ESCUELA TÉCNICA SUPERIOR DE INGENIEROS INDUSTRIALES Y DE TELECOMUNICACIÓN

Titulación:

INGENIERO TÉCNICO EN INFORMÁTICA DE GESTIÓN

Título del proyecto:

DISEÑO, PROGRAMACIÓN Y USO DE MINIROBOTS PARA COMPETICIONES ROBÓTICAS

Gloria Martínez Magallón

Alfredo Pina Calafi

Pamplona, 20 de Abril de 2011

En primer lugar agradezco a Alfredo Pina, tutor de este Proyecto Fin de Carrera, todo su apoyo durante todo el proceso e involucración.

A Patxi Oliva por su ayuda y por los momentos vividos en las competiciones robóticas.

A mis padres, que sin ellos no hubiese sido posible llegar hasta aquí, por su esfuerzo y confianza para poder realizar estos estudios.

A Cristina y Miguel por su apoyo y afecto en todo momento, sin ellos no hubiese sido posible terminar este proyecto.

Finalmente agradecer a la UPNA por financiar la compra de robots y poner los medios necesarios para poder desarrollar este proyecto.

1 OBJETO DEL PROYECTO

Este proyecto de fin de carrera plantea un doble objetivo, por un lado iniciarse en el ámbito de las competiciones robóticas y por otro lado diseñar y programar robots capaces de competir en esas pruebas.

En una primera fase se estudiarán las competiciones robóticas que se desarrollan en España así como las diferentes pruebas y modalidades en las cuales se puede competir (Velocistas, Rastreadores, Baile sincronizado, etc).

En una segunda fase se tratara de diseñar robots competitivos, de estudiar estrategias de juego adaptadas a las modalidades de las pruebas y de programar los robots para poder competir.

Entre los aspectos que interesa trabajar están el estudio de los diferentes kits de robots existentes así como los diferentes entornos de desarrollo y programación (tanto textuales como gráficos).

Este proyecto (como otros de temática similar) persigue generar un conocimiento o "know how" en este ámbito, que permita adquirir las competencias básicas para poder participar en competiciones oficiales tanto a los autores de estos PFC's como a otros estudiantes de la UPNA; todo este "know how" se reflejará en el material generado, tanto electrónico como escrito.

2 INTRODUCCIÓN

2.1 INICIOS DEL MUNDO DE LA ROBÓTICA

El mundo de la robótica ha existido desde siempre, [1]. En el mundo de la literatura se ha escrito mucho sobre la robótica, [2].

Sin duda, el autor más productivo sobre la robótica fue Isaac Asimov (1920-1992), que escribió muchos libros sobre los robots y su iteración con los seres humanos. Se preocupó sobre el miedo que el ser humano tenía hacía los robots, y se esforzó en desarrollar una serie de instrucciones que debería darse a los robots para reducir el riesgo que estos presentaban para los humanos. Así desarrolló, ***Las tres leyes de la robótica.***

1 *Ningún robot causará daño a un ser humano o permitirá, con su inacción, que un ser humano sufra daño*.

2 *Todo robot obedecerá las órdenes que le den los seres humanos, a menos que esas órdenes entren en conflicto con la primera ley*.

3 *Y todo robot debe proteger su propia existencia, siempre que esa protección no entre en conflicto con la primera o la segunda ley.*

En 1942, Asimov escribió por primera vez sobre estas leyes [3]. En los siguientes años escribió otros relatos, que en 1950 fueron recogidos en la colección [4].

A lo largo de la historia se ha ido avanzando en la robótica y hoy en día, la robótica es algo cotidiano, así hoy en día los robots son ampliamente utilizados sobre todo en plantas de manufactura, montaje y embalaje, en transporte, en exploraciones en la Tierra y en el espacio, cirugía, armamento, investigación en laboratorios y en la producción en masa de bienes industriales o de consumo.

2.2 EXPERIENCIA EN EL MUNDO DE LA ROBÓTICA

Para la elaboración de este proyecto de fin de carrera se decidió iniciarse en este mundo, estudiando los diferentes eventos que se llevan a cabo tanto a nivel nacional como a nivel internacional.

Aunque hoy en día los robots están incorporados a vida diaria de las personas, tanto en el mundo laboral como en el personal, cuando se habla de robótica muchas personas sienten que se trata de un ámbito muy complejo, que requiere unos conocimientos de informática, electrónica, mecánica,… muy avanzados. La experiencia obtenida para la elaboración de este proyecto lleva a rechazar esta idea.

Existen muchos kits de robótica que permiten trabajar de manera rápida y fácil con los robots, con el objetivo de tener un inicio en este mundo fácil y divertido.

