gólem es el precursor de Frankenstein de Mary Shelley, que en su libro "The Modern Prometheus" (1818) combina la noción de humanoide con los peligros de la ciencia.

El concepto de autómata significa literalmente entidades dotadas de movimiento y ha tenido siempre un gran interés para el hombre. Los primeros modelos de autómatas dependían de palancas y manivelas, o bien estaban basados en la hidráulica. Después se aplicaron conceptos basados en los avances en la ciencia de la relojería (siglo XIII) y posteriormente con las máquinas de vapor y la electromecánica, Durante el siglo XX, las nuevas tecnologías trasladaron los autómatas al termino de dispositivos útiles.

La palabra robo proviene del término checo *robota*, que significa trabajo forzado servicio obligatorio, o bien del término *robotnik*, que significa siervo, Se utilizó en primer lugar por el dramaturgo Karel Capek, en 1918, en historia corta y posteriormente en 1921, en la obra de teatro titulada R.U.R. (Rossum's Universal Robots). Rossum era un inglés de ficción que utilizaba métodos biológicos para inventar y producir en serie "hombres" que servían a los humanos. Con el tiempo, estos seres se rebelaron y acabaron como raza dominante, sometiendo a la humanidad. Esta idea original hace que se asocie a los robots con seres de apariencia humanoide. Esta imagen, además, se ve reforzada con la película alemana Metrópolis (1926), en la que aparecía un andador eléctrico. Posteriormente, las películas de ciencia ficción (La Guerra de las Galaxias, Terminator, Robocop,…) han continuado reforzando esta idea. (Figura 2.1)

Escena de la obra (R.U.R)

Detras de Escena de la pelicula Metrópilis

C3-PO y R2D2 (La Guerras de las galaxias)

Robocop

Wall-E

Figura 2.1: Robots en la ciencia ficción

En 1942, Isaac Asimov, junto con el autor y editor John W. Campbell, formuló las leyes de la Robótica.

1. Un robot no puede dañar a un ser humano o, por inacción, permitir que un ser humano pueda resulte dañado.
2. Un robot debe obedecer las órdenes dadas por los seres humanos excepto cuando tales órdenes entren en conflicto con la Primera Ley.
3. Un robot debe proteger su propia existencia hasta donde esta protección no entre en conflicto con la Primera o Segunda Ley.

Estas leyes, formuladas dentro del entono de la ciencia ficción, se han tomado como base y referencia histórica en la robótica y la inteligencia artificial, si bien hay abierta una larga controversia respecto a su implantación en el mundo industrial [R.Clarke, 1993]. De hecho, mucha gente se encargó de analizar casos conflictivos que pueden presentarse mediante la aplicación literal de dichas leyes. Tras un análisis de dichos conflictos, posteriormente se añadió una Ley cero orientada a proteger a la humanidad, ajustándose las anteriores en consecuencia.

A finales de los años 40 se inician programas de investigación en los laboratorios Oak Ridge y argonne National Laboratories para desarrollar manipuladores eran del tipo "maestro - esclavo", diseñados para que reprodujeran fielmente los movimientos de brazos y manos realizados por un operario. Fue alrededor de 1948 cuando surgió el progenitor más directo del robot, el *telemanipulador* (Figura 2.2), que fue desarrollado por R.C. Goertz del Argonne National Laboratory, para manipular elementos radioactivos evitando riesgos al operario. Era un dispositivo mecánico en el cual el operario accionaba los controles del manipulador desde una zona segura, mientras observaba a través de un cristal las evoluciones del mismo con los objetos radioactivos. Posteriormente se sustituiría el control mecánico por un control eléctrico.

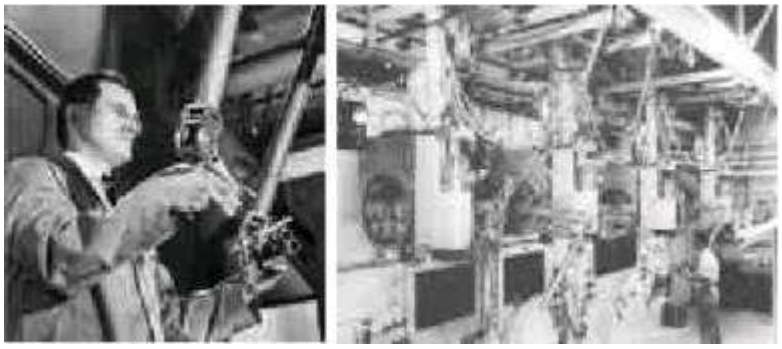

Figura 2.2: Manipulador teleoperados de Goertz

Ralph Moshe de General Electric desarrolló un dispositivo consistente en dos brazos teleoperados controlados por un dispositivo maestro. El desarrollo de telemanipuladores fue seguido muy de cerca por la industria submarina, nuclear y en los años sesenta por la industria espacial, pero el concepto de telemanipulador ha quedado muy reducido a estas industrias hoy en día. Pronto surgió la necesidad de sustituir el control del operario por un ordenador con un programa que controlase los movimientos del manipulador, dando así al concepto de robot.

A mediados de los años 50, George C. Devol desarrolló un dispositivo de transferencia programada articulada. Se trataba de un manipulador cuyas operaciones podían ser programadas y almacenadas. Los desarrollos posteriores de este concepto por el propio Devol y Joseph F. Engelberger condujeron al primer robot industrial, introducido en 1959 por Unimation Inc. Universal Automation) la empresa pionera en el campo de la robótica y fundada por Engelberger. Este dispositivo utilizaba un computador junto con un manipulador que conformaba una máquina que podía ser enseñada para la realización de tareas variadas de forma automática.

En los años 60 en vista de una mayor flexibilidad, se hace necesaria la realimentación sensorial. En 1962, H. A. Ernst publica el desarrollo de una mano mecánica con sensores táctiles controlada por computador. Llamándola MH-1. Este modelo evolucionó adaptándole una cámara de televisión dentro del proyecto MAC. También en 1962, Tomovic y Boni desarrollan una mano con un sensor de presión para la detección del objeto que proporcionaba una señal de realimentación al motor. En 1963 se introduce el robot comercial VERSATRAN por la Amercican Machine and Foundry Company (AMF). Se incorporaron varios robots UNIMATE serie 2000 en las cadenas de fabricación de automóviles de General Motors en 1967 y 1968. En 1968 se publica el desarrollo de un sistema con manos, ojos y oídos (manipuladores, cámaras y micrófonos) por parte de McCarthy en el Stanford Artificial Intelligence Laboratory del Stanford Research Institute. Este sistema era capaz de reconocer mensajes hablados, detectar piezas distribuidas en una mesa y manipularlas de acuerdo con ciertas instrucciones.

Las primeras aplicaciones industriales en Europa, aplicaciones de robots industriales en cadenas de fabricación de automóviles, data de los años 1970 y 1971. En este último año, Khan y Roth analizan el comportamiento dinámico y le control de un brazo manipulador. En 1973 ASEA (actual ABB), industria sueca construyó el primer robot con accionamiento eléctrico, el IRB 6 y más tarde el IRB 60.

Durante la década de los 70, la investigación en robótica se centra en gran parte en el uso de sensores externos para su utilización en tareas de manipulación. Así, en 1073, Bolles y Paul utilizan realimentación visual y de fuerza en el brazo Stanford para el montaje de bombas de agua de automóvil. En 1974 Inue, en el Artificcial Intelligence Laboratory del

MIT, desarrolló trabajos de investigación en los que aplicaba la inteligencia artificial en la realimentación de fuerzas.

En 1975, Will y Grossman, en IBM, desarrollaron un manipulador controlado por computadora con sensores de contacto y fuerza para montajes mecánicos. En 1976 los robots espaciales de la NASA Viking 1 y Viking 2 son utilizados para tomar imágenes en Marte. En 1977 Machine Inteligence Corporation desarrolla un sistema de cisión comercial que se puede integrar en aplicaciones industriales robotizadas.

En 1980 se forma une Europa la Federación Internacional de Robótica (IFR). Los primeros robots tenían configuraciones denominadas antropomórficas y esféricas, de uso para la manipulación, pero es en 1982 cuando Makino de la Universidad de Yamamashi en Japón define el concepto de robot SCARA (Selective Compilanc Assembly Robot Arm) que es un robot con 3 ó 4 grados de libertad orientado al ensamblado de piezas.

2.4 Tipos de robots

Podemos encontrar distintos tipos de robots para facilitar el trabajo al ser humano, con esto podemos encontrar la diferencia de entre los robots industriales y los robots de servicio. Los cuales se diferencian principal mente en la aplicación que realiza cada uno, como también podemos encontrar que existen en distas configuraciones.

2.4.1 Robots de servicio

Los robots de servicio tienen como una de sus principales características es trabajar en entornos no estructurados y al menos parcialmente desconocidos. El servicio que ofrecen es el acceder a zonas inaccesibles o peligrosas para el ser humano que su rentabilidad económica. En la mayoría de los casos se trata de robots teleoperados, si bien la investigación y la tecnología han estado avanzando a pasos agigantados se han empezado a introducir aspectos de autonomía. El control aplicado sobre ellos está basado generalmente en comportamientos reactivos, de forma que ante un cierto evento, reaccionan según su programación.

Los robots de servicio se usan en entornos no industriales, o más concretamente, excluyendo las operaciones de fabricación. Por lo tanto ese utiliza en sectores como la medicina, agricultura, construcción, entretenimiento, etc. Pero también hay que hacer una separación en los robots de producción, transporte y almacenamiento que se usan en estos sectores como los robots móviles en hospitales, oficinas o bibliotecas que se pueden considerar como robots industriales destinados a la producción.

2.5 Robots Industriales

La utilización de robots industriales en el proceso productivo, ha dado lugar al desarrollo de controladores industriales rápidos y potentes, basados en microprocesadores, así como un empleo de servo en bucle cerrado que permiten establecer con exactitud la posición real de los elementos los robots y una desviación o error. Esta evolución ha dado origen a una serie de tipos de robots.

2.5.1 Robots manipuladores

Un robot manipulador de acuerdo a la definición de la Federación Internacional de Robótica (IFR), es una máquina de manipulación automática reprogramable y multifuncional con tres o más ejes que puede posicionar y orientar materia, piezas herramientas o dispositivos especiales para la ejecución de trabajos diversos en la diferentes etapas de producción industrial, ya sea en una posición fija o en movimiento [1]

Figura2.3: Brazo Robot

También lo podemos definir mecánicamente, como robot manipulador (o simplemente manipulador) como una brazo mecánico articulado formando por eslabones conectados a través de uniones o articulaciones que permiten el movimiento relativo entre dos eslabones consecutivos [14]. El movimiento de cada articulación puede ser traslacional, rotacional o una combinación de ambos. Bajo consideraciones razonables, el número de articulaciones en un manipulador determina su número de grados de libertad (g. d. l.). Típicamente, un manipulador dispone de 6 g.d.l. de los cuales 3 ubican la posición de extremo del último eslabón en el espacio y 3 más especifican su orientación.

2.5.2 Robots de repetición o aprendizaje

Estos son manipuladores que se limitan a repetir a una secuencia de movimientos, previamente ejecutada por un operador humano, haciendo uso de un controlador manual o un dispositivo auxiliar. Este tipo de robots, el operario durante la fase de enseñanza se vale de una pistola de programación con diversos pulsadores o teclas, un joystick, o utiliza un

maniquí, o bien desplaza directamente la mano del robot. Actualmente, los robots de aprendizaje son los más conocidos en algunos sectores de la industria, y el tipo de programación que incorporan reciben el nombre de *"gestual"*.

Figure 2.4: Robot Pingüino de aprendizaje

2.6 Robot Móviles

Los robots móviles se realizaron a la necesidad de expandir el campo de aplicación de la Robótica, restringido inicialmente al alcance de una estructura mecánica anclada en uno de sus extremos. Además de incrementar la autonomía limitando todo lo posible la interacción humana. Desde el punto de vista de la autonomía, los robots móviles tienen como precedentes los dispositivos electromecánicos, tales como los denominados "micro-mouse", creados desde los años treinta para desarrollar funciones inteligentes tales como cubrir caminos en laberintos. Esos trabajos de investigación tienen una relación con los vehículos autónomos que comenzaron aplicarse desde los años sesenta en la industria, siendo guiados por cables bajo el suelo o mediante sensores ópticos para seguir líneas trazadas en la planta. Las aplicaciones, hoy en día son muy comunes en muchos procesos de fabricación.

En los años setenta se vuelve a trabajar en el desarrollo de robots móviles dotados de una mayor autonomía. La mayor parte de las experiencias se desarrollan empleado plataformas que soportan sistemas de visión [22]. En los años ochenta el incremento de la capacidad computacional y el desarrollo de nuevos sensores, mecanismos y sistemas de control, permite aumentar la autonomía. Se desarrollan los robots móviles para interiores como para navegación exterior, realizados en la Carnagie Mellon University (Pittsburgh, EE.UU.). Se trata que los robots tengan la suficiente inteligencia como para reaccionar y tomar decisiones basadas en observaciones de su entorno, sin suponer que este entorno es perfectamente conocido.

La autonomía de un robot móvil se basa en el sistema de navegación automática. En estos sistemas se incluyen tareas de planificación, percepción y control. En los robots móviles, el problema de la planificación global de la misión, de la ruta, de la trayectoria y, finalmente, evitar obstáculos. En un robot par interiores, la misión podría consistir en determinar a qué habitación hay que desplazarse, mientras que la ruta establecería el camino desde la posición inicial a una posición en la habitación, definiendo puntos intermedios de paso. Un vehículo puede desviarse de la ruta debido a la acumulación de imprecisiones mecánicas y de control.

Existen numerosos métodos de planificación de caminos para robots móviles que se basan en hipótesis simplificadoras. En particular hay muchos métodos que buscan caminos libres de obstáculos que minimizan la distancia recorrida en un entorno modelado mediante polígono. En otros casos se modela el espacio libre tratando de encontrar caminos por el centro del mismo. Para facilitar la búsqueda existen técnicas de descomposición del espacio en celdas [29], utilización de restricciones de varios niveles de resolución y búsqueda jerarquizada [28] que permiten hacer más eficiente el proceso con vistas a su aplicación en tiempo real. La planificación de la trayectoria puede realizarse también de forma dinámica, considerando la posición actual del vehículo. Por ejemplo, en vehículos con ruedas y tracción convencional, interesa definir trayectorias de curvatura continua que puedan ejecutarse con el menor error posible [12], [23]. Además de las características geométricas y cinemáticas, puede ser necesario tener en cuenta modelos dinámicos de comportamiento del vehículo contemplando la interacción vehículo-terreno. Por otra parte, puede plantearse también el problema de la planificación de la velocidad teniendo en cuenta las características del terreno y del camino que se pretende seguir. Una vez realizada la planificación de la trayectoria, es necesario planificar movimientos concretos y controlar dichos movimientos para mantener al vehículo en la trayectoria planificada. De esta forma, se plantea el problema del seguimiento de caminos, que para vehículos con ruedas se concreta en determinar el ángulo de dirección teniendo en cuenta la posición y orientación actual del vehículo con respecto a la trayectoria que debe seguir. Asimismo, es necesario resolver el problema del control y regulación de la velocidad vehículo. Conviene mencionar también los métodos que permiten la integración de la planificación con el control del vehículo. Entre estos cabe mencionar el de los campos potenciales. La idea consiste en determinar la resultante de fuerzas que atraen el robot hacia el objetivo y que lo repelen de los obstáculos. En cualquier caso, el problema del control automático preciso de un vehículo con ruedas puede resultar más complejo que el de los manipuladores debido a la presencia de restricciones no holonomicas. Los bucles de control se plantean tanto en el espacio de las variables en el espacio de las variables articulares como en coordenadas del mundo, y las ecuaciones de movimientos son complejas, si se considera la interacción con el terreno.

2.6.1 Sistemas de locomoción

Existe una gran variedad de maneras de cómo moverse sobre una superficie; para los robots móviles, las más comunes son las ruedas, las cadenas y las patas.

Para encontrar una buena elección para encontrar un buen sistema, un factor muy importante son: el uso al cual está destinado el robot, el terreno en el que debe de moverse y el nivel de prestaciones que deseemos relacionarlas con la inversión que se pretende hacer.

2.6.2 Robots móviles con ruedas.

También llamados vehículos con ruedas el cual es capaz de un movimiento autónomo las cuales dan a la solución más simple y eficiente para conseguir la movilidad en terrenos suficientemente duros y libres de obstáculos, permitiendo conseguir velocidades relativamente altas. Como limitación más significativa cabe mencionar el deslizamiento en la impulsión. Dependiendo de las características del terreno pueden presentarse también deslizamiento y vibraciones. Los robots móviles emplean diferentes tipos de locomoción mediante ruedas que les confieren características y propiedades diferentes respecto a la eficiencia energética, dimensiones cargas útiles y maniobrabilidad. A continuación se comentan brevemente las características más significativas de los sistemas de locomoción más comunes en los robots móviles.

2.6.2.1 Ackerman

El vehículo de cuatro ruedas convencional. Los vehículos robóticos para exteriores resultan normalmente de la modificación de vehículos convencionales tales como automóviles incluso vehículos más pesados. La rueda delantera interior gira un ángulo ligeramente superior a la exterior $(\theta_1 > \theta_0)$ para eliminar el deslizamiento. Las prolongaciones del ejes de las dos ruedas delanteras intersectan en un punto sobre la prolongación del eje de las ruedas traseras. El lugar de los puntos trazados sobre el suelo por los centros de los neumáticos son circunferencias concéntricas con centro el eje de rotación P_1 en la Figura 2.5. S i no se tienen en cuenta las fuerzas centrífugas, los vectores instantáneos son tangentes a estas curvas.

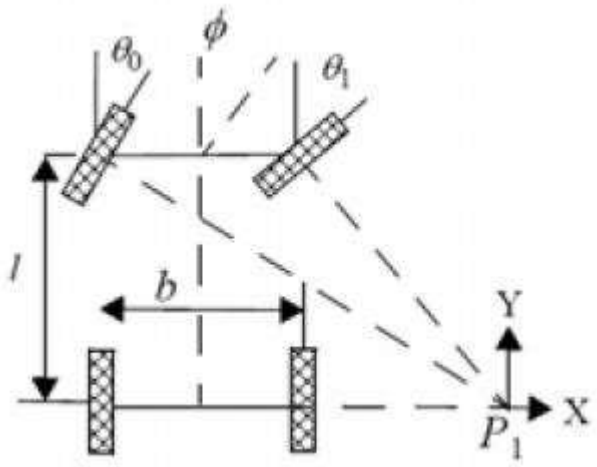

Figura 2.5 : Sistema Ackerman

Figura ¡Error! Utilice la ficha Inicio para aplicar 0 al texto que desea que aparezca aquí.**6: NavLab 2**

En la Figura 2.5 se muestra el NavLab 2, un vehículo autónomo que ha venido empleándose en el Robotics Institute de Carnegie Mellon University desde 1984 para experimentos de navegación autónoma en exteriores, el vehículo es todo terreno el cual es utilizado para reconocimiento ya que cuenta con un software que puede conducir sobre terreno accidentado, evitando los obstáculos, a velocidades de hasta 6 mph y en carretera a 70 m.p.h. En la Figura 2.7 puede observarse el NavLab5, Es un Pontiac Trans Sport 1990, este vehículo es utilizado para experimentos de navegación en carretera, el cual alcanza una velocidad máxima de 90 mph.

Figura2.7: NavLab *5*

2.6.2.2 Triciclo Clásico.

El sistema de locomoción se ilustra en la figura 2.8. La rueda delantera sirve tanto para la tracción como para la dirección. El eje trasero, con dos ruedas laterales, es pasivo y sus ruedas se mueven libremente. La maniobrabilidad es mayor que en la configuración anterior pero puede presentar problemas de estabilidad en terrenos difíciles. El centro de gravedad tiende a desplazarse cuando el vehículo se desplaza por una pendiente, causando la pérdida de tracción.

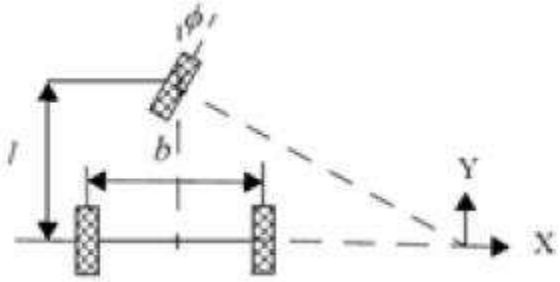

Figura 2.8: Locomoción de triciclo clásico.

2.6.2.3 Configuración síncrono

Todas las ruedas (usualmente 3) se mueven en forma síncrona para dar vuelta y avanzar. Las ruedas están ligadas de forma que siempre apuntan en la misma dirección y para dar vuelta giran las ruedas sobre el eje vertical, por lo que la dirección del chasis se mantiene. En esta configuración se requiere de un mecanismo adicional para mantener el frente del chasis en la dirección de las ruedas (torreta). En la configuración síncrona se evitan los problemas de inestabilidad y de perdida de contacto del diferencial, pero también tiene mayor complejidad.

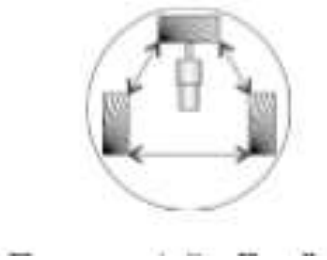

Figura 2.9: Configuración síncrona

2.6.2.4 Configuración diferencial

Está compuesta de dos ruedas colocadas en el eje perpendicular a la dirección del robot. Cada rueda es controlada por un motor, de tal forma que el giro del robot queda determinado por la diferencia de velocidad de las ruedas. Así, para girar a la izquierda, hay que darle una velocidad mayor a la rueda derecha. Para girar a la derecha hay que darle una velocidad mayor a la rueda izquierda. La Figura 2.10 muestra este tipo configuración.

Figura 2.10: Robot con estructura diferencial. (Robot Khepera)

Uno de los problemas que tiene la configuración diferencial es mantener el equilibrio del robot, ya que consta de dos ruedas, por lo cual se agregan ruedas de libre giro. Estas ruedas sirven para mantener el horizontal al robot, por lo que gira libremente según el movimiento. Además, estas ruedas se orientan a la dirección del robot. Todo depende de sus necesidades, se pueden agregar, una o más ruedas de libre giro.

2.6.2.5 Configuración a cadena

Esta configuración es pesada, costosa y compleja. Sin embargo en condiciones adversas de irregularidad del terreno son más eficientes que las configuraciones a ruedas. Esta configuración es llamada también de "oruga". Basada en rodear todas y cada una de las llantas con una misma cadena por cada lado. La configuración "oruga" es muy utilizada por la milicia para la desactivación de bombas y recorrer terrenos inhóspitos como en Marte.

Figura 2.11: Robot configuración oruga

2.6.3 Robots articulados

En un sistema locomoción a "patas" el diseño es un auténtico reto, debido al elevado número de grados de libertad y a la mayor complejidad mecánica y de los algoritmos de control. El desarrollo de estos sistemas es muy atractivo para la exploración espacial, o lugares de difícil acceso para el humano tales como volcanes, rifts (grietas entre dos placas terrestres), etc.

2.6.3.1 Robots bípedos

Existen varios diferentes diseños de robots bípedos uno de los más populares son los denominados "humanoides" con dos patas con varias articulaciones, de manera que sea capaz de emular el caminar del humano. Tiene el inconveniente del equilibrio. En el ser humano, en los oídos interno, existen unos "receptores" del equilibrio que son capaces de

detectar cambios de orientación de la cabeza en las 3 dimensiones. Se puede decir que el ser humano está provisto de acelerómetros biológicos. Este inconveniente ha sido resuelto de distintas maneras, sin embargo, no se ha conseguido que un robot realice la actividad de correr como el ser humano, debido a que el ser humano es capaz de mantener el equilibrio incluso sin ninguno de los pies e contacto en la superficie. Un robot bípedo ha de tener al menos uno de los dos pies en la superficie, así como corredor de marcha.

Figura ¡Error! No hay texto con el estilo especificado en el documento**..12: Robot Bipedo**

Algunos robots Bípedos pueden realizar tareas como bailar sincronizada mente como el robot bípedo Robonova que llevan a cabo rutinas entre 2 o más robots sincronizada mente. Además de Luchar o practicar deportes.

2.6.3.2 Robots insectos

Generalmente contiene diseños simétricos, ya que se emplean por un numero de patas, son capaces de avanzar perfectamente en terrenos muy irregulares. Por lo general suelen tener 6 u 8 patas.

Este tipo de robots se basan en movimientos de parejas de servos: el servo A levanta la pata, el servo B adelanta la posición relativa de la pata respecto del cuerpo, el servo A baja la pata y se reafirma en la superficie, en ese instante el servo B vuelve a su posición inicial, haciendo avanzar el cuerpo. Para que el movimiento sea efectivo no debe de haber incongruencias en la coordinación de los distintos movimientos de las diferentes patas

Figura 2.13: Robot insecto

2.6.3.3 Robots Zoomórficos

Los robots Zoomórficos se constituyen principalmente por sus sistemas de locomoción que imitan a los diversos seres vivos como animales. A pesar de la disparidad morfológica de sus posibles sistemas de locomoción es conveniente agrupar a los robots zoomórficos en dos categorías principales: caminantes y no caminantes.

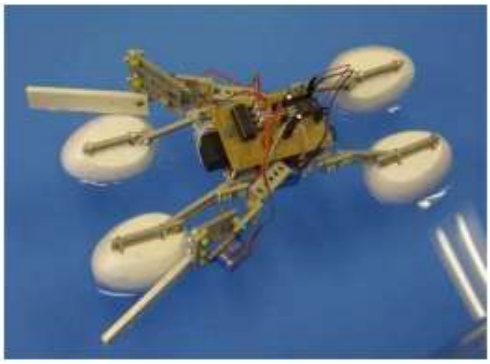

Figura 2.14: Robot flotador

El robot que observamos en la figura 2.14, la cual está suspendida en el agua, utilizando las 4 extremidades para mantener flotando, y dos para transportarse: en este zoorobot podemos ver cierta similitud con un insecto pero realmente la forma no es tan detallada. Entonces encontramos que no siempre un zoorobot tiene una forma exacta a la de un animal, basta con que su desplazamiento sea similar a como se desplaza un animal para considerarlo como un zoorobot.

- Caminantes

Los Robots zoomórficos caminadores son muy numerosos y están siendo experimentados en diversos laboratorios con vistas al desarrollo posterior de verdaderos vehículos terrenos, piloteados o autónomos, capases de evolucionar en superficies muy accidentadas. LAS aplicaciones de estos robots serán interesantes en los caminos de la exploración espacial y en estudio de los volcanes.

En la figura, encontramos el robot Spidernaut, de la NASA, que no es más que un robot arácnido (con forma de araña) diseñado para reparar naves espaciales. Actualmente ese zoorobot es solo ¼ de lo que será su tamaño final, en donde se estima que su peso superara los 270 Kilogramos. Y en el momento de reparar un nave, por su diseño en el cual pueda repartirse en su peso en 8 patas y así pueda generar menos daño que lo que pueda generar una persona.

Figura 2.15: Robot Spidemaur, de la NASA

- No caminantes

El grupo de los no caminantes están muy poco evolucionados. Los experimentos efectuados en Japón basados en segmentos cilíndricos biseladosacoplados aialmente entre si y dotados de un movimiento relativo de rotación. Como por ejemplo de estos podríamos tomar peces, y algunos insectos.

Figura 2.16: Robot lampea

En la figura vemos un ejemplo de un zoorobot no caminador. Este robots es similar a una lamprea (“Las lampreas son peces primitivos, agnatos (sin mandibula), semejantes externamente a la anguila, aunque muy lejanamente emparentados con ella, y con cuerpo gelatinoso y muy resbaladizos, sin escamas y con forma cilíndrica”)

- Mascotas Robot

Las mascotas robots se están convirtiendo en una compañía para las personas y negocios grandes para sus fabricantes. En la actualidad AIBO, es un robot mascota fabricado por SONY, y para hacer su funcionamiento tal como el de una mascota dispone de sensores que le evitan golpearse contra objetos, y una cola que funciona de antena y además de sentido del tacto.

AIBO no es el resultado de una investigación exhaustiva, refleja la fascinación humana para crear vida"; es una combinación de varias tecnologías tales entre las que encontramos la robótica, la inteligencia artificial y la multimedia; AIBO le ha significado a la firma japonesa el ubicar a 692 mil perros robots en diferentes hogares dese 1999.

Figura 2.17: Robot Aibo Sony

2.6.4 Robots subacuáticos

Un robot submarino está diseñado para realizar tareas bajo el agua, las cuales pueden ser realizadas mientras navega o al llegar a un lugar predefinido mediante algún tipo de manipulador. Desde este punto de vista, los robots submarinos pueden hacer dos tipos de misiones:

Misiones de Inspección. Son aquellas misiones que se realizan durante la navegación del robot submarino. En este tipo de tareas no se requiere un brazo manipulador, ni mecanismos para interactuar con el medio ambiente. Una misión de inspección consistirá en recoger imágenes con una o varias cámaras mientras el robot navegan en el agua. Por otro lado, la observación del lecho marino mediante la obtención de la cartografía acústica o en la obtención de datos relativos a la calidad del agua donde navega.

Figura 2.18: Robot de inspección submarina

Misiones de Manipulación. Son las misiones en las que el robot submarino interviene brazos manipuladores y/o herramientas. Para el desarrollo de estas misiones deberá contar con un sistema de visión en tiempo real (en el caso de los ROVs), que proporcionan al operador las imágenes en directo del entorno de operación. Las tareas típicas de manipulación comprenden: el mantenimiento de estructuras subacuáticos; la apertura y cierre de válvulas en instalaciones subacuáticas; la desactivación de minas; el ensamble y desensamble de componentes; la recolección de muestras para estudios arqueológicos, geológicos o ecológicos.

Figura 2.19: Robot submarino ROVs

2.6.5 Robots aéreos

Un vehículo Aéreo no Tripulado (UAV: Unmanned Aerial Vehicle) es un vehículo controlado automáticamente o desde tierra utilizando planes de vuelo programados. [16] Existen distintos tipos y mecanismos en la robótica aérea, como los Predator, Cudricopteros (ArDrone), helicópteros, entre otros.

Por ejemplo la Universidad de Maryland, en Collage Park, EEUU, fue el escenario de la presentación del modelo AD-150, un vehículo aéreo no tripulado desarrollado por American Dynamics Flight Systems. El AD-150 utiliza alta tecnología de propulsión que le

permite despegar verticalmente, así como un control PID para realizar la transición hacia cuelo horizontal, y mantener una capacidad de vuelo aérea muy alta.

Figura 2.20: Robot Aereo AD-150

El MIT, (Boston), está desarrollando el proyecto UAV SWARM Health Management Project, cuyo objetivo es la posibilidad de ejecutar misiones de larga duración con una flota de UAVs en un entorno dinámico. Dado que cada vehículo es limitada, se han de coordinar relevándose para ir a reportar sin descuidar la misión que estén llevando acabo. Los vehículos operan de forma autónoma, y el sistema está controlado por un ser humano.

Figura 2.21: Robot Aereo UAV SWARM

2.6.6 Estructura general de un robot móvil.

Debido a que el robot móvil por lo general está destinado a simular el comportamiento de personas y animales con un nivel de eficiencia similar, la estructura tanto de un robot móvil como de un ser vivo.

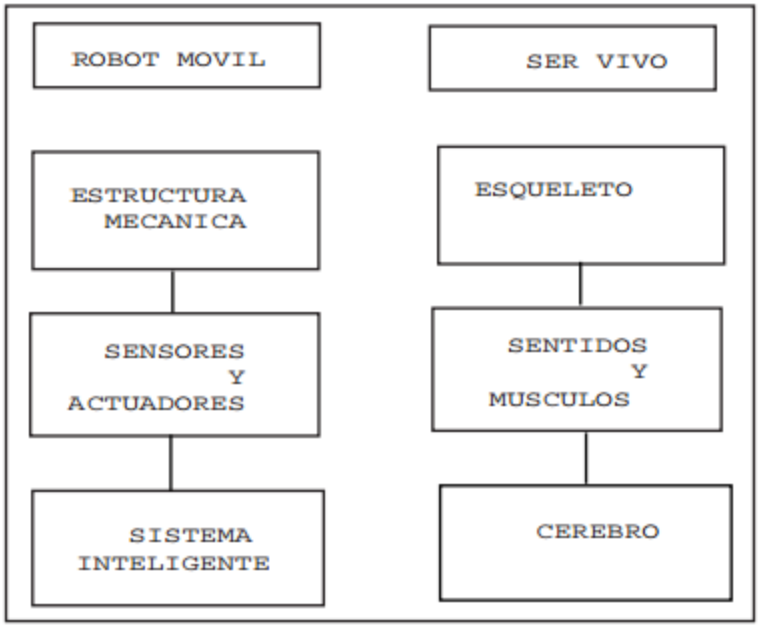

Figura 2.22: Similitudes entre un robot móvil y ser vivo.

La estructura de un robot móvil presentada en la figura, está conformada por diferentes subestructuras, tales como:

- Estructura mecánica: estructura con ruedas, patas y orugas.
- Actuadores: motores, luces, brazos, ruedas y en definitiva cualquier elemento que permita interactuar con el entorno.
- Sensores: sonar, láser, brazos, ruedas y en definitiva cualquier elemento que permita interactuar con el entorno.
- Inteligencia: método, algoritmo, etc. Estos van a permitir, a partir de la información de los sensores, interactuar con el entorno.

2.6.7 Grados de libertad, tipos de ruedas y centro instantáneo de rotación.

Los grados de libertad de un robot, así como los tipos de ruedas son aspectos que intervienen en el proceso de control y análisis de movimiento del robot.

Grados de libertad (GDL): Es cada uno de los movimientos de desplazamiento y rotación que puede realizar el robot.

- Un cuerpo que se mueve en dos dimensiones tiene 3 GDL (una rotación y 2 traslaciones).
- Un cuerpo que se mueve en tres dimensiones tiene 6 GDL (3 rotaciones y 3 traslaciones) .

Sistema holonómico y no holonómico. Un sistema es holonómico si la cantidad de grados de libertad que se pueden controlar es igual a la cantidad de grados de libertad disponible. En un sistema que es no holonómico el robot móvil no podrá desplazarse lateralmente. Esto se ilustra en la figura 2.23.

Un sistema no holonómico, las ecuaciones diferenciales no son integrables en la posición final del robot. El saber la distancia recorrida por cada rueda, no es suficiente para poder calcular la posición final del robot. Hay que conocer como fue ejecutado el movimiento, en función del tiempo.

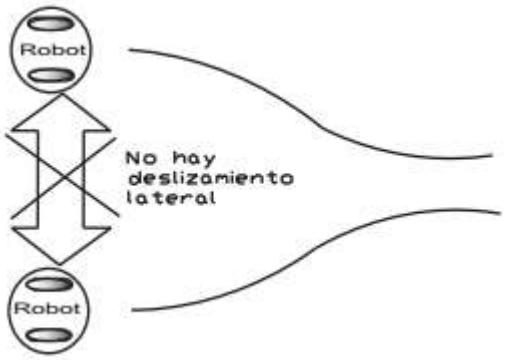

Figura 2.23: Sistema no holonómico

Tipo de ruedas. Las ruedas son el elemento que proporciona la capacidad de movilidad en un robot móvil. Se clasifican como:

- Rueda fija. Figura 2.24(a). El movimiento se produce en la dirección de la rueda. Dónde: ω es la velocidad angular de la rueda, V la velocidad y a_x el vector unitario de dirección.

- Orientación centrada. Figuera 2.24 (b). Gira sobre el eje de la rueda y rota alrededor del eje vertical situado a una distancia d desde el centro de la rueda. Dónde: ω es la velocidad angular de la rueda, V la velocidad y a_x vector unitario de dirección.

- Orientación descentrada. Figura 2.25 (a). Gira sobre el eje de la rueda y rota alrededor del eje vertical situado a una distancia a_x desde el centro de la rueda.

Dónde: ω es la velocidad angular de la rueda, V la velocidad y a_x el vector unitario de dirección.

- Rueda seca. Figura 2.25 (b). Además de moverse en la dirección de la rueda, se mueve en dirección perpendicular a la dirección de la rueda. $V = (r * \omega)a_x + Ua_s$
  Dónde: U es la velocidad de deslizamiento y a_s es un vector unitario en la dirección del deslizamiento, ω es la velocidad angular de la rueda y V la velocidad.

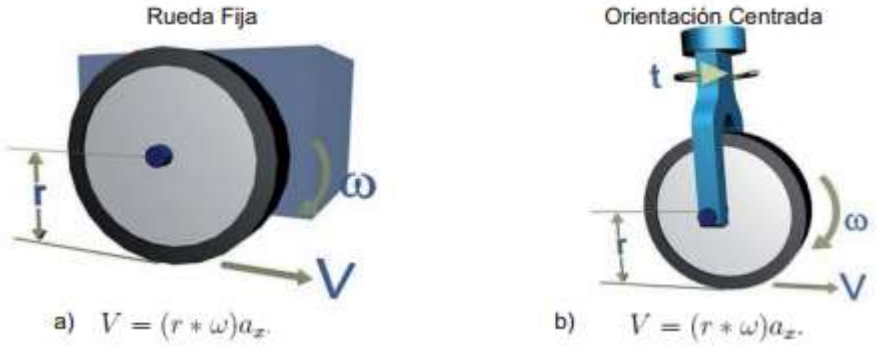

Figura 2.24: a) Rueda fija. b) Orientación centrada

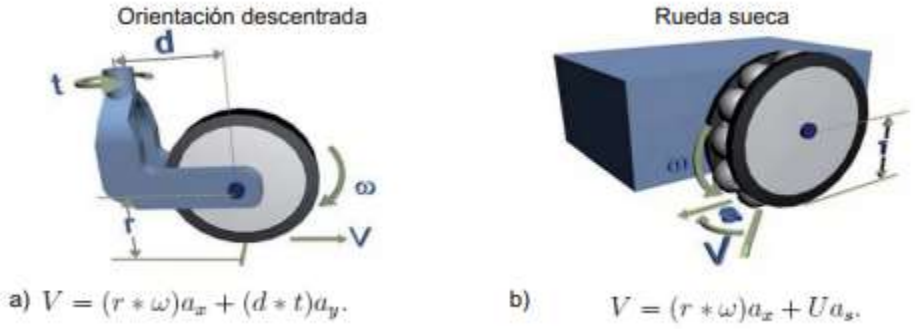

Figura 2.25: a) Orientación descentrada. b) Rueda sueca.

Centro instantáneo de rotación (ICR) o centro instantáneo de curvatura (ICC). Se define como el punto por el cual cruzan los ejes de todas las ruedas; es el punto alrededor del cual el robot gira en un instante determinado.

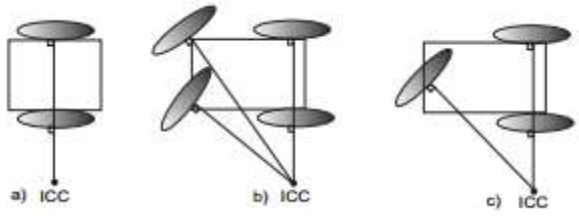

Figura 2.26: ICC en a) Configuración diferencial, b) Configuración ackerman, c) Configuración triciclo.

2.6.8 Sistema de control

La aplicación de una PC en el control de procesos supone un salto tecnológico enorme que se traduce en la implementación de nuevos sistemas de control en el entorno de la Industria. Desde el punto de vista de la aplicación de las teorías de control automático la computadora no está limitada a emular diferentes marcos de trabajo en la implementación de avanzados algoritmos de control. El controlador se encarga de almacenar y procesar la información de los diferentes componentes de los robots.

La definición de un sistema de control es la combinación de componentes que actúan juntos para realizar el control de un proceso. Este control se puede hacer de forma continua, es decir en todo momento o de forma discreta, en cada cierto tiempo. Si el sistema es continuo, el control se realiza con elementos continuos. En cambio, cuando el sistema es discreto el control se realiza con elementos digitales, como el ordenador, por lo que hay que digitalizar los valores antes de su procesamiento y volver a convertirlos tras el procesamiento.

Existen dos tipos de sistemas, sistema en lazo abierto y sistema en lazo cerrado.

1. Sistemas en lazo abierto.

Los sistemas de lazo abierto son aquellos que la salida no tiene influencia sobre la entrada. Estos toman la información acerca de las condiciones de operación que reciben de varios sensores y entonces usan esa información para determinar, ya sea por medios mecánicos o usando medios electrónicos programados, exactamente qué acción debe aplicarse para alcanzar la situación deseada. La precisión en la medición depende eternamente de que tan bien el sistema, de cualquier tipo que sea puede predecir las necesidades del motor basado en su "conocimiento" de las condiciones de operación.

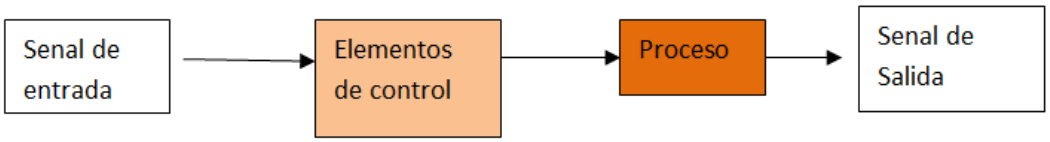

Figura 2.27: Sistema de control de lazo abierto

2. Sistema en lazo cerrado

Un sistema de lazo cerrado son aquellos que la salida influye sobre la señal de entrada. Es decir en un sistema de lazo cerrado o de retroalimentación, la información acerca de cualquier cosa que este siendo controlada es continuamente retro-alimentación al sistema como un dato de entrada. La operación de un termostato en un sistema de calefacción automático es un ejemplo de control de lazo cerrado.

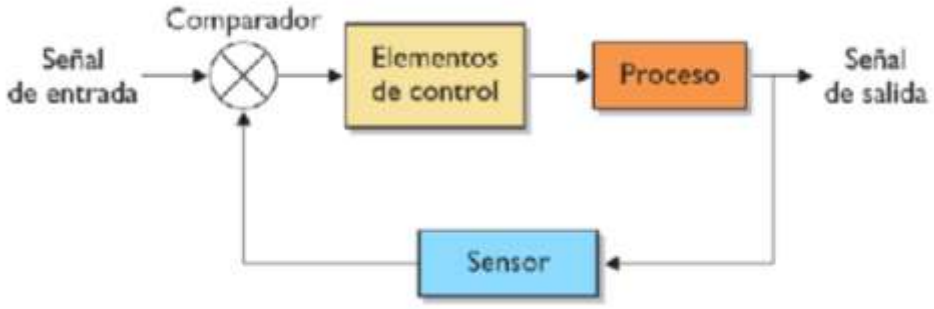

Figura 2.28: Sistema de control de lazo cerrado

El sensor o elemento de medición, es un dispositivo que convierte la variable de salida en otra variable manejable, tal como un desplazamiento, una presión, o un voltaje, que pueda usarse para comparar la salida con la señal de entrada de referencia. Este elemento está en la trayectoria de retroalimentación del sistema en lazo cerrado. El punto de ajuste del controlador debe convertirse en una entrada de referencia con las mismas unidades que la señal de retroalimentación del sensor o del elemento de medición.

Para el ejemplo del termostato, conforme baja la temperatura, el termostato siente el descenso y le indica al horno que añada calor. Tan pronto como la temperatura sube más allá de lo previsto, el termostato siente el resultado de su propia acción de control, el calor producido por el horno y le indica a este que corte el calor.

Un sistema de lazo abierto puede, por ejemplo, sentir al bajar la temperatura y simplemente encender el calor por un predeterminado lapso de tiempo: sin embargo el control de lazo cerrado es automático, la temperatura permanece relativamente constante, y el consumo de energía probablemente se reduce. De todas maneras, el resultado es un control mejor y más preciso.

2.6.9 Esquema general del sistema robot

En la figura 2.29 se muestra el esquema básico de un robot (Silva, 1984). En ella se identifican un sistema mecánico, actuadores, sensores y el sistema de control como elemento básico necesario para cerrar la cadena actuación-medidas-actuación

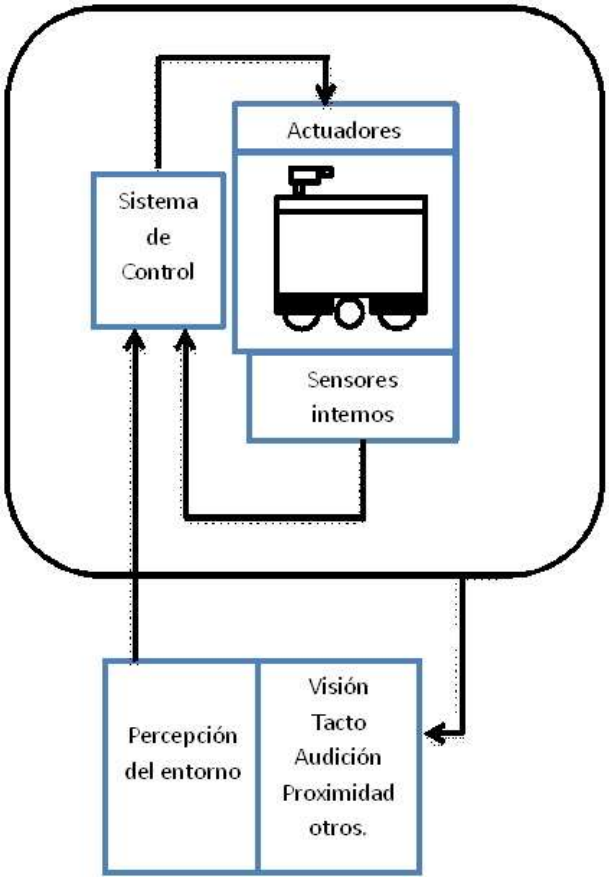

Figura 2.29: Robot y su Interacción con el entorno

En robótica se involucran funciones de control de movimientos, percepción y planificación. En un sentido amplio, el sistema de control involucra tanto bucles de realimentación de la información suministrada por los sensores internos, como del entorno. Los sensores internos miden el estado de la estructura mecánica y, en particular, giros o desplazamientos relativos entre articulaciones, velocidades, fuerzas y pares. Estos sensores permiten cerrar bucles de control de las articulaciones de la estructura mecánica. Los sensores externos permiten dotar de sentidos al robot. La información que suministran es utilizada por el sistema de percepción para aprehender la realidad del entorno. Los sistemas de percepción sensorial hacen posible que un robot pueda adaptar automáticamente su comportamiento en

función de las variaciones que se producen en su entorno, haciendo frente a situaciones imprevistas. Para ello el sistema de control del robot incorpora bucles de realimentación de dicha información sensorial con patrones de referencia.