El primer contacto que se tuvo con el mundo de la robótica fue como juez de mesa de la competición First LEGO League (FLL), en una fase previa celebrada en Pamplona. Se

trata de una competición orientada a jóvenes de 10 a 16 años. En ella se demuestra que no es necesario ser un experto en temas como la programación, la electrónica o la mecánica para presentar un robot capacitado para desarrollar una serie de pruebas de cierto nivel. En esta prueba participan distintos equipos que han tenido un tiempo limitado para preparar un robot de LEGO capaz de resolver las diferentes pruebas establecidas para ese concurso.

Durante el concurso se pudo ver la implicación de todos los miembros, el trabajo en equipo, el apoyo y la satisfacción al ver su trabajo plasmado. Además durante la preparación del concurso, se tiene un objetivo y es tener el robot listo para la competición, lo que requiere una organización y una implicación, además los jóvenes preparan el robot y lo presentan, lo que alimenta la motivación, ya que todo su trabajo se verá reflejado el día de la competición.

De ahí que la robótica sea un gran método para alimentar unas habilidades que hoy en día son esenciales para el desarrollo de las personas, cómo es el trabajo en equipo, implicación al ser responsables del resultado del robot. Este tipo de competiciones son muy importantes para el desarrollo de los jóvenes, el nivel de implicación y trabajo es muy superior que en otros proyectos o asignaturas, de ello que en muchos centros se utilicen para motivar a los estudiantes.

A nivel universitario, se decidió participar en dos competiciones, DeustoBot y Robolid, de ellas se hablará más adelante. Las competiciones robóticas son muy distintas y más que una competición en sí, se trata de una reunión de aficionados a la robótica donde cada uno aprende de los otros, de cada competición se recogen nuevas ideas, nuevos planteamientos, se descubren nuevos kits de robots. En estas competiciones los participantes suelen llevar sus propios robots, en muy pocos casos se ven kits de robótica, sino que son diseños propios. Así se pueden ver robots realizados con motores de coches del Scalextric o viejos motores de electrodomésticos. Así como diseños con cadenas, con rampas, sensores de todos tipos… Lo que te hace ver que con un poco de imaginación se puede lograr realizar un robot competitivo. De estas competiciones además se logró un how now de diferentes pruebas y de diferentes tipos de robots, así se descubrió el kit de Moway, con el que se ha trabajado en la elaboración del proyecto y la prueba Baile Sincronizado. Además se obtuvieron ideas para realizar robots lo más competitivos posibles teniendo en cuenta las limitaciones de ellos.

Al igual que en la FLL se pudo observar que este tipo de competición ayudaba al desarrollo de los jóvenes, en las competiciones universitarias, se llego a la conclusión de que la robótica servía para poner en práctica los conocimientos aprendidos en algunas asignaturas y que era un gran método para entender y afianzar estos conocimientos. Si se estudian los contenidos de la carrera cursada, Ingeniería Técnica de Informática de Gestión, se pueden ver que con la robótica se desarrollan gran parte de estos contenidos como son la programación, los computadores, la electrónica digital o inteligencia artificial.

Existen publicaciones que reafirman los beneficios de las competiciones robóticas. [5] [6].

3 COMPETICIONES ROBÓTICAS

De los seminarios, cursos o grupos de robótica han surgido las diferentes competiciones robóticas, orientadas a su vez a diferentes públicos. Así tenemos competiciones robóticas orientadas a niños, a estudiantes de secundaria, a estudiantes universitarios o todos los públicos. A continuación se incluyen las competiciones más importantes, a nivel nacional e internacional.

Para desarrollar este proyecto de fin de carrera se han estudiado las diferentes competiciones orientadas a estudiantes universitarios. Aunque a nivel nacional existen muchas pruebas, se decidió participar en dos competiciones, 8º Concurso de Microbótica de Euskadi y 7ª edición de ROBOLID, con el objetivo de captar ideas, aprender de ellas, y a partir de ellas evolucionar los robots.

3.1 8º CONCURSO DE MICROBÓTICA DE EUSKADI

El grupo de robótica de la Universidad de Deusto, apoyado por la Facultad de Ingeniería de la Universidad de Deusto organiza todos los años la competición de robots, conocida como “Concurso de Microbótica de Euskadi” o “DeustoBot”.

Este concurso se disputo durante las primeras etapas del proyecto. Se estaba en una etapa de análisis de los diferentes kits de robots y por lo tanto era muy prematura la participación en él. De todas formas se decidió ir formando equipo, con un compañero, Patxi Oliva, que estaba en una etapa más madura de su proyecto de fin de carrera de características similares a éste y por lo tanto estaba preparado para participar en competiciones robóticas.

Durante el concurso se habló con los organizadores la forma que tienen de trabajar para conseguir el éxito de esta competición, ya que en todas las pruebas el número de participantes es alto. La clave principal es tener un grupo de robótica que anime a los estudiantes a entrar en él, principalmente incentivando con el logro de créditos por la participación. Una vez creado un grupo de robótica estable, todos los años con el inicio del año académico se programan una serie de cursos en los que se estudian las pruebas que se presentarán en el concurso. Así los alumnos tienen la posibilidad de crear robots competitivos para ellas. De ahí que la mayoría de los participantes sean de la propia universidad, además esta competición está abierta a otras personas, como estudiantes de formación profesional de institutos de Vizcaya o como era nuestro caso, a estudiantes de otras universidades.

En la edición en la que se participó se presentaban las pruebas típicas de todos los años, como sumo, mini-sumo y rastreadores. Y se presentaba como novedad la prueba de bailarines.

3.1.1 SUMO

En esta prueba dos robots tendrán que luchar entre ellos, ganará el robot que consiga expulsar del ring al otro robot. El ring estará formado por una superficie negra con forma redonda, de radio entre 1,5m y 1,75m. Además tendrá un borde blanco que ayudará al robot a reconocer que está en una situación de riesgo. La prueba consiste en tres asaltos, ganará el que gane 2. Los robots participantes estarán limitados en su tamaño (20x20cm) y peso (3kg).

En esta prueba habrá que diseñar un robot robusto, que resista a los golpes, pero que además tenga mecanismos de ataque importantes. Además habrá que diseñar un robot con la mayoría de sensores posibles, para tener más controlado el entorno. Además la programación del robot, tendrá que tener en cuenta tanto la defensa como el ataque.

En esta competición, debido al número de participantes, se decidió hacer una criba, para participar en la competición el robot debía ser capaz de expulsar una caja vacía y no salirse del ring.

Durante la competición se pudo observar que lo más importante es el diseño del robot, ya que aunque la programación no sea muy óptima si el diseño permite noquear al rival, las posibilidades de ganar serán mucho mayores. El robot ganador de esta prueba tenía como principal ventaja su diseño, tenía una chapa muy fina que le permitía ir muy pegada a la superficie, impidiendo que otros robots la levanten y a su vez permitía deslizarse por debajo del robot contrario, noqueándolo y sacándolo del ring. Así quedaba demostrando la importancia del diseño por encima de la programación.

3.1.2 MINI-SUMO

Se trata de una competición similar a la anterior, pero las dimensiones son más reducidas, tanto como del robot, como del ring.

Al igual que en la prueba de sumo, los robots tuvieron que pasar una criba para poder participar en la competición.

La gran diferencia entre el mini-sumo y el sumo, es que en el caso del mini-sumo, el diseño del robot no es tan importante, ya que su peso y dimensiones lo limitan. En este caso la programación tiene mayor importancia. En esta prueba se participó formando equipo con Patxi Oliva, que preparó un robot con el Kit de Parallax. El robot preparado por Patxi consiguió un segundo puesto, y la principal característica fue la programación

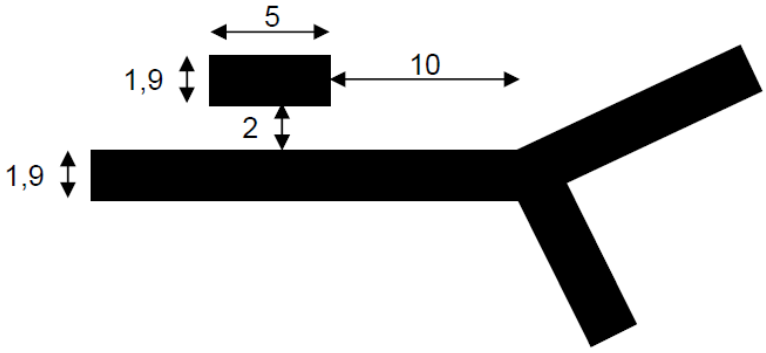
del inicio del combate, teniendo un arranque muy bien preparado y que noqueaba a los rivales.

3.1.3 RASTREADORES

En esta prueba compiten dos robots, se colocan en posiciones opuestas. Ganará el primer robot que consiga alcanzar al otro. Para alcanzar al otro robot, el robot tendrá que ser capaz de seguir la línea negra que marca el camino y a su vez tendrá que elegir bien el camino a tomar en las bifurcaciones para coger el camino más corto.

La pistará estará formado por una superficie blanca, con un circuito cerrado marcado por una línea negra, a los lados de la línea se colocarán unas marcas que indicarán el camino a elegir en la siguiente bifurcación.

Para construir la pista se tendrán en cuenta las siguientes características: Las líneas, tanto las que marquen la bifurcación como la principal tendrán un grosor de 1,9cm. Entre la línea y la marca de bifurcación habrá una distancia de 2cm. La longitud de la marca de bifurcación es de 5 centímetros. Y la distancia desde el fin de la marca de bifurcación hasta la propia bifurcación será de 10cm.



Se trata de una prueba en la que ganará el robot que sea el más rápido y más fiable, es decir, el robot que más rápido recorra el camino más rápido. Por ello para tener un robot competitivo habrá que alcanzar un compromiso entre velocidad y fiabilidad a la hora de tomar el camino correcto.