El desarrollo de sistemas de precepción en Robótica surge a partir de los progresos tecnológicos en sensores tales como los de visión, tacto e, incluso, audición. Sin embargo, la percepción involucra no sólo la captación de la información sensorial, sino también su tratamiento e interpretación. Por tanto, es necesario realizar una abstracción a partir de un cierto conocimiento previo del entorno. Es claro que la complejidad de la percepción artificial depende de lo estructurado que esté dicho entorno.

Es importante la planificación que tiene como objetivo encontrar una trayectoria desde una posición inicial a una posición objetivo, sin colisiones, y minimizando un determinado índice. En el caso más simple, el problema se plantea en un entorno que se supone es conocido y estático. Se supone además que el robot es omnidireccional, que se mueve de una forma eficaz y así ser capaz de seguir el camino en forma perfecta.

2.6.10 Modelado del vehículo

Modelado

Con el propósito de comprender el comportamiento de los robots móviles en configuración diferencial se muestra el modelo cinemático. El cual es obtenido a partir de la teoría mecánica clásica, el cual nos considera las fuerzas que interactúan en el sistema, solamente las relaciones de movimiento.

Se considera al sistema con un robot móvil con dos ruedas en configuración diferencial. En la parte trasera del robot móvil se tiene una rueda de tipo castor que tiene como función el equilibrio del robot móvil. Además cuenta con dos llantas delanteras, izquierdas y derechas, idénticas y paralelas entre sí, cada una de ellas está montada en un servomotor que forma parte de la estructura general del robot. Esta configuración presenta restricciones no holonomicas, particularmente él no puede trasladarse lateralmente.

El modelo idealizado más simple de un robot de dos ruedas se deduce de la figura 2.30 a). La distancia del punto de contacto de una de las ruedas al punto medio del vehículo está dada por l y cada rueda es de radio r . Cada rueda esta accionada por un motor que proporciona giro independiente a cada rueda, esto es, w_d que es la velocidad en una rueda derecha y w_i , velocidad en la rueda izquierda.

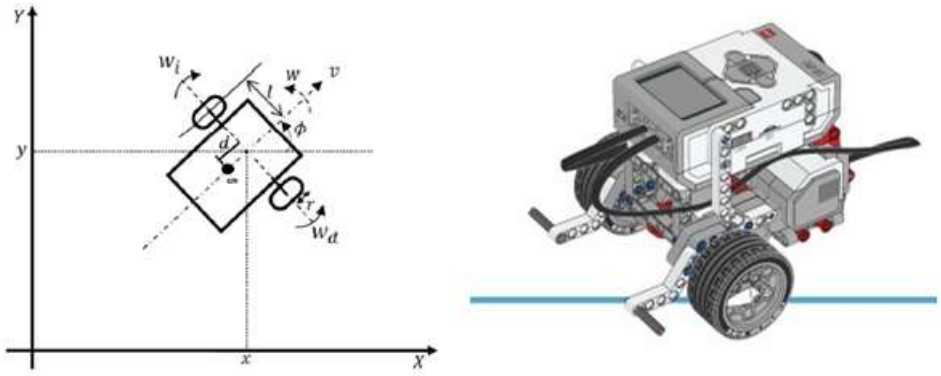

Figura 2.30: a) Modelo de un robot diferencial b) Robot Lego EV3

$v(t)$ Velocidad lineal del punto medio del vehículo

$w(t)$ Velocidad angular del punto medio del vehículo

$w_d(t)$ Velocidad angular de la rueda derecha

$w_i(t)$ Velocidad angular de la rueda izquierda

r es el radio de las ruedas

l distancia de separación entre el eje medio del vehículo y la rueda de tracción

El hecho de que cada llanta está montada en un servo motor diferente implica que el giro sea independiente. La configuración de este tipo de robots móviles posee una llanta trasera independiente sin actuador, denominada también rueda loca ya que esta es utilizada para asegurar la estabilidad al robot.

El modelo del robot se considera en que las ruedas del robot son suficientemente rígidas, no deformables y que giran sin desplazamiento en el espacio de trabajo, además se supone que el terreno en que se encuentra es liso y no tiene inclinaciones.

En este sistema el punto medio del eje del carro es de especial interés, el cual se describe por su ubicación en el plano xy , ya que todo el movimiento del sistema esta resumido en el comportamiento de este punto. Así pues, las ecuaciones que describen la cinemática del robot están dadas por:

$$\dot{x} = \frac{(w_d + w_i)r}{2}\cos\phi \qquad (2.1)$$

$$\dot{y} = \frac{(w_d + w_i)}{2}\sin\phi \qquad (2.2)$$

$$\dot{\phi} = \frac{(w_d - w_i)\,\mathrm{r}}{2l} \qquad (2.3)$$

La siguiente transformación de coordenadas de entrada invertible conduce a un más simple, y muy popular, modelo de cinemática del vehículo de dos ruedas, que es totalmente equivalente a la que ya se encuentra para el monociclo. Este modelo también se le llama como el modelo cinemático de un vehículo no holonomico, la transformación es:

$$\begin{bmatrix} v \\ w \end{bmatrix} = \Gamma \begin{bmatrix} w_d \\ w_i \end{bmatrix} = \begin{bmatrix} \frac{r}{2} & \frac{r}{2} \\ \frac{r}{2l} & \frac{r}{2l} \end{bmatrix} \begin{bmatrix} w_d \\ w_i \end{bmatrix} \Rightarrow \begin{bmatrix} v \\ w \end{bmatrix} = \frac{r}{2} \begin{bmatrix} 1 & 1 \\ \frac{1}{l} & -\frac{1}{l} \end{bmatrix} \begin{bmatrix} w_d \\ w_i \end{bmatrix}$$

Γ es una transformación no singular $\forall \quad r > 0, 1 > 0$, por lo tanto cualquier valor de las velocidades v y w pueden ser obtenidas mediante una adecuada selección de las variables w_i y w_d .

La velocidad lineal $\dot{v}$ está formada por las proyecciones de v hacia los ejes coordenados. La velocidad angular w es el cambio que ocurre en el ángulo ϕ con respecto al tiempo, y se detona por $\dot{\phi}$. De la figura 2.30 a), se observa que $\dot{x}$, $\dot{y}$ y $\dot{\phi}$, puede ser escitas como:

$$\dot{x} = v\cos(\phi) \qquad (2.4)$$

$$\dot{y} = vsen(\phi) \qquad (2.5)$$

$$\dot{\phi} = w \qquad (2.6)$$

El sistema es diferencialmente plano, con las salidas planas dadas por el par de coordenadas x y y que describen el comportamiento del punto medio del eje del carro.

2.6.11 Historia de los Robots Legos Mindstorms.

En 1998 LEGO introdujo al mercado los robots Mindstorms como un juguete educativo orientado a niños y adolescentes. Este desarrollo fue derivado del trabajo colaborativo entre la empresa LEGO y el laboratorio de Ingeniería del MIT. El primer robot fue llamado RCX (*Robotic Command eXplorers*), compuesto por un *brick* (ladrillo) con una capacidad de almacenamiento en la memoria RAM de *32Kb*. E este modelo de *brick.* Este modelo de *brick* incluye únicamente tres puertos de entrada para la interconexión de los motores. En esta versión del robot se integra una bocina, la cual permite reproducir audio precargado al ejecutar alguna acción. La actualización de programas en el robot RCX, desde una computadora personal, se realiza mediante una conexión por un puerto infrarrojo, el cual además permitía establecer una comunicación con otros robots RCX para ejecutar en conjunto algunas acciones (ver Figura 2.31 y Tabla 2.1).

En el 2006 LEGO lanza al mercado el segundo robot de la línea Mindstorms, llamado NXT, compuesto por un *brick* dotado de mayor capacidad de procesamiento que antecesor. Esta versión del NXT es reemplazada en 2009, iniciando la comercialización de la versión NXT *2.0.* Las versiones NXT *1.0* Y NXT *2.0* presentan similares características técnicas en el *brick.* La versión 2.0 del NXT presenta mejoras en el Software de desarrollo e incluye nuevos sensores de luz. En la Figura 1B se muestra en *brick* NXT y en la Tabla 2.1 se presenta un resumen de las características del robot NXT.

En julio del año 2013 se inició la comercialización del tercer modelo del robot LEGO Mindstorms, denominado EV3 (su nombre proviene de la palabra EVolution) (Mindstorms, 2013). El *brick* de este modelo cuenta con una capacidad de 16Mb de memoria flash y 64Mb de memoria RAM, así como una ranura lectora de tarjeta mini SD que soporta una capacidad de 32Gb, la cual permite tener una mayor capacidad de almacenamiento. En la Figura 1C se ilustra el *brick* EV3. Con respecto a dispositivos para establecer la comunicación entre el *brick* y la computadora personal, el robot EV3 cuenta con un puerto USB 2.0, Bluetooth, y con una tarjeta de red inalámbrica (WI-FI) conectada mediante el puerto USB, que permiten programar al robot EV3 de forma inalámbrica. El sistema operativo del robot EV3 está basado en Linux y tiene capacidad de interconexión con dispositivos móviles inteligentes mediante la compatibilidad del sistema operativo con IOS y Android. En el modelo EV3 se incluyen nuevos sensores, tales como un sensor infrarrojo digital (IR), un sensor giroscopio, un sensor de color digital con capacidad de detectar ausencia de color y hasta siete diferentes colores y un generador de señales infrarrojo, que puede usarse también como control remoto del robot EV3.

	RCX	NXT	EV3
Procesador	Hitachi H8/3292 10 – 16 MHz, 16 KB-ROM, 32 KB-RAM	Atmel 32-Bits – ARM7 48MHz, 256KB flash, 64 KB RAM	ARM 9 300 MHz, 16 MB-flash, 64 MB RAM
Puertos	3 puertos para motores 3 puertos para sensores	3 puertos para motores 4 puertos para sensores	4 puertos para motores 4 puertos para sensores
Comunicación	Puertos IR en el frente del *brick* utilizado para comunicación con el equipo de cómputo y con otro RCX	USB 12 Mbps Bluetooth	Bluetooh v2.1 Wi-Fi mediante el puerto USB
Almacenamiento extra	N/A	N/A	Ranura Micro SD
Comunicación	N/A	Android	Android /iOS
Pantalla	LCD	LCD monocromática, 100 x 64 pixeles	LCD monocromática 178 x 128 pixeles
Otras características	-	Co-procesador Atmel 8-Bit AVR 8MHz, 4KB flash, 512 Byte RAM	Auto-detección de dispositivos conectados al robot EV3. Control remoto (IR). Compatible con los motores y sensores del robot NXT *2.0*

Tabla 2.1. Principales características técnicas de los robot LEGO Mindstorms

A) **RCX** B) **NXT** C) **EV3**

Figura2.31: Diferentes tipos de brick (ladrillo) de robot LEGO Mindstorms (LEGO Grupo, 2012).

2.6.11.1 Características de los robots EV3

EV3 es un equipo de robótica que inicia con la idea de construir un robot mediante la unión de piezas y programación, de forma sencilla, a través del ensamblado de bloques con instrucciones concretas (LEGO Group, 2012). EV3 es el tercer robot de la línea LEGO Mindstorms; sus principales características técnicas son: está constituido por un mini-computador o *brick* (ladrillo) con una capacidad de almacenamiento de memoria 64MB e incluye cuatro puertos de entradas utilizados para la conexión de sensores y cuatro puertos de salidas usados para interconectar los motores (cada motor tiene embebido un sensor de rotación), los cuales habilitan la movilidad del robot. Estos motores se caracterizan por permitir movimientos precisos y controlados mediante el sensor de rotación, así como una sincronización de tiempo de ejecución de una acción (rotación) con otros motores. Los sensores proporcionan información del mundo o contexto donde se ubica el robot, esto es, información de los objetos externos que rodean al robot EV3. Esta información interpretada por una aplicación de software que permite detectar objetos, determinar su ruta de desplazamiento e identificar su ubicación o localización. Los sensores básicos del robot EV3 son: luz, sonido tacto, color, ultrasónico y de rotación.

Figura 2.32: Kit robotica LEGO Mindstorms EV3

2.6.12.1 EV3 Brick

Este es el componente central del robot, actúa como cerebro y está compuesto por:

- Un procesador principal de 32 bits, ARM 9 con memoria flash de 16 MB y 64 MB de memoria RAM. Trabaja a 64 MHz.
- Cuatro puertos de entrada que son usados para conectar los sensores, llamados puertos 1, 2, 3 y 4.
- Cuatro puertos de salida que son usados para conectar los motores, llamados puertos A, B, C y D.
- Un puerto USB High speed (480 Mbit/s) que se puede configurar para realizar una comunicación WI-Fiy un módulo interno Bluetooth v2.1.
- Ranura para memoria MICRO SD-CARD, máxima de 32 GB
- Pantalla LCD monocromática de 178 X 128 pixeles.
- Fuente de poder: 6 baterías AA o la batería recargable de LEGO para EV3.

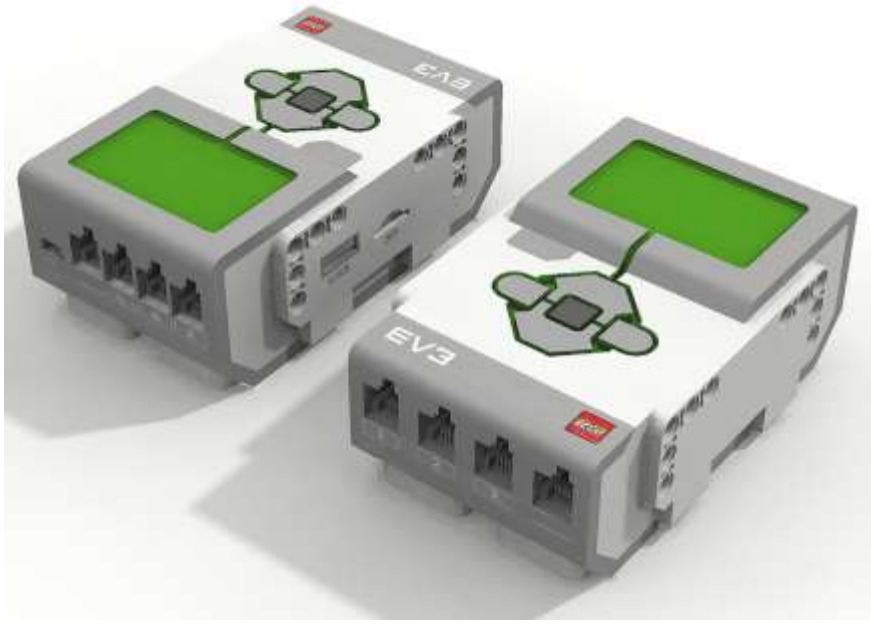

Figura 2.33: Brick EV3

2.6.12.2 Motores

En el Kit EV3 contiene con tres motores que permite al robot realizar tareas como desplazarse, levantar carga, tomar objetos o cualquier otra tarea que requiera energía.

Figura 2.34: Actuadores EV3

Los motores EV3 son servos, esto significa que su posición interna y estado puede ser controlado por una unidad externa, en este caso es controlado por el Brick EV3. Para ellos los motores cuentan con un sensor de rotación que lleva la cuenta de las rotaciones de los ejes del motor. Se puede rotar a una precisión de 1 grado.

La velocidad del motor depende del voltaje de la batería y de la carga. Sin carga, la máxima velocidad es de 100 veces el voltaje.

Articulo	**EV3 Motor Largo**	**EV3 Motor Medio**
Precision	Por 1 angulo por grado	Por 1 angulo por grado
Velocidad de rotacion	160 to 170rpm	240 to 250 rpm
Rotacion de torque	0.21 N*m (30oz*in)	0.08 N*m (11oz*in)
Peso	76 g	36 g
Auto-ID	applicable for EV3 Software	applicable for EV3 Software
Notas	Su es bueno movimiento y lento con	Su movimiento es bueno con

Articulo	EV3 Motor Largo	EV3 Motor Medio
	objetos pesados	elementos ligeros
Voltaje Maximo	9 VDC	9 VDC

Tabla 2.2: Características de actuadores EV3

2.6.12.3 Sensores

Los sensores se clasifican en dispositivos de entrada o de salida. Los dispositivos de entrada se sub-dividen en analógicos y digitales. Los sensores de color, luz, sonido y tacto son dispositivos de entrada analógicos, y el sensor ultrasónico se clasifica como dispositivos de entrada digital. Además existen sensores con mayor grado de especialización para el robot EV3, tales como posicionamiento, dirección, temperatura, y aceleración, entre otros.

Los sensores de permiten al robot EV3 interactuar con el mundo exterior. El kit EV3 tiene cuatro sensores diferentes, los cuales son:

- Sensor de Luz
  Su función es medir la cantidad de luz en una dirección particular. Cuenta con un emisores de Luz en una dirección determinada permitiendo obtener la intensidad de la luz reflejada por le objeto en esa dirección. El valor obtenido es un valor expresado en porcentaje.

  La cantidad de luz reflejada por una superficie depende de varios factores, principalmente del color, la textura y la distancia a la que se encuentra la superficie.
  La siguiente imagen muestra las características del sensor de luz.

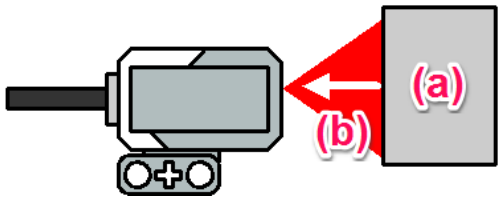

Figura 2.35: Funcion de sensor de luz

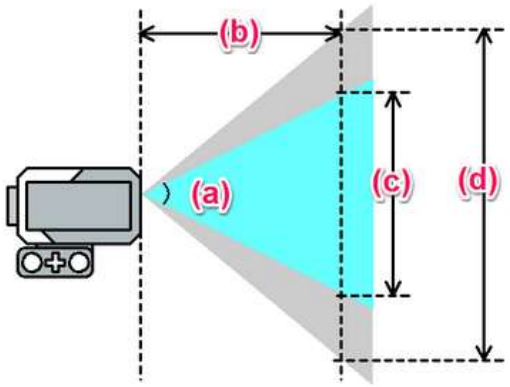

Figura 2.36: Rango de eificiencia del sensor de luz

(a) Rango medible eficiente (45 grados)
(b) Aproximación eficiente 53 mm
(c) Aproximación eficiente 54 mm
(d) Rango medible 88mm

La imagen de arriba muestra el rango de medición del robot LEGO EV3 sensor de color en el modo "color". (Valor medio) La zona de color azul claro es el rango de medición efectiva, aunque área gris capaz de causar falta de reconocimiento.

Especificaciones del sensor

measurement	reflected light of red light, ambient light intensity, color
detectable colors	8 colors (colorless, black, blue, green, yellow, red, white. Brown)
sampling rate	1,000 Hz
Distance	15 to 50mm
Auto-ID	available in EV3 Software

Tabla 2.3: Especificaciones del sensor de luz

Figura 2.37: Sensor de luz

- Sensor Ultrasónico

  Permite la detección de objetos y determina a que distancia se encuentran. El rango de detección esta entre 3cm y 250 cm. Tiene una precisión de +/- 1 cm, para tomar las medidas este sensor utiliza señales sonoras de alta frecuencia que se reflejan en un objeto. El sensor toma la señal de vuelta para la medición.

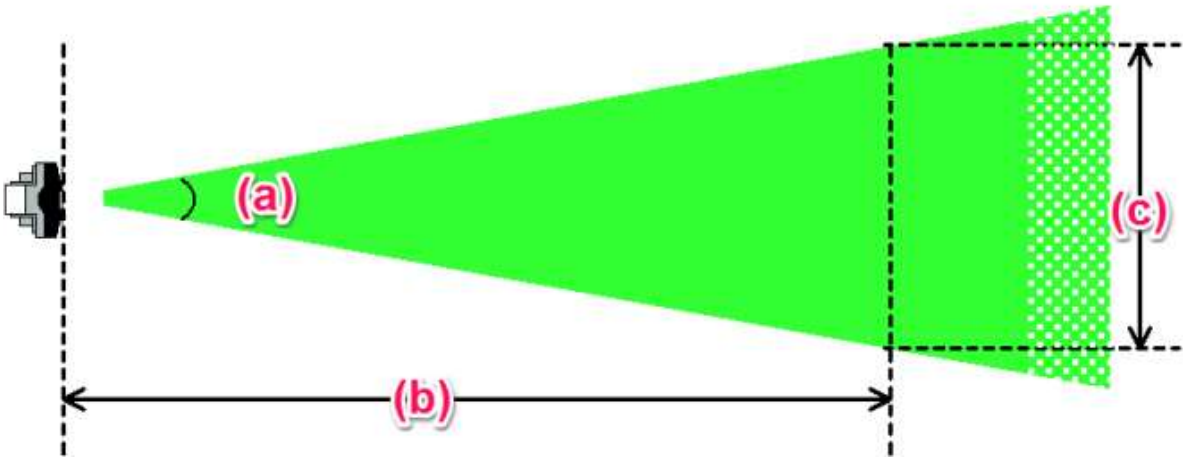

Figura 2.38: Rango de eficiencia del sensor de aproximidad

(a) En 20 grados
(b) Aproximadamente 60 cm
(c) Cerca de 22 cm

Se midió la distancia entre el sensor y el objeto (una caja de papel) de 60 cm antes del sensor. Medimos el ángulo inmediatamente después del sensor detectado cambio de 60 cm a una equivocada. Podemos saber que las medidas del sensor ultrasónico con el rango de ángulo de aproximadamente 20 grados, visto de frente.

- Sensor de Contacto

  Consiste de un pequeno boton interruptor (pushbutton) que permite detectar eventos de presion y leberacion del mismo. Este sensor puede ser utilizado para detener obstaculos muy cercanos del robot.

  El sensor de contacto detecta "on (1) " con el interruptor empujo y "off (0)" si el interruptor presionado. La longitud total del sensor es de 43 mm, sin el interruptor empujado y de 39 mm con el interruptor empujado, el cual fue medido con una pinza verniere.

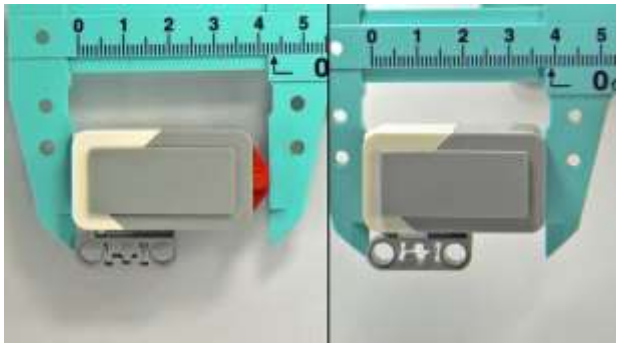

Figura 2.39: Sensor de contacto

Entorno de desarrollo

En esta sección se presenta los lenguajes de programación y entornos de desarrollo soportados por el robot EV3. Entre estos lenguajes de programación se pueden mencionar LEGO MINDSTORMS EV3, Brick Command Center, LEJOS, entre otros, pero solo describiremos los lenguajes más utilizados en los laboratorios de la universidad para desarrollar aplicaciones de software en robot EV3.

2.6.13 Programacion

Existen varias alternativas para el robot LEGO EV3 como la programación por cajas o árbol de decisiones que se proporcionaron el robot EV3, el cual tiene una funcionalidad bastante limitada. Para una programación de alto nivel se utiliza principalmente los siguientes lenguajes.

- RobotC: Entorno de desarrollo integrado que tiene como objetivo la programación del EV3, NXT o RCX bajo el lenguaje de programación C. Una de sus ventajas es que no necesita sustituir el firmware original del robot. Entre sus principales características tenemos una interfaz parecida a las VisualBasic, herramientas para utilizar la depuración de nuestras aplicaciones.

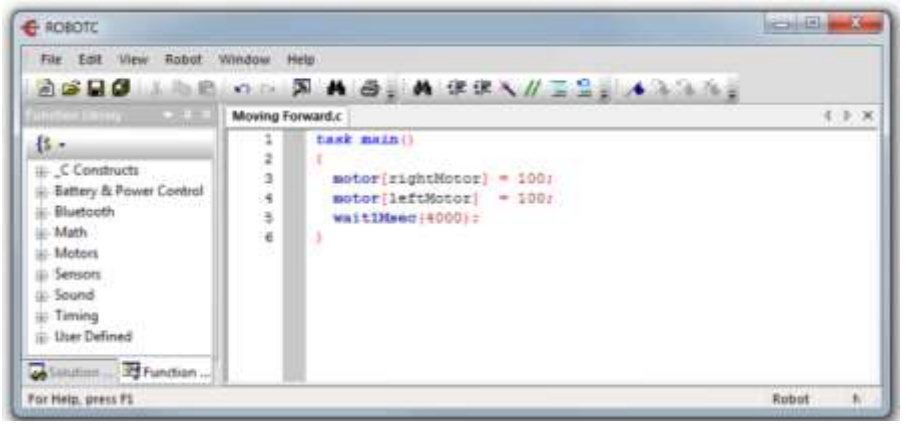

Figura 2.40: Interfaz RobotC

- LejOs: para la programación en este lenguaje es necesario sustituir el firmware original del brick por este. Incluye una máquina virtual de Java, la cual permite al LEGO Mindstorms ejecutar aplicaciones que se han implementado bajo este lenguaje. Incluye funciones para el control de prácticamente todos los sensores y actuadores. Este es el firmeware que se utiliza para el desarrollo de nuestro proyecto.

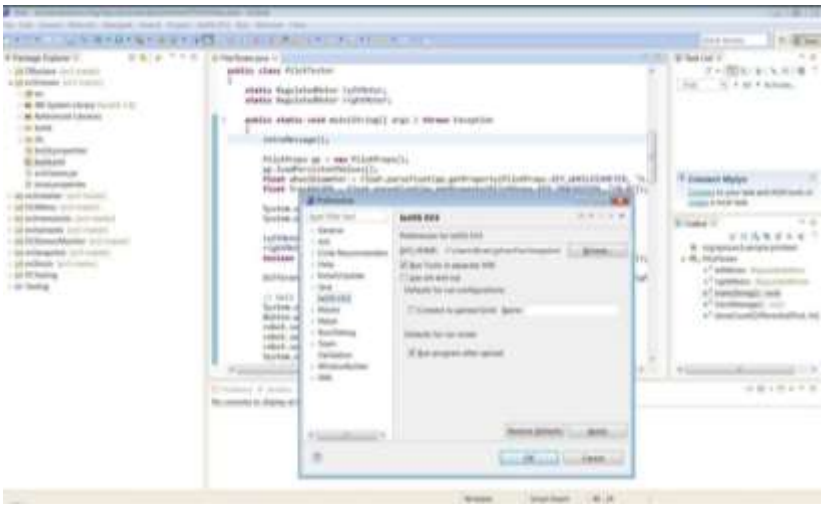

Figura 2.41: Interfaz Lejos

- BricxCC: Brick Command Center es un conoció IDE que soporta programación del RCX con NQC, C, C++, Pascal, Forth, y Java utilizando brickOS, pdForth y Lejos. Con BricxCC se pueden desarrollar programas en NBC y NXC. Tanto NBC como NCX utiliza estándar EV3. Este software está disponible en código abierto.

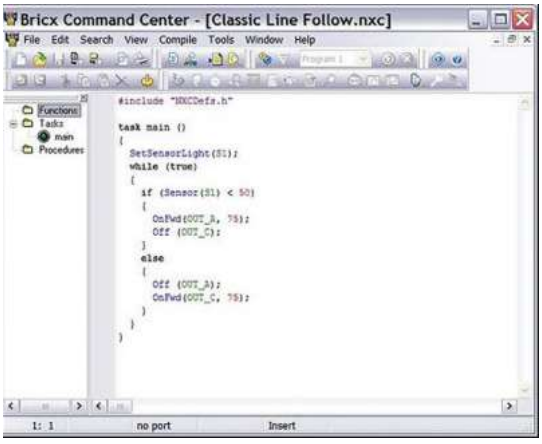

Figura 2.42: Interfaz Bricx C

- EV3: Este software de gran alcance, fácil aprendizaje, fácil de utilizar para el registro de la programación y da datos en LabVIEW de National Instruments. Este software es de diseño gráfico utilizando avances de diseño de software intuitivo. La programación de EV3 se realiza mediante arrastrar y soltar iconos en una línea con el fin de formar los comandos. La interfaz gráfica del lenguaje permite a los estudiantes construir programas sencillos.

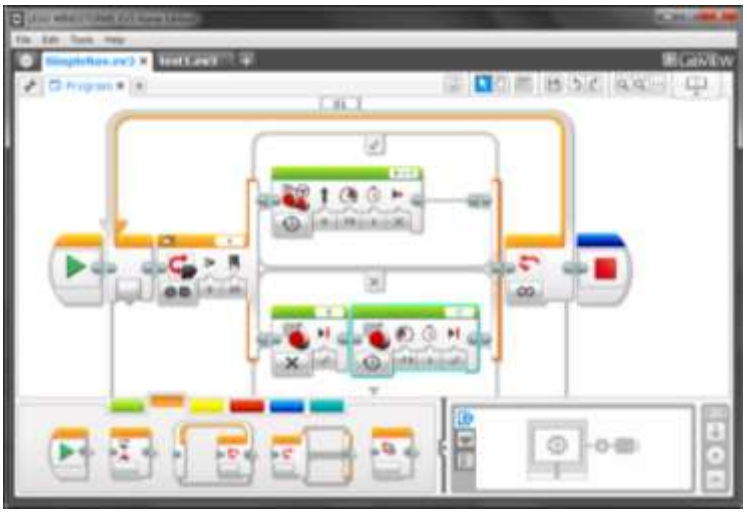

Figura 2.43: Interfaz EV3

Comunicación de Robots móviles LEGO MINDSTORMS EV3 con configuración maestro esclavo.

Los Robots LEGO MINDSTORMS EV3 cuentan con una comunicación vía bluetooth, que puede ser utilizado para comunicarse con un PC u otro LEGO EV3 a través de un protocolo dedicado. La tecnología de Bluetooth en el bloque EV3 y donde puede elegir ajustes específicos de privacidad y de Apple iOS. Además puedes conectarse a otros dispositivos Bluetooth, como por ejemplo otro bloque EV3

La comunicación entre robots EV3 se realizan vía inalámbrica a través de Bluetooth, en donde solo puede estar conectado un máximo de tres equipos, estableciéndose desde un inicio el estado de la conexión Bluetooth de cada robot, eso es, maestro o esclavo. Estas conexiones tienen que ser establecidas antes de ejecutar los programas de cada robot. La actualización de programas en el brick EV3 se puede realizar mediante el puerto USB, Bluetooth y Wi-Fi.

Capítulo 3

Comunicación

3.1 Introducción.

Comunicar, según la Real Academia Española, consiste en "hacer a otro participe de lo que uno tiene". La comunicación es uno de los actos más importantes en un sistema de multirobots, ya que es la llave para obtener todo el potencial de un trabajo colectivo. Los sistemas de multi-robots, al estar orientados a la resolución de problemas, necesitan de la capacidad de comunicación mediante la cual establece estrategias de cooperación.

Los modelos de comunicaciones se dividen en dos grandes grupos.

- Arquitectura de pizarra: Las pizarra s una zona de trabajo común donde se encuentra la información a compartir. Esta pizarra puede ser consultada por todos los robots, del mismo modo que todos pueden dejar información en ella. Puede existir robots con tareas de control específicas sobre la pizarra, así como varias pizarras. De este modo, no hay comunicación directa entre los robots, teniendo toda la información centralizada en cada una de las pizarras.

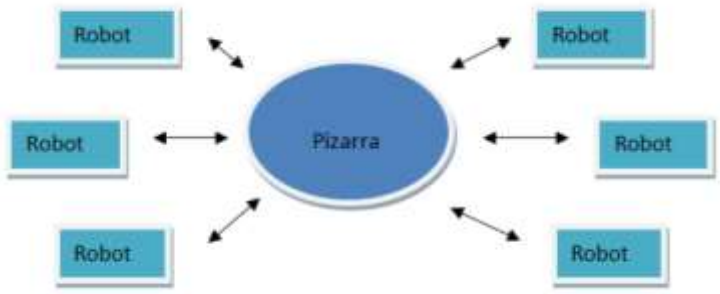

Figura 3.1: Arquitectura Pizarra

- Paso de mensaje: La comunicación se realiza del mismo modo que se realiza entre dos seres humanos, mediante el establecimiento e intercambio directo de mensajes entres dos robots (emisor y receptor). Como ventajas encontramos que es más flexible que la arquitectura de pizarra, además de que no es necesario tener toda la información centralizada.

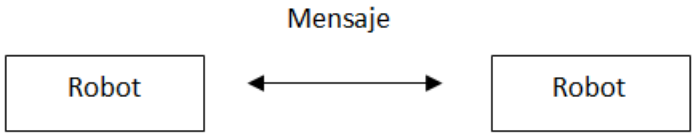

Figura 3.2: Arquitectura paso mensaje

El objetivo de este capítulo es obtener la comunicación entre dispositivos los cuales utilizaremos robots móviles, pero antes se necesita el conocimiento de distintas tecnologías y así poder escoger la adecuada para dicha comunicación. También se obtienen algunas herramientas que se utilizaran para la programación que se llevara a cabo los robots Lego EV3. Y por último se realiza un ejemplo de cómo se enviara el mensaje de robot móvil Lego EV3 maestro a robot móvil Lego EV3 esclavo.

3.2 Antecedente histórico.

Cuando hablamos de sincronía nos referimos al fenómeno mediante el cual, dos o más acontecimientos suceden al mismo tiempo, de manera pareja y equilibrada, simultáneamente. El termino sincronía proviene del griego "**syn**" (que significa juntos o en conjunto) y "**cronos**" (que significa tiempo) por lo cual, puede ser entendido como algo que sucede al mismo tiempo. La sincronía siempre nos habla de una situación en la cual, dos personas o dos elementos actúan de manera conjunta y pareja. De manera general,

definimos sincronía a la propiedad que adquiere un conjunto de "objetos dinámicos" (de una misma o diferentes especie) de manifestar un ritmo o comportamiento como un (generalmente distinto a los ritmos individuales de los objetos considerados), partiendo de ritmos o comportamientos individuales distintos, debido a la presencia de un medio acoplante (un medio físico de conexión) entre ellos, el cual, en la mayoría de los casos, es extremadamente débil.

La primera vez que se obtuvo una observación formal de un fenómeno de sincronía, se atribuye a Christian Huygens en 1673, durante algunos experimentos realizados en la mejora de relojes de péndulo. Esto ocurrió, cuando dos relojes suspendidos de la misma frecuencia y moviéndose en direcciones opuestas, debido al acoplamiento débil a través de la misma viga, es decir a las casi imperceptibles oscilaciones de la viga ocasionadas por el movimiento de ambos relojes. Huygens observo también, que los "tictacs" de ambos relojes se escuchaban al unisonó.

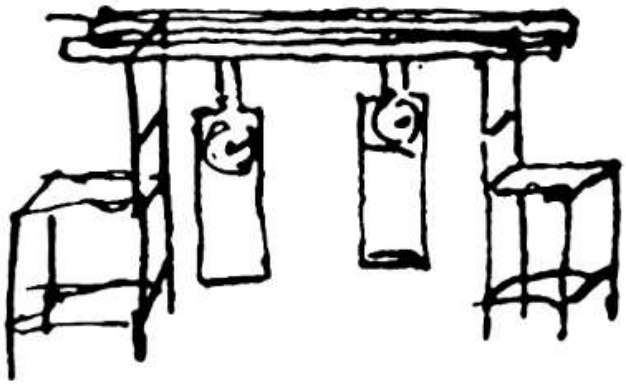

Figura 3.3 Dibujo original de Christian Huygen ilustrando su célebre experimento relojes de péndulo colocados en un soporte común {Pikovsky et al. 2001]

Huygens se preguntó la causa de dicho fenómeno, ya que al colocar ambos relojes sobre una pared (es decir, interrumpió el acoplamiento de las oscilaciones), aquel fenómeno cesaba. A partir de entonces, esta propiedad se viene observando (o apreciando con mucho interés) en sistemas de naturaleza nos proporciona a diario miles y miles de ejemplos de fenómenos colectivos en donde unidades dinámicas se organizan en un estado de sincronía. Es suficiente con observar la luna cada noche y darse cuenta de que nuestro satélite nos muestra siempre la misma cara, debido a que en el curso de los años.

En los últimos años han surgido casos particulares en los cuales no es evidente el alcance y mantenimiento de un estado sincrónico. Algunos de estos casos se presentan con los sistemas, caóticos, los cuales son sistemas dinámicos deterministas en los que la evolución de sus variables con determinadas condiciones iniciales es muy diferente a la evolución de las variables del mismo sistema ante un pequeño cambio sus condiciones iniciales.

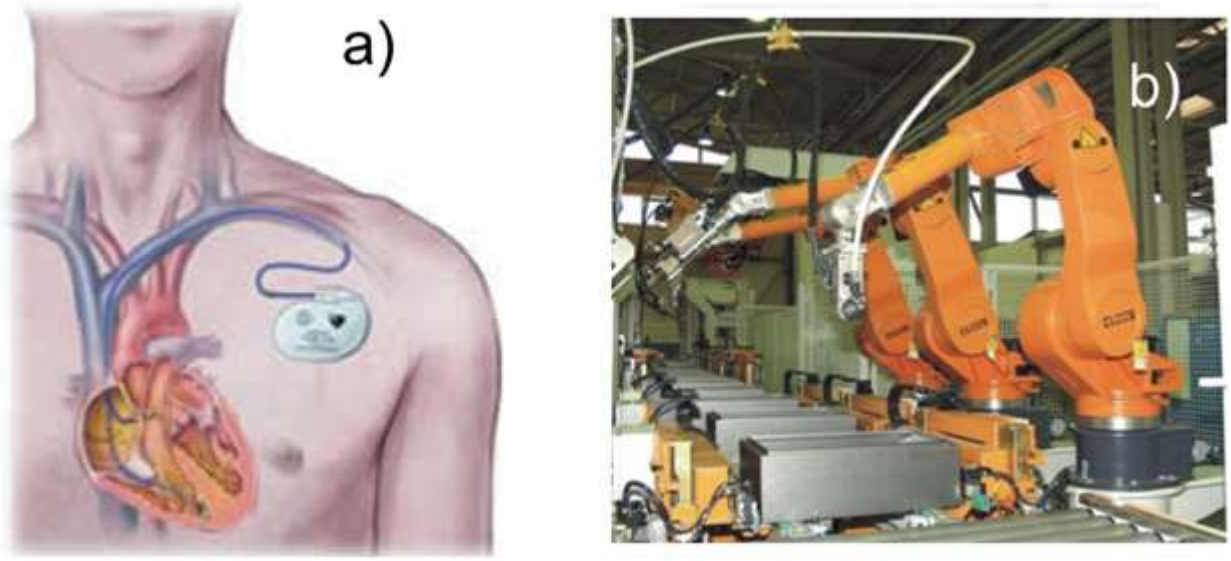

Figura 3.4: Aplicaciones de sincronización en sistemas no lineales en general: a) Marcapasos electrónicos para sincronizar el ritmo cardiaco, b) Robots manipuladores sincronizados en una línea de producción.

3.3 Sistemas biológicos sincronizados.

En las condiciones adecuadas, es posible observar procesos de sincronización en varias especies bilógicas. Entre ellos, una de los comportamientos mes espectaculares en la formación de grupos de individuos que acoplan sus movimientos y se desplazan de forma coordinada, a veces a grandes velocidades.

Figura 3.5: Ejemplos en dos y tres dimensiones de propiedades emergentes que se presentan en sistemas animados e inanimados: (a) Un tornado en el centro de Oklahoma (b) Aves al vuelo (c) NGC 4414, una galaxia espiral típica en la constelación Coma Berenices (d) Un sistema de baja presión saliendo de la costa sureste de Islandia (e) Hormigas australianas agrupándose para comer (f) Manada de Nus azules de barba blanca

Durante mucho tiempo y desde varios enfoques se han tratado de esclarecer las formas de interacción que permiten la sincronía de organismos biológicos y aunque se han logrado responder algunas preguntas relevantes, pero persiste la pregunta central: ¿Cómo se combinan los comportamientos individuales para generar sincronía colectiva? Una de las perspectivas biológicas de la agregación la plantea como una estrategia evolutiva que proporciona beneficios a la mayoría de los organismos del grupo a costa de algunas desventajas.

Por ejemplo: en aves migratorias formar grupos permite ahorrar energía en vuelos largos, eficiente la búsqueda de alimento, aumenta la probabilidad de sobrevivir al ataque de algún depredador y ofrece variabilidad reproductiva a los miembros del grupo. Sin embargo una alta concentración de individuos atrae depredadores, incrementa la competencia reproductiva y por alimento. El sistema existe en un intricado balance entre beneficios y desventajas que el entorno modifica constantemente. También es posible encontrar situaciones en las que organismos vivos se sincronizan aparentemente solo por diversión, por ejemplo, humanos en un estadio de futbol haciendo una ola, desfiles, etc.

Julia Parrish [Parrish et al 2002] propone que los agregados de organismos vivos se distinguen de los que forman partículas inanimadas porque los organismos para responder de varias formas distintas al mismo conjunto de estímulos. Por ejemplo, los peces que forman un cardumen pueden coordinar sus movimientos para adoptar formas muy variadas como reloj de arena, vacuolas, e incluso par comprensiones o expansiones repetitivas, comportamientos que difícilmente observaremos en sistemas inanimados. Adicionalmente, Parrish [Parrish et. A, 1999] menciono ciertos aspectos comunes que se presentan entre los agregados biológicos sincronizados.

1. La forma y el tamaño de los grupos dependen de los recursos disponibles, de la fisiología del organismo, de la actividad predominante (que puede o no determinar el medio) y de las limitaciones en las habilidades de precepción de los individuos.
2. La unidad de grupo puede afectarse cuando se agregan abruptamente nuevos individuos, llegando ocasionalmente a destruir la sincronía de grupos.
3. La forma, la estructura interna y el movimiento del grupo son propiedades emergentes auto-consistentes, esto es, se regulan por el mismo comportamiento del cumulo.
4. La tendencia a agruparse parecería la respuesta evolutiva a minimizar los costos que implica una forma de existencia gregaria.

3.4 Conceptos generales de comunicación

Siempre ha existido la necesidad de transmitir información entre personas, mediante señales, después mediante sonidos guturales, comunicándose siempre con las personas que se encontraban alrededor, pero después nació la necesidad de poder transmitir información hacia personas que se encontraban muy lejanas de la persona que emita el mensaje, para esto se inventaron las telecomunicaciones, entonces se tuvieron que desarrollar sistemas que pudieran procesar las señales de información de manera adecuada.

Dentro del campo de las comunicaciones en la robótica móvil, cada fabricante ofrece unas posibilidades de comunicación según las prestaciones y las capacidades de su producto.

Un sistema de comunicación se compone de un transmisor, un receptor y un medio de comunicación como el que se muestra en la figura. El medio a través del cual se lleva a cabo la transmisión de información puede ser la atmosfera utilizando señales de radiofrecuencia, los alambres conductores mediante el uso de señales de voltaje o corriente, la fibra óptica trasmitir mediante el uso de señales luminosas. En el sistema de la figura 3.2 la estación 1 solamente ocurre en una dirección, pero a veces es necesario que tanto la estación 1 como la estación 2 puedan transmitir y recibir información.

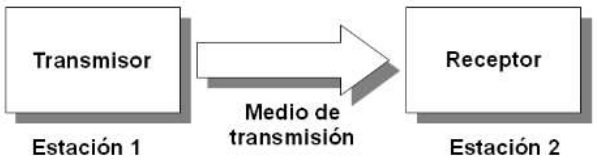

Figura 3.6. Sistema de comunicación

3.4.1. Modos de transmisión.

Los sistemas de comunicación pueden clasificarse de acuerdo a su modo de transmisión, lo cual implica tomar en cuenta, el número de líneas de transmisión, la dirección en que se da el flujo de la información y los instantes en que este flujo ocurre. Entonces tenemos tres modos posibles: *simplex, half duplex, full dúplex*.