Una vez visto los resultados de la prueba, se decidió profundizar en esta prueba, estudiar cómo preparar un robot competitivo con los robots de LEGO. Teniendo en cuenta los resultados vistos, se dedicará más esfuerzo en tener un robot fiable que en tener un robot muy rápido.

3.1.4 BAILE SINCRONIZADO

Antes de llegar al concurso, nunca se había oído hablar de esta prueba, las especificaciones son muy mínimas, sólo se dice que la organización será la que ponga en marcha los robots y marcará el fin. Los robots tendrán que sincronizarse y terminar a la vez, aunque se indique el final en uno de ellos.

Se trata una prueba bastante diferente al resto, en este concurso se utilizaron para participar en ello, parejas de robots Moway. Hablando con los participantes se vio que estos robots se adaptan muy bien a este tipo de pruebas por permitir la comunicación entre ellos de una forma bastante intuitiva. Además el diseño no es muy importante, por lo tanto el hecho de ser robots ya montados no sería un gran problema.

Tras ver la prueba, se decidió trabajar con los robots Moway y más específicamente en esta prueba.

3.2 7ª EDICIÓN ROBOLID

En 2002 un grupo de estudiantes de la Universidad de Valladolid creó la asociación AMUVA, Asociación de Microrobótica de la Universidad de Valladolid. Su objetivo principal era divulgar la robótica entre sus compañeros universitarios e incentivar la participación en concursos de robótica, así cómo organizar un concurso a nivel nacional. Así nació la Robolid, que desde 2002 se organiza todos los años.

AMUVA organiza todos los años unos seminarios en los que se trabajan algunas de las pruebas, normalmente, Rastreadores y Velocistas. Los participantes a estos seminarios son capaces de diseñar, montar y programar robots competitivos para estas pruebas. Miembros de la organización aseguran que estos seminarios son fundamentales para que el número de participantes en la competición sea alto.

Esta competición está abierta a todos los públicos, tanto estudiantes universitarios como aficionados a la robótica, por ello que el nivel de la competición sea alto. Las pruebas que se ofrecen en esta competición son Rastreadores, Velocistas, Programación y Sumo. Además un día antes de la competición la asociación organiza una serie de conferencias, se asistió a la conferencia de Julio Pastor Mendoza [4] sobre la Eurobot (competición robótica a nivel europeo) y la conferencia impartida por Luís Ignacio Día del Dedo y Luís Alberto Pérez García sobre su robot cuadrúpedo.

Tras participar en la competición de Deusto se analizaron las ideas recabadas en ella y se decidió preparar un robot con el kit Mindstorms de LEGO para la competición de Valladolid, con el objetivo de competir en dos pruebas, Rastreadores y Velocistas. A continuación se resumen las normativas de las pruebas que se disputan en la Robolid, se entrará en más detalle en las pruebas preparadas.

3.2.1 RASTREADORES

3.2.1.1 Objetivo

Esta prueba valorará la habilidad de un robot para recorrer un camino sinuoso, previamente establecido, en el menor tiempo posible. Se conjugarán, por lo tanto, dos aspectos importantes: capacidad de detección y seguimiento del camino y rapidez con que se ejecutan las maniobras.

Aquí se percibe una diferencia con la prueba de Rastreadores de Deusto, en este caso, el robot compite de forma individual y gana el robot más rápido y con menos errores. En el caso de Deusto competían dos robots, ganando el robot que consiga alcanzar al otro robot, el camino más corto es el que indican las marcas de bifurcación.

En las dos competiciones se valora la velocidad y la capacidad de detectar las marcas de bifurcación, en el caso de la Robolid, se valora más la capacidad de detectar las marcas frente a la velocidad.

3.2.1.2 Características de los robots participantes

Los robots han de poseer unas dimensiones máximas de 20 cm de ancho y 30 cm de largo, siendo libres la altura y peso. En cualquier caso deben ser completamente autónomos, es decir, no podrán disponer de ningún tipo de conexión o comunicación con el exterior, ni se podrá operar directamente sobre ellos una vez comenzada la prueba.

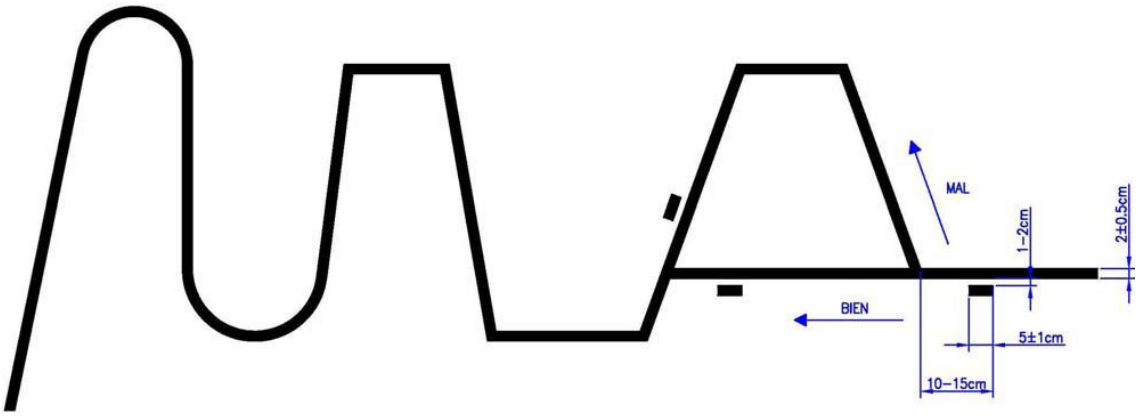
3.2.1.3 Pista de rastreadores

La pista consistirá en una superficie clara con una línea oscura (con diferencia de reflectividad mínima entre ellas de 0,4) de 2±0,5 cm. de grosor, que constituirá el camino a seguir desde la salida hasta la meta. La pista podrá estar confeccionada, en todo o en parte, con material plástico. La superficie de la pista podrá presentar irregularidades sin tener que ser perfectamente plana, y podrá estar iluminada con diferentes niveles de intensidad luminosa, desde muy oscura hasta sobreiluminada. El camino a recorrer puede presentar tantas bifurcaciones y curvas como la organización considere oportuno. Los puntos de salida y meta serán únicos.

Forma de tomar las bifurcaciones: Entre 10 y 15 cm antes de que aparezca una bifurcación, una línea oscura de 2±0,5 centímetros de grosor y 5±1 cm de longitud, separada entre 1 y 2 centímetros de la trayectoria (quedará una zona clara entre marca y trayectoria de entre 1 y 2 cm.) y en el sentido de recorrido de la misma, indicará por cuál de los dos caminos de la bifurcación el Robot deberá seguir obligatoriamente: si está a la izquierda el camino a seguir es el de la izquierda y si está a la derecha el camino se tomará girando a la derecha. El camino a seguir puede presentar curvas circulares de cualquier ángulo de giro, además podrá tener pequeñas subidas y bajadas, giros poligonales (nunca menores de 90º), etc. (Ver figura ejemplo).

El Robot siempre debe seguir el camino a lo largo de la línea que define su trazado sin posibilidad de poder evitar o saltar una parte del mismo. En caso de seguir en una

bifurcación por el camino contrario al indicado, será penalizado. Si un robot permanece parado durante un tiempo superior a 10 segundos quedará eliminado.



3.2.1.4 Sistema de puntuación

La puntuación del robot se hará en base a dos criterios: menor número de errores cometidos y menor tiempo.

La pista se dividirá en cuatro tramos perfectamente identificados. Cada vez que el Robot consiga superar un tramo sin ninguna penalización acumulará 25 puntos, por tanto, la puntuación máxima que puede obtener un Robot sin penalizaciones es de 100 puntos. Además la organización, en función de la pista, pondrá un tiempo máximo para recorrer la pista. Si se supera ese tiempo, se parará el robot y se puntuara en función a donde se encuentre en ese instante.

Penalizaciones:

Cada vez que se tome una bifurcación por el camino no indicado se penalizará con 10 puntos.

Cada vez que el robot se salga de la trayectoria pero sea capaz de encontrarla en menos de 10 segundos será penalizado con 15 puntos.

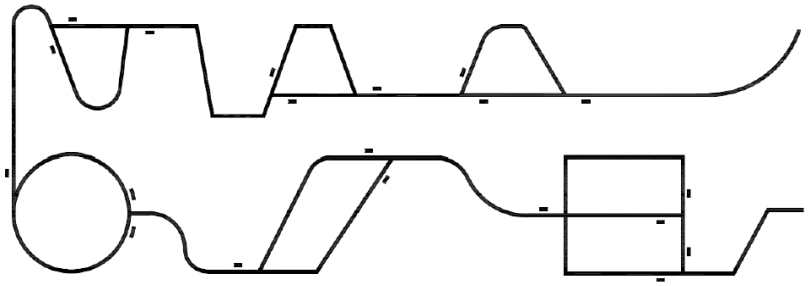
Si se sobrepasa el tiempo máximo estipulado para hacer el recorrido o si el robot se sale del recorrido y no es capaz de retomarlo en 10 segundos se parará el tiempo y se puntuará en función de donde se encuentre.

Ganador:

El ganador será aquel que consiga hacer el recorrido completo con un mayor número de puntos con el menor tiempo posible. La clasificación se hará de acuerdo a la puntuación conseguida, en caso de empate a puntos vencerá aquel que haya tardado menos en hacer el recorrido.

3.2.1.5 Ejemplo de pista

La siguiente imagen es la pista que se utilizó en una edición de la Robolid.