Simplex

En este modo solo hay una línea de comunicación por lo que las transmisiones solamente se hacen en una dirección. Una estación solamente puede ser transmisora o receptora pero no ambos. Como se muestra en la figura 3, en donde solo hay flujo de información de la estación 1 a la estación 2.

Half Duplex

Este modo existe una línea para la comunicación, pero a diferencia del modo simplex aquí se puede realizar transmisiones de información en ambas direcciones, pero no al mismo tiempo, como se muestra en la figura 3.2

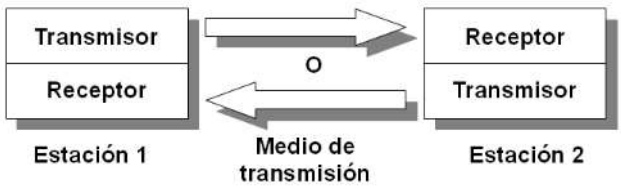

Figure 3.7. Sistema Half Duplex.

Full Duplex

En este modo existen dos líneas de comunicación, una que se ocupa para transmitir y otra para recibir información, teniendo la capacidad de poder realizar las dos operaciones simultáneamente como lo muestra en la figura 3.3

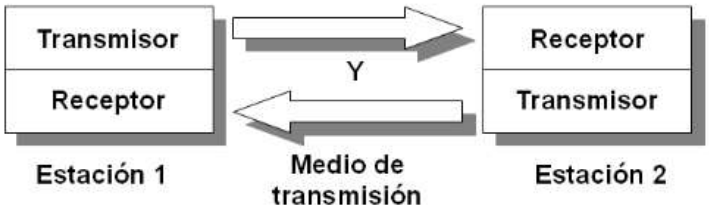

Figure 3.8. Sistema Full Duplex.

3.4.2 Hipótesis del medio

En la hipótesis del medio se describen las características más relevantes respecto al medio de transmisión por donde viajara la información, el modo de operación del canal de comunicación, como la tecnología que se utilizara para realizar la comunicación, etc. También los factores externos que pueden intervenir en la ejecución.

3.4.3 Tipos de mensajes

Existen diferentes tipos de métodos como el stop and wait. Este método tiene los siguientes mensajes: MENSAJE que representa la información que se va a transmitir, ACK para los acuses de recibido positivo, y NACK para los acuses de recibido negativo.

Estos mensajes pueden ser expresados como un conjunto finito de la siguiente manera:

V = {MENSAJE, ACK, NACK} (3.1)

3.4.4 Formato de encapsulación de tipos de mensajes

El formato de encapsulado de un mensaje clásico consta de tres campos como se muestra en figura, y los cuales son:

- **Cabecera:** Es un conjunto de campos de longitud fija que contiene información de control denominada PCI (Protocol Control Information). La información PCI por lo general contiene bits para definir la función del mensaje, el número de secuencia del mismo e información acerca del estado de los equipos y de la conexión.
- **Información:** Este campo contiene los datos de un protocolo de mayor jerarquía.
- **Cola:** Es un conjunto de campos de longitud fija que contiene información de control al final del mensaje y por lo general corresponde a un FCS para la detección de errores. Este campo depende de la información PCI y de los datos presentes en el campo de información.

Figura 3.9: Fotmato de mensaje utilizando cabeceras y colas

3.4.5 Modelos y arquitectura para el diseño de protocolos de comunicación

Para el diseño de protocolo de comunicación proporciona un marco teórico y tecnológico, implementar y administrar dichos protocolos. Se basan en estructuras por capas, lo que permite dividir las distintas tareas que realizara el protocolo de comunicación en módulos. Cada módulo tendrá la capacidad de realizar una sub tarea y de interactuar con otros módulos.

3.4.5.1 Modelo OSI

Creado en 1970, la Organización Internacional de Estandarización (ISO, International Standards Osganization) que es una organización multinacional dedicado a establecer acuerdos mundiales sobre estándares internacionales. Un estándar ISO que cubre todos los aspectos de las redes de comunicación es el modelo de interconexión de Sistemas Abiertos

(OSI, Open System Interconnection). El objetivo del modelo OSI es permitir la comunicación entre sistemas distintos sin necesidad de cambiar la lógica del hardware o el software subyacente.

El modelo OSI divide el proceso de comunicación en varias funciones, las mismas que se encuentran distribuidas en siete capas como se observa en la figura 3.9.

Figura 3.10: El modelo de referencias OSI

Las tareas que realiza cada una de las capas del Modelo OSI son las siguientes:

- **Capa física:** "Se encarga de la transmisión de cadenas de bits no estructurados sobre el medio físico; está relacionada con las características mecánicas, eléctricas, funcionales y de procedimiento para acceder al medio físico.
- **Capa de enlace de datos:** Proporciona un servicio de trasferencia de datos confiable a través del enlace físico; envía bloques de datos (tramas) llevando a cabo la sincronización, el control de error y el flujo.
- **Capa de red:** proporciona intendencia a los niveles superiores respecto a las técnicas de conmutación y de transmisión utilizadas para conectar los sistemas; es responsable del establecimiento, mantenimiento y cierre de las conexiones.
- **Capa transporte:** Proporciona una trasferencia y confiable de datos entre los puntos finales; además, proporciona procedimientos de recuperación de errores y control de flujo entre el origen y el destino.
- **Capa sesión:** Proporciona el control de la comunicación entre las aplicaciones; establece, gestión y cierra las conexiones (sesiones) entre las aplicaciones operadoras.
- **Capa presentación:** proporciona a los procesos de la aplicación independencia respecto a las diferencias en la representación de los datos.
- **Capa aplicación:** Proporciona el acceso al entorno OSI para los usuarios y, también proporciona servicios de información distribuida.

3.4.5.2 La arquitectura TCP/IP

La arquitectura TCP/IP (Transmission Control Protocol, Internet Protocol) no se originó en base al modelo OSI, por tal razón no existe un acuerdo universal para describir las funciones de las capas que conforman esta arquitectura en base a dicho modelo.

TCP/IP está conformada por cuatro capas. Cuando se envían datos, cada capa trata a la información que recibe de la capa superior como datos, añade su cabecera y después la pasa a la capa inferior. Cuando se reciben datos, el procedimiento inverso se lleva acabo.

Las cuatro capas que conforman al modelo TCP/IP, y sus respectivas funciones son:

- **Capa aplicación:** esta capa agrupa las funciones de las capas aplicaciones, presentación y sesión correspondientes al modelo OSI. En esta capa se utiliza sockets y puertos para permitir la comunicación entre aplicaciones. Generalmente las aplicaciones están asociadas con uno o más puertos.
- **Capa Transporte:** en la arquitectura TCP/IP, existen dos protocolos a nivel de la capa de transporte. El transporte TCP que garantiza la transmisión de la información, mientras que el protocolo UDP (User Datagram Protocol) transporta datagramas de un extremo hacia otro sin verificar la fiabilidad de los mismos. Ambos protocolos son usados para diferentes tipos de aplicaciones.
- **Capa Red:** El protocolo internet (IP, Internet Protocol) se utiliza en esta capa para ofrecer el servicio de encaminamiento a través de varias redes. Este protocolo se implementa tanto en los sistemas finales como en los routers intermedios
- **Capa de acceso a la red:** las funciones de las capas enlace de datos y física son agrupadas en esta capa. Existe documentación donde se describe como IP puede utilizar protocolos de capa enlace de datos ya existentes como Ethernet.

En la figura 3.10 muestra las capas que conforman la arquitectura TCP/IP.

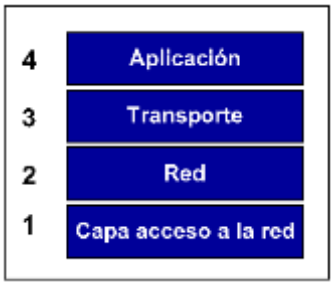

Figura 3.11: Arquitectura TCP/IP

3.5 Comunicaciones inalámbricas

A pesar de que en algunos casos se hace uso de conexiones por cable (normalmente) para la transferencia de los programas a los robots. Todos los robots utilizados para la realización de la presente tesis disponen de comunicación inalámbrica, lo que les brinda cierta autonomía y una gran ventaja de no necesitar cables.

Es importante conocer a fondo los protocolos de comunicación para poder aprovechar al máximo los tiempos de transmisión y la capacidad del canal, para no limitarse al uso de metodologías de alto nivel. Ahora se describen protocolos de comunicación inalámbrica más usuales en la robótica móvil.

3.5.1 Wi-Fi

WLAN (Wireless Local Area Network, en inglés) es un sistema de comunicación de datos inalámbrico flexible, muy utilizado como alternativa a las redes LAN cableadas o como extensión de estas. Utiliza tecnología de radiofrecuencia que permite mayor movilidad a los usuarios al minimizar las conexiones cableadas. La WLAN va adquiriendo importancia n muchos campos, como almacenes o para manufactura, en los que se transmite la información en tiempo real a un terminal central. También son muy populares en los hogares para compartir el acceso a internet entre varias computadoras.

3.5.1.1 Inicios Wi-Fi

Los pioneros en el uso de redes inalámbricas han sido los radioaficionados mediante sus emisoras, que ofrecen una velocidad de 9600 bps. Pero si hablamos propiamente de redes inalámbricos debemos remontarnos el año 1997, en el que organismo regulador IEEE (Institute of Electonics Engineer) publico el estándar 802.11 (802 hace referencia al grupo de documentos que describen las características de la LAN) dedicado a redes LAN inalámbricas.

Dentro del mismo campo y anteriormente, en el año 1995, tenemos, la aparición de Bluetooth, una tecnología de Ericsson con el objetivo de conectar mediante ondas de radio los teléfonos móviles con diversos accesorios. Al poco tiempo se generó un grupo de estudio formado por fabricantes que estaban interesados en esta tecnología para aplicarla a otros dispositivos, como PDAs, terminales móviles o incluso electrodomésticos

Pero el verdadero desarrollo de este tipo de redes surgió a partir de que la FCC, el organismo americano encargado de regular las emisiones radioeléctricas, aprobó el uso civil de la tecnología de transmisiones de espectro disperso (SS o sprend spectrum, en inglés), pese a que un principio lo prohibió por el uso ampliado del espectro. Dicha tecnología ya se usaba en ámbitos militares desde la Segunda Guerra Mundial debido a sus

extraordinarias características en cuanto a la dificultad de su detección y su tolerancia a interferencias eternas.

A pesar de que esta tecnología ya tiene una antigüedad de más de diez años, no ha sido hasta ahora cuando este tipo de redes se ha desarrollado eficazmente debido a la disminución de precios de los dispositivos que la integran. En la actualidad cada vez más se encuentran equipos que pueden competir en precios con los modelos para redes cableadas.

3.5.1.2 Como trabaja WLAN

Se utiliza ondas de radio para llevar la información de un punto a otro sin necesidad de un físico guiado. Al hablar de ondas de radio nos referimos normalmente a portadoras de radio, sobre las que va la información, ya que realizan la función de llevar la energía a un receptor remoto. Los datos a transmitir se superponen a la portadora de radio y de modo pueden ser extraídos exactamente en el receptor fina.

A este proceso se le llama modulación de la portadora por la información que estas siendo transmitida. Si las ondas son trasmitidas a distintas frecuencias de radios, varias portadoras pueden existir en igual tiempo y espacio sin interferir entre ellas. Para extraer los datos el receptor se sitúa en una determinada frecuencia portadora, ignorando el resto. En una configuración típica de LAN sin cables los puntos de acceso (transceiver) conectan la red cableada de un lugar fijo mediante cableado normalizado. El punto de acceso recibe la información, la almacena y la transmite entre la WLAN y la LAN cableada. Un único punto de acceso puede soportar un pequeño grupo de usuarios y puede funcionar en un rango de al menos treinta metros y hasta varios centímetros. El punto de acceso (o la antena conectada al punto de acceso) es normalmente colocado en lo alto pero podrá colocarse en cualquier lugar en que se obtenga la cobertura de radio deseada. El usuario final accede a la red WLAN a través de adaptadores. Estos proporcionan una interfaz entre el sistema de operación de red del cliente y las ondas, mediante una antena.

3.4.1.3 Seguridad

Uno de los problemas principales de este tipo de redes es precisamente la seguridad ya que cualquier persona con una terminal inalámbrica podría comunicarse con un punto de acceso privado si no se dispone de las medidas de seguridad adecuadas. Ducha medidas van encimadas en dos sentidos: por una parte está cifrado de los datos que se transmiten y en otro plano, pero igualmente importante, se considera la autentificación entre los diversos usuarios de la red, En el caso de cifrado se están realizando diversas investigaciones como el encriptado de mensajes utilizando sistemas caóticos para darle más seguridad a los datos que se envían.

3.5.2 Otras alternativas

A pesar que durante el desarrollo de la tesis se han hecho uso de los protocolos Wifi y Bluetooth, hay que tener en cuenta que también existen otros protocolos en el mercado que tiene características que pueden ser utilizadas en comunicación entre dispositivos o en nuestro caso en robots móviles.

3.5.3 Zigbee

Zigbee es el nombre de la especificación de un conjunto de protocolos de alto nivel de comunicación inalámbrica para su utilización con radios digitales de bajo consumo, basada en el estándar IEEE 802.15.4 de redes inalámbricas de área personal (wireless personal area network, WPAN). Su objetivo son las aplicaciones que requieren comunicaciones seguras con baja tasa de transmisión de datos y maximización de vida útil de sus baterías.

La razón de ello son diversas características que lo diferencian de otras tecnologías:

- Su bajo consumo
- Su topología de red de malla
- Su fácil integración (Se puede fabricar nodos con muy poca electrónica).

3.5.4.1 Generalidades de Zigbee

Zigbee utiliza la banda ISM para usos industriales, científicos y médicos; en concreto, 868 MHz en Europa, 915 MHz en Estados Unidos y 2.4 GHz en todo el mundo. Sin embargo, a la hora de diseñar dispositivos, las empresas optaran prácticamente siempre por la banda de 2.4 GHz, por ser libre en todo el mundo. El desarrollo de la tecnología se centra en la sencillez y el bajo coste más que otras redes inalámbricas semejantes de la familia WPAN, como por ejemplo Bluetooth. El nodo ZigBee más completo requiere en teoría cerca de 10% del hardware de un nodo Bluetooth o Wi-Fi típico; es cifra baja al 2% para los nodos más sencillos. No obstante, el tamaño de código en si es bastantemente mayor y se acerca al 50% del tamaño del de Bluetooth. Se anuncian dispositivos hasta 128 kBde almacenamiento.

3.5.4.2 Uso del protocolo Zigbee

Los protocolos ZigBee están definidos para un uso en aplicaciones embebidas con requerimientos muy bajos de transmisión de datos y consumo energético. Se pretende su uso en aplicaciones de propósito general con características auto organizativas y bajo costo (redes mallas en concreto). Puede utilizarse para realizar control industrial, albergar sensores empotrados, recolectar datos médicos, ejercer labores de detección de humo o intrusos o donomica, entre otras aplicaciones. La red en su conjunto utilizara una cantidad

muy pequeña de energía de forma que cada dispositivo individual pueda tener una autonomía de hasta 5 antes de necesitar un recambio en su sistema de alimentación.

3.5.4 Bluetooth

Es una Tecnología de res de área personal inalámbrica (abreviada WPAN), una tecnología de inalámbrica de corto alcance, que se utiliza para conectar dispositivos entres si sin una conexión por cable. A diferencia de la tecnología IrDa (que utiliza una conexión infrarrojo), los dispositivos de Bluetooth no necesitan una línea de visualización directa para comunicarse. Esto hace que su uso se a más flexible y permite la comunicación entre habitaciones en espacios pequeños.

El objetivo de Bluetooth es trasmitir voz o datos entre equipos con circuitos de radio de bajo costo, a través de un rango aproximado de entre diez y cien metros, utilizando poca energía.

La tecnología Bluetooth se diseñó principalmente para conectar dispositivos (como impresoras, teléfonos móviles, artículos para el hogar, auriculares inalámbricos, ratón, teclados, etc.) equipos o PDA (Asistente personal digital) entre sí, sin utilizar una conexión por cable. Bluetooth también se utiliza cada vez más en teléfonos móviles, los cuales les permite comunicarse con equipos y se ha extendido especialmente a los accesorios manos libres.

La tecnología Bluetooth originalmente fue desarrollada por Ericsson en 1994, En febrero de 1998, se formó un grupo llamado Bluetooth Special Interest Group (Bluetooth, SIG) con más de 200 compañías, dentro de las cuales se encontraban Agere, Ericcssion, IBM, Intel, Microsoft, Motorola, Nokia y Toshiba. Su objetivo era desarrollar las especificaciones para Bluetooth 1.0 que se publicaron en Julio de 1999.

El nombre de Bluetooth proviene del rey danés Harald l (910-986), cuyo apoco era Harald l Blatand que significaba “hombre de tez oscura” (en inglés “blue-toothed), quien logro la unificación de Suecia y Noruega, e introdujo el Cristianismo en Escandinavia.

3.6 Características

El funcionamiento se encuentra en ir cambiando de frecuencia y mantenerse en cada una un “slot” de tiempo para después volver a saltar otra diferente. Es conocido entre salto y salto en tiempo transcurre es muy pequeño, concretamente unos 650 microsegundos con lo que al cabo de 1 segundo se puede haber cambiado 1600 veces de frecuencia.

El estándar de Bluetooth define 3 clases de transmisores, cuyo alcance varía en función de su potencia radiada:

Clase	Potencia (perdida de señal)	Alcance
L	100 mW (20 dBm)	100 metros
Ll	2, 5 mW (4 dBm)	15-20 metros
Lll	1 mW (0 dBm)	10 metros

Tabla 3.1: Clases de transmisores

A diferencia de la tecnología IrDa, la principal competencia, que utiliza radiación de luz para enviar datos, Bluetooth utiliza ondas de radio (ne la banda de frecuencia de 2.4 GHz) para comunicarse. Como consecuencia, los dispositivos Bluetooth no necesitan estar visualmente comunicados para intercambiar datos. Esto significa que los dos dispositivos pueden comunicarse incluso si se encuentran separados por un muro, como también pueden detectarse entres si sin la participación del usuario, siempre y cuando uno se encuentre dentro del alcance del otro.

3.6.1.1 Como funciona

El estándar Bluetooth, del mismo modo que WIFI, utiliza la técnica FHSS (Frequency Hoppinf Spread Spectrum en español Espectro ensanchado por saltos de frecuencia). Que en dividir la banda de frecuencia de 2.402 – 2.480 GHz en 79 canales (denominados saltos) de 1 MHz de ancho cada uno y, después, transmitir la señal utilizando una secuencia de canales que sea conocida tanto para la estación emisora como para la receptora.

Por lo tanto, al cambiar de canales con una frecuencia de 1600 veces por segundo, el estándar Bluetooth puede enviar la interferencia con otras señales de radio.

3.6.1.2 Principio de comunicación

El estándar Bluetooth se basa en el modo de operación maestro/esclavo. El término “piconet” se utiliza para hacer referencia a la red formada por un dispositivo y todos los dispositivos que se encuentren dentro de su rango. Pueden coexistir hasta 10 piconets dentro de una sola área de cobertura. Un dispositivo maestro se puede conectar simultáneamente con hasta 7 dispositivos esclavos activos (255 cuando se encuentran en modo espera). Los dispositivos en una piconet poseen una dirección lógica de 3 bits, para un máximo de 8 dispositivos. Los dispositivos que se encuentren en el modo espera se sincronizaran, pero no tienen su propia dirección física en la piconet.

En realidad, en un momento determinado, el dispositivo maestro solo puede conectarse con un solo esclavo al mismo tiempo. Por lo tanto. Rápidamente cambia de esclavo para que parezca que se está conectando simultáneamente con todos los dispositivos esclavos.

Bluetooth permite que dos piconets puedan conectarse entre sí para formar una red más amplia denominada “scatternet, utiliza ciertos dispositivos que actúan como puentes entre los piconets.

3.6.1.3 Como se establece las conexiones

El establecimiento de una conexión entre dispositivos Bluetooth sigue un procedimiento relativamente complicado para garantizar un cierto grado de seguridad, como el siguiente:

- **Modo pasivo:** Durante el uso normal, un dispositivo funciona en modo pasivo, es decir que está escuchando la red.
- **Solicitud (Búsqueda de punto de acceso):** El establecimiento de una conexión comienza con una fase denominada solicitud, durante la cual el dispositivo maestro envía una solicitud a todos los dispositivos que encuentra dentro de su rango, denominados punto de acceso. Todos los dispositivos que reciben la solicitud responden con su dirección.
- **Paginación (Sincronización con los puntos de acceso):** El dispositivo maestro elige una dirección y se sincroniza con el punto de acceso mediante una técnica denominada paginación, que principalmente consiste en la sincronización de su reloj y frecuencia con el punto de acceso.
- **Descubrimiento de servicio del punto de acceso):** De esta manera se establece un enlace con el punto de acceso que le permite al dispositivo maestro ingresar a una fase de descubrimiento del servicio del punto de acceso, mediante un protocolo denominado SDP (Service Discovery Protocol, en español Protocolo de descubrimiento de servicios).
- **Creación de un canal con el punto de acceso:** Cuando la fase de descubrimiento del servicio ha finalizado, el dispositivo maestro está preparado para crear un canal de comunicación con el punto de acceso, mediante el protocolo L2CAP. Según cuales sean las necesarias del servicio, se puede establecer un canal adicional, denominado RFCOMM que funciona por el canal L2CAP, para proporcionar un puerto serial virtual, De hecho, algunas aplicaciones se han diseñado para que puedan conectarse a un puerto estándar, independientemente del hardware utilizado. Por ejemplo, se han diseñado ciertos programas de navegación en carretera para la conexión con cualquier dispositivo GPS Bluetooth (GPS, Global Positionning System [sistema de posicionamiento global], un sistema de localización geográfica por satélite para encontrar las coordenadas geográficas de un dispositivo móvil o de un vehículo).
- **Emparejamiento mediante el PIN (Seguridad):** El punto de acceso puede incluir un mecanismo de seguridad denominado emparejamiento, que restringe el acceso solo a los usuarios autorizados para brindarle a la piconet cierto grado de protección. El emparejamiento se realiza con una clave cifrada comúnmente conocida como “PIN” (PIN significa Personal Information Number [Número de identificación personal]). Para esto, el punto de acceso le envía una solicitud de emparejamiento al dispositivo maestro. La mayoría de las veces se le solicitara al usuario que ingrese el PIN del punto de acceso. Sin el PIN recibido es correcto, se lleve a cabo la conexión. En el modo seguro, el PIN se enviara cifrado con una segunda clave para evitar poner en riesgo la señal

- **Utilización de la red:** Cuando el emparejamiento se activa, el dispositivo maestro puede utilizarse libremente el canal de comunicación establecido y empezar con el intercambio de mensajes a los dispositivos maestro-esclavo.

3.6.2 Evolución de las versiones del protocolo

Bluetooth v 1.0 y 1.0D

Esta versión de Bluetooth fue introducida desde 1998. Las versiones 1.0 y 1.0B tenían demasiados problemas y restricciones para los fabricantes lo empezaron a desarrollar con éxito. El principal problema fue la falta de intemporalidad entre los dispositivos.

Bluetooth V 1.1

Esta versión 1.1 fue la primera con éxito de la tecnología Bluetooth. Bluetooth 1.1 corregía muchos de los problemas que se encontraron en anterior versiones.

Bluetooth v 1.2

En versión era completamente compatible con la 1.1. Muchos de los dispositivos Bluetooth, como los móviles, empezaron a venderse con la nueva especificación 1.2. Esta versión incorporaba una velocidad de transmisión superior a 721kbit/s, técnicas de interferencias, conexiones síncronas extendidas que reducían la latencia en transferencias de audio e introdujo un control de flujo y de retransmisión por L2CAP.

Bluetooth v 2.0 + EDR

La versión 2.0 + EDR fue anunciada por el SIG en 2004, era totalmente compatible con la versión 1.2 y empezó a aparecer en los dispositivos en 2005. Esta versión introdujo el Enhanced Data Rate (EDR) para mejorar los ratios de transferencia de datos consiguiendo una tasa de transferencia teórica de 3 Mbit/s y en la práctica de 2.1 Mbit/s. También era capaz de proporcionar menos consumo de energía a través de un ciclo de trabajo reducido. Esta versión incorporo la posibilidad de la creación de piconets, varias redes de dispositivos esclavos colectados a un maestro interconectados entre sí.

Bluetooth v 2.1 + EDR

En esta versión se mejoró la cabecera de realización de pairing (SSP) aumentando la seguridad entre dispositivos y de nuevo mejoro el consumo de energía en modo de baja potencia.

Bluetooth v 3.0 + HS

Aunque si distribución todavía está en proceso y poco dispositivos disfrutan de esta versión, la mejora más importante que incorpora es la de soportar el estándar IEEE 802.11 (típicamente de WI-FI), lo que permite una velocidad de trasferencia de 24Mbit/s.

3.6.3 Comparativa entre protocolos

Con el fin de comprar las distintas tecnologías en una misma tabla 3.1 se puede observar las diferencias entre los protocolos el cual intenta hacerse con el control de ciertos dispositivos según su funcionalidad y sus características. Además se sintetiza las características más importantes ente los protocolos WIfi, Bluetooth y Zigbee.

CATEGORÍA	ZIGBEE	BLUETOOTH	Wi-FI
Distancia	50-1600m	10m	50m
Extensión	Automática	Ninguna	Depende de la red
Consumo (Duración)	Años	Meses	Horas
Complejidad	Simple	Complicada	Muy complicada
Vel. de transferencia	259Kbps	24Mbps	54Mbps
Rango de frecuencias	868MHz, 916MHz, 2.4GHz	2.4GHz	2.4GHz
Nodos posibles	65535	7	50
Tiempo de enlace	30ms	>10s	>3s
Coste	Bajo	Bajo	Alto
Seguridad	128 bits	64bits, 128bits	SSID

Tabla 3.2: Principales comparaciones entre los protocolos de comunicación inalámbrica

3.6.4 Protocolos para la tecnología Bluetooth

Los protocolos utilizados para la tecnología Bluetooth se basa en el modelo OSI, y utiliza una arquitectura de protocolo de divide las diversas funciones de red en un sistema de niveles. En conjunto permiten el intercambio transparente de información entre aplicaciones diseñadas de acuerdo con dicha especificación, y fomentan la interoperabilidad entre productos de diferentes fabricantes.

En la figura 3.11 se muestra la las capas de protocolo Bluetooth y su comparación con el modelo OSI.

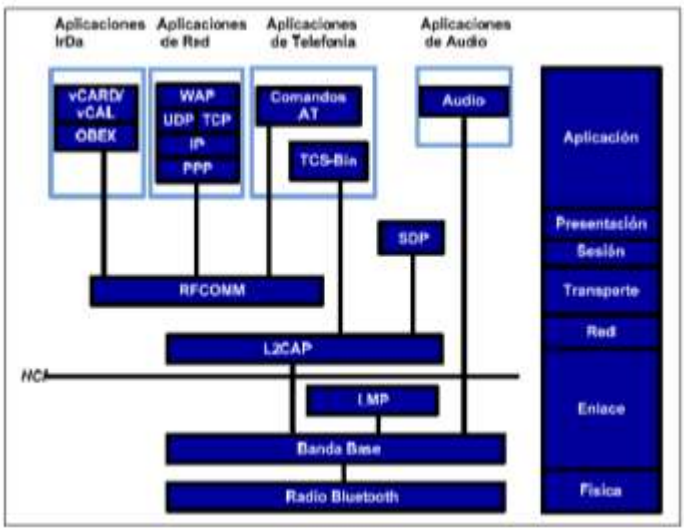

Figura 3.12: Comparacion de protocolo Bluetooth y modelo OSI

- **Protocolos del núcleo Bluetooth:** Banda base, protocolo LMP (Link Manager Protocol), Protocolo L2CAP (Logical Link Control and Adaptation Protocol). Protocolo SDP (Service Discovery Protocol).
- **Protocolos de situación de cable:** RFCOMM (Radio Frecuency Communication).
- **Protocolos de control de telefonía:** TCS-Binario (Telephony Control Protocol), Command AT.
- **Protocolos adaptados:** PPP (Point-to-Point Protocol), UDP (User Datagram Protocol), TCP (Transmission Control Protocol), IP (internet Protocol), OBEX (Object Exchange), WAP, (Wireless Applicacion Protocol).

Ya que los dispositivos dentro del sistema de implementación se utilizara la versión Bluetooth 2.1 EDR (Enhanced Data Rate) que contiene el Robot LEGO EV2, se describirá en los siguientes apartados los protocolos que forman parte del núcleo de Bluetooth siguiendo esta especificado.

3.6.5 Banda de Frecuencia

Bluetooth opera en la banda ISM de 2.4Ghz, usando 79 canales de frecuencia con un ancho de banda de 1 MHz cada uno. En la tabla 2.1 se muestra los rangos de frecuencia para diferentes regiones del mundo.

Región	Rango de frecuencia (MHz)	Canales RF (MHz)
Europa y Estados Unido	2400-2483.5	f = 2402+k k=0……, 78
Francia	2446.5-2483.5	f = 2454+k k=0,….,22
España	2445-2475	f = 2449+k k=0,….,22

Tabla 3.3: Banda de frecuencia y canales RF altos.

Dado que la banda ISM está abierta, el sistema de radio Bluetooth se encuentra expuesto a múltiples interferencias por los que utiliza la técnica FHSS (Frecuency-Hopping Spread Spectrum).

Velocidad

Se tienen dos modos de velocidad:

- **Modo Básico:** En este modo se emplea la modulación GFSK (Gaussian Frequency Shift Keying). Se tiene una velocidad de 1 Mbps.
- **Modo EDR (Enhanced Data Rate):** Este modo se emplea en dos tipos de modulaciones. La modulación $\pi/4-DQPSK$ (Differential Quadrature Phase Shift Keying) que permite una velocidad de 2Mbps y la modulación 8-DPSK (Differential Phase Shift Keying) que permite una velocidad de 3 Mbps.

Potencia

De acuerdo a la potencia de transmisiones Bluetooth define tres clases de dispositivos, que se muestran en la tabla 3.4

El equipo receptor debe poseer una sensibilidad de al menos -70 dBm y la tasa de error admisible (BER, Bit Error Rate) debe ser menor o igual a 0.1 % en modo de velocidad básico y menor o igual a 0.01% en modo de velocidad EDR. Estos valores se encuentran establecidos en la especificación Bluetooth versión 2.1 con EDR [Simple pairing withepaper Version V10r00. Bluetooth sig. 3 septiembre del 2006]

Clase	Potencia de salida máxima	Potencia de salida mínima	Alcance
1	100 nW (20 dBm)	1 mw (0 dBm)	100 m
2	2.5 mW (4 dBm)	0.25 mW (-6 dBm)	10 m
3	1 mW (0 dBm)	----------	1 m

Tabla 3.4: Niveles de potencia en Bluetooth

3.6.6 Banda Base

En la banda base se encuentra el controlador del enlace. Entre las tareas que realiza esta capa esta la sincronización, transmisión, de la información, división lógica de canales, control de enlace, direccionamiento y formato de paquetes. También se define la red básica de Bluetooth: piconect y scatterner, que se muestra en la figura3.12. Una piconect como ya se había dicho en el apartado# es una colección de dos o más dispositivos Bluetooth compartiendo el mismo canal físico (están sincronizados a un reloj común y usan una misma secuencia de salto de frecuencia). Está conformada por un dispositivo maestro y puede llegar tener hasta siete dispositivos esclavos activos además de dispositivos esclavos en modo park

La topología Bluetooth permite la interconexión de varias piconect formando una scatternet. Un dispositivo puede pertenecer a diferentes piconects pero solo estar activo en una de ellas a la vez. En ocasiones el dispositivo que es el maestro de una piconet puede ser esclavo de potra piconet.

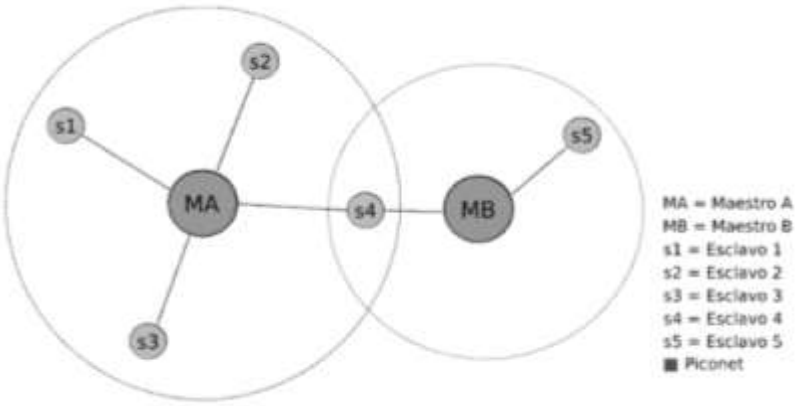

Figura 3.12: Topologia bluetooth piconect

3.6.7 Canales físicos

Se define por una secuencia pseudoaleatoria de salto escogida de entre 79 canales (para las regiones de Francia y España se tiene 23 canales disponibles) de radiofrecuencia disponibles en la banda ISM de 2.4 Ghz usando la técnica FHSS. Tanto el dispositivo transmisor como el receptor deben usar la misma secuencia de frecuencia de portadora.

La secuencia de salto es única para cada piconet. Todos los dispositivos conectados a la piconet están sincronizados con el canal en salto y tiempo. La velocidad de salto es de 1600 saltos/segundo en el estado de conexión y de 3200 santos/segundo en estado Inquiry y page.

Para emular una transmisión full dúplex los canales físicos utilizan el esquema TDD (Time División Dúplex), en el que cada canal está dividido en ranuras de tiempo a slots, cada slot corresponde a una frecuencia de salto y tiene una duración de 625 us, Los dispositivos Bluetooth alternan entre la transmisión y recepción de datos de un slot a otro. El rango de numeración va de 0 a $2^{27}-1$ en forma cíclica con una duración de 2^{27} . El maestro comienza su transmisión en los slots pares, mientras que los esclavos lo hacen en los slots impares para evitar fallas en la transmisión. En la figura 3.14 se muestra la transmisión entre un dispositivo maestro y un esclavo.

Los dispositivos Bluetooth usan paquetes para la transmisión de sus datos, cada paquete corresponde a un slot pero para permitir comunicaciones más eficientes se usa también paquetes multislot que pueden extenderse hasta una duración de 5 slots.

Los canales físicos son identificados por el código de acceso que se encuentra en el paquete, junto con el reloj y la dirección del dispositivo Bluetooth maestro.

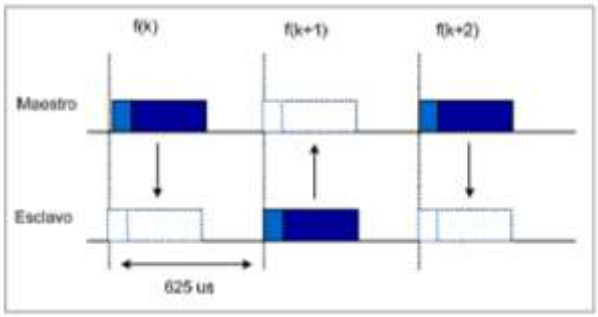

Figura 3.14: Transmisión entre maestro y esclavo

Bluetooth define cuatro canales físicos:

- **Piconet:** Este canal es usado para la comunicación entre los dispositivos conectados.
- **Adapted Piconet:** Este canal ofrece la posibilidad de utilizar su subconjunto de los 79 canales de radio frecuencia (mínimo 20) para evitar las frecuencias en las que la interferencia es demasiado alta y facilitar la coexistencia con otros sistemas es la misma banda de frecuencia. Además el dispositivo maestro en su trasmisión anterior.
- **Page-Scan:** En este canal se envían las solicitudes de Page hacia los dispositivos con los cuales se requiere establecer una conexión. Los dispositivos que estén preparados para aceptar conexiones escuchan a través de este canal dichas solicitudes y envían sus respectivas respuestas.

- **Inquiry-Scan:** Es para que lo dispositivos que quieran descubrir otros dispositivos envíen sus solicitudes de Inquiry. Los dispositivos que esperan ser descubiertos escuchan a través de este canal a la espera de las solicitudes de Inquiry y envían sus respectivas respuestas

3.6.8 Enlaces físicos

Dentro del sistema de Bluetooth un enlace físico es concepto virtual y representa una conexión de Banda Base entre dispositivos Bluetooth. Es un enlace punto a punto entre el dispositivo maestro y esclavo y siempre se encuentra presente cuando el esclavo esta sincronizado dentro de la piconet.

Hay dos tipos de enlaces físicos:

- **Activo:** Un enlace físico entre un dispositivo maestro y un esclavo es activo si existe un trasporte lógico ACL entre los dispositivos
- **Aparcado:** Existe un enlace físico de este tipo entre un dispositivo Bluetooth maestro y un esclavo cuando el esclavo no necesita participar en el canal de la piconet trasmitiendo datos pero aun desea permanece sincronizado con el canal.

3.6.9 Transportes lógicos

Se tiene definidos 5 tipos de trasportes lógicos que se pueden establecer entre el dispositivo maestro y los esclavos.

a. **SCO (Synchronous Connection-Oriented)**
   Es un trasporte simétrico, punto a punto entre el maestro y un esclavo especifico. Para establecerlo se reservan dos slots consecutivos en intervalos regulares. La reserva de los slots la realiza el dispositivo maestro cuando se establece conexión con el dispositivo esclavo. Puede ser considerado como una conexión de conmutación de circuitos. Este tipo de enlace no requiere asegurar la entrega de paquetes y es utilizado principalmente para la transmisión de voz, soportando una tasa de transferencia de 64Kbps; los paquetes SCO nunca son retransmitidos.
   El maestro puede soportar hasta tres enlaces SCO a un mismo esclavo o diferentes esclavos, mientras que los esclavos pueden soportar tres enlaces SCO de un mismo maestro, o dos enlaces SCO si estos provienen de diferentes maestros.

b. **eSCO (Extended Sysnchronous Connection-Oriented)**
   Es un transporte punto a punto que puede ser simétrico o asimétrico entre el dispositivo maestro y un esclavo específico. También realiza la reserva de slots, pero adicionalmente brinda la posibilidad de realizar un número limitado de retransmisiones. Si las retransmisiones son requeridas estas pueden llevarse a cabo en los slots que sigue a los reservados.

c. ACL (Asynchronous Connection-Oriented)

Es un trasporte punto-multipunto entre el maestro y uno o más esclavos activos dentro de la piconet, puede ser simétrico o asimétrico. No hay reserva de canal, usa slots por demanda, los mismos que pueden ser 1, 3 o 5 slots consecutivos. Es considerado como una conexión de conmutación de paquetes Todo dispositivo esclavo activo dentro de la piconet conocido como ACL, por defecto el mismo que es creado cuando un dispositivo se une a la piconet. Solamente puede existir un enlace ACL entre cada para maestro-esclavo.

Este transporte es usado para intercambiar información de control y datos de usuario por lo que se aplica la retransmisión de los paquetes errados para asegurar la integridad de la información.

Los paquetes ACL no direccionados a un esclavo específico son considerados paquetes de broadcast y son leídos por todos los esclavos.

d. ASB (Active Slave Broadcast)

Es usado para transportar tráfico de usuario L2CAP a todos los dispositivos que se encuentren actualmente conectados al canal físico que es usado por el ASB. El tráfico es unidireccional del dispositivo maestro de la piconet (hacia los esclavos. No es confiable ya que no se realizan retransmisiones.

e. PSB (Parked Slave Broadcast)

Es usado para la comunicación entre el dispositivo maestro y esclavo que se encuentra en modo park, la igual que el ASB no es confiable. Lleva tráfico de control LMP y de usuario L2CAP.

3.6.10 Formato de paquete Bluetooth.

En Bluetooth los datos son enviados en paquetes y siguen el formato Little Endian, en el que el bit menos significativo LSB es el primero en ser enviado por el aire.

El formato general del paquete para el modo de velocidad básica se muestra en la figura 3.15.

Cada paquete contiene tres entidades: el código de acceso, la cabecera y la carga útil.

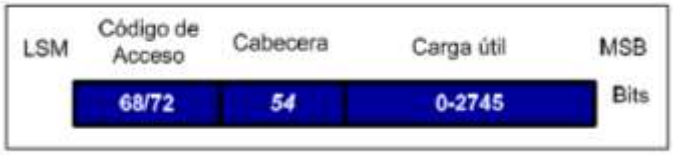

Figura 3.15: Formato general de un paquete Banda base para el modo básico

Para el modo de velocidad EDR se tiene el formato de paquete mostrado en la figura 3.16.

Figura 3.16: Formato general de un paquete de Banda Base para el modo EDR

Cada paquete EDR está formado por 6 entidades: el código de acceso, la cabecera, un periodo de guarda (GUARD), una secuencia de sincronización (SYNC), la carga útil EDR y una cola.

El código de acceso y la cabecera son idénticos en formato y modulación que los paquetes de modo básico, mientras que la secuencia de sincronización, la carga útil EDR y la cola usan el esquema de modulación EDR. EL tiempo de guarda (campo GUARD del paquete de Banda Base para el EDR) permite la transición entre los esquemas de modulación y tiene una duración entre 4.75 us y 5.25 us. La secuencia de sincronización consta de 11 símbolos DPSK generados a partir de una secuencia de bits preestablecidos y la cola consta de 2 símbolos cuyos bits deben ser todos cero.

a. Código de acceso

El código de acceso identifica todos los paquetes intercambiados sobre el canal físico.

Los cuales existen tres tipos de código de acceso:

- **CAC (Channel Access Code):** Identifica una piconet y es incluido en todos los paquetes transmitidos en la piconet.
- **DAC (Device Access Code):** Se utiliza en el procedimiento de establecimiento de conexión.
- **IAC (Inquiry Access Code):** Se utiliza en el procedimiento de búsqueda de dispositivos.

b. Cabecera.

Contiene información de control de enlace y está formada por 6 campos como se muestra en la figura 3.17

Figura 3.17: Formato de la cabecera del paquete Banda Base

- **Direccion LT_ADDR:** A Cada dispositivo esclavo activo en la piconet dentro de la piconet se le asigna una dirección LT_ADDR para distinguirlos individualmente. Indica el dispositivo esclavo destino del paquete es una transmisión maestro-esclavo e indica el dispositivo esclavo fuente en una transmisión esclavo-maestro. La dirección 000 está reservada para paquetes de broadcast.
- **Tipo:** Este campo especifica qué tipo de paquete es usado y depende del tipo de transporte lógico asociado por el paquete.
- **Flujo:** Se emplea para el control de flujo. Este bit en 0 indica una señal de parada y el bit en 1 una señal para continuar la transmisión. Esta señal solo concierne a los paquetes ACLs
- **ARQN:** Se utiliza para informar una transferencia exitosa de la carga útil con CRC y puede ser un ACK con el bit ARQN = 1 o un NAK con el bit ARQN = 0 con el que se está solicitando una retransmisión. El éxito de la recepción es comprobado mediante el cálculo de CRC.
- **SEQN:** Proporciona una secuencia numérica para ordenar el flujo de paquetes de datos. Por cada nuevo paquete de datos con CRC transmitido el bit SEQN es invertido.
- **HEC:** Para la verificación la integridad de la cabecera. Se usa el polinomio generador $x^8 + x^7 + x^5 + x^2 + x + 1$.

La cabecera consta de 18 bits y es codificada con FEC de tasa 1/3 que se basa es repetir 3 veces cada bit dado como resultado una cabecera de 54 bits de longitud.

c. Carga útil

Transporta la información de las capas superiores y se puede distinguir dos campos: el capo de voz (síncrono) y el campo de datos (asíncrono) como se muestra en la imagen en la figura 3.18.

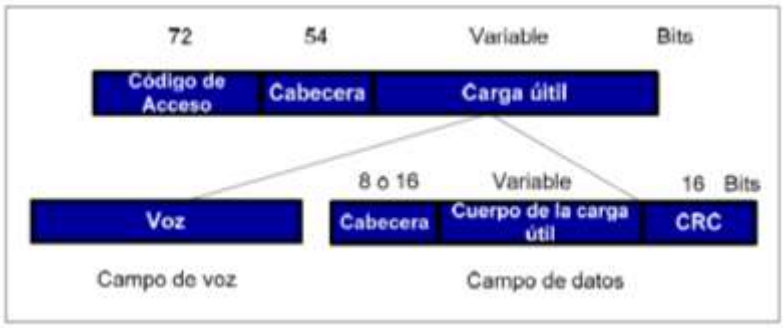

Figura 3.18: Formato del campo carga útil del paquete de Banda Base en modo de velocidad Basico

El campo de datos se encuentra formado por una cabecera que lleva información de control, un cuerpo de la carga útil que contiene los datos propiamente dichos que

puede ser información LMP o L2CAP y un CRC de 16 bits para la verificación de errores en la carga útil. Para su cálculo se emplea el polinomio generador $x^{16} + x^{12} + x^5 + 1$.

3.6.11 Establecimiento de una conexión

Para el establecimiento de una conexión en Bluetooth los dispositivos pueden estar en ciertos estados como son:

- **Inquiry:** Este estado es utilizado para descubrir otros dispositivos dentro del área de cobertura. En este estado el dispositivo adquiere información de los dispositivos como su dirección BD_ADDR.
- **Scan:** Cuando un dispositivo Bluetooth esta modo STANDBY (dormido), periódicamente escucha el canal, esperando a ser descubiertos por otros dispositivos.
- **Page:** Es un estado de búsqueda que es utilizado, generalmente, luego del estado de Inquiry para establecer la conexión.

3.6.12 Modos de ahorro de energía

Los modos de energía permiten optimizar su uso de acuerdo a los requerimientos y son:

- **Hold:** El maestro puede ordenar al esclavo quedarse en modo Hold. Durante este periodo no hay comunicación posible entre esclavo y maestro. Cuando el periodo expira el esclavo vuelve al canal y permanece sincronizado.
- **Park:** El esclavo también puede ser puesto en modo Park. En este caso el esclavo entra a un ciclo de trabajo en donde los intervalos de escucha del maestro son muy largos.
- **Sniff:** El esclavo no escucha todas las ranuras de tiempo, sino que solo escucha algunas. Para entrar al modo Sniff, el esclavo y maestro deben acordar en que ranura el esclavo podrá atención al canal.

3.6.13 Protocolo L2CAP (Logica Link Control and Adaptation Protocol)

Se encarga de adaptar los protocolos de capas superiores al protocolo de Banda Base. Brinda servicios de datos orientados y no orientados a conexión a las capas superiores. L2CAP permite a las capas superiores enviar y recibir paquetes de datos de hasta 64 Kbytes de longitud y está definido únicamente para enlaces ACL.

L2CAP sigue un modo de comunicación basado en canales lógicos que permite el flujo de datos entre entidades L2CAP en dispositivos remotos.

Los canales L2CAP pueden ser:

- Canales de señalización bidireccionales que transportan comandos. Estos canales son usados para crear y establecer canales orientados a conexión y para negociar cambios en las características de estos canales.
- Canales orientados a conexión para conexiones punto a punto bidireccionales.
- Canales unidireccionales no orientados a conexión que soportan conexiones punto-multipunto, permitiendo que una entidad local L2CAP sea conectada a un grupo de dispositivos remotos.

Las funciones principales del Protocolo L2CAP son:

- Multiplexacion de protocolos de capas superiores.
  L2CAP multiplexa los protocolos de capas superiores con el fin de enviar carios protocolos cobre el canal Banda Base.
- Segmentación y Re ensamblado
  Los paquetes de datos definidos por el protocolo Banda Base están limitados en tamaño. Los pequeños grandes L2CAP deben ser segmentados en varios paquetes antes de transmitirse.

  En el receptor los paquetes pequeños reciben la Banda Base son re ensamblados en paquetes L2CAP.

3.7 Componentes de software

Como cualquier dispositivo programable el EV3 requiere de elementos software y el robot EV3 contiene algunos elementos ya embebidos de fábrica y estos elementos son:

- Un sistema operativo :
  EL EV3 brick tiene su propio firmware que puede ser considerado como un sistema operativo. Este viene almacenado en la memoria flash por lo que no será borrado si se apaga o se retira la batería. Para este frimeware se tiene un sistema de programación grafico llamad Lego Mindstorms EV3. Este lenguaje de alto nivel funciona por medio de bloques con funcionalidades específicas, que se conectan generando rutinas que luego son transferidas al procesador del robot para su posterior ejecución.

  Este frimware puede ser reemplazado por otros tipos de firmware que ofrezcan un mejor desempeño, características adicionales o soporte a algún lenguaje de programación específico. En el presente proyecto se reemplazara el firmware

original y se utilizara el sistema operativo lejos (Lego Java Operating System) que permite el desarrollo de programas Java para el robot.

El proceso de remplazo de firmware original por el lejos se muestra en el ANEXO E.5.

- Herramientas para la programación
  Se dispone de diferentes entornos de desarrollo por ejemplo el firmware lejos que se utilizara se tiene un IDEs Java como por ejemplo el IDE NetBeams que se utilizara en el presente proyecto.

3.7.1 Lejos (Lego Java Operating System)

Lejos es un proyecto *open source* que permite a los desarrolladores realizar programas Java para los robots Lego Mindstorms. Su descarga es gratuita

Reemplaza al firmware original y emplea una máquina virtual Java que se encarga de interpretar el bytecode para comunicarse al EV3 brick.

Lejos contiene algunas características las cuales benefician a la utilizar el software y estas son:

- Soporte para programar orientada a objetos (Java).
- Es un proyecto open source con vario colaboradores.
- Tiene soporte para multiplataforma: Windows, Linux, MAC OS.
- Tiene soporte para comunicación Bluetooth
- Proporciona alta precisión en el control de motores del robot
- Soporta monitoreo remoto y seguimiento de los programas lejos desde el PC

Lejos cuenta con una API llamada EV3 (java for Lego Mindstorms) API que está formada por varias clases que permitan darle funcionalidad a los programas. Los paquetes de comunicación se muestran en la tabla 3.5.

Paquete de comunicación	Descripción
java.io	Soporte para entrada y salida
javax.bluetooth	Clases estándar de Bluetooth.
java.microedition.io	J2ME I/O
lejos.ev3.comm	Clases de comunicación EV3
lejos.nt.socket	Soporte para sockets vía PC.
lejos.net.remote	Acceso remoto EV3 sobre Bluetooth

Tabla 3.5: Herramientas para comunicación

Los paquetes que permiten la administración de EV3 brick, sensores, motores se muestran en la tabla 3.6.

Paquetes de administración	Descripción
Javax.microedition.lcdui	Clases J2ME para interfaces de usuario
Javax.microedition.location	Clases J2ME para soporte de servicio de localización
Lejos.gps	Soporte GPS
Lejor.ev3	Proporciona acceso a los sensores motores EV3
Lejos.ev3.addon	Proporciona acceso para hardware que no ha sido incluido en el kit Lego EV3
Lejos.ev3.debug	Depuración de las clases de hilos.
Lejos.util	Utilidades

Tabla 3.6: Herramientas para sensores y actuadores.

3.7.2 Clase de uso de sensores de luz y tacto

Para accedes al sensor de luz se emplea la clase LigthSensor del paquete lejos.EV3

Se tiene dos construcciones:

LightSensor (SensorPort puerto)

LightSensor (SensorPort, boolean foco)

El parámetro de entrada puerto puede tomar el valor de las siguientes constantes que representan a los cuatro puertos EV3 Brick donde se conectan los sensores:

- SensorPort.S1.
- Sensor.Port.S2.
- Sensor.Port.S3.
- Sensor.Port.S2.

El parámetro de foco permite encender o apagar el led del sensor dependiendo si tiene valor de true o flase.

El método que permite obtener la lectura del sensor de luz:

Int readValue()

3.7.3 Clases para el uso del sensor de tacto

Para acceder al sensor de tacto se emplea la clase TouchSensor del paquete lejos.EV3.

Su constructor es:

TouchSensor (SensorPort puerto)

Esta clase dispone del método:

Boolean isPressed ()

Este método chequea si el sensor está presionado, en caso afirmativo devuelve el valor de True.

3.7.4 Clase manejo de motores

Se tiene la clase de Motor del paquete de lejos.ev3. Esta clase es una abstracción de los motores EV3 y proporciona una interfaz para cada puerto a los que se conectan los motores, los cuales son: Motor.A, Motor.B y Motor.C. Algunos de los métodos que permiten el control de los motores se muestran en la Tabla 3.7.

Métodos	Descripción
void backward()	Provoca que el motor rote hacia atrás.
void forward()	Provoca que el motor rote hacia adelante.
void stop()	Provoca que el motor se detenga.
void setSpeed(int velocidad)	Establece la velocidad el motor en grados por segundo. La máxima velocidad es 900 con 8 V.
void rotate(int ang)	Provoca que el motor rote en un ángulo (*ang*) especifico.
int getTachoCount()	Obtiene la cuenta del tacómetro.
void resetTachoCount()	Pone la cuenta del tacómetro en cero.

Tabla 3.7: Métodos para el control de motores

Algunos métodos para el control de motores ev3

3.7.5 Clases manejo de Botones

En esta clase se tiene Botón con 5 instancias que representa a 5 botones las cuales son:

- Button.ENTER
- Button.UP
- Button.DOWN
- Button.LEFT
- Button.RIGHT

Algunos de los métodos de las clases e muestran en la tabla

Método	Descripción
boolean isPressed()	Chequea si el botón se encuentra presionado.
void waitForPress()	Espera hasta que un botón sea presionado.
int readButtons()	Lee el estado de los botones.

Tabla 3.8: Métodos para botones del bick

Algunos métodos de la clase Button

Además se debe añadir un listener implementando la interfaz ButtonListener para escuchar los eventos generados por los botones. Esto se lo realiza mediante el método:

Void addButtonListener(ButtonListener listener)

La interfaz ButtonListener implementa los métodos:

- **Void buttonPressed (Button btn):** es llamado cada vez que un botón ha sido presionado. Permite determinar qué acción se desea realizar según el botón presionado.
- **Void buttonReleassed (Button btn):** es llamado cuando el botón ha sido liberado

3.7.6 APIS de JAVA para Bluetooth

Las APIS permiten crear aplicaciones para controlar el Bluetooth de dispositivos. Estas librerías que se utilizaran en el presente proyecto para establecer una comunicación Bluetooth.

API javax.bluetooth

Este API contiene las clases y métodos necesarios para establecer una comunicación Bluetooth entre dispositivos. Permite establecer conexiones a nivel de protocolo L2CAP y el protocolo RFCOMM.

Las acciones que se deben llevar a cabo en el dispositivo cliente para poder comunicarse con el servido son:

- Búsqueda de dispositivos
- Establecimiento de la conexión
- Enviar y recibir datos

Mientras que en el servidor se deben llevar a cabo las siguientes acciones:

- Crear una conexión servidora

- Aceptar conexión de los clientes
- Enviar y recibir datos

Para llevar a cabo las acciones antes mencionadas se disponen de diferentes clases que se detallaran a continuación.

3.7.7 Clases Básicas para administración de dispositivos.

Se tiene clases que representan a los objetos Bluetooth esenciales, como son LocalDevice y RemoteDevice. Además se tiene las clases DeviceClass y BluetoothStateException que dan soporte a la clase LocalDevice.

- **Clase LocalDevice:** Proporciona acceso y control sobre el dispositivo Bluetooth local. Permite obtener información del mismo como por ejemplo el modo de conectividad, la dirección Bluetooth, y el nombre del dispositivo o fiendly-name. Esta clase es el punto de partida para las acciones que se vayan a llevar a cabo mediante este API. Los métodos principales de esta clase con:
  - **LocalDevice getLocalDevice():** Obtiene el objeto que represente al dispositivo local.
  - **String getAddress():** Obtiene la dirección Bluetooth del dispositivo local.
  - **Boolean setDiscoverable (int mode):** Establece el modo de detección de dispositivos. Los modos que se pueden establecer mediante el parámetro de entre mode, se muestra en la tabla 3.9
  - **DiscoveryAgent getDiscoveryAgent():** obtiene el objeto Discovery Agent para el dispositivo local.

- **Clase RemoteDevice:** De forma similar, esta clase representa al dispositivo Bluetooth remoto y permite obtener información como el nombre, la dirección Bluetooth del dispositivo remoto y otros términos asociados a la conexión. A continuación se describen algunos métodos de esta clase:
  - **RemoteDevice getRemoteDevice (javax.microedition.io.Connection):** Obtiene el objeto que representa al dispositivo Bluetooth remoto asociado a la conexión correspondiente. El parámetro de entrada con representa dicha conexión.
  - **String getBluetoothAddress():** obtiene la dirección Bluetooth del dispositivo remoto.
  - **String getFriendlyName():** obtiene el nombre del dispositivo Bluetooth remoto

Nombre de la constante	Descripción	Valor
DiscoveryAgent.GIAC	Pone a los dispositivos en un modo detectable general.	10390323
DiscoveryAgent.LIAC	Pone a los dispositivos en un modo detectable limitado. Son visibles ante otros dispositivos durante un periodo de tiempo limitado.	10390242
DiscoveryAgent. NOT_DISCOVERABLE	En este modo los dispositivos no podrán ser descubiertos por otros.	0

Tabla 3.9: Modo para establecer tipo de búsqueda

- **Clase DeviceClass:** A través de esta clase se puede obtener de dispositivos. Por ejemplo se conoce si el dispositivo es un teléfono, un ordenador, EV3 u otra clase de dispositivo.
- **Clase BluetoothStateException():** Representa la excepción que se produce cuando un dispositivo no puede cumplir con una solicitud que normalmente si la soportaría debido al estado actual del sistema. Por ejemplo algunos dispositivos no permiten realizar el proceso de inquirí cuando el dispositivo se encuentra conectado a otro dispositivo.

3.7.8 Búsqueda de dispositivos.

La búsqueda de dispositivos es realizada por el equipo Cliente para poder detectar el equipo al que desea conectarse.

Para poder realizar la búsqueda de dispositivos es necesario crear un único objeto DiscoveryAgent de la clase DiscoveryAgent. Este objeto se le puede obtener a partir del método getDiscoveryAgent() del objeto LocalDevice como en la figura 3.19.

```
LocalDevice equipo = LocalDevice.getLocalDevice();

DiscoveryAgent discoveryAgent = equipo.getDiscoveryAgent();

DiscoveryAgent discoveryAgent =
LocalDevice.getLocalDevice().getDiscoveryAgent();
```

Figura 3.19: Ejemplo para obtener DiscoveryAgent

Los métodos de la clase DiscoveryAgent que van a permitir realizar y cancelar búsquedas de dispositivos son:

- **Boolean startlnquiry (int accessCode, DiscoverListener listener):** Permite iniciar el proceso de búsqueda de dispositivos (proceso de Inquiry). Este método requiere que se implemente la interfaz DiscoveryListener, esta interfaz será notificada cada vez que un nuevo dispositivo sea encontrado o el proceso de búsqueda haya terminado.

  Este método requiere dos argumentos de entrada. El primero argumento es un valor entero que representa el modo de conectividad que se ha definido para la búsqueda. Podría tomar los valores de las constantes DiscoveryAgent.GIAC o DiscoveryAgent.LlAC que se muestra en la tabla# Generalmente se usa DiscoveryAgent.GIAC para realizar una búsqueda general. El segundo argumento es un objeto que implementara la cual se notificara los dispositivos que se vaya descubriendo.
- **Boolean cancellnquiry (DiscoveryListener listener):** Este método cancela la búsqueda de dispositvos que se encuentre en proceso.
- **RemoteDivece[] retriveDevices (int option):** Este método se usa para obtener lista de dispositivos ya conocidos. Devuelve un arreglo de objetos RemoteDevice. El argumento de entrada opción puede tomar los valores que se muestran en la tabla 3.10.

Nombre de la constante	Descripción	Valor
DiscoveryAgent.CACHED	Recupera los dispositivos que fueron descubiertos mediante un proceso de *inquiry* previo.	0
DiscoveryAgent.PREKNOWN	Recupera los dispositivos con los cuales el dispositivos local interactùa frecuentemente y que han sido agregados como pre-conocidos.	1

Tabla 3.10: Valores que puede tomar el argumento de entrada del método retriveDevice

3.7.9 Comunicación

Una vez que la aplicación Bluetooth haya realizado la búsqueda de dispositivos, se procederá a realizar el establecimiento de la conexión con el dispositivo seleccionado, para posteriormente poder realizar la transmisión de datos.

Para establecer la conexión se utiliza el perfil SSP que debe seguir diferentes pasos dependiendo es cliente o servidor.

- Servidor SSP

Para crear una conexión se hace uso de la clase Conector perteneciente al paquete javax.microedition.io. Esta clase permite abrir una conexión mediante el método Connector.open(), el cual recibe un argumento de entrada del tipo String que contiene un URL (Uniform Resourse Locator). Este URL debe contener el protocolo, la dirección Bluetooth, el UUID (identificador de servicio), y parámetros de seguridad opcionales. Así el formato para el URL del dispositivo servidor es el siguiente.

Btspp://localhost:UUID;parámetros

En la tabla 3.11 se muestran los diferentes parámetros que se pueden usar en el URL.

Nombre	Descripción	Valores permitidos	Cliente o Servidor
master	Especifica si el dispositivo debe ser el maestro de la conexión.	*true, false*	Ambos
authenticate	Especifica si el dispositivo remoto debe ser autenticado antes de establecer la conexión.	*true, false*	Ambos
encrypt	Especifica si el enlace debe ser encriptado.	*true, false*	Ambos
authorize	Especifica si todas las conexiones hacia el dispositivo deben recibir una autorización para usar el servicio.	*true, false*	Servidor
name	Especifica el nombre de servicio.	Cualquier *String* válido	Servidor

Tabla 3.11: Parámetros que se pueden usar en el URL de conexión

El método Connector.open() devuelve un objeto StreamConnection-Notifier (interface del paquete javax.microedition.io) para la conexión servidora. Este objeto es un notificador que permitirá escuchar las conexiones entrantes de los clientes a través del método acceptAndOpen().

El método acceptAndOpen() retorna un objeto StreamConnection (interfaz del paquete javax.microedition.io) que permite obtener y abrir los flujos (streams) de entrada y salida de datos a través de los métodos openDataInputStream() y openDataOutputStream() respectivamente. Adicionalmente tiene el método close() que permite cerrar la conexión.

Para enviar y leer los datos se puede empelar los métodos read() y write() de la clase DataInputStream y DataOutputStream respectivamente pertenecientes al paquete java.io.

En la figura 3.20 se muestra un ejemplo de la creación de una conexión para un servidor SPP.

```
StreamConnectionNotifier notificador =
(StreamConnectionNotifier)Connector.open("btspp://
localhost:123456789ABCDE;name=Servidor");
StreamConnection conn = notificador.acceptAndOpen();
DataOutputStream salida = conn.openOutputStream();
DataInputStream entrada = conn.openInputStream();
//lee los datos recibidos
byte datosRecividos[ ] = new byte;
entrada.read(datosRecividos);
//Se envia datos através del flujo de salida
salida.write(dato);
// Se cierra la conexión y los flujos de entrad y salida
entrada.close();
salida.close();
con.close();
```

Figura 3.20: Ejemplo para la creación de una conexión para un servidor SPP

- Cliente SPP
  Al igual que en el servidor se emplea el método open() de la clase Connector para abrir una conexión. La diferencia está en los campos del URL. El formato para el URL del dispositivo cliente es el siguiente:

  Btspp://direccionBluetooth:CN;Patametros

  Dónde:
  - direccionBluetooth: es la dirección del dispositivo con el cual se quiere conectar.
  - CN: Es el número de canal usado por el cliente para conectarse con el servidor.
  - Parámetros: Representa los parámetros adicionales que se pueden usar. Estos se muestran en la tabla 3.11

Un ejemplo de un URL para un dispositivo cliente es:

Btspp://00803DD8901:1;master=false; encrypt=false

El método Connector.open() devuelve el objeto StreamConnection que representa la conexión. Al igual que en el servidor a través de este objeto se puede obtener los flujos de datos de entrada y salida para él envió y recepción de datos como se muestra en el ejemplo de la fgura3.21

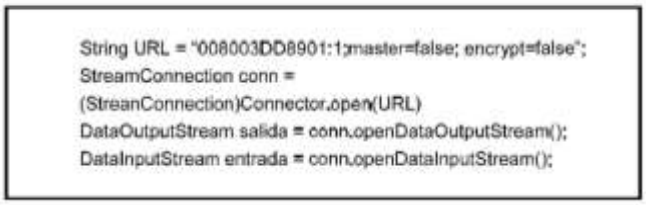

Figura 3.21: Ejemplo para la creación de una conexión para un Cliente SPP

3.7.10 Clase para comunicación Bluetooth

Se tiene varias clases que permiten establecer la comunicación mediante Bluetooth con otros dispositivos.

- Búsqueda de dispositivo
  Si no se tiene conocimiento sobre los dispositivos que se encuentran alrededor se debe
  llevar a cabo el proceso de Inquiry. Para la búsqueda de dispositivos se utiliza el siguiente método:

  Vector inquire(int maxDevices, int timeout, byte[] code)

  Donde:

  - maxDevices: representa el máximo número de dispositivos a descubrir.
  - Timeout: representa el tiempo de espera (1.28 s).
  - Code: representa la clase de dispositivos que se va a buscar.

  Este método regresara un vector con los dispositivos encontrados a partir de los cuales se obtiene los objetos RemoteDevice.

- Conexión con un dispositivo Bluetooth
  Una vez que se tiene el RemoteDevice se puede establecer la conexión con un dispositivo específico mediante el siguiente método de la Clase Bluetooth:

  BTConnection connct (RemoteDevice dispostivoRemoto)

  Este método devuelve un objeto BTConnection que representa la conexión Bluetooth establecida. Si no se pudo llevar a cabo la conexión regresa null

- Intercambio de datos entre el EV3 con otro EV3

Cuando sea ya establecida la conexión a través del objeto BTConexion se obtiene un objeto StreamComexion (interfaz del paquete javax.microedition.io) que permite obtener y abrir los flujos (streams) de entrada y salida a través de los métodos openDataInputStream() y openDataOutputStream() respetvivament.

Empleado los métodos read() y write() de la clase DataInputStream y DataOutPutStream, podemos enviar y leer datos, los cuales pertenecen al paquete java.io.

- Escuchar conexión Bluetooth
  Este método solo se utilizara para comunicar otros dispositivos como una PC, Móvil, más de un EV3, etc. Ya que el EV3 se encontrara como servidor y deberá escuchar las solicitudes de conexión de los clientes para ello se tiene el método de la clase:

  BTConnection waitForConnection()

  Este método retornara un objeto BTConnection() que representa la conexión Bluetooth establecida. Una vez establecida la conexión se realizara el intercambio de datos.

3.7.11 Threads

La robustez de lejos puede soportar JAVA Threads (hilos) que permite realizar tareas concurrentes. Ya que en la programación de robots se puede tener varios threads como por ejemplo:

- Un thread para administrar la locomoción del sistema
- Un thread para administrar la comunicación Bluetooth
- Un thread que pueda administrar los sensores del robot EV3

Una forma de implementar un thread en Java es crear una clase que se derive de la clase Thread. Esta clase se encuentra en el paquete java.lang. Al hacer esto se debe sobre escribir el método run(), dentro de este método ira el código que se desea que se ejecute cuando inicie el thread como se muestra en la siguiente figura 3.22.

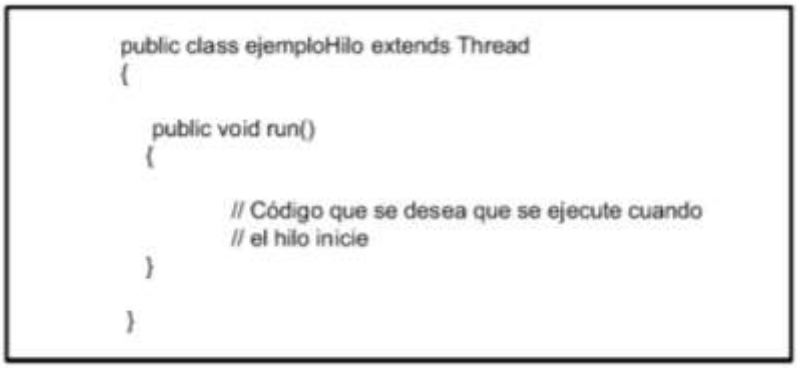

Figura 3.22: Ejemplo para utilizar Thread

En tareas concurrentes se debe sincronizar los objetos para que no sean accedidos por varios therads a la vez. Una forma de hacerlo es mediante la synchronized como se muestra en la figura 3.23. Esto hace que si un therad es haciendo uso de un objeto, un segundo no puede hacerlo hasta que el primero lo deje.

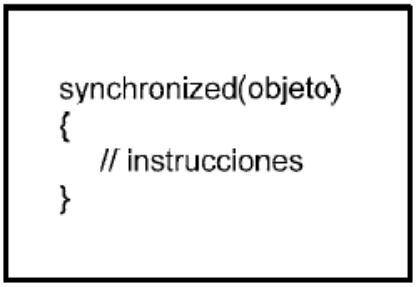

Figura 3.23: Ejemplo de uso de synchronized

El API de java para sincronización también proporciona los métodos wait() y notify() del paquete java.lang

El método wait () espera a que una condición ocurra y es usado dentro de un bloque que ha sido sinconizado.

El método notify() notifica a un thread que se encuentra esperando que la condición ocurra.

3.8 Arquitectura de software

La arquitectura de software se refiere a las estructuras de un sistema, compuestas de elementos con propiedades visibles de forma eterna y las relaciones que existen entre ellos

El termino elemento puede referirse a distintas entidades relacionadas con el sistema. Los elementos pueden ser entidades que existen en tiempo de ejecución (objetos, hilos), entidades lógicas que existen en tiempo de desarrollo (clases, componentes) y entidades físicas (nodos). Por otro lado, las relaciones entre elementos depende de propiedades visibles (o públicas) de los elementos, quedando ocultos los detalles implementación. Finalmente, cada conjunto de elementos relacionados de un tipo particular corresponde a una estructura distinta, de ahí que la arquitectura está compuesta por distintas estructuras.

A continuación se describirán brevemente las arquitecturas software más generalizadas con el fin de ofrecer una visión más general.

3.8.1 Plana

La arquitectura plana integra todos los componentes de la aplicación en el mismo espacio de direccionamiento que el sistema operativo y fue tradicionalmente adoptado por muchos sistemas en tiempo real.

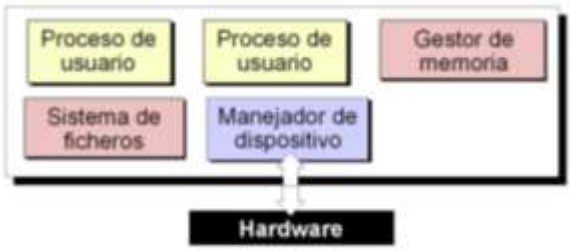

Figura 3.24: Arquitectura plana

Es una arquitectura muy utilizada en sistemas empotrados pequeños o procesadores, como los procesadores digitales de señales (DSPs).

3.9 Comunicación entre Robots

Una característica más importante de los robots es que poseen una habilidad para comunicarse. El paradigma de comunicación es adoptado es el paso de mensaje asíncrono. Cada robot tiene una pequeña pila de entrada de mensajes donde el runtime de almacenamiento de mensajes enviados por otros robots. Así mismo, cada vez que un mensaje es añadido a la cola de mensaje, el robot que lo recibe es notificado. Sin embargo, el instante en el que el robot agarra el mensaje de la cola y lo procesa depende únicamente del programador.

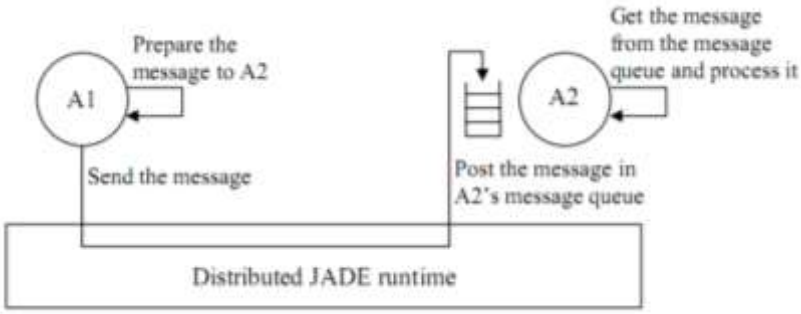

Figura 3.24: Cola de mensaje JADE

3.9.1 Envió de mensaje

Al enviar un mensaje de un robot a otro robot es tan simple como rellenar los valores de un objeto de tipo RCLMessage y llamar al método send() de la clase RobotEV3. A continuación vamos a enviar un mensaje de información al robot esclavo avisándole "Hola".

```
RCLMessage msg = new RCLMessage(ACLMessage.INFORM);
                 msg.addReceiver(new AID"EV3", AID.ISLOCALNAME));
                 msg.setLanguage("Castellano");
                 msg.setContent("Hola");
                 send(msg);
```

3.9.2 Recepción de mensajes

Como sabemos el runtime de JADE automáticamente almacena los mensajes recibidos en la cola de llegada tan pronto como sea recibido. Un robot puede tener un mensaje de su propia cola de mensajes utilizando el método receive(). Este método devuelve el primer mensaje en la cola de mensaje (y lo elimina) o devuelve null en caso de que la cola de mensajes este completamente vacía. Del mismo modo, disponemos de una función

bloqueante para la recepción de mensajes, que detendrán el comportamiento que la invoque hasta que un mensaje con las características necesarias sea recibido. Para ello, debemos invocar a la función blockingReceive(). A continuación mostramos un ejemplo de aplicación de la primera de estas funciones:

```
RCLMessage msg = receive()
                    if(msg!= null)
                            //Hemos recibido un mensaje y pasamos a procesarlo
```

En el caso de la espera bloqueante para un mensaje sería muy similar al caso anterior; con la diferencia de que no es necesario comprobar que el mensaje no es null, ya que siempre se ejecuta dicha instrucción tenemos un mensaje listo para procesar.

Capitulo 4

Resultados

4.1 Introducción

En este capitulo se presenta los aspectos de implementación a nivel practico de los resultados obtenidos en el desarrollo de esta tesis. La implementación de las trayectorias para los robots multiples se implementaron en un dispositivo comercial de la marca LEGO, llamado Lego Mindstorms EV3 [12]. Este dispositivo como ya antes mencionado en el capitulo 2 cuenta con las caracterisitcas adecuadas, en software y hardware, para el desarrollo de la comunicación entre los robots móviles EV3 via bluetooth como se mencino en el capitulo 3 utilizando algunas herramientas para realizar dicha comunicación.

Entonces, El estudio y el análisis para realizar un movimiento colectivo en distintas trayectorias, es implementado en el Kit LEGO EV3, el cual se armo como un robot móvil en configuración diferencial y asi mismo se probara cada uno de los sensores como actuadores para conocer sus limitantes, como sus alcances y asi realizar las trayectorias predefinidas en dos robot móvil EV3.

4.2 Caracterización del Robots Mindstorms EV3

La caracterización de lo Servomotores se realizó para conocer sus alcances y deficiencias, buscando obtener el mejor provecho de los robots móviles EV3.

Se realizó una caracterización de los Servo motores de los Robots LEGO EV3 que se utilizaran maestro y esclavo en nuestra aplicación. La caracterización consta de las siguientes pruebas:

- Movimiento derecho del Robot maestro y esclavo.

Figura 4.1: Desplazamiento hacia adelante

- Movimiento en reversa del robot maestro y esclavo.

Figura 4.2: Desplazamiento hacia atras

- Vuelta 360 grados hacia la izquierda de los Robots maestro y esclavo.

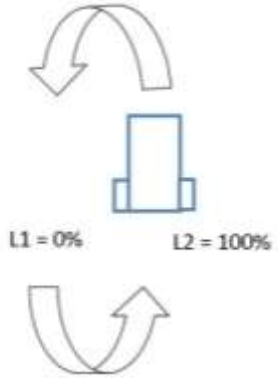

Figura 4.3: Giro de 360 grados hacia la izquierda

- Vuelta 360 grados hacia la derecha de los Robots maestro y esclavo

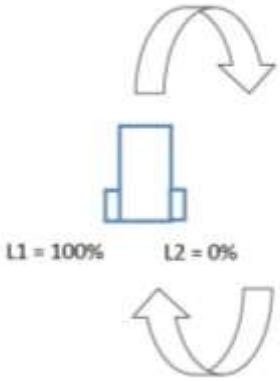

Figura 4.4: Giro de 360 grados hacia la derecha

.

- Vuelta hacia la izquierda de 360 grados con velocidad del motor L1 y L2 del robot LEGO EV3 maestro y esclavo.

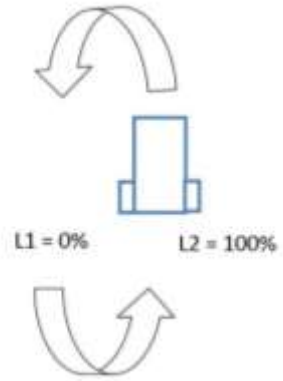

Figura 4.5 Giro de 360 grados hacia la izquierda L1 = 0% y L2 = 100%

- Vuelta hacia la derecha de 360 grados con la velocidad del motor L1 y L2 del robot LEGO EV3 maestro y esclavo.

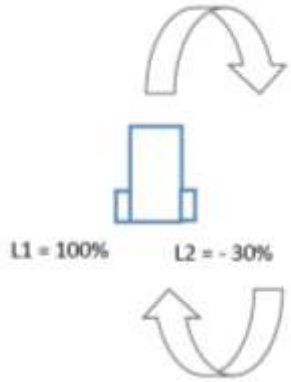

Figura 4.6: Giro de 360 grados hacia la derecha L1 = 100% y L2 = 0%

Caracterización del robot móvil LEGO EV3M Diferencial maestro movimiento en línea recta

Porcentaje del motor	Tiempo	Distancia
100%	0:03:12 seg	1 mts
90%	0:03:13 seg	1 mts
80%	0:03:25 seg	1 mts
70%	0:03:62 seg	1 mts
60%	0:04:13 seg	1 mts
50%	0:04:90 seg	1 mts
40%	0:06:37 seg	1 mts
30%	0:08:56 seg	1 mts
20%	0:12:93 seg	1 mts
10%	0:24:25 seg	1 mts
1%	3:29:66 seg	1 mts

Tabla 4.1: Caracterización robot EV3M desplazamiento línea recta

Tabla de caracterización de robot móvil LEGO EV3E diferencia Esclavo movimiento en línea recta

Porcentaje del motor	Tiempo	Distancia
100%	0:03:10 seg	1 mts
90%	0:03:11 seg	1 mts
80%	0:03:23 seg	1 mts
70%	0:03:60 seg	1 mts
60%	0:04:10 seg	1 mts
50%	0:04:88 seg	1 mts
40%	0:06:35 seg	1 mts
30%	0:08:53 seg	1 mts
20%	0:12:90 seg	1 mts
10%	0:24:23 seg	1 mts
1%	3:29:64 seg	1 mts

Tabla 4.2: Caracterización robot EV3E desplazamiento línea recta

Tabla de Caracterización de robot móvil LEGO EV3M diferencial movimiento de reversa en línea recta

Porcentaje de motores hacia atrás	Tiempo	Distancia
-100%	0:03:13 seg	1 mts
-90%	0:03:20 seg	1 mts
-80%	0:03:28 seg	1 mts
-70%	0:03:32 seg	1 mts
-60%	0:04:38 seg	1 mts
-50%	0:04:93 seg	1 mts
-40%	0:06:38 seg	1 mts
-30%	0:08:56 seg	1 mts
-20%	0:11:93 seg	1 mts
-10%	0:24:23 seg	1 mts
-1%	3:16:96 seg	1 mts

Tabla 4.3: Caracterización robot EV3M reversa

Tabla de Caracterización de robot móvil LEGO EV3E diferencial movimiento de reversa en línea recta

Porcentaje de motores hacia atrás	Tiempo	Distancia
-100%	0:03:10 seg	1 mts
-90%	0:03:17 seg	1 mts
-80%	0:03:25 seg	1 mts
-70%	0:03:29 seg	1 mts
-60%	0:04:35 seg	1 mts
-50%	0:04:90 seg	1 mts
-40%	0:06:35 seg	1 mts
-30%	0:08:53 seg	1 mts
-20%	0:11:90 seg	1 mts
-10%	0:24:20 seg	1 mts
-1%	3:16:93 seg	1 mts

Tabla 4.4: Caracterización robot EV3E reversa

Tabla de caracterización de robot móvil LEGO EV3M maestro diferencial dando una vuelta a la izquierda de 360° con los motores L1 modificara su valor en porcentajes negativos del -100% al -10% y L2 modificara su valor en porcentajes positivos del 100% al 10%

Porcentaje de motor L1	Porcentaje de motor L2	Tiempo	Distancia
-100%	100%	0:01:35 seg	360 grados
-90%	90%	0:01:25 seg	360 grados
-80%	80%	0:01:30 seg	360 grados
-70%	70%	0:01:35 seg	360 grados
-60%	60%	0:01:56 seg	360 grados
-50%	50%	0:01:84 seg	360 grados
-40%	40%	0:02:09 seg	360 grados
-30%	30%	0:02:66 seg	360 grados
-20%	20%	0:04:19 seg	360 grados
-10%	10%	0:08:78 seg	360 grados

Tabla 4.5: Caracterización robots EV3M L1 = - y L2 = +

Tabla de caracterización de robot móvil LEGO EV3E esclavo diferencial dando una vuelta a la izquierda de 360° con los motores L1 modificara su valor en porcentajes negativos del -100% al -10% y L2 modificara su valor en porcentajes positivos del 100% al 10%

Porcentaje de motor L1	Porcentaje de motor L2	Tiempo	Distancia
-100%	100%	0:01:34 seg	360 grados
-90%	90%	0:01:23 seg	360 grados
-80%	80%	0:01:29 seg	360 grados
-70%	70%	0:01:34 seg	360 grados
-60%	60%	0:01:55 seg	360 grados
-50%	50%	0:01:83 seg	360 grados
-40%	40%	0:02:08 seg	360 grados
-30%	30%	0:02:65 seg	360 grados
-20%	20%	0:04:18 seg	360 grados
-10%	10%	0:08:77 seg	360 grados

Tabla 4.6: Caracterización robot EV3E L2 = - y L2 = +

Tabla de caracterización de robot móvil LEGO EV3M maestro diferencial una vuelta a la derecha de 360 grados con los motores L1 modificara su valor en porcentajes positivos del 100% al 10% y L2 modificara su valor en porcentajes negativos del -100% al -10%

Porcentaje de motor L1	Porcentaje de motor L2	Tiempo	Distancia
100%	-100%	0:01:38 seg	360 grados
90%	-90%	0:01:34 seg	360 grados
80%	-80%	0:01:39 seg	360 grados
70%	-70%	0:01:43 seg	360 grados
60%	-60%	0:01:50 seg	360 grados
50%	-50%	0:01:81 seg	360 grados
40%	-40%	0:02:12 seg	360 grados
30%	-30%	0:02:88 seg	360 grados
20%	-20%	0:04:16 seg	360 grados
10%	-10%	0:09:19 seg	360 grados

Tabla 4.7: Caracterización robot móvil EV3M L1 = + y L2 = -

Tabla de caracterización de robot móvil LEGO EV3M maestro diferencial una vuelta a la derecha de 360 grados con los motores L1 modificara su valor en porcentajes negativos del 100% al 10% y L2 modificara su valor en porcentajes positivos del -100% al -10%

Porcentaje de motor L1	Porcentaje de motor L2	Tiempo	Distancia
100%	-100%	0:01:37 seg	360 grados
90%	-90%	0:01:33 seg	360 grados
80%	-80%	0:01:38 seg	360 grados
70%	-70%	0:01:42 seg	360 grados
60%	-60%	0:01:49 seg	360 grados
50%	-50%	0:01:80 seg	360 grados
40%	-40%	0:02:11 seg	360 grados
30%	-30%	0:02:87 seg	360 grados
20%	-20%	0:04:15 seg	360 grados
10%	-10%	0:09:18 seg	360 grados

Tabla 4.8: Caracterizacion robot móvil EV3E L1 = + y L2 = -

Tabla caracterización de robot móvil LEGO EV3M maestro diferencial una vuelta a la derecha de 360 grados con los motores L1 modificara su valor en porcentajes positivos del 100% al 10% y L2 mantendrá su valor en 0%

Porcentaje de motor L1	Porcentaje de motor L2	Tiempo	Distancia
100%	0%	0:02:19 seg	360 grados
90%	0%	0:02:28 seg	360 grados
80%	0%	0:02:31 seg	360 grados
70%	0%	0:02:59 seg	360 grados
60%	0%	0:02:81 seg	360 grados
50%	0%	0:03:25 seg	360 grados
40%	0%	0:03:90 seg	360 grados
30%	0%	0:05:32 seg	360 grados
20%	0%	0:08:27 seg	360 grados
10%	0%	0:15:87 seg	360 grados

Tabla 4.9: Caracterización robot EV3M L1 = + y L2 = 0%

Tabla caracterización de robot móvil LEGO EV3E esclavo diferencial una vuelta a la derecha de 360 grados con los motores L1 modificara su valor en porcentajes positivos del 100% al 10% y L2 mantendrá su valor en 0%

Porcentaje de motor L1	Porcentaje de motor L2	Tiempo	Distancia
100%	0%	0:02:18 seg	360 grados
90%	0%	0:02:27 seg	360 grados
80%	0%	0:02:30 seg	360 grados
70%	0%	0:02:58 seg	360 grados
60%	0%	0:02:80 seg	360 grados
50%	0%	0:03:24 seg	360 grados
40%	0%	0:03:89 seg	360 grados
30%	0%	0:05:31 seg	360 grados
20%	0%	0:08:26 seg	360 grados
10%	0%	0:15:86 seg	360 grados

Tabla 4.10: Caracterización robot EV3E L1 = + y L2 = 0%

Tabla de caracterización de robot móvil LEGO EV3M maestro diferencial una vuelta a la izquierda de 360 grados con los motores L1 no modificara su valor y se mantendrá en 0%, mientras L2 modificara su valor de 100% al 10%

Porcentaje de motor L1	Porcentaje de motor L2	Tiempo	Distancia
0%	100%	0:02:16 seg	360 grados
0%	90%	0:02:41 seg	360 grados
0%	80%	0:02:53 seg	360 grados
0%	70%	0:02:60 seg	360 grados
0%	60%	0:02:94 seg	360 grados
0%	50%	0:03:35 seg	360 grados
0%	40%	0:04:12 seg	360 grados
0%	30%	0:05:82 seg	360 grados
0%	20%	0:08:50 seg	360 grados
0%	10%	0:16:00 seg	360 grados

Tabla 4.11: Caracterizacion robot móvil EV3M L1 = 0% y L2 = +

Tabla de caracterización de robot móvil LEGO EV3M maestro diferencial una vuelta a la izquierda de 360 grados con los motores L1 no modificara su valor y se mantendrá en 0%, mientras L2 modificara su valor de 100% al 10%

Porcentaje de motor L1	Porcentaje de motor L2	Tiempo	Distancia
0%	100%	0:02:18 seg	360 grados
0%	90%	0:02:27 seg	360 grados
0%	80%	0:02:30 seg	360 grados
0%	70%	0:02:58 seg	360 grados
0%	60%	0:02:80 seg	360 grados
0%	50%	0:03:24 seg	360 grados
0%	40%	0:03:89 seg	360 grados
0%	30%	0:05:31 seg	360 grados
0%	20%	0:08:26 seg	360 grados
0%	10%	0:15:86 seg	360 grados

Tabla 4.12: Caracterización robot móvil EV3E L1 = 0% y L2 = +

Tabla de Caracterización de robot móvil maestro diferencial una vuelta a la derecha de 360 grados con los motores L1 modificara su valor en porcentajes positivos del 100% al 10% y L2 mantendrá su valor en -30%

Porcentaje de motor L1	Porcentaje de motor L2	Tiempo	Distancia
100%	-30%	0:01:75 seg	360 grados
90%	-30%	0:01:93 seg	360 grados
80%	-30%	0:02:10 seg	360 grados
70%	-30%	0:02:77 seg	360 grados
60%	-30%	0:03:11 seg	360 grados
50%	-30%	0:03:52 seg	360 grados
40%	-30%	0:04:29 seg	360 grados
30%	-30%	0:05:99 seg	360 grados
20%	-30%	0:08:57 seg	360 grados
10%	-30%	0:16:17 seg	360 grados

Tabla 4.13: Caracterización EV3M con L1 = + y L2 = - 30%

Tabla de Caracterización de robot móvil maestro diferencial una vuelta a la derecha de 360 grados con los motores L1 modificara su valor en porcentajes positivos del 100% al 10% y L2 mantendrá su valor en -30%

Porcentaje de motor L1	Porcentaje de motor L2	Tiempo	Distancia
100%	-30%	0:01:74 seg	360 grados
90%	-30%	0:01:92 seg	360 grados
80%	-30%	0:02:09 seg	360 grados
70%	-30%	0:02:76 seg	360 grados
60%	-30%	0:03:10 seg	360 grados
50%	-30%	0:03:51 seg	360 grados
40%	-30%	0:04:28 seg	360 grados
30%	-30%	0:05:97 seg	360 grados
20%	-30%	0:08:56 seg	360 grados
10%	-30%	0:16:16 seg	360 grados

Tabla 4.14: Caracterización EV3E con L1 = + y L2 = - 30%

Tabla de Caracterización de robot móvil LEGO EV3M maestro diferencial una vuelta a la derecha de 360 grados con los motores L1 se mantendrá con un valor de -30% y L2 modificara su valor de 100% a 10%.

Porcentaje de motor L1	Porcentaje de motor L2	Tiempo	Distancia
-30%	100%	0:01:55 seg	360 grados
-30%	90%	0:01:56 seg	360 grados
-30%	80%	0:02:10 seg	360 grados
-30%	70%	0:02:77 seg	360 grados
-30%	60%	0:03:11 seg	360 grados
-30%	50%	0:03:52 seg	360 grados
-30%	40%	0:04:29 seg	360 grados
-30%	30%	0:05:99 seg	360 grados
-30%	20%	0:08:57 seg	360 grados
-30%	10%	0:16:17 seg	360 grados

Tabla 4.15: vuelta derecha 360 grados EV3M L1 = - 30 y L2 = +

Tabla de Caracterización de robot móvil LEGO EV3M maestro diferencial una vuelta a la izquierda de 360 grados con los motores L1 se mantendrá con un valor de -30% y L2 modificara su valor de 100% a 10%.