3.2.2 VELOCISTAS

3.2.2.1 Objetivo

El concurso de velocidad consistirá en una carrera de persecución entre dos robots en una pista cerrada, comenzando en puntos opuestos y avanzando en el mismo sentido (la pista será simétrica respecto a dos ejes, garantizando que ambos robots encuentran tramos de pista similares en su recorrido). El objeto, por tanto, será la consecución de altas velocidades de marcha en un recorrido perfectamente preestablecido.

3.2.2.2 Características de los robots participantes

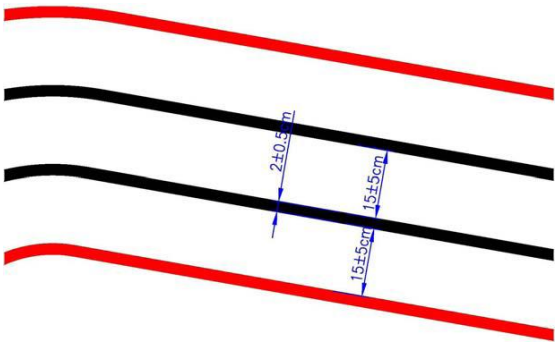
Los robots han de poseer unas dimensiones máximas de 20 cm de ancho y 30 cm de largo, siendo libres la altura y peso. En cualquier caso deben ser completamente autónomos, es decir, no podrán disponer de ningún tipo de conexión o comunicación con el exterior, ni se podrá operar directamente sobre ellos una vez comenzada la prueba.

3.2.2.3 Pista de velocistas

La pista estará formada por una sola calle de 15±5 cm de anchura, delimitada por dos líneas oscuras de 2±0.5 cm de anchura cada una, sobre una superficie clara. La salida se realizará desde el centro de la pista y los robots podrán seguir cualquiera de las dos líneas o navegar entre ambas. Se establecerá unos límites de navegación interior y exterior a la pista, mediante dos líneas rojas, a una distancia mínima de 15 ± 5 cm de la misma, de modo que si alguna parte de robot alcanza estos límites será descalificado de la carrera en la que esté compitiendo.

El radio de curvatura de la pista siempre será superior a 40±5 cm. La pista podrá tener curvas en diferentes sentidos, aunque obviamente será cerrada.

La superficie de la pista podrá presentar pequeñas irregularidades, sin tener que ser perfectamente lisa. Además la pista podrá estar iluminada con diferentes niveles de intensidad luminosa, desde muy oscura hasta sobre iluminada.



3.2.2.4 Sistema de puntuación

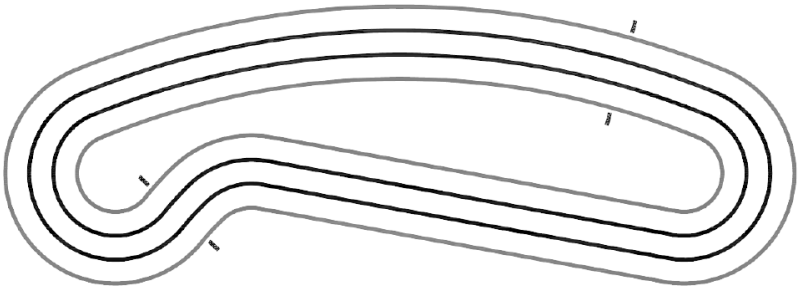
Las pruebas se evaluarán siguiendo el siguiente esquema:

Ronda de clasificación: Participarán todos los robots individualmente computándose el tiempo que tardan en dar un número determinado de vueltas al circuito, cada robot dispondrá de dos intentos. El objeto es seleccionar a los 8 mejores robots que pasarán a la siguiente ronda.

Rondas eliminatorias y finales: En las siguientes rondas las parejas se establecerán según el orden establecido en la ronda de clasificación: el primero con el último, el segundo con el penúltimo, etc. Las rondas eliminatorias se realizarán al mejor de tres mangas, considerándose ganador de una manga a aquel robot que sea capaz de alcanzar a su contrincante. Si transcurrido un tiempo mínimo de 3 minutos y ningún robot ha alcanzado al otro, los jueces podrán detener la carrera, quedando a su decisión el ganador de la prueba, teniendo en cuenta la distancia entre los dos robots en el momento de la finalización de la misma. En cualquier caso, la decisión de los jueces será inapelable.

3.2.2.5 Ejemplo de pista

La siguiente imagen es la pista que se utilizó en una edición de la Robolid.



3.2.3 PROGRAMACIÓN

La prueba valorará la habilidad de los participantes para programar un microcontrolador PIC en lenguaje C ó ensamblador, ante un suceso real. Existen un total de 5 fases que se evaluarán individualmente, en donde no todas son independientes.