Porcentaje de motor L1	Porcentaje de motor L2	Tiempo	Distancia
-30%	100%	0:01:74 seg	360 grados
-30%	90%	0:01:92 seg	360 grados
-30%	80%	0:02:09 seg	360 grados
-30%	70%	0:02:76 seg	360 grados
-30%	60%	0:03:10 seg	360 grados
-30%	50%	0:03:51 seg	360 grados
-30%	40%	0:04:28 seg	360 grados
-30%	30%	0:05:97 seg	360 grados
-30%	20%	0:08:56 seg	360 grados
-30%	10%	0:16:16 seg	360 grados

Tabla 4.16: vuelta izquierda 360 grados EV3E L1 = - 30 y L2 = +

4.3Trayectoria evasión de obstáculos

El sensor de color EV3 contiene dos piezas principales de hardware;

1. El color de detección foto-resistencia
2. Un proyector LED.

El sensor de color es capaz de medir ya sea luz ambiente en general o específicamente la luz roja reflejada y puede identificar 8 tipos de colores. El proyector LED puede ser encendido o apagado y colores específicos del LED se puede controlar de forma independiente. Una pared de plástico se incluye entre el proyector LED y el sensor de color para evitar el ruido del proyecto. Una representación del sensor de proporciona en la figura 4.7.

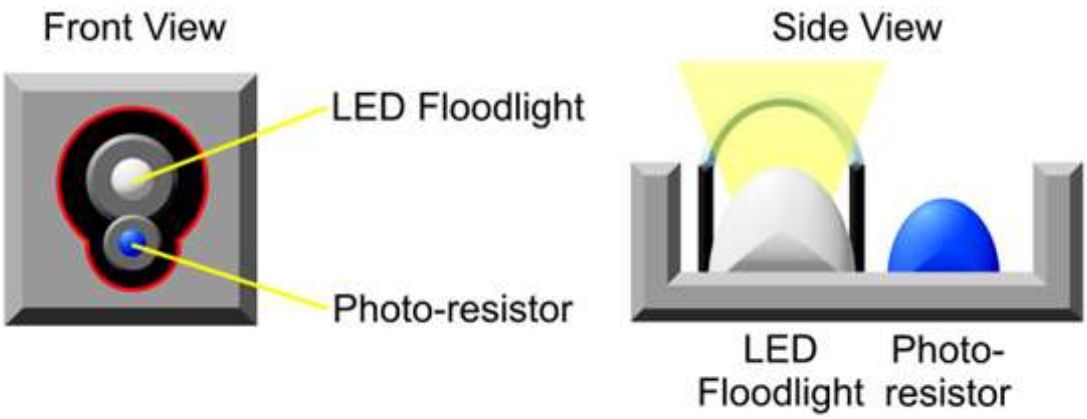

Figura 4.7: Representación de sensor de luz internamente

Además, cuando se utiliza el hardware del sensor de color, por lo general es importante mantener el sensor lo más cerca posible del objeto que se está leyendo. El sensor lleva un promedio de todo el campo de visión y para tener resultados más precisos será deseable mantener este campo en la mínima visa posible.

En el programa de EV3 brick permitirá el control de los sensores y servomotores conectados a él. Se realizaran varios paquetes se describen a continuación:

- **AppRobot_Main:** Este paquete contendrá la clase Main, la misma que permitirá iniciar los objetos a partir de las clases que se encuentren trabajando en el paquete.

- **AppRobotServer.control.servos:** este paquete contendrá las clases necesarias que permitirán regular la velocidad de los dos servomotores y también el control de los mismos.

  Se tiene la clase de Motor del paquete de lejos.ev3. Esta clase es una abstracción de los motores EV3 y proporciona una interfaz para cada puerto a los que se conectan los motores, los cuales son: Motor.A, Motor.B. Algunos de los métodos que permiten el control de los motores se muestran en la Tabla 4.17

Métodos	Descripción
void backward()	Provoca que el motor rote hacia atrás.
void forward()	Provoca que el motor rote hacia adelante.
void stop()	Provoca que el motor se detenga.
void setSpeed(int velocidad)	Establece la velocidad el motor en grados por segundo. La máxima velocidad es 900 con 8 V.
void rotate(int ang)	Provoca que el motor rote en un ángulo (*ang*) especifico.
int getTachoCount()	Obtiene la cuenta del tacómetro.
void resetTachoCount()	Pone la cuenta del tacómetro en cero.

Tabla 4.17: Métodos para el control de motores EV3

Algunos métodos para el control de motores ev3

- **AppRobotServer.utileria.pantalla:** Las pantallas que permitirán la interacción del usuario con EV3M y el EV3E se encontrara en este paquete.
- **AppRobotServer.control.sensores:** Dentro de este paquete se encontrara la clase que permitirá el control de sensores de luz.

  Dentro del paquete se empela la clase LigthSensor del paquete lejos.EV3

Se tiene dos construcciones:

LightSensor (SensorPort puerto)

LightSensor (SensorPort, boolean foco)

El parámetro de entrada puerto puede tomar el valor de las siguientes constantes que representan a los cuatro puertos EV3 Brick donde se conectan los sensores:

- SensorPort.S1.
- Sensor.Port.S2.
- Sensor.Port.S3.
- Sensor.Port.S2.

El parámetro de foco permite encender o apagar el led del sensor dependiendo si tiene valor de true o false.

El método que permite obtener la lectura del sensor de luz:

Int readValue()

Para demostrar el funcionamiento del sensor de luz y el funcionamiento del robot móvil Lego EV3 en forma diferencial. Se diseñó una trayectoria, la cual contenía cuatro obstáculos de diferentes colores. Cada obstáculo contiene un diferente color como el rojo, verde, amarillo y azul, ya que estos colores son algunos que puede detectar el sensor de luz del Robot móvil Lego EV3.

El robot llevara una velocidad constante en dirección recta, si el encuentra un obstáculo de color rojo y el robot de detendrá, girara hacia su derecha así evitando una colisión. El robot móvil seguirá avanzando si encuentra otro obstáculo el girara de nuevo hacia la derecha, y seguirá en línea recta hasta encontrar otro obstáculo de diferente color. Al encontrar otro obstáculo el de un color distinto el girara hacia la derecha y después continuara avanzando en la misma dirección, hasta encontrar la dirección de salida.

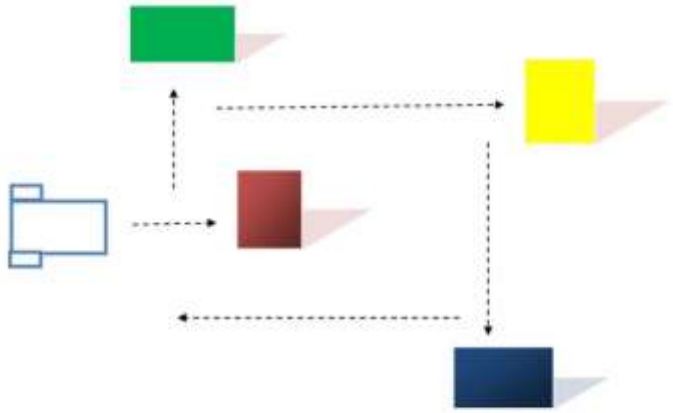

Figura 4.8: Representación de trayectoria de evasión de obstáculos

Para realizar esta trayectoria se tomó en cuenta el siguiente hardware con algunas restricciones:

- Brick EV3 (apartado)
- Servoamplificadores (apartado): Para el control de los servomotores el EV3M tiene una pequeñas restricciones que utilizaremos
  - diferencia en los motores L1 y L2, el cual el motor L1 tiene un desfase respecto al L2. Se compensara con cuando se desee hacer un movimiento hacia adelante L1 = 61% y L2 = 60%.
  - Sera necesario girar 90 en su propio eje con dirección derecha e izquierda, haciendo una interpolación a 3 puntos
  - Como el sensor Luz tiene muy poco alcance al llegar a detectar el obstáculo tendrá que retroceder L1 =- 61% y L2 = -60%

- Sensor de Luz (apartado): Los colores de los obstáculos serian diferentes, y ya que el alcance del sensor es de 15mm a 50mm, para esta trayectoria utilizaremos una distancia de 15 mm.

Se definirá un esquema organizacional que ayudara a especificar nuestra programación.

Identificación de Robot: Robot móvil EV3M
Misión de robot: Realizar una trayectoria evadiendo obstáculos.
Objetivo: Realizar una trayectoria evadiendo obstáculos para que no colisionar.
Responsabilidades: Trayectoria de evasión de obstáculos que cada uno de los obstáculos tendrá un diferente color y con ayuda del sensor evadiremos los obstáculos.
Restricciones: *Se debe asegurar que los robot no sufran colisiones entre ellos*

Tabla4.18: Esquema organizacional de robot móvil EV3M para trayectoria evasión de obstáculos

El esquema detectara que el robot EV3M seguirá una trayectoria que tendrá una serie de obstáculos de diferente color cada uno. Las cuales utilizaremos distintas clases y variables para el control de los sensores, el control de los servomotores y la pantalla para iniciar el programa. En la figura# nos muestra la secuencia que se utilizaremos para realizar la trayectoria de evasión de obstáculos. Para realizara trayectoria deseada evadiendo los obstáculos que se presenten seguirá el siguiente procedimiento:

- Se moverá el robot EV3M con el servomotor L1 = 61%, L2 = 60% hasta encontrar un obstáculo, que para esta trayectoria se optó por un de color rojo.
- Al detectar el primer obstáculo girara 90° con los servomotores L1 = -30% y L2 = 100% por 437 ms
- Avanzaran el robot EV3M derecho con los servomotores en L2 = 61%, L2 = 60% en hasta que el sensor detecte un obstáculo.
- Cuando detecte un obstáculo que en nuestra trayectoria será de color verde el robot EV3M girara en una vuelta de 90° hacia la derecha con L1 = 100%, L2 = -30% por 375 ms.
- El Robot EV3M seguirán moviéndose hacia adelante con los servomotores en L1 = 61% y L2 = 60% hasta que detecte un obstáculo.
- Detectando un obstáculo que tendrá un color amarillo giraran 90° hacía la derecha con L1 = 100%, L2 = -30%.
- Seguirán avanzando con los servomotores en L1 =61% y L2 = 60%.
- Hasta que detecte un obstáculo de color azul giraran 90° hacia la derecha con L1 = 100%, L2 = -30%.
- Completando la trayectoria avanzara hacia el punto de inicio con los motores L1 = 61% y L2 =60 por 3000 ms.

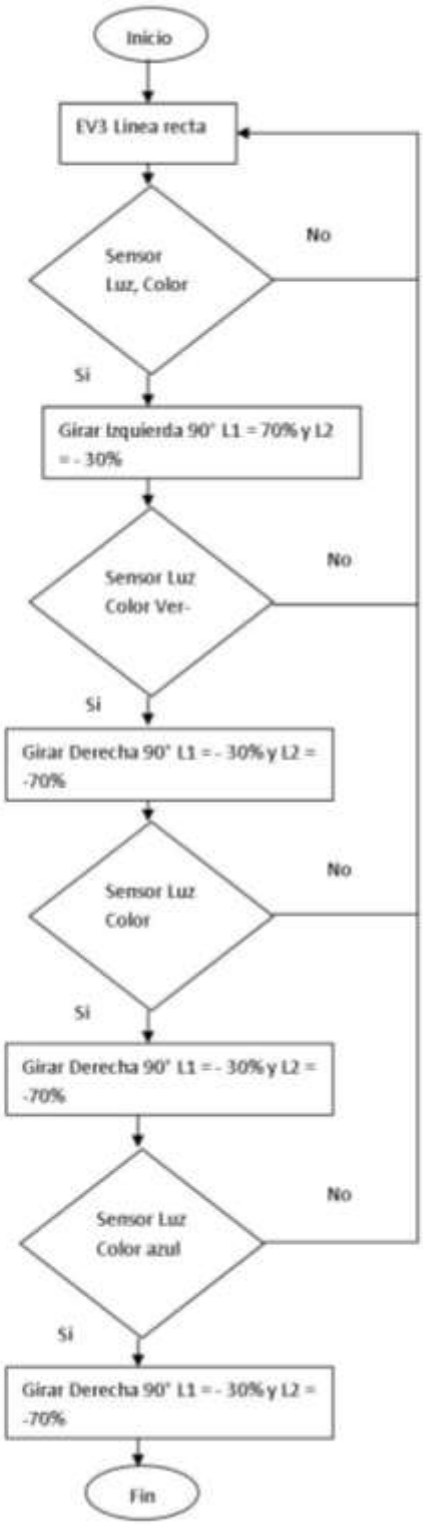

Figura 4.9 Diagrama de flujo de trayectoria evasión de obstáculos

4.4 Trayectoria Sinusoidal en topología maestro esclavo

El EV3 brick soporta comunicación inalámbrica Bluetooth mediante el un chip CSR BlueCore 4 EXT. Este chip implementa la especificación Bluetooth versión 2.1 con EDR, trabaja como un dispositivo Bluetooth de clase 2, lo que significa que puede comunicarse hasta una distancia máxima de 10 metros aproximadamente.

La comunicación se realiza mediante canales, donde un EV3 funciona como dispositivo maestro y otros EV3 se comunican cómo se muestra en la figura 4.11

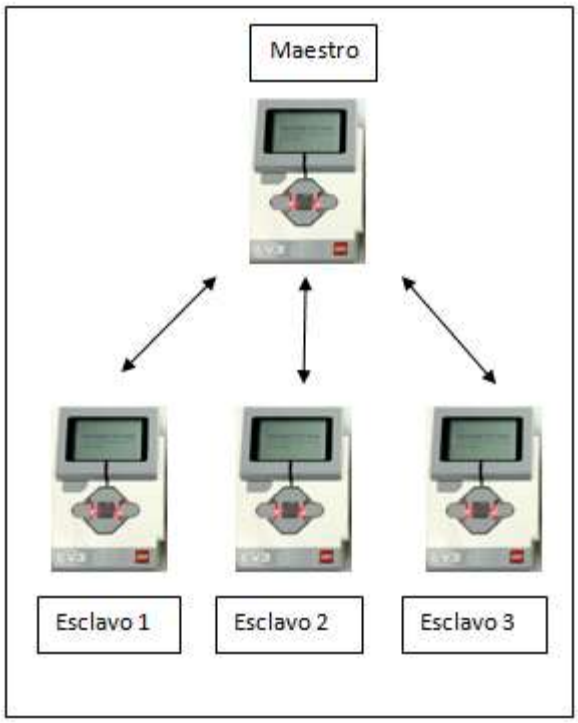

Figura 4.10: Comunicación entre EV3 usando Bluetooth

El EV3 brick puede conectarse inalámbrica con otros tres dispositivos al mismo tiempo pero solamente comunicarse con un dispositivo a la vez.

El EV3 tiene cuatro canales para comunicarse vía Bluetooth. El canal 0 es siempre utilizado por un EV3 esclavo para comunicarse con el EV3 maestro y los canales 1, 2 y 3 son para la comunicación del dispositivo maestro hacia los diferentes esclavos.

Un EV3 no es capaz de funcionar como maestro y esclavo al mismo tiempo a que esto puede causar la perdida de datos los dispositivos EV3. Pero en esta aplicación se utilizara en maestro para enviar y recibir información, que solo utilizaremos un esclavo.

4.4.1 Descripción de topología maestro – esclavo

Los mensajes del protocolo de comunicación serán intercambiados sobre el canal de comunicación full dúplex, que será configurado utilizando el perfil SPP (*Serial Port Profile*) de la tecnología inalámbrica Bluetooth versión 2.1 con EDR. El cual se tendrá un enlace de velocidad efectiva síncrona 108.8 Kbp y dependiendo del tamaño del mensaje que se va a transmitir.

El protocolo de comunicación será orientado a la conexión para la transmisión de la información y ofrecerá los siguientes servicios:

- Preservación de secuencia
- Sincronización de unidades de datos
- Control de flujo

El protocolo de comunicación deberá ser capaz de enviar un comando a la vez y limitar el número de retransmisiones del temporizador *flush tiemout*. Por lo tanto utilizara para el control de flujo el método *stop and wait.*

4.4.2 Planteamiento de protocolo de comunicación en JAVA.

En la implementación estará dividido en dos fases los cuales en el maestro y el esclavo será similar su programa. Ya que su movimiento será en forma sinusoidal, los dos empezaran su trayectoria paralelamente. Así que realmente está constituido por el mantenimiento individual de cada uno de los robots del sistema. La funcionalidad de los robots se puede resumir en la siguiente secuencia de etapas que deben ser tenidas en cuenta en el proceso global.

1. El protocolo de comunicación
2. La programación de los robots maestro (EV3M) y esclavo (EV3E)

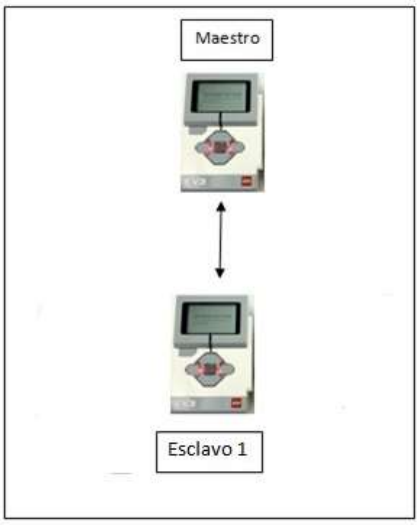

Figura 4.11: Comunicación en topología Maestro-Esclavo

El protocolo de comunicación intercambiara mensajes con el robot móvil EV3M y el robot móvil EV3E a través de canales internos síncronos, los mismos que simulan a los métodos Java.

Dentro de los métodos, el usuario podrá enviar 4 mensajes hacia el protocolo de comunicación que son:

- **pCon :** Permite al usuario realizar una petición de conexión al protocolo de comunicación.
- pTx: permitirá al usuario realizar una petición de transmisión de comandos al protocolo de comunicación.
- pFinCon: Permite al usuario realizar una petición de finalización de conexión.
- Msg: Corresponde al mensaje que desea enviar el maestro, el cual este mensaje representa un mensaje de ejecución de comandos.

El Robot EV3E será capaz de comunicarse con el protocolo de comunicación utilizando los siguientes mensajes:

- pTx: permite el robot realizar una petición de transmisión de acuses de recibido al protocolo de comunicación.
- ack: corresponde a un ACK que desea enviar EV3E luego de haber recibido un msg correcto.
- Nack: corresponde a un NACK que desea enviar el EV3E luego de haber recibido un msg incorrecto.

Las reglas y procedimientos para el protocolo de comunicación son:

- El usuario debe iniciar el servicio (iniciar conexión, como trayectoria)
- El usuario EV3M solicitara el establecimiento de la conexión
- Una vez establecida la conexión, el usuario EV3M podrá transmitir mensajes hacia el EV3E, el protocolo de comunicación finalizara e informara al usuario EV3M del acontecimiento.
- Si el usuario EV3M transmite un mensaje y llega correctamente el EV3E, este enviara un ACK, caso contrario enviara un NACK.
- Si no se recibe un ACK luego de haber realizado todas las retransmisiones posibles, el protocolo de comunicación informara al usuario del acontecimiento y finalizara la conexión.
- El EV3E podrá enviar mensaje de acuse de recibido cuando arranque el programa. Una vez realizada la lectura, esta procederá avanzar con la trayectoria sinusoidal.
- El temporizado se activara cuando se envíen mensajes de ejecución de comando. Si llega un ACK, este temporizador se detendrá inmediatamente, caso contrario si llega NACK, un acuse de recibido no llega ninguno se realizaran la transmisión hasta que el usuario elimine la conexión si es necesario.
- El usuario EV3M podría finalizar la conexión en cualquier instante, ya sea desde el EV3M o el EV3E. En cualquier de los casos, el protocolo de comunicación informara el acontecimiento.

Los procesos que intervienen en el modelo de validación del protocolo de comunicación para describir las reglas y procedimientos del mismo son las siguientes:

- RXR/RXM
  - Permiten la recepción de los datos en el EV3M y el EV3E respectivamente.
  - Si RXR recibe un mensaje, instantáneo al proceso TXR para devolver una respuesta. La respuesta se recibirá en el proceso RXM
  - RXM se encarga de simular el método de control de flujo *stop and wait*.

- TXT/TXM
  - Son procesos que solo existen mientras se transmite un mensaje.
  - TXR es el encargado de simular el no determinismo del canal de comunicación y también él envió del arranque del programa a ejecutarse.
  - TXM es el encargado de transmitir cualquier mensaje proveniente del EV3M.

Los paquetes de lejos en Java y sus funciones se describen a continuación:

- **AppRobot_Main:** Este paquete contendrá la clase Main, la misma que permitirá iniciar los objetos a partir de las clases que se encuentren trabajando en el paquete.
- **AppRobotServer.control.servos:** este paquete contendrá las clases necesarias que permitirán regular la velocidad de los dos servomotores y también el control de los mismos.
- **AppRobotServer.utileria.pantalla:** Las pantallas que permitirán la interacción del usuario con EV3M y el EV3E se encontrara en este paquete.

Los mensajes del protocolo de comunicación utilizaran un formato de mensaje con cabecera y colas, utilizando Bluetooth. Los mensajes se clasifican en:

- Mensajes de datos: que pueden ser de ejecución de comando, color, y fin de conexión.
- Mensaje de control: que pueden ser de acuses de recibido positivos (ACK) o negativos (NACK) para cada uno de los tipos de mensajes de datos.

La información que conformara la cabecera estará definida por las siguientes criterios y se muestran en la figura#

- Un bit Tp para identificar el tipo de mensaje.
- Un bit de Sec para establecer el número de secuencia del mensaje para el control de flujo.
- Un bit A/N para identificar si se trata de un ACK o un NACK.
- Dos bits Clase para identificar el tipo de mensaje de control.
- Un bit Tx para identificar el origen del mensaje, debido a que un mensaje de datos podrá ser utilizado para distintos propósitos, dependiendo de quién envié el mensaje.
- Un bit F para indicar cuándo se debe finalizar la conexión.

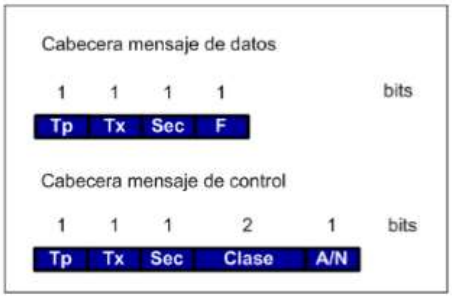

Figura 4.12: Cabecera de los mensajes de datos y de control del protocolo de comunicación

Los valores de los bits que conforman la cabecera del protocolo de comunicación y su descripción se muestran en la tabla 4.19 y 4.20.

Bit	Valor	Descripción
Tp	0	Mensaje de datos.
	1	Mensaje de control.
Tx	0	Transmite el teléfono celular.
	1	Transmite el brazo robot.
Sec	0	Número de secuencia.
	1	Número de secuencia.
F	0	No finaliza la conexión.
	1	Finaliza la conexión.
Clase	00	*NACK*.
	01	*ACK* de comando
	10	*ACK* de fin de conexión.
	11	*ACK* de mensaje de color.

Tabla 4.19: Valores de los bits de la cabecera para el protocolo de comunicación

Valores y descripción de los bits que conforman la cabecera del protocolo de comunicación.

Bit	Valor	Descripción
A/N	0	*ACK*
	1	*NACK*

Tabla 4.20: Valores de bits de la cabecera

Requerimientos necesarios a considerar

Para la implementación hay ciertos requerimientos, además de consideraciones:

- El protocolo de comunicación debe realizar varias tareas simultaneas, por lo cual, se debe recurrir a la programación concurrente. Java utilizara procesos ligeros para permitir la creación de sistemas multitarea, por tal razón, se deben tener cuidado ya que estos comparten recursos entre si de una manera no determinística.
- Todos los hilos de ejecución que se produzcan debido al protocolo de comunicación debe ejecutarse una sola vez durante el ciclo.
- El protocolo de comunicación será implementando basado en el modelo OSI, el mismo que permite el uso de procesos pares entre ellos ya que los dos utilizaran librerías Java comunes.
- El acceso a los servicios del SAP debe ser inmediato e independiente de la clase para facilitar la implementación de comunicación.

El protocolo de comunicación deberá ser diseñado tomando en cuenta principalmente la propiedad de modularidad. Cada servicio que este ofrecerá deberá corresponder a una clase o a un método, dependiendo la flexibilidad para futuros cambios.

En la figura 4.13 muestra cómo se podrían implementar el protocolo de comunicación. Muestra los servicios del protocolo de comunicación vía Bluetooth de los robots móvil EV3M y EV3E.

Figura 4.13: Servicios de comunicación del robot EV3

.

Cada servicio del protocolo de comunicación estará representado por una clase y estarán agrupados según se planteó anteriormente. Debido a que algunos servicios pueden ser hilos de ejecución, la clase protocolo de comunicación será un clase singleton para evitar la clonación innecesaria de hilos de ejecución.

Encapsulada en la clase singleton protocolo de comunicación se encuentra la clase PuntoAccesoServicio, la misma que se está compuesta por todos los servicios que se pueden brindar el protocolo de comunicación.

Para solicitar un servicio a la clase PuntosAccesoServicio, se necesita del manejo de las primitivas. Para implementar las primitivas se utilizó el método CubbyHole. Este método implementa un slot que puede tener solo un objeto a la vez para la comunicación entre hilos. Un hilo coloca un objeto en el slot y otro lo toma a través de dos métodos. Si otro hilo trata de poner un objeto en el slot cuando este se encuentre ocupado, este hilo se bloquea hasta que el slot se libere. De esta manera, cualquier clase podrá solicitar un servicio al SAP de una manera segura (ver el anexo).

Para la petición (colocación del objeto parecido en el slot) y confirmación (liberación del slot) de servicios utilizados el método CubbyHole se utiliza las siguientes instrucciones:

```
Protocolo.obtenerSAP().peticionServicio(byte servicio);

Boolean conf = Protocolo.obtenerSAP().confirmacion Servicio();
```

Los mensajes del protocolo de comunicación utilizaran un formato de mensaje con cabeceras y colas, debido a que se utilizaran el modulo interno Bluetooth que contiene el robot móvil LEGO EV3.

Figura 4.14: Servicio de protocolo de comunicación implementado en los robots LEGO

Se definirá un esquema organizacional que ayudara a especificar nuestra programación.

Identificación de Robot: Robot móvil EV3M
Misión de robot: Realizar comunicación entre maestro y esclavo, siguiendo una trayectoria Sinusoidal.
Objetivo: Obtener comunicación con robot móvil EV3E y realizar una trayectoria sinusoidal.
Responsabilidades: Búsqueda de dispositivo por canal 1, iniciar programa trayectoria sinusoidal.

Restricciones: *Se debe asegurar que los robot no sufran colisiones entre ellos*

Tabla 4.21: Esquema de propósito general para trayectoria sinusoidal en robot móvil maestro

Identificación de Robot: Robot móvil EV3E
Misión de robot: Realizar comunicación entre maestro y esclavo, siguiendo una trayectoria Sinusoidal.
Objetivo: Obtener comunicación con robot móvil EV3M y realizar una trayectoria sinusoidal.
Responsabilidades: Búsqueda de dispositivo por canal 1, iniciar programa trayectoria sinusoidal.
Restricciones: *Se debe asegurar que los robot no sufran colisiones entre ellos*

Tabla 4.22: Esquema de propósito general para trayectoria sinusoidal dé robot esclavo

Con los esquemas podemos detectar que el propósito general de es similar en el Robot maestro y el Robot esclavo. Las cuales se describirán cada una de las clases y variables que se utilizaran para realizar la comunicación. El la figura# y figura# nos muestra la secuencia que se utilizaremos para realizar la comunicación con una topología maestro-esclavo. Después de encontrar la conexión se realizara una trayectoria sinusoidal, que consta con el movimiento en forma de onda sinusoidal, ya que al iniciar el programa se ejecutara él envió de mensaje del Robot móvil maestro EV3M hacia el robot móvil esclavo EV3E. El motor L 1 = 0%, L2 = 100% por 1000 ms, y avanzar L1 = 61% L2 = 60% en 1000 ms, después dar una vuelta 180 hacia la derecha con L1 = 100%, L2 = 10% por 2000 ms. Quedando en pendiente de 45°, la cual seguirá derecho hasta con L1 = 61%, L2 = 60% por 3000 ms. Al terminar girara 180° con L1 = 10%, L2 = 100% por 2000 ms, y así repitiendo de la secuencia para el robot móvil esclavo EV3E, como se muestra en la figura#

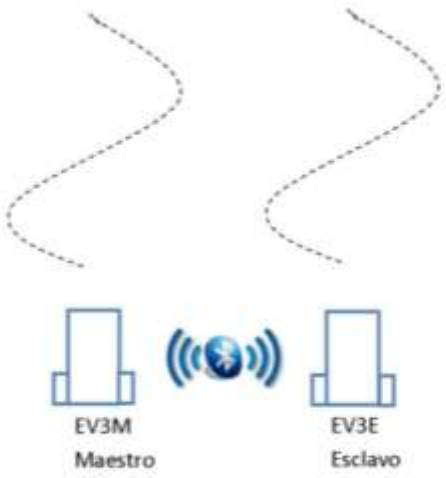

Figura 4.15: Representación de trayectoria sinusoidal

Figura 4.16: Diagrama de flujo de comunicación para el robot móvil maestro

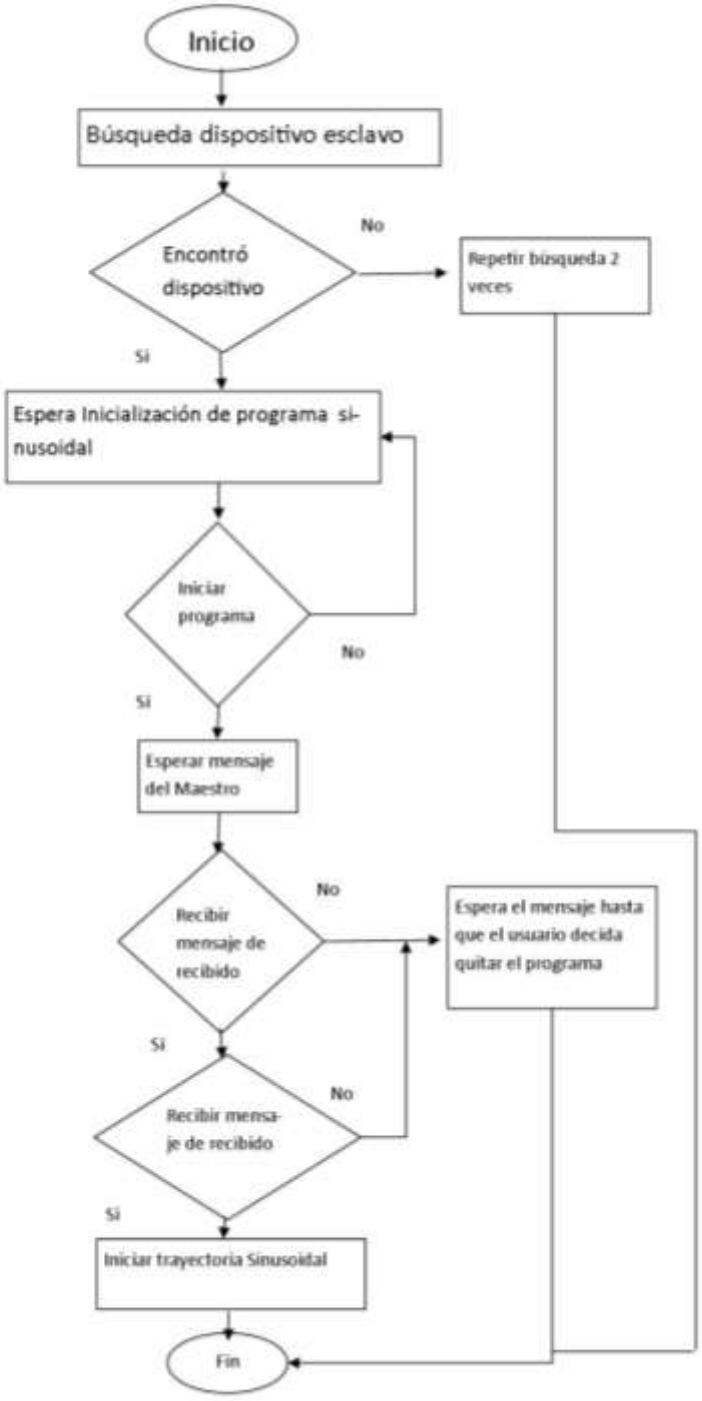

Figura 4.17: Diagrama de flujo de comunicación para el robot móvil esclavo

Valores y descripción de los bits que conforman la cabecera del protocolo de comunicación.

La carga útil está definida por el siguiente criterio.

- Se escogen 5 bits para indicar comandos de ejecución, los cuales para la trayectoria se utilizaran solo un bit y los otros se utilizaran para próximos programas. Cuando los cinco bits se encuentren en cero, significa que se trata de un mensaje de finalización de la conexión.

La figura 4.23 muestra la carga útil de los mensajes de datos, que en esta trayectoria solo se utilizara un bit y en la tabla 4.23 muestra los valores que corresponden a los otros programas tomando la carga útil.

Carga útil de los mensajes de datos del protocolo de comunicación.

Bit	Valor	Descripción
Trayectoria Sinusoidal	01	Inicio de trayectoria
Trayectoria en forma de 8	10	Inicio de trayectoria
Trayectoria de posiciones diferentes	11	Inicio de trayectorias
detección de EV3	0	No detección EV3
	1	detección de EV3

Tabla 4.23: Carga útil de mensajes de datos del protocolo de comunicación

Valores y descripción de los bits que conforman la carga útil del protocolo de comunicación

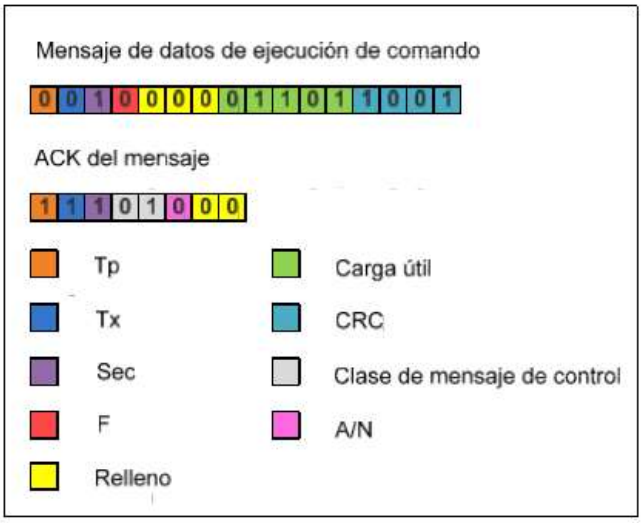

Figura 4.18: Ejemplo de dos tipos de mensaje del protocolo de comunicación

4.5Trayectoria Ocho en topología maestro esclavo

Se desarrolló el mismo protocolo de comunicación con la diferencia en la selección de la carga útil, lo cual se pretende él envió del mismo mensaje para todos los programas, modificando dos bits al enviar el mensaje utilizando el valor 10 en vez del 01 como en la trayectoria sinusoidal como muestra en la tabla 4.24. Se realizara la misma comunicación con las mismas clases las cuales solo cambiaran la trayectoria.

Bit	Valor	Descripción
Trayectoria Sinusoidal	01	Inicio de trayectoria sinusoidal
Trayectoria en forma de 8	10	Inicio de trayectoria en forma de 8
Trayectoria de posiciones diferentes	11	Inicio de trayectorias
detección de EV3	0	No detección EV3
	1	detección de EV3

Tabla 4.24: Carga útil de mensajes de datos del protocolo de comunicación pero señala la trayectoria a utilizar

Valores y descripción de los bits que conforman la carga útil del protocolo de comunicación.

Para la implementación hay ciertos requerimientos, además de consideraciones:

- El protocolo de comunicación debe realizar varias tareas simultaneas, por lo cual, se debe recurrir a la programación concurrente. Java utilizara procesos ligeros para permitir la creación de sistemas multitarea, por tal razón, se deben tener cuidado ya que estos comparten recursos entre si de una manera no determinística.
- Todos los hilos de ejecución que se produzcan debido al protocolo de comunicación debe ejecutarse una sola vez durante el ciclo.
- El protocolo de comunicación será implementando basado en el modelo OSI, el mismo que permite el uso de procesos pares entre ellos ya que los dos utilizaran librerías Java comunes.
- El acceso a los servicios del SAP debe ser inmediato e independiente de la clase para facilitar la implementación de comunicación

Se definirá un esquema organizacional que ayudara a especificar nuestra programación.

Identificación de Robot: Robot móvil EV3M
Misión de robot: Realizar comunicación entre maestro y esclavo, siguiendo una trayectoria en forma de ocho.
Objetivo: Obtener comunicación con robot móvil EV3E y realizar una trayectoria forma de ocho.
Responsabilidades: Búsqueda de dispositivo por canal 1, iniciar programa trayectoria forma de ocho.
Restricciones: *Se debe asegurar que los robot no sufran colisiones entre ellos*

Tabla 4.25: Esquema de propósito general para trayectoria forma ocho en robot maestro

Identificación de Robot: Robot móvil EV3E
Misión de robot: Realizar comunicación entre maestro y esclavo, siguiendo una trayectoria forma de ocho.
Objetivo: Obtener comunicación con robot móvil EV3M y realizar una trayectoria en forma de ocho.
Responsabilidades: Búsqueda de dispositivo por canal 1, iniciar programa trayectoria en forma de ocho.
Restricciones: *Se debe asegurar que los robot no sufran colisiones entre ellos*

Tabla 4.26: Esquema de propósito general para trayectoria forma ocho en robot esclavo

Con los esquemas podemos detectar que el propósito general de es similar en el Robot maestro y el Robot esclavo. Las cuales se describirán cada una de las clases y variables que se utilizaran para realizar la comunicación. El la figura# y figura# nos muestra la secuencia que se utilizaremos para realizar la comunicación con una topología maestro-esclavo. Después de encontrar la conexión se realizara una trayectoria en forma de ocho, que consta con el siguiente procedimiento:

- Se moverá el robot EV3M y EV3E con el servomotor L1 = 0%, L2 = 100% por 1000ms

- Avanzaran el robot EV3 M y EV3E en L2 = 61%, L2 = 60% en 1000 ms.
- Giraran el robot EV3M y EV3E en una vuelta de 180° hacia la derecha con L1 = 100%, L2 = 10% por 2000 ms.
- Quedando en una pendiente de 45°, que seguirán con los servomotores en L1 = 61% y L2 = 60% por 3000 ms.
- Giraran 360° con L1 = 10%, L2 = 100% hasta quedar en un pendiente de 45°.
- Al quedar en una pendiente de 45°, seguirán los servomotores en L1 =61% y L2 = 60%
- Giraran 360°con L1 = 100%, L2 = 10%, haciendo una ocho completo

Como se muestra en la figura 4.19.

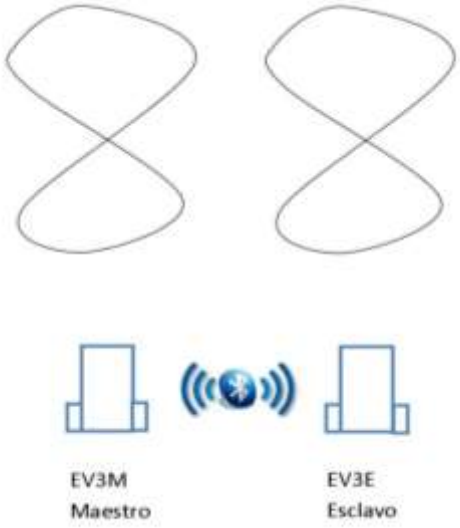

Figura 4.19: Trayectoria en forma de ocho

Las reglas y procedimientos para el protocolo de comunicación son las mismas que en la trayectoria sinusoidal:

- El usuario debe iniciar el servicio (iniciar conexión, como trayectoria)
- El usuario EV3M solicitara el establecimiento de la conexión
- Una vez establecida la conexión, el usuario EV3M podrá transmitir mensajes hacia el EV3E, el protocolo de comunicación finalizara e informara al usuario EV3M del acontecimiento.
- Si el usuario EV3M transmite un mensaje y llega correctamente el EV3E, este enviara un ACK, caso contrario enviara un NACK.
- Si no se recibe un ACK luego de haber realizado todas las retransmisiones posibles, el protocolo de comunicación informara al usuario del acontecimiento y finalizara la conexión.
- El EV3E podrá enviar mensaje de acuse de recibido cuando arranque el programa. Una vez realizada la lectura, esta procederá avanzar con la trayectoria sinusoidal.
- El temporizado se activara cuando se envíen mensajes de ejecución de comando. Si llega un ACK, este temporizador se detendrá inmediatamente, caso contrario si llega NACK, un acuse de recibido no llega ninguno se realizaran la transmisión hasta que el usuario elimine la conexión si es necesario.
- El usuario EV3M podría finalizar la conexión en cualquier instante, ya sea desde el EV3M o el EV3E. En cualquier de los casos, el protocolo de comunicación informara el acontecimiento.

4.6 Sincronía de dos robots utilizando sensor de proximidad en topología maestro esclavo.

En este ejemplo consiste en desarrollar un una sincronización de movimientos entre dos robots situados en un determinado entorno, con un movimiento distinto en cada uno de los robot móvil lego EV3M y EV3E. Comunicándose entre sí por medio de Bluetooth y utilizando el sensor de a proximidad para saber la ubicación de cada uno. Como se muestra en la figura 4.20.

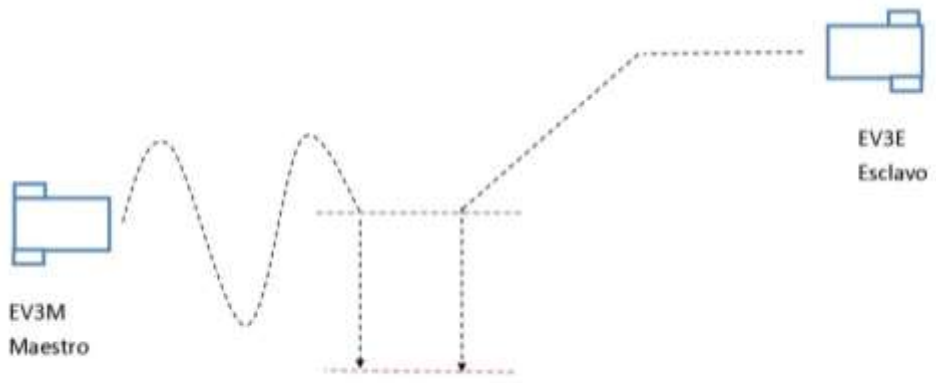

Figura 4.20: Trayectoria de sincronía de dos robots móviles en diferente punto y con diferente trayectoria

En el ejemplo se plantea el desarrollo de un programa (Anexo 5) para el control necesario de los robots móviles en configuración diferencial LEGO EV3, como se explicaba el movimiento en el los capítulos 2 y capitulo 3. La programación se basa en la comunicación entre dispositivos y los componentes básicos para la sensorización, controlados por los brick de los robots LEGO EV3. El componente hardware que se utilizara son los siguientes.

- Brick: *brick* (ladrillo) con una capacidad de almacenamiento de memoria 64MB e incluye cuatro puertos de entradas utilizados para la conexión de sensores y cuatro puertos de salidas usados para interconectar los motores (cada motor tiene embebido un sensor de rotación), los cuales habilitan la movilidad del robot
- Servo Motores: Los motores EV3 son servos, esto significa que su posición interna y estado puede ser controlado por una unidad externa, en este caso es controlado por el Brick EV3. Estos motores se caracterizan por permitir movimientos precisos y controlados mediante el sensor de rotación, así como una sincronización de tiempo de ejecución de una acción (rotación) con otros motores.

Figura 4.21: Actuadores de robot EV3

- Sensor de a proximidad: Permite la detección de objetos y determina a que distancia se encuentran. El rango de detección esta entre 3cm y 250 cm. Tiene una precisión de +/- 1 cm, para tomar las medidas este sensor utiliza señales sonoras de alta frecuencia que se reflejan en un objeto. El sensor toma la señal de vuelta para la medición. Como se muestra en la siguiente imagen.

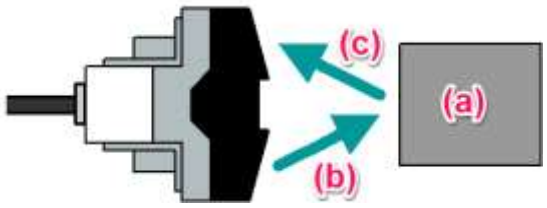

Figura 4.19: representación del sensor de proximidad

Este ejemplo trata de especificar la sincronización los cuales el robot maestro EV3M y el esclavo EV3E llegan a en distintas condiciones iniciales en el plano cartesiano. En este caso los robots tienen un objetivó en común y es converger hacia el mismo punto. Podemos identificarlos en dos clases, un robot jefe, que puede ser el encargado de controlar el proceso de distribución de órdenes y el robot subordinado.

En este caso resulta evidente la especificación de los roles jefe y subordinado, desarrollando un esquema de propósitos de organizacional. Este esquema se divide en maestro y esclavo que contiene En las tabla 4.27 y 4.28 quedara reflejado el esquema organizacional el maestro y el esclavo, que nos ayudara a especificarlas en nuestra programación.

- La identificación robot
- Misión de robot
- Objetivos
- Responsabilidades
- Restricciones

Identificación de Robot: Robot móvil EV3M
Misión de robot: El principal propósito del robot móvil es poder realizar una formación de robot en líneas con el robot EV3E.
Objetivo: llegar a una formación paralela con el EV3E

Responsabilidades: Tareas para llegar a la formación, arrancando en un posición diferente, avanzando en línea recta por 2 seg. Después girar 45° y seguir avanzando. Hasta detectar al EV3E por medio de un sensor de a proximidad y formarse paralelamente con el EV3E
Restricciones: *Se debe asegurar que los robot no sufran colisiones entre ellos*

Tabla 4.27: Proposito organizacional de robot Maestro EV3M

Identificación de Robot: Robot móvil EV3E
Misión de robot: El principal propósito del robot móvil es poder realizar una formación de robot en líneas con el robot EV3M.
Objetivo: llegar a una formación paralela con el EV3M
Responsabilidades: Tareas para llegar a la formación, arrancando en un posición diferente, avanzando en modo sinusoidal. Hasta detectar al EV3M por medio de un sensor de a proximidad y formarse paralelamente con el EV3E
Restricciones: *Se debe asegurar que los robot no sufran colisiones entre ellos*

Tabla 4.28: Proposito organizacional de robot Esclavo EV3E

Analizando los diagramas desarrollados, de detecta que el programa se dividirá en varias clases, una de ellas es la clase de comunicación como ya se estaba manejando anteriormente en el apartado#. En la secuencia de control de trayectoria del maestro mostrado en la figura 4.20, después de encontrar la conexión con el esclavo, se iniciara el programa enviando en mensaje de arranque de trayectoria. Avanzara por 2 segundos en línea recta y girara por 45° con L1 = 70% y L2 = - 30%. Dirigirse en la misma pendiente hasta detectar el EV3E, con el sensor de a proximidad = 30 cm y enviar mensaje de detección de objeto al EV3E. Detenerse hasta recibir el mensaje de recibido y girar 90° para alinearse paralelamente con el Robot EV3E. Avanzar en línea recta como se muestra en la figura#.

En la figura 4.23 Muestra el diagrama de flujo que nos servirá para la programación del esclavo. Al empezar con la comunicación enviara un mensaje de recibido al EV3M para empezar la comunicación utilizando el mismo método de comunicación como en el apartado#. Así mismo arrancando el programa recibirá el mensaje para arrancar con la trayectoria y al mismo tiempo enviara el robot EV3M el mensaje de recepción de mensaje. Avanzara en forma sinusoidal hasta la detección del robot EV3M o recibir el mensaje de detección. Si detecta al robot EV3M, detenerse por 5 seg y enviar mensaje de detección al robot EV3M, alinearse con el EV3M y

arrancar en línea recta. Si recibe el mensaje de detección detenerse por 5 seg, alinearse y avanzar en línea recta.

Maestro

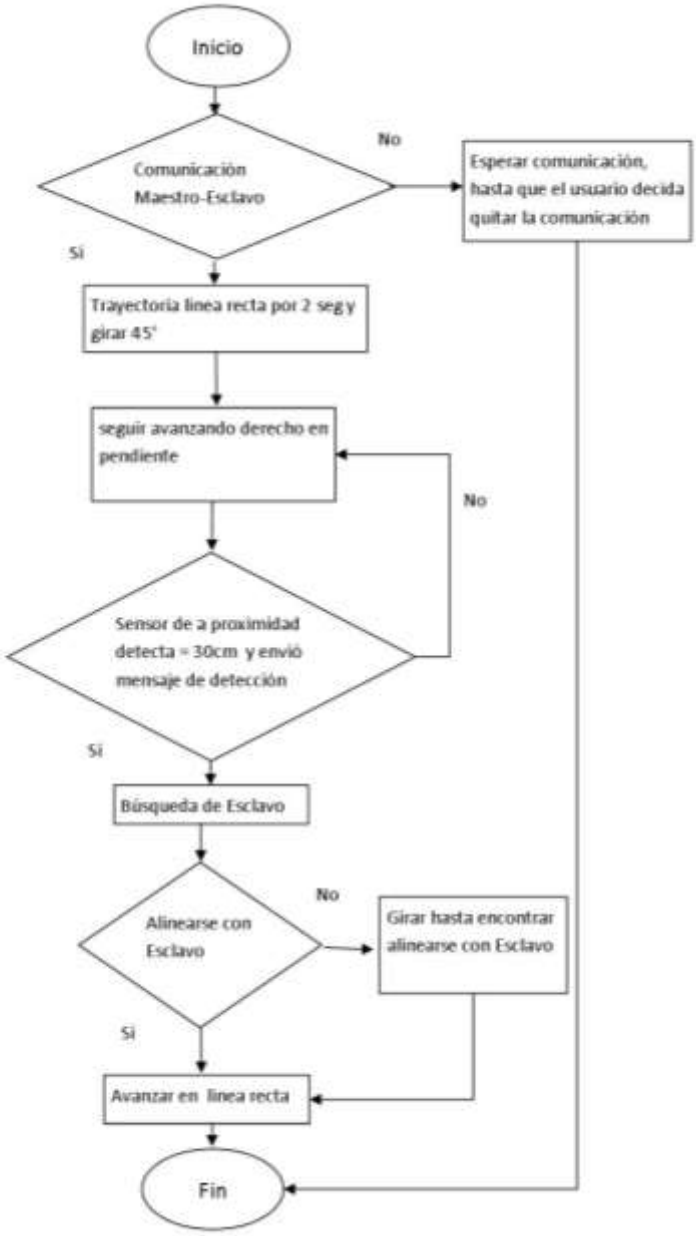

Figura 4.23: Diagrama de flujo de trayectoria síncrona de robots para maestro

Esclavo

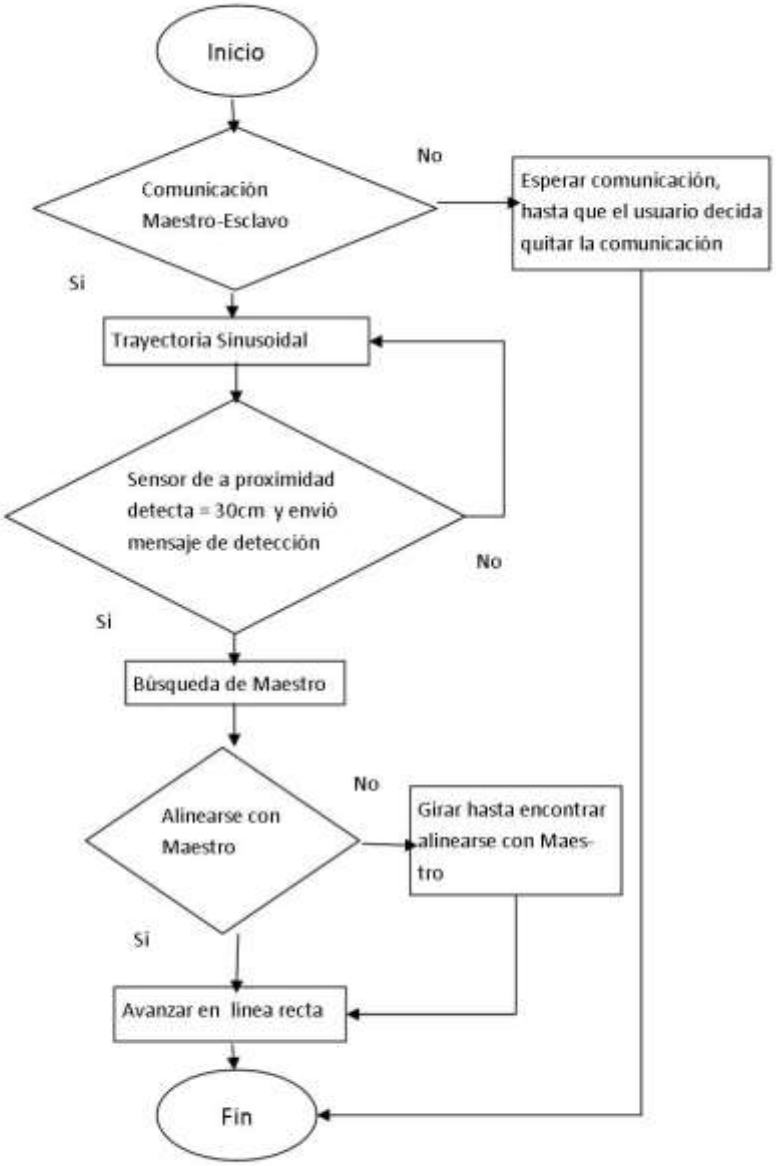

Figura 4.24: Diagrama de flujo de trayectoria síncrona de robots para esclavo

Bit	Valor	Descripción
Trayectoria Sinusoidal	01	Inicio de trayectoria
Trayectoria en forma de 8	10	Inicio de trayectoria
Trayectoria de posiciones diferentes	11	Inicio de trayectorias
detección de EV3	0	No detección EV3
	1	detección de EV3

Tabla 4.29: Carga útil de mensajes de datos del protocolo de comunicación pero señala la trayectoria posiciones diferentes

En los dos caso en el maestro y el esclavo es tener una formación y moverse en línea recta, pero viendo la integridad de cada uno de los robots evitando colisionar. Este último está realmente constituido por el mantenimiento individual de cada uno de los robots del sistema. La funcionalidad de los robots se puede resumir en la siguiente secuencia de etapas que deben ser tenidas en cuenta en el proceso global.

1. Establecimiento de conexión Maestro – Esclavo
2. Envió de mensajes con éxito
3. Desplazamiento de robots de la posición de inicio con diferente trayectoria
4. Detección de Robot
5. Alineación de robots
6. Avanzar en línea recta

La relación existente entre el hardware de los robots móviles para el programa, atreves del diagrama secuencial como se ha estado usando (apartado#), donde se indica él envió de y recepción de paquetes de datos y los plazos temporales entre dichos envíos. Mas él envió de la retención de robot, las cuales las unidades de tiempo absoluto dado por el reloj de sistema. Así por ejemplo, se expresa que entre dos envíos consecutivos de información por parte del hardware robot transcurren 100 ms y que el tiempo que debe pasar desde que se recibe un paquete de información hasta que se envié un paquete de ordenes no debe superar dichos 100 ms.

Evento	Descripción	Patrón de llegada	Tiempo mínimo
dec_robot_ev3m	Detectar robot	Periódico	300 ms
dec_robot_ev3e	Detectar robot	Periódico	300 ms
ini_tra_ev3m	Inicio trayectoria maestro	Periódico	300 ms
Ini_tra_ev3e	Inicio trayectoria esclavo	Periódico	300 ms

Tabla 4.30: Secuencia de etapas para detección de robot

4.7 Aplicaciones

Una vez que hemos barrido los fundamentos teóricos y los conceptos más interesantes de los sistemas robóticos vamos a hacer un repaso a las aplicaciones típicas que se abordan con grupos de robots. Por cada una de ellas identificaremos los conceptos más representativos.

4.7.1 Recolección o foraging

La recolección consiste en encontrar y recoger una serie de objetos diseminados por el entorno. En situaciones reales estas técnicas se pueden usar para recoger sustancias toxicas, participar en situaciones de rescate, detectar minas, etc. Los mecanismos de cooperación en estos entornos pueden ser variados, desde repartir los robots por zona disjuntas, hasta construir cadenas con ellos para recolectar los objetos. Incluso pueden repartirse aleatoriamente y que emerja un comportamiento colectivo. Los equipos de robots para recolección suelen ser homogéneos y el control es totalmente distribuido. En cuanto al tipo de comunicación empleado la habitualidad es utilizar interacción sensorial para evitar colisiones. Los sistemas más avanzados emplean comunicaciones para hacer un mejor reparto de zona a explorar y optimizar el tiempo empleado.

Figura 4.25: Robot Sally y Shannon (izquierda) pertenecientes a Georgia Institute of Technology participando en la competencia de saqueo Find life on Mars (derecha) del AAAI-97 Robot Competion.

La figura 4.1 muestra instantes de la cooperación Find Life on Mars [26], inspirada en la exploración de Marte. Los robots participantes deben recolectar objetos y pelotas específicamente coloreados y depositarlos en una zona común. Los robots *Sally y Shannon* fueron programados por *George Institute of Tecnology* para abordar problemas desde un punto de vista cooperativo. Los dos robots estaban gobernados por varias capas de control. La capa de bajo nivel, mas reactiva, se encargaba de implementar los comportamientos reactivos como por ejemplo es-quitar_obstaculos o ir_a_objetivo. El resultado final era un comportamiento emergente donde los dos robots no comunicaban explícitamente entre sí pero si colaboraban minimizando el tiempo de recolección.

4.7.2 Formaciones

Esta especialidad aparece cuando se combina la navegación y los grupos de robots. Consiste en el control de múltiples individuos moviéndose juntos en formación. Los trabajos en el ámbito de las formaciones en vehículos no tripulados despertaron un interés militar en el Ministerio de Defensa de Estados Unidos durante la década de los 90. El control de las formaciones de vehículos resulta interesante para realizar funciones de escolta o exploración. Un ejemplo de formaciones lo podemos encontrar en [10]. El método propuesto usa sensores locales y no externos y asigna a uno de los robots el rol de líder, por lo tanto es un sistema centralizado. Al haber un robot actuando como líder, el equipo es por definición heterogéneo, pues los roles diferenciados de cada tipo de robot marcan la heterogeneidad. Es este caso concreto la cooperación se produce mediante interacciones sensoriales detectando la distancia y orientación entre los vehículos. El robot que juega el papel de líder se mueve según un planificador de trayectorias disponibles. El objetivo es mantener una distancia aproximada entre las parejas de robots y que todos tengan una orientación determinada. Los parámetros involucrados en las decisiones son la distancia consiguen dar los comandos de movimiento adecuados a cada uno. Mediante cálculos matemáticos se consiguen dar los comandos e movimiento adecuados a cada robot para que la formación y pasar a formas más lineales para evitar el contacto, etc.

Un segundo ejemplo que se adecua a la aplicación de formación en un proyecto desarrollado en *Georgia Tech Mobile Robot Laboratory*, utilizando vehículos terrestres no tripulados. En la figura 4.4 se muestra un convoy real de vehículos durante una demostración, así como las cuatro formaciones clásicas más empleadas (línea, columna, diamante y cuña). Es este caso, uno de los vehículos actuaban de líder y no tenía que preocuparse de mantener la formación. Esto nos indica que el sistema es heterogéneo. Los vehículos estaban equipados con GPS y esto permitía conocer con realidad exactitud la posición de cada uno. En cuanto al mecanismo de comunicación el proyecto desarrollo varias técnicas siendo las más empleadas y basadas en comunicación explicita.

Figura 4.26: Equipo de cuatro camiones militares propiedad de DARPA durante el proyecto *Demo II Project* (izquierda) y formaciones en línea, columna, diamante y cunia (derecha)

4.7.3. Gestión de almacenamiento.

En grandes centros de distribución, almacenes o pequeños negocios con gran volumen de movimiento de productos es habitual dedicar mucho tiempo a la gestión del inventario. Esta gestión involucra tareas de llenado de determinados compartimientos o vaciado cuando los clientes lo solicitan. Un equipo de robots cooperantes puede ser una muy buena solución para este problema, liberando a las personas de esta tarea repetitiva y mejorando la eficiencia de la gestión de los recursos. Cuando disponemos de varios agentes compartiendo el mismo miedo también pueden surgir colisiones. Este problema puede interpretarse como un problema de acceso a un recurso (carreteras, aire, etc.) y se puede resolver mediante reglas, prioridades o comunicación. Desde punto de vista también puede verse como un problema de plantificación de movimientos entre varios robots.

Figura 4.27: Robots de *Kiva Systems* (Izquierda) e Robots puesto en marcha en instalaciones en Denver (derecha) donde los robots realizan trabajos de almacén.

La empresa *Kiva Systems* [4], [15] ofrece al mercado de los centros de distribución un equipo de cientos robots formando un enjambre (*swarm*) para encargarse del inventario. Esta empresa afianzada en Boston ha desarrollado un sistema que permite transportar estantería de carga a diferentes sitios de un almacén de manera coordinada. *Kiva Systems* no ha publicado datos concretos sobre la tecnología de control de los robots, ni de su diseño mecánico o su algoritmo de asignación de recursos. Sin embargo, los resultados que han obtenido hablan por sí solos y se podría decir que representan la punta de lanza en cuanto a resultados reales en esta área se refiere.

4.7.4. Manipulación coordinada

Existen muchos trabajos sobre manipulación coordinada el cual consiste en arrastrar cajas usando robots con brazos, o de manera más genérica, manipular objetos entre varios agentes. Un dato interesante de esta especialidad es que la cooperación puede llevarse a cabo sin que cada robot sepa de la existencia del resto. Algunos trabajos interesantes en esta línea son [27] y [2].

En la figura 4.6 podemos observar los experimentos realizados por Joel M. Esposito en [9], donde plantea la cuestión del posicionamiento de nuevos miembros a la hora de trasportar coordinadamente objetos. En su caso, los robots son grandes grupos de barcos autónomos a escala. Su trabajo está inspirado en la literatura actual sobre manipulación y agarre de objetos con manos robóticas. Su enfoque es afrontar el problema con un control distribuido y utilizando un equipo de robots, equipados con módulos de comunicaciones inalámbrica para intercambiar información entre ellos.

Figura 4.28: Seis barcos autónomos remolcando de manera coordinada una barcaza simulada

En el trabajo [24] se implementó dos brazos manipuladores industriales cooperan para trabajar sobre el mismo objeto. Esta necesidad surge para soportar objetos muy grandes o pesados que de manera individual no podrían sujetarse. El trabajo explora los mecanismos para trazar trayectorias compatibles con los dos manipuladores y que eviten cualquier obstáculo.

4.7.5 Futbol Robótico

EL futbol robótico es un campo de pruebas interesante que está cobrando mayor interés año a año. Además de reto tecnológico que requiere desarrollar un sistema completo perceptivo, locomotor, estratégico, que incluya autolocalización y cooperación entre sus miembros, tiene el ingrediente extra de ser competitivo. Un equipo de robots frente a otro equipo de robots con objetivos opuestos a los suyos, desde el punto de vista de coordinación es un escenario muy propicio a establecer roles entre los miembros de cada equipo (portero, defensor, delantero, etc.) con estrategias de posicionamiento y comportamientos específicos.

La percepción también se beneficia de la cooperación (como se muestra en la figura 4.9) y la estimación de los objetos interesantes del partido, como la pelota, puede realizarse de manera más precisa intercambiando información entre varios robots.

Existe otra variable en el futbol robótico con las mismas reglas pero con diferente entorno: Un entorno virtual simulado. Los simuladores como [6] proporcionan una excelente plataforma para trabajar con sistemas de robots. Simulan los limitados sistemas perceptivos de los robots, permiten comunicar información entre los miembros del equipo utilizando primitivas de tipo *say*, e incluso se simulan los errores habituales tanto en los comandos de movimiento como en las capturas de sensoriales. Los simuladores Stage y Gazebo [64] son

otros ejemplos de simuladores de referencia con capacidades de comunicación entre robots. En particularidad Gazebo permite simular comunicaciones entre equipos de robots y Stage permite simular comunicaciones dentro de grupos muy numerosos o enjambres.

Figure 4.29: Simulador Soccer Server empleado en la competencia RoboCup (izquierda) e instantes de un partido de una competencia organizada por la FIRA (derecha

Capitulo 5

5 Analisis de resultados y conclusiones

5.1 Conclusiones

A lo largo de la elaboración de este trabajo de tesis se han adquirido los conocimientos de las técnicas que permitieron dar solución al problema planteado. Es importante hacer mención que tanto el estudio de las técnicas de comunicación maestro-esclavo entre dispositivos, como implementación a nivel práctico son la base esencial de este trabajo.

La base importante de este trabajo fue la integración de robots móviles en configuración diferencial, como el movimiento de cada uno de ellos con la asignación de una tarea específica demostrando el comportamiento individual como en conjunto. Algunas tareas que los robots móviles pueden ejecutar son: la limpieza de residuos tóxicos, la transportación y manipulación de objetos, exploración y vigilancia, y tareas de búsqueda como rescate. Sin lugar a duda hay tareas que son más factibles a ejecutar para robots que por humanos, debido al alto grado de peligro que estas pueden simbolizar para los humanos.

Debido a lo anterior, se estudió la problemática del comportamiento individual, como sensores, actuadores, comunicación y la programación para la manipulación de cada uno de los robots que se usarían. Como también el comportamiento colectivo de dos robots móviles para ejecutar varias tareas específicas, que se basa en el traslado de cada robot de un punto inicial a un punto final; sujeto a las restricciones físicas del sistema, sujetos a su comunicación, también considerando la evasión de obstáculos que se encontrasen el espacio de trabaja, así como el evitar choques entre los robots utilizados para este trabajo.

Para poder realizar lo anterior, se realizaron pruebas experimentales como la caracterización de los robots móviles de Mindstorms EV3 de LEGO para saber el comportamiento, como ventajas y desventajas de cada uno de ellos. Se propusieron distintas trayectorias para conocer su comportamiento en distintas áreas de trabajo, además se desarrollarlo la comunicación vía Bluetooth entre los robots móviles y se desarrollaron distintos programas para el control del seguimiento de trayectorias. Los programas están basados en la configuración, caracterización y propiedades que poseen los robots móviles. Esto permite que las variables utilizadas para controlar los robots sean variables independientes a salidas, para el seguimiento de trayectorias.

Con base a lo anterior, basta con especificar las trayectorias propuestas para los robots; por un lado se estudió a cada uno de los robots por utilizar, con distintos movimientos que utilizarían en las diferentes trayectorias. Así al saber las ventajas y desventajas podemos compensar errores que tenga en sus movimiento y los sensores, para trasladarse en el marco de trabajo ya sea con obstáculos y evitando colisiones. Al conocer cada uno de los alcances de los robots nos permite, que la forma de control implementada hacia los robots esté sujeta en distintas restricciones, que son los movimientos no autorizados por su configuración diferencial.

Ahora, con respecto a la planeación de la comunicación entre robots móviles para un movimiento colectivo de los mismos; por un lado, se estudió las formas de comunicación aceptada por el robot móvil mindstorms EV3 de LEGO, basándonos en la comunicación punto a punto vía Bluetooth. Para otro se implementó el estudio ya existente de los robots móviles EV3, los cuales al tener una comunicación maestro-esclavo se plantearon trayectorias para llevar a la experimentación dicho estudio y que ellos realizaran movimientos sincronizados sin importar sus condiciones iniciales.

Para resolver el caso de la comunicación de robots móviles, se formuló como un problema de programación de envío y recepción de mensajes, utilizando protocolos, como distintas herramientas en la programación. El software utilizado fue Lejos que se utilizó el lenguaje de programación en JAVA, este lenguaje tiene distintos paquetes que encontraron soluciones para la optimización de la transmisión y recepción de los mensajes enviados entre dispositivos, así como la comunicación por vía Bluetooth.

5.2 Análisis de resultados

Durante el desarrollo de este trabajo de investigación se han mostrado resultados de cada actividad realizada; esto permite un análisis a nivel general de todos los resultados mostrados en esta tesis.

Primero abordamos el concepto particular de cada uno de los robots móviles a utilizar, que se puede concluir que es una clase particular de robots que poseen la capacidad de moverse en un entorno con base en la programación que este posea. Cada robots utilizado son los Mindstorms EV3 creados por la compañía LEGO, como ya se explicó antes realizando una caracterización con distintos movimientos, que fueron hacia adelanta, de reversa y vuelta de 360°. Estos movimientos fueron en diferentes velocidades para observar el rendimiento de los servomotores.

La trayectoria de evasión de obstáculos utilizando el sensor de luz, después de haber investigado sus alcances y limitación, se desarrolló un programa para garantizar que los robots pueden evitar un choque con algún obstáculo de diferente color que se encuentra en el medio de trabajo.

Las trayectoria de forma sinodal y forma de ocho entre maestro-esclavo, garantizo el estudio de técnicas de comunicación vía Bluetooth utilizando herramientas de programación para la transmisión de mensajes y ese mensaje llegue con éxito al receptos para la ejecución de la trayectoria, de igual manera el maestro necesita un mensaje de que fue recibida la información. Al tener comunicación entre robots es un aporte para partir a realizar distintas aplicaciones entre robots móviles y empezar a realizar trabajos colectivos ayudando al ser humano en tareas donde le es difícil llegar.

Trayectoria en distintos puntos de los robots maestros y esclavo, utilizando las técnicas anteriores se realizan una trayectoria diferente para el robot maestro y para el robot esclavo, además iniciando en distintos puntos hasta llegar a tener una sincronía en sus movimientos.

5.3 Trabajos futuros

Por último, se puede decir que el objetivo general de esta tesis se concluyó y sus objetivos particulares plateados, los resultados permitieron comprobar algunos aspectos teóricos.

Entre los trabajos futuros se encuentran:

- Aplicar un controlador eficiente para la manipulación de los robots móviles.
- Emplear más robots móviles para la realización de un trabajo colectivo, como mover objetos pesados, entre otras aplicaciones.
- Realizar tareas simultáneas con una topología maestro-esclavo la cual es maestro tendrá en controlador y la manipulación de los demás esclavos.
- Migrar la comunicación vía Bluetooth a Wi-Fi para extender los robots móviles EV3
- Los robots móviles pueden ser programados con distintos lenguajes de programación, así tiene la virtud de programar y comunicar distintos robots

como los ArDrone ya que también tiene un software libre, asi tener comunicación con los EV3 para atacar distintos problemas como la formación, patrullaje, consenso, entre otros.

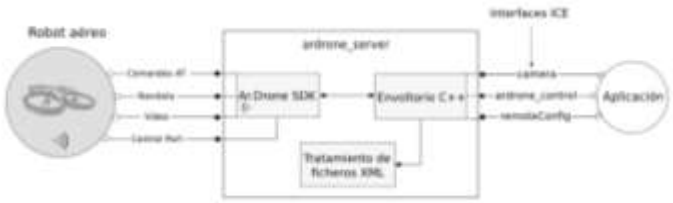

Figura 5.1: Diagrama de comunicación de drone controlada con interfaz

Bibliografía

[1] BARRIENTOS, A., Peñin. L. F. Balaguer, C., and Aracil, R. (2007). Fundamentos de Robótica. Mc Graw Hill, España, second edition.

[2] B. Tung and L. Kleinrock. Distributed Control Methods. In HPDC, peges 206-215, 1993.

[3] Bluetooth Special Interest Goup, "Specification of the Bluetooth System Covered core package version: 2.0+EDR"

[4] E. Guizzo. Three engineers, hundreds of robots, one warehouse. Spectrum, IEEE, 45(7) 26-34, July 2008.

[5] Enfoque holónomico basado en agentes para el control de organizaciones de robots móviles. Cervera, A., Soriano, A., Gómez, J., Valera, A., Valles, M., Giret, A. Jornadas de Automatica 2011. Sevilla.

[6] I. Noda, C. F. I. Noda, H. Matsubara, K, Hiraki and I. Frank. Soccer server: A tool for research on multiagent systems. Applied Artificial Intelligence, 1998

[7] Java Agent DEvelopmnt Framework. http://jade.tilab.com/

[8]Java APIs for Bluetooth. URL: http://download.oracle.com/javame/config/

[9] J. M. Esposito. Distributed grasp synthesis for swarm manipulation with application to autonomous tugboats. *IEEE Conference on Robotics and Automation*, pages 1489-1494, 2008

[10] J.P. Desai, J. Ostrowski, and V. Kumar. Controlling formations of multiple robots. In in Procceding of the IEEE International conference on Robotics and Automation, page 2864 – 2869, 1998.

[11] J. Salazar. Optimizacion Matematica: Ejemplos y aplicaciones. Universidad de la Laguna, spain, 2003

[12] KANAYAMA I. and B. Hartman, (1989). "Smooth local path planning for autonomous vehicles" Proc. IEEE Conference on Robotics and Automation

[13][Kanayama and Hartman, 1986] Kanayama S and L.S. Davis, (1986) "Multiresolution path planning for mobile robots". IEEE Journal of Robotic and Automation,

[14] KELLY, R. and Santibanez. V. (2003). Control de Movimiento de Robots Manipuladores. Prentice Hall.

[15] K. Harada, S. Kajita, F. Kanehiro, K. Fujiwara, K. Yokoi, and H. Hirukawa. Real-time planning of humanoid robot`s gait for force controlled manipulation. In ICRA, pages 616-622, 2004

[16] L. Bass, P. Clements, R. Kazman, Software Architecture in Practice, 2nd Edition, Addison Wesley, 2003

[17] LabView. http://www.ni.com/academic/midstorms/esa/

[18] leJOS NXJ API. URL http://lejos .sourceforge.net.nxt/api/index.html

[19]leJOS NXJ Tutorial. URL: http://lejos.sourceforge.net/nxt/nxj/tutorial/index.htm

[20] LEGO. http:// midstorms.lego.com 2012.

[21][Moravec H. y A. Elfes, 1985] Moravec H. and A. Elfes (1985) "High resolution maps from a wide-angle sonar" Proc, IEEE Conference on Robotics and Automation.

[22] MORAVEC H "Robot rover Visual navigation". UMI Reseach Press.

[23] NELSON W., (1989) "Continuous curvature path for autonomous vehicles". Proc. IEEE Int. Conference on Robotics and Automation.

[24]R. M. C. Bodduluri, J. M. McCarthy, and J. E. Bobrow. Planning movement for two puma manipulation holding the same object. In the *First International Symposium on experimental Robotics I,* Pages 579-593, London, UK, 1990.

[25] Running JADE under Eclipse. http://wrjih.wordpress.com/2008/11/29/running-jade-under-eclupse/

[26] Sixth Annual Mobile Robot Competition at Gerogia Institute of Technology. http://www.cc.gatech.edu/ai/robot-lab/reseach/aaai97/ ,1998.

[27] S. Sen, M. Sekaran, and J. Hale. Learning to Coordinate without Sharing Information. In Procceding of the Twelfth National Conference on Artificial Intelligence, pages 426-431, Seattle, WA, 1994

[28][Stenz, 1990] Stenz R.H. (1990) "Multisolution constraint modelling for mobile robot path- planning". "Vision and Navigation: The CMU NavLab"Kluwer Academic Plublishers, Capitulo 1

[29][Thorpe, 1984] Thorpe C., (1984) “FIDO: visión and navigation for a robot rover”, PH. D Thesis, Carnegie Mellon University.

[30][Todd Jochem and Dean Pomerleau, 1995] Tood Jochem and Dean Pormerleau (1995) “PANS: A portable Navigation Platform” IEEE Symposium on Intelligent vehicle.

[31] Y.Wang, Y. Liu, Ding H., and Y. Xu. Obstacle-avoidance path planning for soccer vehicles using particle swarm optimization. In IEEE international Conference on Robotics and Biomimetic, RIBIO, pages 1233-1238pp.-, Kunming China, 2006. IEEE.

[32] Y. Wang, P. Chen, and J. Yugang. Trajectory planning for an unmanned ground vehicle group usin augmented particle swarm optimization in a dynamic environment. In IEEE international Conference on Systems, Man and Cybernetics, pages 4341 – 4346 pp.-, San Antonio, TX, 2009.

Anexo A

Instalación del JAVA JDL (Java Development Kit).

Para poder trabajar con el lenguaje de programación Java es necesario instalar lo que se conoce como JDK o Java Development Kit, el cual provee herramientas de desarrollo para la creación de programas de Java. Puede instalarse en una computadora loca o una unidad de red una, una vez que ha sido descargada del siguiente enlace:

http://www.oracle.com/technetwork/java/javase/downloads/jdk-6u26-download-400750.html

Una vez descargado, solo tenemos que activar el ejecutable y seguir los pasos del asistente para instalar el JDK en nuestra computadora.

Instalación del driver USB de LEGO

Para que nuestra computadora pueda comunicarse con todos los robots Lego EV3, independientemente de la versión de firmeware que tenga instalada, es necesario instalar los drivers del conector Bluetooth USB de lego. Podemos descargarlos dese el siguiente enlace:

http://mindstorms.lego.com/en-us/support/file/default.aspx

Una vez instalado el driver, será necesario agregar todos los robots como dispositivos conocidos por el interfaz Bluetooth, ya que si no se encuentran conectados entre sí no podemos comunicarlos entre ellos. Para ello, vamos a “Configuración” -> “Dispositivos Bluetooth” -> “Agregar…”, con lo que podemos agregar cada uno de los robots que se quieren interconectar, introduciendo el PIN cuando nos sea necesario.

Instalación de LeJOS EV3

LeJOS EV3 cuenta con su propio firmeware, es necesario reemplazar el original que tiene los robots. Para ello se debe descargar el software desde:

http://LeJOS.sourseforce.net/ev3-downloads.php

Unas vez instalado en la computadora , el software incluye una biblioteca de clases de java (Classes.jar) que implementan a LeJOS EV3 Application Programming Interface (API), herramientas de PC para actualizar el firmeware del robot, cargar los programas, después, depurarlos, un API de PC para escribir programas de PC que se comunican con los

programas de LEJOS EV3 a través de Bluetooth o USB, Wifi, otras muchas funciones y programas de ejemplos.

El robot se puede comunicar con la computadora por medio de USB y Bluetooth. Para comunicarse por medio de USB se utiliza el firmware a la versión de LeJOS EV3, y con esto los robots quedan preparados para programarlos.

Para realizar aplicaciones, pruebas y rutinas, se ha utilizado el entorno de trabajo eclipse, por su popularidad entre programadores en Java y por la familiaridad personal con la que se contaba anteriormente.

Instalación y configuración de la plataforma JADE

En el siguiente apartado se explica la configuración e instalación de la plataforma.

Para poder utilizar el framework de JADE es necesario instalar y configurar el equipo. Lo primero que tendremos que hacer es descargar las librerías necesarias desde la página web del proyecto http://jade.tilab.com .

CLASSPATH=/ruta/jade/lib/jade.jar:/ruta/jade/lib/iiop.jar:/ruta/jade/lib/http.jar

Donde la ruta indica donde tenemos las librarías de JADE que acabamos de descargar.

Configuración del entorno de desarrollo (Eclipse)

Desde http://www.eclipse.org/downloads se descargara la última versión de ECLIPSE IDE para desarrolladores JAVA.

Compilación y ejecución para EV3

LeJOS proporciona una aplicación para descargar los programas en el EV3, el hecho de complicar con ECLIPSE y luego usar otra aplicación para descargar los datos EV3. Así de esta manera se configura para tener la opción de descargar los programas directamente en el brick, sin necesidad de la aplicación de LeJOS. Para esto hay que configurarlo de la siguiente manera:

Con un proyecto abierto, se hace clic en “Run” -> “External Tools Configurations…”

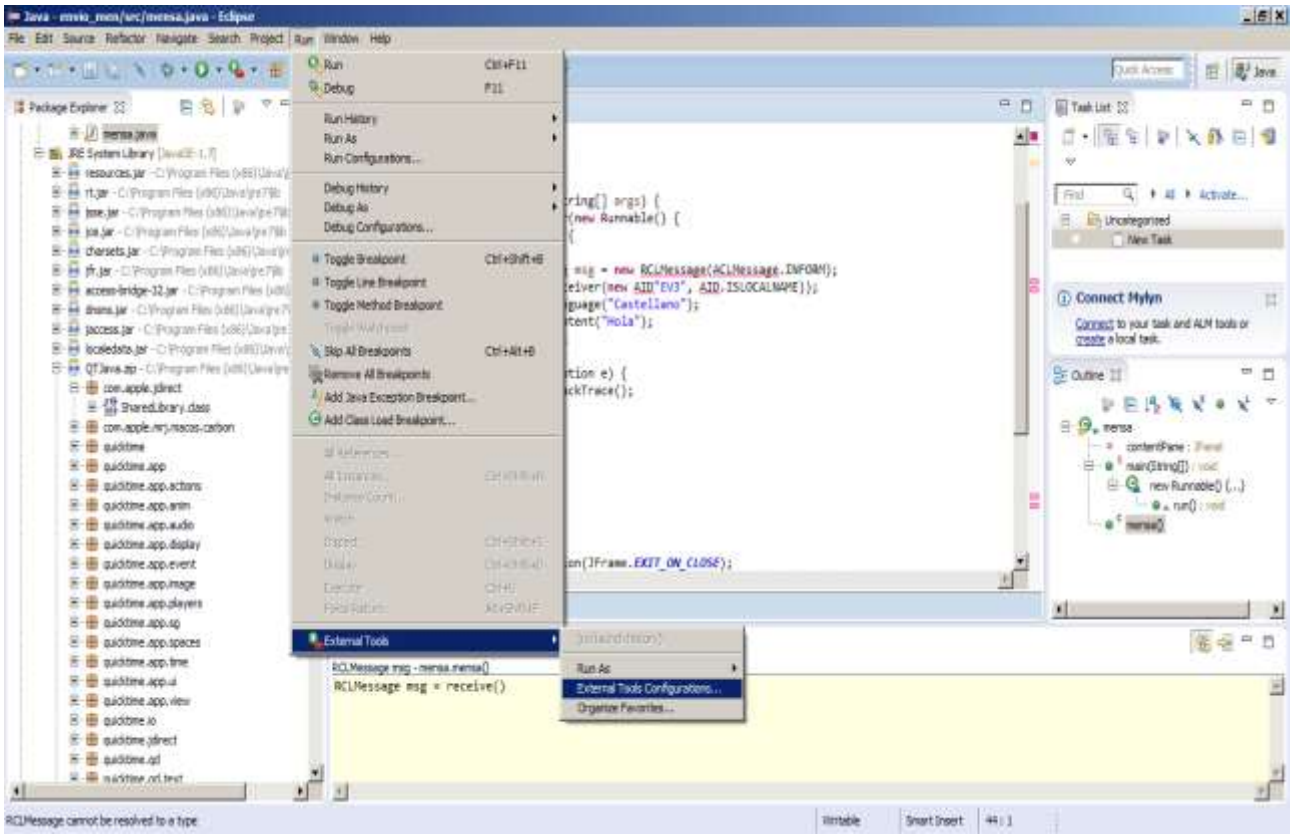

Figura A.1: Imagen de para compliar proyecto

Nos muestra una ventana donde hay que seleccionar “Program” en la parte de la izquierda y el icono de “New launch configuration”. Como nombre se le puede dar “LeJOS Download” por ejemplo, y en la pestaña “Main” hay que rellenar el campo “Location” con la ruta donde se encuentra “LeJOSdl.bat” proporcionado por LeJOS. (Debera estar en %ruta_de LeJOS_EV3%/bin). Este ejecutable es el que le permite descargar directamente al EV3, lo que se está haciendo es vincular el ECLIPSE a esa aplicación para poder hacerlo directamente sin salir del entorno. Siguiendo con la configuración en el campo “Working Directory” hay que introducir “${Project_loc}\bin” (sin comillas) y el campo “Arguments” introducirá “${java_type_name}” (sin comillas).

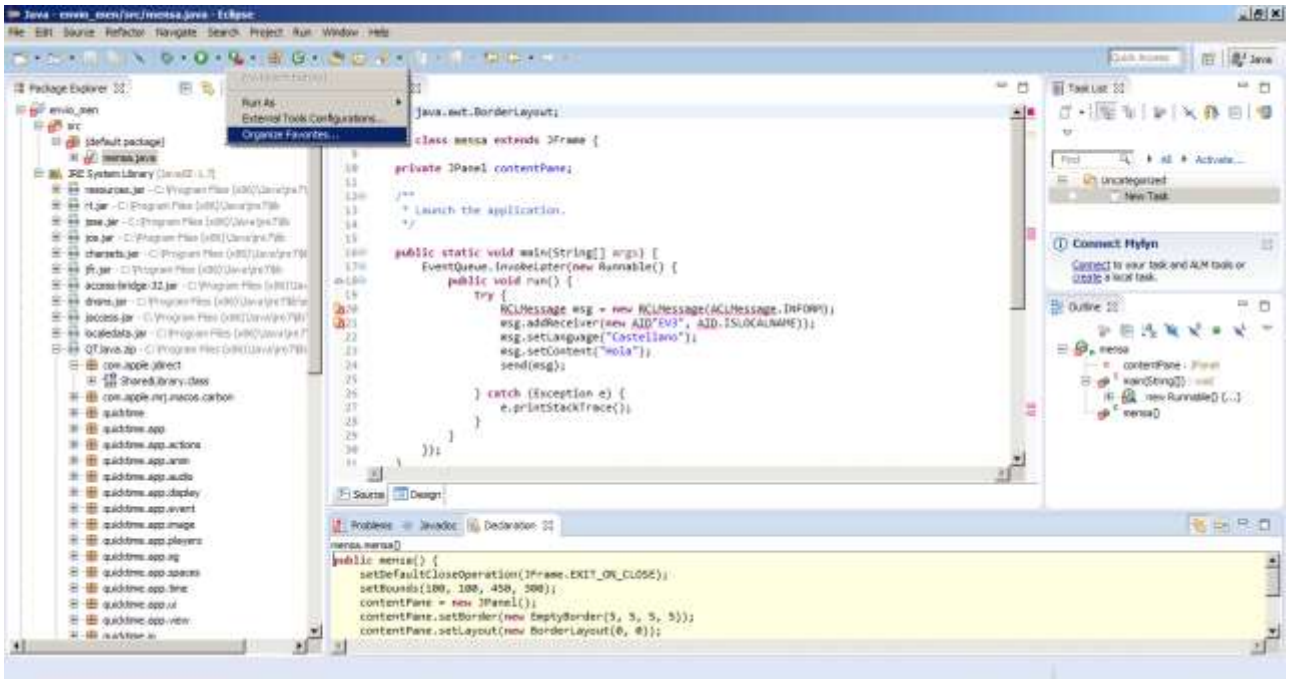

Figura A.2: Imagen de pantalla para organizar favoritos

Dar click en “Add” en la ventana nueva que aparecerá y luego seleccionar la opción de “LeJOS Download” (o el nombre que se le haya querido dar a la configuración definida anteriormente). Presionar “OK”, luego “OK” y de nuevo “OK”.

Figura A.3: Imagen de pantallas para añadir lejos al interfaz

De este modo bastara con hacer “Run” que se ha configurado para indicar a Eclipse que debe compilar y descargar el programa en el EV3 mediante USB.

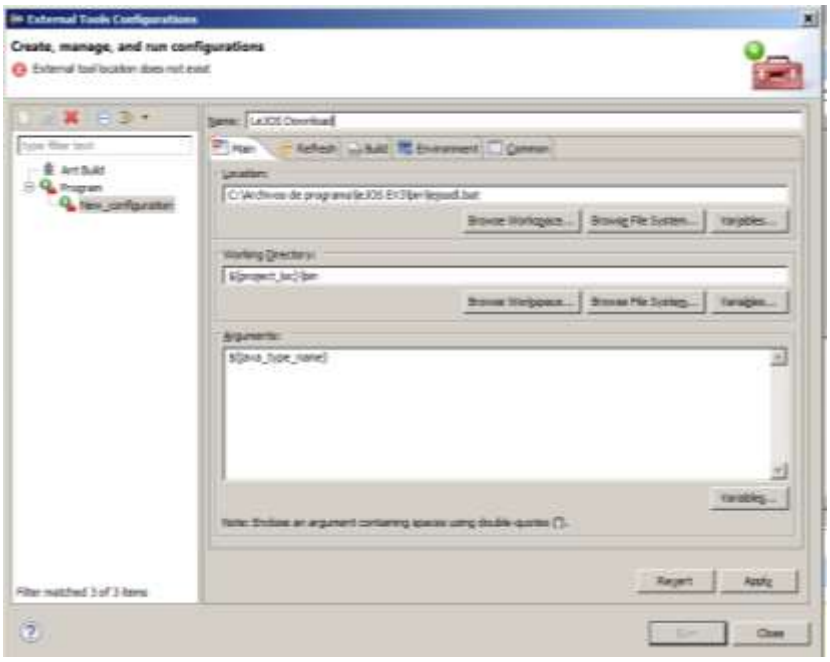

Figura A.4: Imagen de pantalla para correr el programa Lejo en Eclipse

Cuando ya se ha creado la configuración, se puede poner un acceso directo haciendo clic en el icono de “Run” -> “Organize Favorites...”

Anexo B

Programacion de trayectorias

AnexoB.1 programa de caracterización hacia adelante.

El programa del maestro y es esclavo será igual para los dos.

```
package pack;

import lejos.hardware.motor.Motor;
import lejos.utility.Delay;

public class Motorforwad {

	public static void main(String[] args) {

		public Motorfowad(){
		try{
			Motor.A.setSpeed(100); //La velocidad en porcentaje
			//el numero que están en parente es el que se va modificar
			Motor.B.setSpeed(100); //La velocidad en porcentaje
			//el numero que estan en parente es el que se va modificar

			Motor.A.forward();  //motor A utilizar
			Motor.B.fotward();  //motor B utilizar
			Delay.msDelay(10000);     //timepo de espera
			motorA.stop();            //Parar Motor A
			motorB.stop();     //Parar Motor B

			}
		}




	}

}
```

Anexo B.2 programa de caracterización hacia atrás.