La prueba se desarrollará dentro del tiempo estipulado por la organización, siendo la programación del código fuera de este tiempo causa de penalización para el participante. Durante el desarrollo de la prueba los programadores dispondrán de un

intento para simular el resultado de su programa, sobre un ordenador concreto habilitado por la organización. El programa podrá ser modificado hasta la finalización de la prueba.

Prueba de ROBOLID 09

Esta prueba consistirá en desarrollar un programa el cual debe realizar 5 fases. Estas serán supervisadas por un tribunal los cuales serán los encargados de dar la puntuación de 2 puntos, 1 punto o 0 puntos, en relación a los objetivos cumplidos, la mitad de ellos cumplidos o fase no operativa respectivamente. El ganador de esta prueba será la persona que obtenga la máxima puntuación. En caso de empate será el tribunal el que se encargue de evaluar el programa desarrollado de forma que el código más óptimo sea aquel que gane la prueba. La prueba se desarrollará sobre un circuito sobre el cual se desplazará un vehículo de forma que realice varias vueltas sobre la pista, y las fases se realicen un mínimo de dos veces para comprobar el correcto funcionamiento del programa realizado.

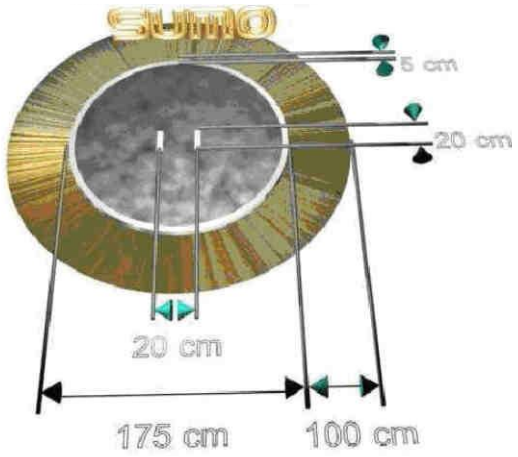
3.2.4 SUMO

En el juego luchan dos Robots de dos equipos diferentes, estando formado cada uno de los equipos por uno o varios jugadores (quienes han hecho el Robot).

Los Robots compiten dentro del Área de Combate según las normas que a continuación se expondrán, para obtener puntos efectivos (llamados puntos Yuhkoh). Un combate lo gana el Robot que consigue más puntos efectivos.

Los Robots deberán tener unas dimensiones tales que quepan dentro de una caja de 20x20 cm. con los sensores de contacto (microinterruptores) plegados en caso que fuera necesario. El peso máximo de los Robots será de 3000 gramos incluyendo todas las partes. La tolerancia en peso será del 1%.

El Ring será circular, de color negro, de 175 cm. de diámetro y situado a una altura de 5 cm. respecto al suelo. Señalando el límite exterior del Ring, habrá una línea blanca circular de 5cm. de ancho. La tolerancia de todas las medidas indicadas anteriormente será del ±5%.



Los enfrentamientos consistirán en 3 asaltos (ó combate) de (como máximo) 3 minutos cada uno. Entre asalto y asalto habrá un tiempo máximo de 1 minuto.

Para el comienzo del enfrentamiento se llamarán a los dos equipos participantes. Se realizarán como máximo tres avisos, y si en el plazo de 1 minuto desde el último aviso uno de los equipos no compareciera se otorgaría directamente la victoria al equipo compareciente.

Ganará el combate el Robot que más asaltos gane.

3.3 OTRAS COMPETICIONES

3.3.1 COMPETICIONES NACIONALES

ALCABOT – HISPABOT

Desde mayo de 2000 el Departamento de Electrónica de la Universidad de Alcalá organiza actividades relacionadas con la Robótica Móvil.

Web de contacto: http://www.alcabot.org/

Mes del concurso: Abril.

Pruebas: Velocistas, Rastreadores, Sumo, Eurobot (eliminatoria nacional para la fase Europea) y Robocup JR (eliminatoria nacional para la fase Europea).

MADRID-BOT

Madrid-Bot es el concurso de micro-robótica organizado por los Centro que imparten las enseñanzas del Ciclo Formativo de Grado Superior de Desarrollo de Productos Electrónicos en la Comunidad Autónoma de Madrid. La primera edición se celebró en 2006.

Web de contacto: http://www.madridbot.org/

Mes del concurso: Marzo.

Pruebas: Velocistas, Rastreadores, Sumo, Prueba libre y Prueba libre para robot LEGO.

COSMOBOT

La competición de robótica CosmoBot, organizada conjuntamente por CosmoCaixa , el Museo de la ciencia de la Obra Social "la Caixa" y RoboticSpot se disputa en Cosmocaixa Alcobendas

Web de contacto: http://www.roboticspot.com/cosmobot/index.php

Mes del concurso: Marzo.

Pruebas: Velocistas y Sumo.

3.3.2 COMPETICIONES INTERNACIONALES

EUROBOT

Se trata de una competición organizada en Europa desde el año 1998, se disputa cada año en un país diferente, está orientada a jóvenes europeos y de otros continentes. En algunos países, como en España, se hacen rondas eliminatorias, para limitar el número de participantes.