El programa es idéntico en el maestro y el esclavo

```
package pack;

import lejos.hardware.motor.Motor;
import lejos.utility.Delay;
```

```
public class Motorforwad {

	public static void main(String[] args) {

		public Motorfowad(){
		try{
			Motor.A.setSpeed(100); //La velocidad en porcentaje
			//el numero que están en parente es el que se va modificar
			Motor.B.setSpeed(100); //La velocidad en porcentaje
			//el numero que estan en parente es el que se va modificar

			Motor.A.backward(); //motor A utilizar
			Motor.B.backward(); //motor B utilizar
			Delay.msDelay(10000);     //timepo de espera
			motorA.stop();            //Parar Motor A
			motorB.stop();     //Parar Motor B

			}
		}
	}

}
```

Anexo B.3 programa de caracterización con vuelta hacia la izquierda con L1 =-10 0 y L2 = 100, los motores L1 y L2 se modificara para la velocidad de la vuelta de 360 robot móvil EV3. El programa del maestro y el esclavo serán iguales en la calibración.

```
package pack;

import lejos.hardware.motor.Motor;
import lejos.utility.Delay;

public class Motorforwad {

	public static void main(String[] args) {

		public Motorfowad(){
		try{
			Motor.A.setSpeed(-100); //La velocidad en porcentaje
			//el numero se quedara en 0
			Motor.B.setSpeed(100); //La velocidad en porcentaje
			//el numero que estan en parente es el que se va modificar

			Motor.A.forward();  //motor A utilizar
			Motor.B.forward();  //motor B utilizar
			Delay.msDelay(10000);     //timepo de espera
			motorA.stop();            //Parar Motor A
			motorB.stop();     //Parar Motor B

			}
		}
	}
```

```
}
```

Anexo B.4 programa de caracterización con vuelta hacia la derecha con L1 =100 y L2 = -100, los motores L1 y L2 se modificara para la velocidad de la vuelta de 360 del robot móvil EV3. El programa del maestro y el esclavo serán iguales en la calibración.

```
package pack;

import lejos.hardware.motor.Motor;
import lejos.utility.Delay;

public class Motorforwad {

	public static void main(String[] args) {

		public Motorfowad(){
		try{
			Motor.A.setSpeed(100); //La velocidad en porcentaje
			//el numero se moificara, el cual se estará diminuyendo de 10
			Motor.B.setSpeed(-100); //La velocidad en porcentaje
			//el numero se quedra en 0

			Motor.A.forward();  //motor A utilizar
			Motor.B.forward();  //motor B utilizar
			Delay.msDelay(10000);     //timepo de espera
			motorA.stop();            //Parar Motor A
			motorB.stop();     //Parar Motor B

			}
		}
	}

}
```

Anexo B.5 programa de caracterización con vuelta hacia la derecha con L1 =100 y L2 = 0, el motor L1 se modificara para la velocidad de la vuelta de 360 del robot móvil EV3. El programa del maestro y el esclavo serán iguales en la calibración.

```
package pack;

import lejos.hardware.motor.Motor;
import lejos.utility.Delay;

public class Motorforwad {

	public static void main(String[] args) {
```

```
		public Motorfowad(){
		try{
			Motor.A.setSpeed(100); //La velocidad en porcentaje
			//el numero se moificara, el cual se estará diminuyendo de 10
			Motor.B.setSpeed(0); //La velocidad en porcentaje
			//el numero se quedra en 0

			Motor.A.forward();  //motor A utilizar
			Motor.B.forward();  //motor B utilizar
			Delay.msDelay(10000);     //timepo de espera
			motorA.stop();            //Parar Motor A
			motorB.stop();     //Parar Motor B

			}
		}
	}

}
```

Anexo B.6 programa de caracterización con vuelta hacia la izquierda con L1 =0 y L2 =100, el motor L2 se modificara para la velocidad de la vuelta de 360 del robot móvil EV3. El programa del maestro y el esclavo serán iguales en la calibración.

```
package pack;

import lejos.hardware.motor.Motor;
import lejos.utility.Delay;

public class Motorforwad {

	public static void main(String[] args) {

		public Motorfowad(){
		try{
			Motor.A.setSpeed(0); //La velocidad en porcentaje
			//el numero se moificara, el cual se estará diminuyendo de 10
			Motor.B.setSpeed(100); //La velocidad en porcentaje
			//el numero se quedra en 0

			Motor.A.forward();  //motor A utilizar
			Motor.B.forward();  //motor B utilizar
			Delay.msDelay(10000);     //timepo de espera
			motorA.stop();            //Parar Motor A
			motorB.stop();     //Parar Motor B

			}
		}
	}

}
```

Anexo B.7 programa de caracterización con vuelta hacia la izquierda con L1 = -30 y L2 =100, pero el motor de L2 se modificara para la velocidad de la vuelta del robot móvil EV3. El programa del maestro y el esclavo serán iguales en la calibración.

```
package pack;

import lejos.hardware.motor.Motor;
import lejos.utility.Delay;

public class Motorforwad {

	public static void main(String[] args) {

		public Motorfowad(){
		try{
			Motor.A.setSpeed(-30); //La velocidad en porcentaje
			//el numero se moificara, el cual se estará diminuyendo de 10
			Motor.B.setSpeed(100); //La velocidad en porcentaje
			//el numero se quedra en 0

			Motor.A.forward();  //motor A utilizar
			Motor.B.forward();  //motor B utilizar
			Delay.msDelay(10000);     //timepo de espera
			motorA.stop();            //Parar Motor A
			motorB.stop();     //Parar Motor B

			}
		}
	}

}
```

Anexo B.8 programa de caracterización con vuelta hacia la derecha con L1 = 100 y L2 =-30, pero el motor de L1 se modificara para la velocidad de la vuelta del robot móvil EV3. El programa del maestro y el esclavo serán iguales en la calibración.

```
package pack;

import lejos.hardware.motor.Motor;
import lejos.utility.Delay;

public class Motorforwad {

	public static void main(String[] args) {

		public Motorfowad(){
		try{
			Motor.A.setSpeed(100); //La velocidad en porcentaje
			//el numero se moificara, el cual se estará diminuyendo de 10
			Motor.B.setSpeed(-30); //La velocidad en porcentaje
			//el numero se quedra en 0

			Motor.A.forward();  //motor A utilizar
			Motor.B.forward();  //motor B utilizar
			Delay.msDelay(10000);     //timepo de espera
			motorA.stop();            //Parar Motor A
			motorB.stop();     //Parar Motor B

			}
		}
	}

}
```

Anexo B.9 Trayectoria de evasión de obstáculos

Clase Main, En esta clase mandas llamar los puertos que se utilizaran, el tipo de sensor que se utilizara y la configuración del sensor. Ya que utilizaremos modo colores que serán el rojo, azul, amarillo y verde.

```
public class Main {

	public static void main(String[] args) throws InterruptedException {
		final EV3ColorSensor colorSensor = new
EV3ColorSensor(SensorPort.S2);
		final SensorMode mode = colorSensor.getColorIDMode();
		final ColorRecognizerThread colorRecognizerThread = new
ColorRecognizerThread(colorSensor);

		colorRecognizerThread.start();

		Buttom.waitForAnyPress();
	}

}
```

Clase ColorSensor, La case realiza la operación de la trayectoria, primero importando la librería de los servo motores, y después se realizaran los casos de la trayectoria. Que son al detectar el color rojo, después el verde, seguido del azul y al final el color rojo.

```
package AppRobotSever.utileria;

import lejos.hardware.motor.Motor;
import lejos.utility.Delay;

public class ColorSensor extends Thread {

	private EV3ColorSensor colorSensor;

	public ColorRecognizerThread(final EV3ColorSensor colorSensor){
		this.colorSensor = colorSensor;

	}
	public void run() {
		while(true){
			final int colorId = colorSensor.getColorID();
			try{
				Motor.A.setSpeed(61); //La velocidad en porcentaje
		//el numero que estan en parente es el que se va modificar
				Motor.B.setSpeed(60); //La velocidad en porcentaje
		//el numero que estan en parente es el que se va modificar
			switch (colorId){
			//Rojo
			case 0;
```

```
Button.LEDPattern(2);
Delay.msDelay(1000);      //timepo de espera
motorA.stop();            //Parar Motor A
motorB.stop();      //Parar Motor B
boolean rojo;
if(rojo){
      Motor.A.setSpeed(-61); //La velocidad en porcentaje
//el numero que estan en parente es el que se va modificar
      Motor.B.setSpeed(-60); //La velocidad en porcentaje
//el numero que estan en parente es el que se va modificar

      Motor.A.forward();  //motor A utilizar
      Motor.B.fotward();  //motor B utilizar
      Delay.msDelay(1000);      //timepo de espera


      Motor.A.setSpeed(-30); //La velocidad en porcentaje
//el numero que estan en parente es el que se va modificar
      Motor.B.setSpeed(100); //La velocidad en porcentaje
//el numero que estan en parente es el que se va modificar

      Motor.A.forward();  //motor A utilizar
      Motor.B.fotward();  //motor B utilizar
      Delay.msDelay(4370);      //timepo de espera

      Motor.A.setSpeed(100); //La velocidad en porcentaje
//el numero que estan en parente es el que se va modificar
      Motor.B.setSpeed(100); //La velocidad en porcentaje
//el numero que estan en parente es el que se va modificar

      Motor.A.forward();  //motor A utilizar
      Motor.B.fotward();  //motor B utilizar
      Delay.msDelay(1000);      //timepo de espera
}else{

}
break;

//Verde
case 1;
Button.LEDPattern(1);
Delay.msDelay(1000);      //timepo de espera
motorA.stop();            //Parar Motor A
motorB.stop();      //Parar Motor B
boolean verde;
if(verde){
      Motor.A.setSpeed(-61); //La velocidad en porcentaje
//el numero que estan en parente es el que se va modificar
      Motor.B.setSpeed(-60); //La velocidad en porcentaje
//el numero que estan en parente es el que se va modificar

      Motor.A.forward();  //motor A utilizar
      Motor.B.fotward();  //motor B utilizar
      Delay.msDelay(1000);      //timepo de espera
```

```
            Motor.A.setSpeed(100); //La velocidad en porcentaje
        //el numero que estan en parente es el que se va modifica
            Motor.B.setSpeed(-30); //La velocidad en porcentaje
        //el numero que estan en parente es el que se va modificar

            Motor.A.forward();  //motor A utilizar
            Motor.B.fotward();  //motor B utilizar
            Delay.msDelay(3750);      //timepo de espera

            Motor.A.setSpeed(100); //La velocidad en porcentaje
        //el numero que estan en parente es el que se va modificar
            Motor.B.setSpeed(100); //La velocidad en porcentaje
        //el numero que estan en parente es el que se va modificar

            Motor.A.forward();  //motor A utilizar
            Motor.B.fotward();  //motor B utilizar
            Delay.msDelay(1000);      //timepo de espera
        }else{

        }
        break;

        // Amarillo
        case 2:
            Button.LEDPattern(4);
            Delay.msDelay(1000);      //timepo de espera
            motorA.stop();            //Parar Motor A
            motorB.stop();      //Parar Motor B
            boolean verde;
            if(verde){
            Motor.A.setSpeed(100); //La velocidad en porcentaje
    //el numero que estan en parente es el que se va modificar
            Motor.B.setSpeed(-30); //La velocidad en porcentaje
        //el numero que estan en parente es el que se va modificar

                    Motor.A.forward();  //motor A utilizar
                    Motor.B.fotward();  //motor B utilizar
                    Delay.msDelay(3750);      //timepo de espera

            Motor.A.setSpeed(100); //La velocidad en porcentaje
        //el numero que estan en parente es el que se va modificar
            Motor.B.setSpeed(100); //La velocidad en porcentaje
        //el numero que estan en parente es el que se va modificar

                    Motor.A.forward();  //motor A utilizar
                    Motor.B.fotward();  //motor B utilizar
                    Delay.msDelay(1000);      //timepo de espera

            Motor.A.setSpeed(100); //La velocidad en porcentaje
        //el numero que estan en parente es el que se va modifica
            Motor.B.setSpeed(100); //La velocidad en porcentaje
        //el numero que estan en parente es el que se va modificar

                    Motor.A.forward();  //motor A utilizar
```

```
                Motor.B.fotward();  //motor B utilizar
                Delay.msDelay(1000);      //timepo de espera
            }else{

            }
            threadSleep(2000);
            Button.LEDPattern(0);
            System.exit(0);


            //Azul
        case 3:

            Button.LEDPattern(3);
            Delay.msDelay(1000);      //timepo de espera
            motorA.stop();            //Parar Motor A
            motorB.stop();      //Parar Motor B
            boolean azul;
            if(verde){
            Motor.A.setSpeed(100); //La velocidad en porcentaje
        //el numero que estan en parente es el que se va modificar
            Motor.B.setSpeed(-30); //La velocidad en porcentaje
        //el numero que estan en parente es el que se va modificar

                Motor.A.forward();  //motor A utilizar
                Motor.B.fotward();  //motor B utilizar
                Delay.msDelay(3750);      //timepo de espera

            Motor.A.setSpeed(100); //La velocidad en porcentaje
        //el numero que estan en parente es el que se va modificar
            Motor.B.setSpeed(100); //La velocidad en porcentaje
        //el numero que estan en parente es el que se va modificar

                Motor.A.forward();  //motor A utilizar
                Motor.B.fotward();  //motor B utilizar
                Delay.msDelay(1000);      //timepo de espera

            Motor.A.setSpeed(100); //La velocidad en porcentaje
        //el numero que estan en parente es el que se va modificar
            Motor.B.setSpeed(100); //La velocidad en porcentaje
        //el numero que estan en parente es el que se va modificar

                Motor.A.forward();  //motor A utilizar
                Motor.B.fotward();  //motor B utilizar
                Delay.msDelay(10000);     //timepo de espera
            }else{

            }


            Motor.A.setSpeed(61); //La velocidad en porcentaje
    //el numero que estan en parente es el que se va modificar
            Motor.B.setSpeed(60); //La velocidad en porcentaje
    //el numero que estan en parente es el que se va modificar
            Motor.A.forward();  //motor A utilizar
```

```
                        Motor.B.fotward();  //motor B utilizar
                        Delay.msDelay(3000);      //timepo de espera
                        motorA.stop();      //Parar Motor A
                        motorB.stop();      //Parar Motor B
                        break;
                        default:
                              Button.LEDPattern(0);
                  }
             }
            }
      }
      private void threadSleep(final int ms){
            try{
                  Thread.sleep(ms);
            }catch (InterruptedException e){
                  e.printStackTrace();
            }
      }

}
```

Anexo B.10 Trayectoria sinusoidal

Robot móvil Maestro EV3M

Clase MetodHilos, manda llamar a los hilos en la paquete del robot.

```
package AppRobot.mindRobot.utileria.intefaces;

public class MetodHilos {
	public abstract void terminarHilos();
}
```

Clase ParametrosBluetooth, manda llamar a los parámetros del protocolo de Bluetooth, los canales que se utilizaran en nuestro programa, asi como autentificación y el modo que tendrá nuestro nuestro robot ya sea maestro o esclavo en la red.

```
package AppRobot.midRobot.utilieria.interfaces;

public class ParametrosBluetooth {
	public static final String Protocolo_Bluetooth = "btspp://";
	public static final String CANAL_UNO = "1";
	public static final String CANAL_DOS = "2";
	public static final String CANAL_TRES = "3";
	public static final String AUTENTICACION = "authenticate=false";
	public static final String ENCRIPCION = "encrupt=false";

	public static final String MASTER = "master=false";

}
```

Clase Busqueda, Esta clase busca el robot móvil EV3 esclavo (EV3E) para realizar la comunicación por Bluetooth. Se manda llamar cada vez que se requiera hacer la conexión el robot Maestro al esclavo, si el tiempo.

```
package AppRobot.midRobot.utileria.pantallas;

import AppRobot.mindRobot.utileria.interfaces.*;
import AppRobot.midRobot.utileria.*;
import javax.microedition.lcdui.*;
import protocoloRAP.*;

public class Busqueda extends Formulario implements {
	private String txtNombreEV3;
	private Alerta errorConexion;
```

```
	public Busqueda(String titulo,ControlRobot control, Alerta errorConeion,
Display disp){
		super(titulo,disp);
		this.controlRobot = control;
		this.errorConexion = errorConexion;
		String nombreEV3 = "EV3E";
		this.txtlblNombreEV3 = new
CajaTexto(nombreEV3,20,TextFiled.LAYOUT_LEFT);
		this.ingresarltem(this.txtNombreEV3);
		this.ingresarItem(this.txtNombreEV3);
	}
	private void administradorConexion(Espera espera)throws
InterruptedException, IOException{
		String nombreEV3 = this.lblNombreEV3.getString();
		boolean respuesta = this.peticionConexionServidor(nombreEV3,1);
		if(respuesta){
			this.controlRobot.mostrarFormulario();
			espera.terminarHilo();

		}else{
			this.mostrarFormulario();
			espera.terminarHilo();
			this.errorConexion.mostrarAlerta();
		}
	}
	private boolean peticionConexionServidor(String nombreEV3,int canal)
throws InterruptedException, IOException{
		Protocolo.obtenerSAP().establecerNombreEV3(nombreEV3);
		Protocolo.obtenerSAP().establecerCanalComunicacion(canal);
Protocolo.obtenerSAP().peticionServicio(Busqueda.P_ESTABLECIMIENTO_CONEXION);
		boolean respuestaSAP =
Protocolo.obtenerSAP().confirmacionServicio();
		if(respuestaSAP){
			return true;
		}else{
			return false;
		}
	}
	public void HiloAdministradorConexion(Espera espera){
		final Espera esp = espera;
		Runnable Admin = new Runnable(){
			public void run(){
				try{
					administradorConexion(esp);
				}catch(Exception ex){

				}
			}
		};
		Thread TAdmin = new Thread(Admin);
		TAdmin.start();
		}

}
```

Clase Espera, le otorga el tiempo a la espera de la búsqueda, sincronizando la transmisión entre los dispositivos encontrados.

```
package AppRobot.mindRobot.utileria.pantallas;

import AppRobot.mindRobot.utileria.intefaces.*;
import AppRobot.mindRobot.utileria.*;
import javax.microedition.lcdui.*;
import java.io.*;

public class Espera extends Formulario implements Runnable InterfazMetodosHilos
{
	private String[] pathsimagenes = {"/"}
	private boolean ejecutar;
	private Thread inicarEspera;
	public Espera (String titulo, Display disp){
		super(titulo,disp);
		this.ejecutar = true;
		try{
			this.img = new imagen(this.pathsimagenes[0]);
			this.ingresarImagen(this.img.obtenerImagen());
			this.get(0).setLayout(Item.LAYOUT_CENTER);
		}catch (IOEception ex){

		}
		this.iniciarEspera = new Thread(this);
	}
	public void run(){
		this.iniciarEspera();
	}
	private void iniciarEspera(){

		while(this.ejecutar !=false){
			synchronized(this){
				for(int i + 0;<this.pathsimagenes.length;i++){
					this.deleteAll();
					try{
						this.img = new
Imagen(this.pathsimagenes[i]);

	this.ingresarImagen(this.img.obtenerImagen());

	this.get(0).setLayout(Item.LAYOUT_CENTER);
						Thread.sleep(300);
					}catch(Exception ex){

					}
				}
			}
		}
	}
	public void ejecutarFormularioEspera(){
```

```
			this.iniciarEspera.start();
		}
		public boolean obtenerEstadoHiloEsperar(){
			if(this.iniciarEspera.isAlive()){
				return true;
			}else{
				return false;
			}
		}
		public synchronized void terminarHilo(){
			this.ejecutar = false;
		}

}
```

Clase Protocolo, esta clase manda llamar a la clase se Acceso para todo los servicio se los protocolos de comunicación a utilizar

```
package protocoloRAP.singleton;

import protcoloRAP.SAP.*;

public class Protocolo {
	private static final Protocolo protocoloSingleton = new Protocolo();
	private static PuntoAccesoServicio sap;
	private Protocolo(){
		Protocolo.sap = new PuntoAccesoServicio(true);
	}
	public static Protcolo obtenerInstanciaProtocolo(){
		return protocoSingleton;
	}
	public static PuntoAccesoServicio obtenerSAP(){
		return Protocolo.sap;
	}

}
```

Clase TxEscuchar, se encagar de obtener la trama y la transfeencia en la transmisión.

```
package AppRobot.midRobot.utileria.peticionTXListeners;

import AppRobot.midRobot.utileria.interfaces.*;
import protocoloRAP.*;
import java.io.*;


public final class TxExcuchar extends Thread  {
	private int a;

	private int s;
	private boolean ejecutar;
	public PeticionTxEscuchar(boolean iniciarTherad){
```

```
		this.a = 0;
		this.s = 0;
		if (iniciarThread)
			this.start();
	}
	public void run(){
		this.iniciarPeticiones();
	}
	public void run(){
		this.iniciarPeticiones();
	}
	private void iniciarPeticiones(){
		while(this.ejecutar != false){
			synchronized(this){
				try{
					this.wait();
				}catch(InterruptedException ex){

				}
			}
		}
	}
	public synchronized void peticionTxEscucahar(byte[] datos,boolean
commando){
		this.HiloPeticionTransmision(dato,this.a,commando);
		if(this.a ==0){
			this.a++;
		}else{
			this.a--;
		}
	}
	private boolean peticionTransmision(byte[] datos,boolean commando)throws
InterruptedException{
		Protocolo.obtenerSAP().establecerTipoTramaEnviar(comnado);
		Protocolo.obtenerSAP().establecerDatosEnviar(datos);
		try{

	Protocolo.obtenerSAP().peticionServicio(PeticionTxEscuchar.P_TRANSFERENCI
A_DATOS);
			boolean respuestaSAP =
Protcolo.obtenerSAP().confirmacionServicio();
			if(respuestaSAP){

	Protocolo.obtenerSAP().obtenerTxRx().establecerSecuenciaEnviada(this.a);
				establecerSecuenciaEnviada(this.a);
				this.s = this.a;
				return true;
			}else{
				return false;
			}
		}catch(IOException ex){
			return false;
			}
	}
```

```
	private void HiloPeticionTransmision(byte[] datos, int
numeroSecuencia,boolean commando){
		final byte[]trama = datos;
		final int sec = numeroSecuencia;
		final boolean com = comando;
		Runnable runTran = new Runnable(){
			public void run(){
				try{
					if(trama != null){

	Protocolo.obtenerSAP().establecerNumeroSecuenciaTX(sec);
						boolean respuesta =
peticionTransmision(trama,com);
						if(respuesta){

						}else{

						}
					}else{

					}
				}catch(Exception ex){

				}
			}
		};
		Thread transferencia = new Thread(runTrain);
		transferencia.start();
	}
	public synchronized void terminarHilo(){
		this.ejecutar = false;
		this.notifyAll();
	}

}
```

Clase PuntoAccesoServicio, esta clase brindara todos los servicio del protocolo de comunicación (ver tabla#) el cual como se explico en el apartado# se implementa para utilizar solo slot de comunicacion

```
package ProtocoloRAP.SAP;
package pack;

import protocoloRAP.servicio.OperacionesTramas.*;
import protocoloRAP.servicio.Sensores.*;
import protocoloRAP.servicio.transmision.*;
import AppRobotServer.utileria.interfaces.*;
import AppRobotServer.control.sensores.*;
import AppRobotServer.control.trayetoria.*;
import AppRobotServer.archivo.*;
import protocolo.canalRFCOMM.*;
```

```
import lejos.ev3.*;
import java.io.*;



public class PuntoAccesoServicio {

	private CanalRFCOMM canalRFCOMM;
	private ConexionBTsvr con;
	private boolean ejecutar;
	private byte[] servicio;
	private Sensores sns;
	private Servos svs;
	private OperacionesTramasve oT;
	private byte[] tramaEnviar;
	private Archivo archivoMotorA,archivoMotorB;
	public PuntoAccesoServicio(boolean iniciarSAP){
		this.txrx = null;
		this.sns = null;
		this.svs = null;
		this.canalRFCOMM = null;
		this.tramaEnviar = null;
		this.archivoMotorA = new Archivo
(PuntoAccesoServicio.Nombre_Archivo_A);
		this.archivoMotorB = new Archivo
(PuntoAccesoServicio.Nombre_Archivo_B);
		this.oT = new OperacionesTramasvr();
		this.ejecutar = true;
		this.con = new ConexionBTsvr(false);
		if(iniciarSAP)
			this.star();
	}
	public void run(){
		this.iniciarSAP();
	}
	private void iniciarSAP(){
		while (this.ejecutar !=false){
			synchronized (this){
				try{
					this.wait();

				}catch(interruptedException ex){

				}
			}
		}
	}
	public synchronized void peticionServicio (byte mensaje){
		while(this.servicio !=null){
			try{
				this.wait();

			}catch(interruptedException ex){
```

```
                        }
                }
                byte[]mensajeC = (mensaje);
                this.servicio = mensajeC;
                this.notifyAll();
        }
        public synchronized boolean confirmacionServicio() throw
interruptedException{
                while(this.servicio == null){
                        this.wait();
                }
                boolean respuesta = this.determinacionServicio(this.sevicio);
                this.servicio = null;
                return respues;
        }
        private boolean determinacionServicio(byte[] peticion){
                switch(peticion[0]){
                case PuntoAccesoServicio.P_ESTABLECIMIENTO_CONEXION;
                this.con.start();
                try{
                        boolean estadoCon =
this.confirmacionServicio().conexionEstablecida();
                        if(estadoCon){
                                this.canalRFCOMM = this.con.obtenerCanalRFCOMM();
                                this.txrx = new TXRX (this.canalRFCOMM,true);
                                this.sns = this.txrx.sensores();
                                return
PuntoAccesoServicio.ACK_Establecimiento_Conexion;

                        }else {
                                return
PuntoAccesoServicio.ACK_ESTABLECIMIENTO_CONEXION;
                        }
                }catch(Exception ex){
                        return PuntoAccesoServicio.NACK_ESTABLECIMIENTO_CONEXION;
                }
                case PuntoAccesoServicio.P_TRANSFERENCIA_DATOS:
                        byte[] trama = this.tramaEnviar;
                        try{
                                this.txrx.txTramma(trama);

                        }catch(IOException ex){
                                return PuntoAccesoServicio.ACK_TRANS_TRAMA_SAP;
                        }
                        return PuntoAccesoServicio.ACK_TRANS_TRAMA_SAP;
                case PuntoAccesoServicio.P_FINALIZACION_CONEXION;
                this.svs.pararAA();
                this.svs.pararID();
                boolean trayectoria = this.svs.obtenerTraayectoria();
                }
                this.finalizarConexion();
                return PuntoAccesoServicio.ACK_FINALIZACION_CNOEION;
                default:
                        this.svs.paparAA();
                        this.svs.pararID();
```

```
                    LCD.clearDisplay();
                    LCD.drawString("Error",0,0);
                    this.finalizarConexion();
                    return PuntoAccesoServicio.ERROR;

        }
}
private void finalizarConexion(){
        if(this.archivoMotorA != null){
                this.archivoMotorA = null;
        }
        if (this.archivoMotorB != null){
                this.archivoMotorB = null;
        }
        if(this.sns.isAlive()){
                this.sns.terminarHilo();
                this.sns = null;
        }else {
                this.sns = null;

        }
        if(this.svs != null){
                this.svs = null;
        }
        if(this.txtrx != null){
                if(this.txtrx.isAlive()){
                        this.txrx.terminarHilo();
                        this.txrx = null;
                }else {
                        this.txtrx = null;
                }
        }
        if(this.canalRFCOMM != null){
                this.canalRFCOMM.desconectar();
                this.canalRFCOMM = null;
        }
        if(this.con != null){
                this.con = null;
        }
        this.terminarHilo();
        System.gc();
}
public OperacionesTramasvr obtenerOperacionesTrama(){
        return this.oT;
}
public synchronized void establecerTramaEnviar(byte[]tramaEnvia){
        this.tramaEnviar = tramaEnviar;
}
public synchronized Archivo obenerArchivoA(){
        return this.archivoMotorA;
}
public synchronized Archivo obtenerArchivoB(){
        return this.archivoMotorB;
}
public synchronized void terminarHilo(){
```

```
        this.ejecutar = false;
        this.notifyAll();
}
}
```

Clase RFCOMM, como ya se explico antes esta clase es la encargada de establecer la configuración en el canal de transmisión.

```
package protocoloRAP.canalRFCOMM;
package pack;

import javax.microedition.io.*;
import java.io.*;



public class canalRFCOMM {

        private StreamConnection strcon;
        private DataInputStream streamEntrada;
        private DataOutnputStream streamSalida;
        public canalRFCOMM (StreamConnection strcon){
                this.strcon = strcon;
                this.streamEntrada = this.strcon.openDataInputStream();
                this.streamSalida = this.strcon.openDataOutputStream();

        }
        public byte[] recibirTrama()throws IOException {
                if(this.streamEntrada.available() == 0){
                        return null;
                }else {
                        byte tramaRecibida[] = new
byte[this.streamEntrada.available()];
                        this.streamEntrada.read(tramaRecibida);
                        return tramaRecibida;
                }
        }
        public void enviarTrama (byte[]trama){
                final DataOutputStream dout = this.streamSalida;
                final byte[] datos = trama;
                Runnable env = new Runnable(){
                        public void run(){
                                try{
                                        dout.write(datos);
                                        dout.flush();

                                }catch (IOException ex){

                                }
                        }
                };
                Thread tenv = new Thread(env);
                tenv.start();
```

```
	}
	public DataInputStream obtenerStreaEntrada(){
		return this.streamEntrada;

	}
	public DataOutputStream obtenerStreamSalida(){
		return this.streamSalida;
	}
	public StreamConnection obtenerStreamConexion(){
		return this.strcon;
	}
	public void desconectar(){
		try{
			if(this.streamEntrada != null){
				this.streamEntrada.close();
			}
		}catch(Exception e){

		}
		try{
			if(this.streamSalida != null)
				this.streamSalida.close();
		}catch(Exception e){

		}
		try{
			if(this.streamSalida != null)
				this.streamSalida.close();
		}catch(Exception e){

		}
		try{
			if(this.strcon != null)
				this.strcon.close();
		}catch (Exception e){}
			this.streamEntrada = null;
			this.streamSalida = null;
			this.strcon = null;
	}
}
```

Clase de ConeionBTsvr, Esta clase realiza la conexión entre robots móviles Lego EV3 maestro (EV3M) y EV3 ESCLAVO (EV3E). Activando el modulo de Bluetooth, realiza la sincronización entre dispositivos y la espera de envió de señal RF mientras encuentra el dipositivo.

```
Package protocoloRAP.servicios.conexion;

import protocoloRAP.canalRFCOMM.*;
```

```
import javax.microedition.io.*;
import lejos.ev3.comm.*;
import lejos.ev3.*;

public class ConeionBTsvr {

	private boolean inicioConexion;
	private CanalRFCOMM canalRFCOMM;
	public ConeionBTsvr(boolean iniciarervicor){
		if(iniciarServidor)
			this.start();
	}
	public void run(){
		this.inicarServidor();
	}
	private void inicarServidor(){
		try{
			LCD.drawString("Esperando",0,1);
			LCD.drawString(" conexion..", 0, 2);
			BTConnection btcon = Bluetooth.waitForConnection();
			btcon.setIOMode(EV3Connection.RAW);
			this.canalRFCOMM = new CanalRFCOMM
					((StreamConnection)btcon);
			synchronized (this){
				this.inicioConexion = true;
				notify();
			}
		}catch(Exception ex){

		}
	}
	public boolean conexionEstablecida() throws InterruptedException{
		int i = 0;
		while(true){
			synchronized(this){
				this.wait();
			}
			i = i + 1;
			if(i==1){
				break;
			}
		}
		if(this.inicioConeion){
			return true;
		}else{
			return false;
		}
	}
	public CanalRFCOMM obtenerCanalRFCOMM(){
		return this.canalRFCOMM;
	}
}
```

Clase Tramasvr, Realiza la trama la cual será enviada.

```
package protocloRAP.servicios.OperacionesTramas;


public class Tramasvr {
	public Tramasvr(){

	}
	private int obtenerCRInt(byte[] arreglo){
		int crc = 0;
		int i = 0;
		int datos = 0;
		int datosInvertidos=0;
		int poly = 25;
		int opXor = 0;
		datos = this.arregloBytesAInt(arreglo);
		datosInvertidos = this.obtenerNumero(datos,16);
		if(datosInvertidos%2==0){
			opXor = (int)(datosInvertidos^0);
		}else{
			opXor = (int)(datosInvertidos^poly);

		}
		for (i=1;i<12;i++){
			crc= (int)(opXor>>>1);
			if(crc%2==0){
				opXor = (int)(crc^poly);
				crc = opXor;
			}


		}
		crc= (int)(opXor>>>1);
		crc = obtenerNumero(crc,4);
		return crc;
	}
	private int obtenerNumero(int valor, int bits){
		int[] tabla =
{1,2,3,8,16,32,64,128,256,512,1024,2048,4096,8192,16384,32768};
		int numero = valor;
		int nBits = bits;
		int numero = valor;
		int nBits = bits;
		int numeroInvertido = 0;
		for (int j = nBits-1; j>=0;j--){
			if(numero%2!=0){
				numeroInvertido = numeroInverido + tabla[j];
			}
			numero = numero>>>1;

		}
		return numeroInvetido;
	}
```

```
private byte[] intArregloBytes(int valor){
	byte[] resultado;
	byte b1 = (byte)((valor>>8)& 0xff);
	byte b0 = (byte)(valor & 0xff);
	resultado = new byte[]{b1, b0};
	return resultado;

}
public byte[] tramatrayectoriaocho(byte[] valorOcho,int sec){
	byte ocho = valorOcho[0];
	int crc = 0;
	int TramaEnteros = 0;
	byte[] trama = {(byte)64,(byte)0};
	switch(Ocho){
	case 10:
		trama[1]=(byte)160;
		break;
	case 15:
		trama[1] = (byte)240;
		break;
		default:
	}
	if(sec == 1){
		trama[0] = (byte)(trama[0]|(byte)32);
	}
	TramaEnteros = this.arregloBytesAInt(trama);
	crc = this.obtenerCRCInt(trama);
	TramaEnteros = TramaEnteros + crc;
	System.out.printIn(TramaEnteros:"+""\n");
	System.out.println(Integer.toBinaryString(TramaEnteros)+"\n");
	trama = this.intAreglosBytes(TramaEnteros);
	return trama;
}
public byte[] tramaFinalizacion(){
	int crc = 0;
	int TramaEnteros = 0;
	byte[] trama = {(byte)80,(byte)0};
	TramaEnteros = this.arreglosBytesAInt(trama);
	crc = this.obtenerCRCInt(trama);
	TramaEnteros = TramaEnteros + crc;
	System.out.println("TxFin(int):"+"\n");
	System.out.println(TramaEnteros + "\n");
	System.out.println("TxFin(bin):"+"\n");
	System.out.println(Integer.toBinaryString(TramaEnteros)+"\n");
	trama = this.intArregloBytes(TramaEnteros);
	return trama;
}
public byte[]tramaControl(int sec, int claseControl){
	int TramaEnteros = 0;
	byte[]trama = {(byte)0};
	switch(calsesControl){
	case 0:
		trama[0] = (byte)196;
		break;
	case 1:
```

```
                trama[0] = (byte)200;
                break;
            case 2:
                trama[0] = (byte)208;
                break;
            case 3:
                trama[0] = (byte)216;
                break;
                default:

            }
            if(claseControl=0){
                if(sec==1){
                    trama[0] = (byte)(trama[0]|(byte)32);

                }
            }
            TramaEnteros = this.arregloBytesAInt(trama);
            return trama;
        }
        public byte[]obtenerParamTramaMAxLong(byte[]trama){
            int TramaRecividaEntero = this.aregloBytesAInt(trama);
            System.out.println("Rx(int):"+"\n");
            System.out.println(TramaRecividaEntero + "\n");
            System.out.println("Rx(bin):"+"\n");

        System.out.println(Integer.toBinaryString(TramaRecividaEntero)+"\n");
            int tipoTrama = 0;
            int transmisor = 0;
            int sec= 0;
            int datos = 0;
            int comando = 0;
            int treyectoria = 0;
            int tipoTramaDatos = 0;
            byte[] tramaAck = {(byte)0};
            byte[] parametrosTrama2 =
{(byte)0,(byte)0,(byte)0,(byte)0,(byte)0,(byte)0};
            int fin = 0;
            int ack = 0;
            tipoTrama = (TramaRecividaEntero>>>15)&1;
            int verificarCRC = obtenerCRCInt(trama);
            int (verificarCRC == 0){
                tipoTrama == (TramaRecividaEntero>>>15)&1;
                int verificarCRC = obtenerCRCInt(trama);
                if (verificarCRC == 0){
                    transmisor = (TramaRecividaEntero>>>15)&1;
                    if(tipoTrama==0){
                        sec = (TramaRecividaEntero>>>14)&1;
                        if(transmisor==0){
                            sec = (TramaRecividaEntero>>>13)&1;
                        tipoTramaDatos = (TramaRecividaEntero>>>12)&1;
                                switch (tipoTramaDatos){
                                case 0:
                                    datos =
                                (TramaRecividaEntero>>>4)&31;
```

```
                                        if(datos==10){
                                              ocho=datos;
                        tramaAck = this.tramaControl(sec,3);
                                        }else if (datos == 15){
                                              ocho=dato;
                  tramaAck = this.tramaControl(treyectoria, claseControl);
                                        }else {
                                              comando = datos;
                              tramaAck = this.tramaControl(sec,1);
                                        }
                                        break;
                                    case 1:
                                        fin = 1;
                        tramaAck = this.tramaControl(sec, claseControl);
                                        break;
                                        default:
                                    }
                                }
                        }
                        else{
                              sec = (TramaRecividaEntero>>>13)&1;
                              tramaAck = this.tramaControl(sec, 0);
                              ack = 1;
                        }
                        int tramaAckInt = this.arregloBytesAInt(tramaAck);
                        parametrosTrama2[0] = (byte)fin;
                        parametrosTrama2[1] = (byte)comando;
                        parametrosTrama2[2] = tramaAck[0];
                        parametrosTrama2[3] = (byte)sec;
                        parametrosTrama2[4] = (byte)ack;
                        parametrosTrama2[5] = (byte)ocho;
                        byte[] acuseB = this.arregloBytesAInt(acuseB);
                        if (ack ==0){
                              System.out.println("ACKTx(int):" + "\n");
                              System.out.println(acuse + "\n");
                              System.out.println("ACKTx(bin):"+"\n");

      System.out.println(Integer.toBinaryString(acuse)+"\n");

                        }else{
                              System.out.println("NACKTX(int):"+"\n");
                              System.out.println(acuse + "\n");
                              System.out.println("NACKTx(bin):" + "\n");

      System.out.println(Integer.toBinaryString(acuse)+"\n");
                        }
                        return parametrosTrama2;

                  }
            }
      }
}
```

Clase TXRX, Es para la tranferencia de datos entre robots móviles LEGO EV3

```
package protocoloRAP.Servicios.transmision;

import AppRobotServer.utileria.interfaces.*;
import AppRobotServer.control.sensores.*;
import AppRobotServer.control.servos.*;
import protocoloRAP.canalRFCOMM.*
import protocoloRAP.singleton.*;
import lejos.ev3.*;
import java.io.*;


public class TxRx {
	private CanalRFCOMM canalRFCOMM;
	private byte[] tramaRecibida;
	private boolean ejecutar;
	private Sensores sns;
	private Servos svs;
	public TxRx (CanalRFCOMM canalRFCOMM,boolean inicarThread){
		this.sns = new Sensores(true);
		this.svs = this.sns.obtenerServos();
		this.CanalRFCOMM = canalRFCOMM;
		this.ejecutar = true;
		if(iniciarThread)
			this.start();
	}
	public void run(){
		this.iniciarRX();
	}
	private void iniciarRX(){
		while(this.ejecutar != false){
			synchronized (this){
				try{
					this.wait(1);
					this.adminitradorRX();
				}catch(InterruptedException ex){
				}catch(IOException ex){

				}
			}
		}
	}
	public synchrnized void txTrama(byte[]trama)throws IOException{
		this.canalRFCOMM.enviarTrama(trama);
	}
	public synchronized boolean chequearBuffer(int tamanioTrama)throws
IOEception{
```

```
			int datosDisponibles = this.canalRFCOMM.
					obtenerStreamEntrada().availabe();
			if(datosDisponibles != tamanioTrama){
				return false;
			}else{
				return true;
			}
		}

			}
public Sensores obtenerSensores(){
		return this.sns;
}
public Servos obtenerSevos(){
		return this.svs;
}
private void administradorRx()throws IOException, InterruptedException{
		if(this.chequearBuffer(2)){
			this.tramaRecivida = this.canalRFCOMM.recibirTrama();
			byte[]prametros = Protocolo.obtenerSAP().

		obtenerOperacionesTrama().obtenerParamTramaMaxLong(tramaRecibida);
			byte[] acuseEnviar = {parametros[2]};
			Protocolo.obtenerSAP().establecerTramaEnviar(acuseEnviar);

		Protocolo.obtenerSap().peticionServicio(TxRx.P_TRANSFERENCIA_DATOS);
			boolean respuestaSAP =
Protocolo.obtenerSAP().confirmacionServicio();
			if(respuestaSAP == TxRx.ACK_TRANS_TRAMA_SAP){
				if(parametros[4]== 0){
					switch(parametros[0]){
					case(byte)0:
						if(parametros[5]!=0){
							byte Mov = parametros[5];
							if(Mov==TxRx.MovL1){
								this.svs.establecerMovL1(true);
							}else if(Mov==TxRx.MovL2){
								this.svs.establecerMovL2(false);
							}else{

							}else{
								byte comando = parametros[1];
								this.ejecutarComando(comando);
							}
							break;
							case(byte)1:

		Protocolo.obtenerSAP().peticionServicio(TxRx.P_FINALIZACION_CONEXION);
							boolean respuesta =
Protocolo.obtenerSAP().confimacionServicio();
							if(respuesta){
								LCD.clearDisplay();
								LCD.drawString("Conexion",0,0);

		LCD.drawString("finalizada!!!",2,1);
```

```
                                        }
                                        break;
                                        default:
                                  }
                            }else{

                            }else{

      Protocolo.obtenerSAP(.peticionServicio(TxRx.P_FINALIZACION_CONEXION);
                            }
                      }
                }
                public synchronized void terminalHilo(){
                      this.ejecutar=false;
                      this.notifyAll();
                }
          }
```

Clase Sevos, es la trayectoria que realizara y al activar la rutina de la trayectoria forma de sinusoidal

```
package pack;

import lejos.hardware.motor.Motor;
import lejos.robotics.navigation.DifferentialPilot;

package AppRootServer.control.servos;

import AppRobotServer.utileria.interfaces.*;
import protocoloRAP.sngleton.*;

import lejos.utility.Delay;
import lejos.ev3.*;

import java.net.MalformedURLException;
import java.rmi.NotBoundException;
import java.rmi.RemoteException;




import lejos.hardware.*;
import lejos.remote.ev3.RemoteEV3;

public class lazos {

      DifferentialPilot pilot;
      boolean loop;

      public static void main(String[] args) {
            new Metodo();
```

```
	}
	public lazos(){
		pilot = new DifferentialPilot(1.5f, 6, Motor.A, Motor.B);
		for (int i = 0; i < 10; i++){
			travelAndRotate();
		}
		}

	}

	public void travelAndRotate() {
		pilot.travel(12);
		pilot.rotate(90);

			Motor.A.setSpeed(0);
			Motor.B.setSpeed(100);
			Motor.A.forward();
			Motor.B.fotward();
			Delay.msDelay(1000);


			Motor.A.setSpeed(61);
			Motor.B.setSpeed(60);
			Motor.A.forward();
			Motor.B.fotward();
			Delay.msDelay(2000);

			Motor.A.setSpeed(61);
			Motor.B.setSpeed(60);

			Motor.A.forward();
			Motor.B.fotward();
			Delay.msDelay(3000);


			Motor.A.setSpeed(10);
			Motor.B.setSpeed(100);

			Motor.A.forward();
			Motor.B.fotward();
			Delay.msDelay(2000);


	}

}
```

Robot móvil Esclavo EV3E

Clase MetodHilos, manda llamar a los hilos en la paquete del robot.

```
package AppRobot.mindRobot.utileria.intefaces;
```

```
public class MetodHilos {
	public abstract void terminarHilos();
}
```

Clase ParametrosBluetooth, manda llamar a los parámetros del protocolo de Bluetooth, los canales que se utilizaran en nuestro programa, asi como autentificación y el modo que tendrá nuestro nuestro robot ya sea maestro o esclavo en la red.

```
package AppRobot.midRobot.utilieria.interfaces;

public class ParametrosBluetooth {
	public static final String Protocolo_Bluetooth = "btspp://";
	public static final String CANAL_UNO = "1";
	public static final String CANAL_DOS = "2";
	public static final String CANAL_TRES = "3";
	public static final String AUTENTICACION = "authenticate=false";
	public static final String ENCRIPCION = "encrupt=false";

	public static final String MASTER = "master=false";

}
```

Clase Busqueda, Esta clase busca el robot móvil EV3 esclavo (EV3E) para realizar la comunicación por Bluetooth. Se manda llamar cada vez que se requiera hacer la conexión el robot Maestro al esclavo, si el tiempo.

```
package AppRobot.midRobot.utileria.pantallas;

import AppRobot.mindRobot.utileria.intefaces.*;
import AppRobot.mindRobot.utileria*;
import javax.microedition.lcdui.*;
import protocoloRAP.*;
import java.io.*;


public class BuquedaEV3 {

	private ControlRobotEV3E controlRobot;
	private Alerta errorConexion;
	public Busqueda(String titulo,ControlRobot control,Alerta errorConexion,
Display){
		super(titulo,disp);
		this.controlRobot = control;
		this.erroConexion = "errorConexion";
		String textEV3 = "Ingrese el nombre del robot";
		String nombreEV3 = "EV3E"
				this.txtNombreEV3 = new stringl (textoEV3);
				this.ingresarItem(this.txtNombreEV3);
				this.ingresarItem(this.lblNombreEV3);

	}
```

```
	private void administrarConexion(Espera espera)throws
	interruptedException.IOEception{
		String nombreEV3 = this.lblNombreEV3.getString();
		boolean respuesta = this.peticionConexionServidor(nombreEV3, 1);
		if(respuesta){
			this.controRobot.mostrarFormulario();
			espera.terminarHilo();
		}else{
			this.mostrarFormulario();
			espera.terminarHilo();
			this.errorConexion.mostrarAlerta();

		}
	}
	private boolean peticionConexionServidor(String nombreEV3,int canal)
throws interruptedException,IOException{
		Protocolo.obtenerSAP().establecerNombreEV3(nombreEV3);
		Protocolo.obteberSAP().establecerCanalComunicacion(canal);

	Protocolo.obtenerSAP().peticionServicio(Busqueda.P_Establecimiento_Conexi
on);
		boolean respuestaSAP =
Protocolo.obtenerSAP().confirmacionServicio();
		if(respuestaSAP){
			return true;
		}else{
			return false;
		}
	}
	public void HiloAdminitradorConexion(Espera espera){
		final Espera esp = espera;
			Runnable admin = new Runnable(){
			public void run(){
				try{
					administradorConexion(esp);
				}catch (Exception ex){

				}
			}
		};
		Thread TAdmin = new Thread(admin);
		TAdmin.start();
	}
}
```

Clase Protocolo, esta clase manda llamar a la clase se Acceso para todo los servicio se los protocolos de comunicación a utilizar

```
package protocoloRAP.singleton;

import protcoloRAP.SAP.*;
```

```
public class Protocolo {
	private static final Protocolo protocoloSingleton = new Protocolo();
	private static PuntoAccesoServicio sap;
	private Protocolo(){
		Protocolo.sap = new PuntoAccesoServicio(true);
	}
	public static Protcolo obtenerInstanciaProtocolo(){
		return protocoSingleton;
	}
	public static PuntoAccesoServicio obtenerSAP(){
		return Protocolo.sap;
	}

}
```

Clase PuntoAccesoServicio, esta clase brindara todos los servicio del protocolo de comunicación (ver tabla#) el cual como se explico en el apartado# se implementa para utilizar solo slot de comunicacion

```
package ProtocoloRAP.SAP;
package pack;

import protocoloRAP.servicio.OperacionesTramas.*;
import protocoloRAP.servicio.Sensores.*;
import protocoloRAP.servicio.transmision.*;
import AppRobotServer.utileria.interfaces.*;
import AppRobotServer.control.sensores.*;
import AppRobotServer.control.trayetoria.*;
import AppRobotServer.archivo.*;
import protocolo.canalRFCOMM.*;
import lejos.ev3.*;
import java.io.*;



public class PuntoAccesoServicio {

	private CanalRFCOMM canalRFCOMM;
	private ConexionBTsvr con;
	private boolean ejecutar;
	private byte[] servicio;
	private Sensores sns;
	private Servos svs;
	private OperacionesTramasve oT;
	private byte[] tramaEnviar;
	private Archivo archivoMotorA,archivoMotorB;
	public PuntoAccesoServicio(boolean iniciarSAP){
		this.txrx = null;
		this.sns = null;
		this.svs = null;
		this.canalRFCOMM = null;
```

```
            this.tramaEnviar = null;
            this.archivoMotorA = new Archivo
(PuntoAccesoServicio.Nombre_Archivo_A);
            this.archivoMotorB = new Archivo
(PuntoAccesoServicio.Nombre_Archivo_B);
            this.oT = new OperacionesTramasvr();
            this.ejecutar = true;
            this.con = new ConexionBTsvr(false);
            if(iniciarSAP)
                  this.star();
      }
      public void run(){
            this.iniciarSAP();
      }
      private void iniciarSAP(){
            while (this.ejecutar !=false){
                  synchronized (this){
                        try{
                              this.wait();

                        }catch(interruptedException ex){

                        }
                  }
            }
      }
      public synchronized void peticionServicio (byte mensaje){
            while(this.servicio !=null){
                  try{
                        this.wait();

                  }catch(interruptedException ex){

                  }
            }
            byte[]mensajeC = (mensaje);
            this.servicio = mensajeC;
            this.notifyAll();
      }
      public synchronized boolean confirmacionServicio() throw
interruptedException{
            while(this.servicio == null){
                  this.wait();
            }
            boolean respuesta = this.determinacionServicio(this.sevicio);
            this.servicio = null;
            return respues;
      }
      private boolean determinacionServicio(byte[] peticion){
            switch(peticion[0]){
            case PuntoAccesoServicio.P_ESTABLECIMIENTO_CONEXION;
            this.con.start();
            try{
                  boolean estadoCon =
this.confirmacionServicio().conexionEstablecida();
```

```
                        if(estadoCon){
                              this.canalRFCOMM = this.con.obtenerCanalRFCOMM();
                              this.txrx = new TXRX (this.canalRFCOMM,true);
                              this.sns = this.txrx.sensores();
                              return
PuntoAccesoServicio.ACK_Establecimiento_Conexion;

                        }else {
                              return
PuntoAccesoServicio.ACK_ESTABLECIMIENTO_CONEXION;
                        }
                  }catch(Exception ex){
                        return PuntoAccesoServicio.NACK_ESTABLECIMIENTO_CONEXION;
                  }
                  case PuntoAccesoServicio.P_TRANSFERENCIA_DATOS:
                        byte[] trama = this.tramaEnviar;
                        try{
                              this.txrx.txTramma(trama);

                        }catch(IOException ex){
                              return PuntoAccesoServicio.ACK_TRANS_TRAMA_SAP;
                        }
                        return PuntoAccesoServicio.ACK_TRANS_TRAMA_SAP;
                  case PuntoAccesoServicio.P_FINALIZACION_CONEXION;
                  this.svs.pararAA();
                  this.svs.pararID();
                  boolean trayectoria = this.svs.obtenerTraayectoria();
                  }
                  this.finalizarConexion();
                  return PuntoAccesoServicio.ACK_FINALIZACION_CNOEION;
                  default:
                        this.svs.paparAA();
                        this.svs.pararID();
                        LCD.clearDisplay();
                        LCD.drawString("Error",0,0);
                        this.finalizarConexion();
                        return PuntoAccesoServicio.ERROR;

            }
      }
      private void finalizarConexion(){
            if(this.archivoMotorA != null){
                  this.archivoMotorA = null;
            }
            if (this.archivoMotorB != null){
                  this.archivoMotorB = null;
            }
            if(this.sns.isAlive()){
                  this.sns.terminarHilo();
                  this.sns = null;
            }else {
                  this.sns = null;

            }
            if(this.svs != null){
```

```
			this.svs = null;
		}
		if(this.txtrx != null){
			if(this.txtrx.isAlive()){
				this.txrx.terminarHilo();
				this.txrx = null;
			}else {
				this.txtrx = null;
			}
		}
		if(this.canalRFCOMM != null){
			this.canalRFCOMM.desconectar();
			this.canalRFCOMM = null;
		}
		if(this.con != null){
			this.con = null;
		}
		this.terminarHilo();
		System.gc();
	}
	public OperacionesTramasvr obtenerOperacionesTrama(){
		return this.oT;
	}
	public synchronized void establecerTramaEnviar(byte[]tramaEnvia){
		this.tramaEnviar = tramaEnviar;
	}
	public synchronized Archivo obenerArchivoA(){
		return this.archivoMotorA;
	}
	public synchronized Archivo obtenerArchivoB(){
		return this.archivoMotorB;
	}
	public synchronized void terminarHilo(){
		this.ejecutar = false;
		this.notifyAll();
	}
	}
```

Clase RFCOMM, como ya se explico antes esta clase es la encargada de establecer la configuración en el canal de transmisión.

```
package protocoloRAP.canalRFCOMM;
package pack;

import javax.microedition.io.*;
import java.io.*;



public class canalRFCOMM {

	private StreamConnection strcon;
	private DataInputStream streamEntrada;
	private DataOutnputStream streamSalida;
	public canalRFCOMM (StreamConnection strcon){
		this.strcon = strcon;
		this.streamEntrada = this.strcon.openDataInputStream();
		this.streamSalida = this.strcon.openDataOutputStream();

	}
	public byte[] recibirTrama()throws IOException {
		if(this.streamEntrada.available() == 0){
			return null;
		}else {
			byte tramaRecibida[] = new
byte[this.streamEntrada.available()];
			this.streamEntrada.read(tramaRecibida);
			return tramaRecibida;
		}
	}
	public void enviarTrama (byte[]trama){
		final DataOutputStream dout = this.streamSalida;
		final byte[] datos = trama;
		Runnable env = new Runnable(){
			public void run(){
				try{
					dout.write(datos);
					dout.flush();

				}catch (IOException ex){

				}
			}
		};
		Thread tenv = new Thread(env);
		tenv.start();
	}
	public DataInputStream obtenerStreaEntrada(){
		return this.streamEntrada;

	}
	public DataOutputStream obtenerStreamSalida(){
		return this.streamSalida;
```

```
	}
	public StreamConnection obtenerStreamConexion(){
		return this.strcon;
	}
	public void desconectar(){
		try{
			if(this.streamEntrada != null){
				this.streamEntrada.close();
			}
		}catch(Exception e){

		}
		try{
			if(this.streamSalida != null)
				this.streamSalida.close();
		}catch(Exception e){

		}
		try{
			if(this.streamSalida != null)
				this.streamSalida.close();
		}catch(Exception e){

		}
		try{
			if(this.strcon != null)
				this.strcon.close();
		}catch (Exception e){}
			this.streamEntrada = null;
			this.streamSalida = null;
			this.strcon = null;
	}
}
```

Clase de ConeionBTsvr, Esta clase realiza la conexión entre robots móviles Lego EV3 maestro (EV3M) y EV3 ESCLAVO (EV3E). Activando el modulo de Bluetooth, realiza la sincronización entre dispositivos y la espera de envió de señal RF mientras encuentra el dipositivo.

```
Package protocoloRAP.servicios.conexion;

import protocoloRAP.canalRFCOMM.*;
import javax.microedition.io.*;
import lejos.ev3.comm.*;
import lejos.ev3.*;

public class ConeionBTsvr {

	private boolean inicioConexion;
```

```
	private CanalRFCOMM canalRFCOMM;
	public ConeionBTsvr(boolean iniciarervicor){
		if(iniciarServidor)
			this.start();
	}
	public void run(){
		this.inicarServidor();
	}
	private void inicarServidor(){
		try{
			LCD.drawString("Esperando",0,1);
			LCD.drawString(" conexion..", 0, 2);
			BTConnection btcon = Bluetooth.waitForConnection();
			btcon.setIOMode(EV3Connection.RAW);
			this.canalRFCOMM = new CanalRFCOMM
					((StreamConnection)btcon);
			synchronized (this){
				this.inicioConexion = true;
				notify();
			}
		}catch(Exception ex){

		}
	}
	public boolean conexionEstablecida() throws InterruptedException{
		int i = 0;
		while(true){
			synchronized(this){
				this.wait();
			}
			i = i + 1;
			if(i==1){
				break;
			}
		}
		if(this.inicioConeion){
			return true;
		}else{
			return false;
		}
	}
	public CanalRFCOMM obtenerCanalRFCOMM(){
		return this.canalRFCOMM;
	}
}
```

Clase Tramasvr, esta clase recibe la información y la ejecuta después de enviar el mensaje de confirmación al maestro.

```
package protocloRAP.servicios.OperacionesTramas;
```

```
public class Tramasvr {
	public Tramasvr(){

	}
	private int obtenerCRInt(byte[] arreglo){
		int crc = 0;
		int i = 0;
		int datos = 0;
		int datosInvertidos=0;
		int poly = 25;
		int opXor = 0;
		datos = this.arregloBytesAInt(arreglo);
		datosInvertidos = this.obtenerNumero(datos,16);
		if(datosInvertidos%2==0){
			opXor = (int)(datosInvertidos^0);
		}else{
			opXor = (int)(datosInvertidos^poly);

		}
		for (i=1;i<12;i++){
			crc= (int)(opXor>>>1);
			if(crc%2==0){
				opXor = (int)(crc^poly);
				crc = opXor;
			}


		}
		crc= (int)(opXor>>>1);
		crc = obtenerNumero(crc,4);
		return crc;
	}
	private int obtenerNumero(int valor, int bits){
		int[] tabla =
{1,2,3,8,16,32,64,128,256,512,1024,2048,4096,8192,16384,32768};
		int numero = valor;
		int nBits = bits;
		int numero = valor;
		int nBits = bits;
		int numeroInvertido = 0;
		for (int j = nBits-1; j>=0;j--){
			if(numero%2!=0){
				numeroInvertido = numeroInverido + tabla[j];
			}
			numero = numero>>>1;

		}
		return numeroInvetido;
	}
	private byte[] intArregloBytes(int valor){
		byte[] resultado;
		byte b1 = (byte)((valor>>8)& 0xff);
		byte b0 = (byte)(valor & 0xff);
		resultado = new byte[]{b1, b0};
```

```
        return resultado;

    }
    public byte[] tramatrayectoriaocho(byte[] valorOcho,int sec){
        byte ocho = valorOcho[0];
        int crc = 0;
        int TramaEnteros = 0;
        byte[] trama = {(byte)64,(byte)0};
        switch(Ocho){
        case 10:
              trama[1]=(byte)160;
              break;
        case 15:
              trama[1] = (byte)240;
              break;
              default:
        }
        if(sec == 1){
              trama[0] = (byte)(trama[0]|(byte)32);
        }
        TramaEnteros = this.arregloBytesAInt(trama);
        crc = this.obtenerCRCInt(trama);
        TramaEnteros = TramaEnteros + crc;
        System.out.printIn(TramaEnteros:"+""\n");
        System.out.println(Integer.toBinaryString(TramaEnteros)+"\n");
        trama = this.intAreglosBytes(TramaEnteros);
        return trama;
    }
    public byte[] tramaFinalizacion(){
        int crc = 0;
        int TramaEnteros = 0;
        byte[] trama = {(byte)80,(byte)0};
        TramaEnteros = this.arreglosBytesAInt(trama);
        crc = this.obtenerCRCInt(trama);
        TramaEnteros = TramaEnteros + crc;
        System.out.println("TxFin(int):"+"\n");
        System.out.println(TramaEnteros + "\n");
        System.out.println("TxFin(bin):"+"\n");
        System.out.println(Integer.toBinaryString(TramaEnteros)+"\n");
        trama = this.intArregloBytes(TramaEnteros);
        return trama;
    }
    public byte[]tramaControl(int sec, int claseControl){
        int TramaEnteros = 0;
        byte[]trama = {(byte)0};
        switch(calsesControl){
        case 0:
              trama[0] = (byte)196;
              break;
        case 1:
              trama[0] = (byte)200;
              break;
        case 2:
              trama[0] = (byte)208;
              break;
```

```
		case 3:
			trama[0] = (byte)216;
			break;
			default:

		}
		if(claseControl=0){
			if(sec==1){
				trama[0] = (byte)(trama[0]|(byte)32);

			}
		}
		TramaEnteros = this.arregloBytesAInt(trama);
		return trama;
	}
	public byte[]obtenerParamTramaMAxLong(byte[]trama){
		int TramaRecividaEntero = this.aregloBytesAInt(trama);
		System.out.println("Rx(int):"+"\n");
		System.out.println(TramaRecividaEntero + "\n");
		System.out.println("Rx(bin):"+"\n");

	System.out.println(Integer.toBinaryString(TramaRecividaEntero)+"\n");
		int tipoTrama = 0;
		int transmisor = 0;
		int sec= 0;
		int datos = 0;
		int comando = 0;
		int treyectoria = 0;
		int tipoTramaDatos = 0;
		byte[] tramaAck = {(byte)0};
		byte[] parametrosTrama2 =
{(byte)0,(byte)0,(byte)0,(byte)0,(byte)0,(byte)0};
		int fin = 0;
		int ack = 0;
		tipoTrama = (TramaRecividaEntero>>>15)&1;
		int verificarCRC = obtenerCRCInt(trama);
		int (verificarCRC == 0){
			tipoTrama == (TramaRecividaEntero>>>15)&1;
			int verificarCRC = obtenerCRCInt(trama);
			if (verificarCRC == 0){
				transmisor = (TramaRecividaEntero>>>15)&1;
				if(tipoTrama==0){
					sec = (TramaRecividaEntero>>>14)&1;
					if(transmisor==0){
						sec = (TramaRecividaEntero>>>13)&1;
					tipoTramaDatos = (TramaRecividaEntero>>>12)&1;
							switch (tipoTramaDatos){
							case 0:
								datos =
							(TramaRecividaEntero>>>4)&31;
								if(datos==10){
									ocho=datos;
				tramaAck = this.tramaControl(sec,3);
								}else if (datos == 15){
									ocho=dato;
```

```
                        tramaAck = this.tramaControl(treyectoria, claseControl);
                                                }else {
                                                        comando = datos;
                                        tramaAck = this.tramaControl(sec,1);
                                                }
                                                break;
                                        case 1:
                                                fin = 1;
                                tramaAck = this.tramaControl(sec, claseControl);
                                                break;
                                                default:
                                        }
                                }
                        }
                        else{
                                sec = (TramaRecividaEntero>>>13)&1;
                                tramaAck = this.tramaControl(sec, 0);
                                ack = 1;
                        }
                        int tramaAckInt = this.arregloBytesAInt(tramaAck);
                        parametrosTrama2[0] = (byte)fin;
                        parametrosTrama2[1] = (byte)comando;
                        parametrosTrama2[2] = tramaAck[0];
                        parametrosTrama2[3] = (byte)sec;
                        parametrosTrama2[4] = (byte)ack;
                        parametrosTrama2[5] = (byte)sen;
                        byte[] acuseB = this.arregloBytesAInt(acuseB);
                        if (ack ==0){
                                System.out.println("ACKTx(int):" + "\n");
                                System.out.println(acuse + "\n");
                                System.out.println("ACKTx(bin):"+"\n");

        System.out.println(Integer.toBinaryString(acuse)+"\n");

                        }else{
                                System.out.println("NACKTX(int):"+"\n");
                                System.out.println(acuse + "\n");
                                System.out.println("NACKTx(bin):" + "\n");

        System.out.println(Integer.toBinaryString(acuse)+"\n");
                        }
                        return parametrosTrama2;

                }
        }
    }
}
```

Clase TXRX, Es para la tranferencia de datos entre robots móviles LEGO EV3

```
package protocoloRAP.Servicios.transmision;
```

```
import AppRobotServer.utileria.interfaces.*;
import AppRobotServer.control.sensores.*;
import AppRobotServer.control.servos.*;
import protocoloRAP.canalRFCOMM.*
import protocoloRAP.singleton.*;
import lejos.ev3.*;
import java.io.*;


public class TxRx {
	private CanalRFCOMM canalRFCOMM;
	private byte[] tramaRecibida;
	private boolean ejecutar;
	private Sensores sns;
	private Servos svs;
	public TxRx (CanalRFCOMM canalRFCOMM,boolean inicarThread){
		this.sns = new Sensores(true);
		this.svs = this.sns.obtenerServos();
		this.CanalRFCOMM = canalRFCOMM;
		this.ejecutar = true;
		if(iniciarThread)
			this.start();
	}
	public void run(){
		this.iniciarRX();
	}
	private void iniciarRX(){
		while(this.ejecutar != false){
			synchronized (this){
				try{
					this.wait(1);
					this.adminitradorRX();
				}catch(InterruptedException ex){
				}catch(IOException ex){

				}
			}
		}
	}
	public synchrnized void txTrama(byte[]trama)throws IOException{
		this.canalRFCOMM.enviarTrama(trama);
	}
	public synchronized boolean chequearBuffer(int tamanioTrama)throws
IOEception{
		int datosDisponibles = this.canalRFCOMM.
				obtenerStreamEntrada().availabe();
		if(datosDisponibles != tamanioTrama){
			return false;
		}else{
			return true;
		}
	}

		}
```

```
public Sensores obtenerSensores(){
	return this.sns;
}
public Servos obtenerSevos(){
	return this.svs;
}
private void administradorRx()throws IOException, InterruptedException{
	if(this.chequearBuffer(2)){
		this.tramaRecivida = this.canalRFCOMM.recibirTrama();
		byte[]prametros = Protocolo.obtenerSAP().

	obtenerOperacionesTrama().obtenerParamTramaMaxLong(tramaRecibida);
		byte[] acuseEnviar = {parametros[2]};
		Protocolo.obtenerSAP().establecerTramaEnviar(acuseEnviar);

	Protocolo.obtenerSap().peticionServicio(TxRx.P_TRANSFERENCIA_DATOS);
		boolean respuestaSAP =
Protocolo.obtenerSAP().confirmacionServicio();
		if(respuestaSAP == TxRx.ACK_TRANS_TRAMA_SAP){
			if(parametros[4]== 0){
				switch(parametros[0]){
				case(byte)0:
					if(parametros[5]!=0){
						byte Mov = parametros[5];
						if(Mov==TxRx.MovL1){
							this.svs.establecerMovL1(true);
						}else if(Mov==TxRx.MovL2){
							this.svs.establecerMovL2(false);
						}else{

						}else{
							byte comando = parametros[1];
							this.ejecutarComando(comando);
						}
						break;
						case(byte)1:

	Protocolo.obtenerSAP().peticionServicio(TxRx.P_FINALIZACION_CONEXION);
						boolean respuesta =
Protocolo.obtenerSAP().confimacionServicio();
						if(respuesta){
							LCD.clearDisplay();
							LCD.drawString("Conexion",0,0);

	LCD.drawString("finalizada!!!",2,1);
						}
						break;
						default:
					}
				}else{

				}else{

	Protocolo.obtenerSAP(.peticionServicio(TxRx.P_FINALIZACION_CONEXION);
				}
```

```
                }
        }
        public synchronized void terminalHilo(){
                this.ejecutar=false;
                this.notifyAll();
        }
}
```

Clase Sevos, es la trayectoria que realizara y al activar la rutina de la trayectoria forma de sinusoidal

```
package pack;

import lejos.hardware.motor.Motor;
import lejos.robotics.navigation.DifferentialPilot;

package AppRootServer.control.servos;

import AppRobotServer.utileria.interfaces.*;
import protocoloRAP.sngleton.*;

import lejos.utility.Delay;
import lejos.ev3.*;

import java.net.MalformedURLException;
import java.rmi.NotBoundException;
import java.rmi.RemoteException;




import lejos.hardware.*;
import lejos.remote.ev3.RemoteEV3;

public class lazos {

        DifferentialPilot pilot;
        boolean loop;

        public static void main(String[] args) {
                new Metodo();


        }
        public lazos(){
                pilot = new DifferentialPilot(1.5f, 6, Motor.A, Motor.B);
                for (int i = 0; i < 10; i++){
                        travelAndRotate();
                }
                }

        }
```

```
	public void travelAndRotate() {
		pilot.travel(12);
		pilot.rotate(90);

			Motor.A.setSpeed(0);
			Motor.B.setSpeed(100);
			Motor.A.forward();
			Motor.B.fotward();
			Delay.msDelay(1000);


			Motor.A.setSpeed(61);
			Motor.B.setSpeed(60);
			Motor.A.forward();
			Motor.B.fotward();
			Delay.msDelay(2000);

			Motor.A.setSpeed(61);
			Motor.B.setSpeed(60);

			Motor.A.forward();
			Motor.B.fotward();
			Delay.msDelay(3000);


			Motor.A.setSpeed(10);
			Motor.B.setSpeed(100);

			Motor.A.forward();
			Motor.B.fotward();
			Delay.msDelay(2000);


	}

}
```

Anexo B.11 Trayectoria en forma ocho

Clase Tramasvr, Realiza la trama la cual será enviada.

```
package protocloRAP.servicios.OperacionesTramas;


public class Tramasvr {
	public Tramasvr(){

	}
	private int obtenerCRInt(byte[] arreglo){
		int crc = 0;
		int i = 0;
		int datos = 0;
		int datosInvertidos=0;
		int poly = 25;
		int opXor = 0;
		datos = this.arregloBytesAInt(arreglo);
		datosInvertidos = this.obtenerNumero(datos,16);
		if(datosInvertidos%2==0){
			opXor = (int)(datosInvertidos^0);
		}else{
			opXor = (int)(datosInvertidos^poly);

		}
		for (i=1;i<12;i++){
			crc= (int)(opXor>>>1);
			if(crc%2==0){
				opXor = (int)(crc^poly);
				crc = opXor;
			}


		}
		crc= (int)(opXor>>>1);
		crc = obtenerNumero(crc,4);
		return crc;
	}
	private int obtenerNumero(int valor, int bits){
		int[] tabla =
{1,2,3,8,16,32,64,128,256,512,1024,2048,4096,8192,16384,32768};
		int numero = valor;
		int nBits = bits;
		int numero = valor;
		int nBits = bits;
		int numeroInvertido = 0;
		for (int j = nBits-1; j>=0;j--){
			if(numero%2!=0){
				numeroInvertido = numeroInverido + tabla[j];
			}
```

```
            numero = numero>>>1;

        }
        return numeroInvetido;
}
private byte[] intArregloBytes(int valor){
        byte[] resultado;
        byte b1 = (byte)((valor>>8)& 0xff);
        byte b0 = (byte)(valor & 0xff);
        resultado = new byte[]{b1, b0};
        return resultado;

}
public byte[] tramatrayectoriaocho(byte[] valorOcho,int sec){
        byte ocho = valorOcho[0];
        int crc = 0;
        int TramaEnteros = 0;
        byte[] trama = {(byte)64,(byte)0};
        switch(Ocho){
        case 10:
              trama[1]=(byte)160;
              break;
        case 15:
              trama[1] = (byte)240;
              break;
              default:
        }
        if(sec == 1){
              trama[0] = (byte)(trama[0]|(byte)32);
        }
        TramaEnteros = this.arregloBytesAInt(trama);
        crc = this.obtenerCRCInt(trama);
        TramaEnteros = TramaEnteros + crc;
        System.out.printIn(TramaEnteros:"+""\n");
        System.out.println(Integer.toBinaryString(TramaEnteros)+"\n");
        trama = this.intAreglosBytes(TramaEnteros);
        return trama;
}
public byte[] tramaFinalizacion(){
        int crc = 0;
        int TramaEnteros = 0;
        byte[] trama = {(byte)80,(byte)0};
        TramaEnteros = this.arreglosBytesAInt(trama);
        crc = this.obtenerCRCInt(trama);
        TramaEnteros = TramaEnteros + crc;
        System.out.println("TxFin(int):"+"\n");
        System.out.println(TramaEnteros + "\n");
        System.out.println("TxFin(bin):"+"\n");
        System.out.println(Integer.toBinaryString(TramaEnteros)+"\n");
        trama = this.intArregloBytes(TramaEnteros);
        return trama;
}
public byte[]tramaControl(int sec, int claseControl){
        int TramaEnteros = 0;
        byte[]trama = {(byte)0};
```

```
		switch(calsesControl){
		case 0:
			trama[0] = (byte)196;
			break;
		case 1:
			trama[0] = (byte)200;
			break;
		case 2:
			trama[0] = (byte)208;
			break;
		case 3:
			trama[0] = (byte)216;
			break;
			default:

		}
		if(claseControl=0){
			if(sec==1){
				trama[0] = (byte)(trama[0]|(byte)32);

			}
		}
		TramaEnteros = this.arregloBytesAInt(trama);
		return trama;
	}
	public byte[]obtenerParamTramaMAxLong(byte[]trama){
		int TramaRecividaEntero = this.aregloBytesAInt(trama);
		System.out.println("Rx(int):"+"\n");
		System.out.println(TramaRecividaEntero + "\n");
		System.out.println("Rx(bin):"+"\n");

	System.out.println(Integer.toBinaryString(TramaRecividaEntero)+"\n");
		int tipoTrama = 0;
		int transmisor = 0;
		int sec= 0;
		int datos = 0;
		int comando = 0;
		int treyectoria = 0;
		int tipoTramaDatos = 0;
		byte[] tramaAck = {(byte)0};
		byte[] parametrosTrama2 =
{(byte)0,(byte)0,(byte)0,(byte)0,(byte)0,(byte)0};
		int fin = 0;
		int ack = 0;
		tipoTrama = (TramaRecividaEntero>>>15)&1;
		int verificarCRC = obtenerCRCInt(trama);
		int (verificarCRC == 0){
			tipoTrama == (TramaRecividaEntero>>>15)&1;
			int verificarCRC = obtenerCRCInt(trama);
			if (verificarCRC == 0){
				transmisor = (TramaRecividaEntero>>>15)&1;
				if(tipoTrama==0){
					sec = (TramaRecividaEntero>>>14)&1;
					if(transmisor==0){
						sec = (TramaRecividaEntero>>>13)&1;
```

```
                        tipoTramaDatos = (TramaRecividaEntero>>>12)&1;
                                    switch (tipoTramaDatos){
                                    case 0:
                                          datos =
                                    (TramaRecividaEntero>>>4)&31;
                                          if(datos==10){
                                                ocho=datos;
                  tramaAck = this.tramaControl(sec,3);
                                          }else if (datos == 15){
                                                ocho=dato;
            tramaAck = this.tramaControl(treyectoria, claseControl);
                                          }else {
                                                comando = datos;
                        tramaAck = this.tramaControl(sec,1);
                                          }
                                          break;
                                    case 1:
                                          fin = 1;
                  tramaAck = this.tramaControl(sec, claseControl);
                                          break;
                                          default:
                                    }
                              }
                  }
                  else{
                        sec = (TramaRecividaEntero>>>13)&1;
                        tramaAck = this.tramaControl(sec, 0);
                        ack = 1;
                  }
                  int tramaAckInt = this.arregloBytesAInt(tramaAck);
                  parametrosTrama2[0] = (byte)fin;
                  parametrosTrama2[1] = (byte)comando;
                  parametrosTrama2[2] = tramaAck[1];
                  parametrosTrama2[3] = (byte)sec;
                  parametrosTrama2[4] = (byte)ack;
                  parametrosTrama2[5] = (byte)ocho;
                  byte[] acuseB = this.arregloBytesAInt(acuseB);
                  if (ack ==0){
                        System.out.println("ACKTx(int):" + "\n");
                        System.out.println(acuse + "\n");
                        System.out.println("ACKTx(bin):"+"\n");

System.out.println(Integer.toBinaryString(acuse)+"\n");

                  }else{
                        System.out.println("NACKTX(int):"+"\n");
                        System.out.println(acuse + "\n");
                        System.out.println("NACKTx(bin):" + "\n");

System.out.println(Integer.toBinaryString(acuse)+"\n");
                  }
                  return parametrosTrama2;

            }
      }
```

```
	}
}
```

Clase Sevos, es la trayectoria que realizara y al activar la rutina de la trayectoria forma de ocho

```
package pack;

import lejos.hardware.motor.Motor;
import lejos.robotics.navigation.DifferentialPilot;

package AppRootServer.control.servos;

import AppRobotServer.utileria.interfaces.*;
import protocoloRAP.sngleton.*;

import lejos.utility.Delay;
import lejos.ev3.*;

import java.net.MalformedURLException;
import java.rmi.NotBoundException;
import java.rmi.RemoteException;



import lejos.hardware.*;
import lejos.remote.ev3.RemoteEV3;

public class lazos {

	DifferentialPilot pilot;
	boolean loop;

	public static void main(String[] args) {
		new Metodo();


	}
	public lazos(){
		pilot = new DifferentialPilot(1.5f, 6, Motor.A, Motor.B);
		for (int i = 0; i < 10; i++){
			travelAndRotate();
		}
		}

	}

	public void travelAndRotate() {
		pilot.travel(12);
		pilot.rotate(90);
```

```
                    Motor.A.setSpeed(0);
                    Motor.B.setSpeed(100);
                    Motor.A.forward();
                    Motor.B.fotward();
                    Delay.msDelay(1000);


                    Motor.A.setSpeed(100);
                    Motor.B.setSpeed(10);
                    Motor.A.forward();
                    Motor.B.fotward();
                    Delay.msDelay(2000);

                    Motor.A.setSpeed(61);
                    Motor.B.setSpeed(60);

                    Motor.A.forward();
                    Motor.B.fotward();
                    Delay.msDelay(3000);


                    Motor.A.setSpeed(10);
                    Motor.B.setSpeed(100);

                    Motor.A.forward();
                    Motor.B.fotward();
                    Delay.msDelay(2000);


                    Motor.A.setSpeed(61);
                    Motor.B.setSpeed(60);

                    Motor.A.forward();
                    Motor.B.fotward();
                    Delay.msDelay(2000);



                    Motor.A.setSpeed(100);
                    Motor.B.setSpeed(10);

                    Motor.A.forward();
                    Motor.B.fotward();
                    Delay.msDelay(2000);
        }

}
```

Anexo B.12 Trayectoria en posiciones distintas

Clase Tramasvr, Realiza la trama la cual será enviada.

```
package protocloRAP.servicios.OperacionesTramas;


public class Tramasvr {
	public Tramasvr(){

	}
	private int obtenerCRInt(byte[] arreglo){
		int crc = 0;
		int i = 0;
		int datos = 0;
		int datosInvertidos=0;
		int poly = 25;
		int opXor = 0;
		datos = this.arregloBytesAInt(arreglo);
		datosInvertidos = this.obtenerNumero(datos,16);
		if(datosInvertidos%2==0){
			opXor = (int)(datosInvertidos^0);
		}else{
			opXor = (int)(datosInvertidos^poly);

		}
		for (i=1;i<12;i++){
			crc= (int)(opXor>>>1);
			if(crc%2==0){
				opXor = (int)(crc^poly);
				crc = opXor;
			}


		}
		crc= (int)(opXor>>>1);
		crc = obtenerNumero(crc,4);
		return crc;
	}
	private int obtenerNumero(int valor, int bits){
		int[] tabla =
{1,2,3,8,16,32,64,128,256,512,1024,2048,4096,8192,16384,32768};
		int numero = valor;
		int nBits = bits;
		int numero = valor;
		int nBits = bits;
		int numeroInvertido = 0;
		for (int j = nBits-1; j>=0;j--){
			if(numero%2!=0){
				numeroInvertido = numeroInverido + tabla[j];
			}
```

```
            numero = numero>>>1;

        }
        return numeroInvetido;
}
private byte[] intArregloBytes(int valor){
        byte[] resultado;
        byte b1 = (byte)((valor>>8)& 0xff);
        byte b0 = (byte)(valor & 0xff);
        resultado = new byte[]{b1, b0};
        return resultado;

}
public byte[] tramatrayectoriaocho(byte[] valorOcho,int sec){
        byte ocho = valorOcho[0];
        int crc = 0;
        int TramaEnteros = 0;
        byte[] trama = {(byte)64,(byte)0};
        switch(Ocho){
        case 10:
              trama[1]=(byte)160;
              break;
        case 15:
              trama[1] = (byte)240;
              break;
              default:
        }
        if(sec == 1){
              trama[0] = (byte)(trama[0]|(byte)32);
        }
        TramaEnteros = this.arregloBytesAInt(trama);
        crc = this.obtenerCRCInt(trama);
        TramaEnteros = TramaEnteros + crc;
        System.out.printIn(TramaEnteros:"+""\n");
        System.out.println(Integer.toBinaryString(TramaEnteros)+"\n");
        trama = this.intAreglosBytes(TramaEnteros);
        return trama;
}
public byte[] tramaFinalizacion(){
        int crc = 0;
        int TramaEnteros = 0;
        byte[] trama = {(byte)80,(byte)0};
        TramaEnteros = this.arreglosBytesAInt(trama);
        crc = this.obtenerCRCInt(trama);
        TramaEnteros = TramaEnteros + crc;
        System.out.println("TxFin(int):"+"\n");
        System.out.println(TramaEnteros + "\n");
        System.out.println("TxFin(bin):"+"\n");
        System.out.println(Integer.toBinaryString(TramaEnteros)+"\n");
        trama = this.intArregloBytes(TramaEnteros);
        return trama;
}
public byte[]tramaControl(int sec, int claseControl){
        int TramaEnteros = 0;
        byte[]trama = {(byte)0};
```

```
		switch(calsesControl){
		case 0:
			trama[0] = (byte)196;
			break;
		case 1:
			trama[0] = (byte)200;
			break;
		case 2:
			trama[0] = (byte)208;
			break;
		case 3:
			trama[0] = (byte)216;
			break;
			default:

		}
		if(claseControl=0){
			if(sec==1){
				trama[0] = (byte)(trama[0]|(byte)32);

			}
		}
		TramaEnteros = this.arregloBytesAInt(trama);
		return trama;
	}
	public byte[]obtenerParamTramaMAxLong(byte[]trama){
		int TramaRecividaEntero = this.aregloBytesAInt(trama);
		System.out.println("Rx(int):"+"\n");
		System.out.println(TramaRecividaEntero + "\n");
		System.out.println("Rx(bin):"+"\n");

	System.out.println(Integer.toBinaryString(TramaRecividaEntero)+"\n");
		int tipoTrama = 0;
		int transmisor = 0;
		int sec= 0;
		int datos = 0;
		int comando = 0;
		int treyectoria = 11;
		int tipoTramaDatos = 0;
		byte[] tramaAck = {(byte)0};
		byte[] parametrosTrama2 =
{(byte)0,(byte)0,(byte)0,(byte)0,(byte)0,(byte)0};
		int fin = 0;
		int ack = 0;
		tipoTrama = (TramaRecividaEntero>>>15)&1;
		int verificarCRC = obtenerCRCInt(trama);
		int (verificarCRC == 0){
			tipoTrama == (TramaRecividaEntero>>>15)&1;
			int verificarCRC = obtenerCRCInt(trama);
			if (verificarCRC == 0){
				transmisor = (TramaRecividaEntero>>>15)&1;
				if(tipoTrama==0){
					sec = (TramaRecividaEntero>>>14)&1;
					if(transmisor==0){
						sec = (TramaRecividaEntero>>>13)&1;
```

```
                        tipoTramaDatos = (TramaRecividaEntero>>>12)&1;
                                    switch (tipoTramaDatos){
                                    case 0:
                                          datos =
                                    (TramaRecividaEntero>>>4)&31;
                                          if(datos==10){
                                                ocho=datos;
                  tramaAck = this.tramaControl(sec,3);
                                          }else if (datos == 15){
                                                ocho=dato;
            tramaAck = this.tramaControl(treyectoria, claseControl);
                                          }else {
                                                comando = datos;
                        tramaAck = this.tramaControl(sec,1);
                                          }
                                          break;
                                    case 1:
                                          fin = 1;
                  tramaAck = this.tramaControl(sec, claseControl);
                                          break;
                                          default:
                                    }
                              }
                  }
                  else{
                        sec = (TramaRecividaEntero>>>13)&1;
                        tramaAck = this.tramaControl(sec, 0);
                        ack = 1;
                  }
                  int tramaAckInt = this.arregloBytesAInt(tramaAck);
                  parametrosTrama2[0] = (byte)fin;
                  parametrosTrama2[1] = (byte)comando;
                  parametrosTrama2[2] = tramaAck[1];
                  parametrosTrama2[3] = (byte)sec;
                  parametrosTrama2[4] = (byte)ack;
                  parametrosTrama2[5] = (byte)ocho;
                  byte[] acuseB = this.arregloBytesAInt(acuseB);
                  if (ack ==0){
                        System.out.println("ACKTx(int):" + "\n");
                        System.out.println(acuse + "\n");
                        System.out.println("ACKTx(bin):"+"\n");

System.out.println(Integer.toBinaryString(acuse)+"\n");

                  }else{
                        System.out.println("NACKTX(int):"+"\n");
                        System.out.println(acuse + "\n");
                        System.out.println("NACKTx(bin):" + "\n");

System.out.println(Integer.toBinaryString(acuse)+"\n");
                  }
                  return parametrosTrama2;

            }
      }
```

```
	}
}
```

Clase Sevos, es la trayectoria que utilizara el robot móvil maestro EV3M que realizara y al activar la rutina de la trayectoria forma de sinusoidal

```
import lejos.hardware.Brick;
import lejos.hardware.BrickFinder;
import lejos.hardware.Button;
import lejos.hardware.motor.Motor;
import lejos.hardware.port.Port;
import lejos.hardware.sensor.EV3UltrasonicSensor;
import lejos.robotics.navigation.DifferentialPilot;

import lejos.utility.Delay;

public class UltrasonicDistance {

	DifferentialPilot pilot;
	Ultrasonic ultrasonic;
	public static void main(String[] args) {
		new UltrasonicDistance();


	}
	public UltrasonicDistance(){
	pilot = new DifferentialPilot(1.5f, 6 Motor.A, Motor.B);
	Brick b = BrickFinder.getDefaut();
	Port s2 = b.gePort("S2");
	EV3UltrasonicSensor us = new EV3UltrasonicSensor(s2);
	ultrasonic = new Ultrasonic(us.getMode("Distance"));

	while (true) {
		Delay.msDelay(2);
		Motor.A.setSpeed(-61);
				Motor.B.setSpeed(-60);

				Motor.A.forward();
				Motor.B.fotward();
				Delay.msDelay(1000);

				Motor.A.setSpeed(100);
				Motor.B.setSpeed(-30);
				Motor.A.forward();
				Motor.B.fotward();
				Delay.msDelay(3750);

				Motor.A.setSpeed(100);
				Motor.B.setSpeed(100);

				Motor.A.forward();
				Motor.B.fotward();
```

```
                        Delay.msDelay(1000);
                float distance = ultrasonic.distance();
                if (distance < 0.3){

                        motorA.stop();
                        motorB.stop();
                        Delay.msDelay(1000);

                        Motor.A.setSpeed(100);
                        Motor.B.setSpeed(-30);
                        Motor.A.forward();
                        Motor.B.fotward();
                        Delay.msDelay(1000);

                        Motor.A.setSpeed(61);
                        Motor.B.setSpeed(60);
                        Motor.A.forward();
                        Motor.B.fotward();
                        Delay.msDelay(1000);

                }else {
                        pilot.stop();
                }

                if (Button.ESCAPE.isDown()){
                        pilot.stop();
                        us.close();
                        System.exit(0);
                }
        }
        }

}
```

Clase main, Esta clase manda llamar al puerto donde se encuentra el sensor y lo configura para la aplicación deseada.

```
public class Main {

        public static void main(String[] args) throws InterruptedException {
                final EV3ColorSensor colorSensor = new
EV3ColorSensor(SensorPort.S2);
                final SensorMode mode = colorSensor.getColorIDMode();
                final ColorRecognizerThread colorRecognizerThread = new
ColorRecognizerThread(colorSensor);

                colorRecognizerThread.start();

                Buttom.waitForAnyPress();
        }

}
```

Clase Sevos, es la trayectoria que utilizara el robot móvil Esclavo EV3E que realizara y al activar la rutina de la trayectoria forma de sinusoidal

```
import lejos.hardware.Brick;
import lejos.hardware.BrickFinder;
import lejos.hardware.Button;
import lejos.hardware.motor.Motor;
import lejos.hardware.port.Port;
import lejos.hardware.sensor.EV3UltrasonicSensor;
import lejos.robotics.navigation.DifferentialPilot;

import lejos.utility.Delay;

public class UltrasonicDistance {

	DifferentialPilot pilot;
	Ultrasonic ultrasonic;
	public static void main(String[] args) {
		new UltrasonicDistance();


	}
	public UltrasonicDistance(){
	pilot = new DifferentialPilot(1.5f, 6 Motor.A, Motor.B);
	Brick b = BrickFinder.getDefaut();
	Port s2 = b.gePort("S2");
	EV3UltrasonicSensor us = new EV3UltrasonicSensor(s2);
	ultrasonic = new Ultrasonic(us.getMode("Distance"));

	while (true) {
		Delay.msDelay(2);
		Motor.A.setSpeed(61);
		Motor.B.setSpeed(60);
		Motor.A.forward();
		Motor.B.fotward();

		Motor.A.setSpeed(70);
		Motor.B.setSpeed(-30);
		Motor.A.forward();
		Motor.B.fotward();
		Delay.msDelay(1000);

		float distance = ultrasonic.distance();
		if (distance < 0.3){

			motorA.stop();
			motorB.stop();
			Delay.msDelay(1000);

			Motor.A.setSpeed(100);
			Motor.B.setSpeed(-30);
```

```
                Motor.A.forward();
                Motor.B.fotward();
                Delay.msDelay(1000);

                Motor.A.setSpeed(61);
                Motor.B.setSpeed(60);
                Motor.A.forward();
                Motor.B.fotward();
                Delay.msDelay(1000);

            }else {
                pilot.stop();
            }

            if (Button.ESCAPE.isDown()){
                pilot.stop();
                us.close();
                System.exit(0);
            }
        }
        }

}
```