Web de contacto: http http://www.eurobot.org/eng/

Mes del concurso: Mayo.

A diferencia de otras pruebas, en esta competición se realiza una única prueba que cambia todos los años, la organización publica con un tiempo determinado las reglas que especifican la prueba y los equipos tienen que ser capaces de preparar el robot

ROBOCUP

Se trata de una competición a nivel mundial. Cada año se realiza en un país diferente, así se ha realizado en Japón, Francia, Alemania, EEUU, entre otros.

Web de contacto: http://www.robocup.org/

Mes del concurso: Abril.

Pruebas: Fútbol, Rescate, @Home y Junior.

4 HERRAMIENTAS Y PRUEBAS ROBÓTICAS

4.1 MOWAY

Moway es una herramienta desarrollada por la empresa Minirobots de Barakaldo. Se trata de un robot cuya vocación es ser útil en la enseñanza. Permite a los estudiantes iniciarse en el mundo de la programación de robots, para ello cuenta con un software fácil y eficaz de comunicación con los robots y sus dispositivos de entrada y salida.

Esta herramienta es muy útil para otros ámbitos, como en la investigación y en la industria. Hoy en día se está trabajando en la investigación de los sistemas embebidos, el robot Moway se caracteriza por ser un dispositivo móvil capaz de procesar pequeños programas en su interior o de manera remota siendo además un dispositivo móvil. Moway permite realizar aplicaciones colaborativas sencillas gracias a su comunicación por radio. Es una herramienta a tener en cuenta en la rama de la robótica móvil aplicada. Moway es un elemento muy utilizado en aplicaciones industriales para el desarrollo de pruebas. Así se puede utilizar para el desarrollo de prototipos de bajo costo de plataformas móviles, maquetas con vehículos o sistemas embebidos.

4.1.1 MINIROBOTS

Minirobots es una empresa joven, fue creada en 2008. Su sede se encuentra en Barakaldo. Se trata de una empresa de investigación que centra su actividad en la investigación de la robótica educacional. Su éxito se debe a la búsqueda de soluciones innovadoras y de valor para los centros educativos.

Minirobots es una empresa innovadora dividida en tres vías de desarrollo, como son proyectos de ingeniería a medida en el diseño y puesta en marcha de productos, soporte preventa y postventa tanto a usuarios finales como a distribuidores o educadores y comercialización de productos.

Su producto principal es el kit Moway, éste fue creado por su grupo de ingenieros que trabajan en analizar las necesidades de estudiantes de universidades, formación

profesional y incluso institutos de enseñanza secundaria y diseñar soluciones que satisfagan dichas necesidades.

En 2008 recibió el premio “Premio Barakaldo Empresa Innovadora” por su robot educacional en la modalidad “mejor iniciativa dirigida a empresas con una idea innovadora”. Este premio es otorgado por Inguralde, Organismo Autónomo para el Desarrollo Integral de Barakaldo.

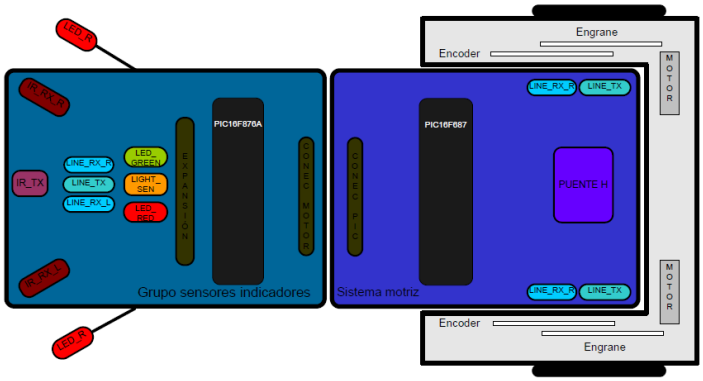
4.1.2 KIT MOWAY

Una de las herramientas de estudio ha sido el kit de Moway. Se trata de una herramienta muy completa en cuanto a diseño y sensores, que se detallan a continuación.

El kit Moway con el que se va trabajar en este proyecto viene con dos robots, tres módulos de Radio-Frecuencia y una llave RF-USB.

4.1.2.1 Robot Moway

Es autónomo y programable, cuenta con gran cantidad de sensores integrados. Pesa tan sólo 100 gramos. Su fácil manejo y programación le hacen ser un producto muy valorado por todos los interesados en la robótica, especialmente a los principiantes y sorprendiendo a los expertos.



4.1.2.1.1 Microcontrolador.

Utiliza como microcontrolador principal el PIC16F876. Ha sido fabricado por la empresa Microchip Technology Inc. Las características del microcontrolador son las siguientes:

Microprocesador Risc de 8 bits
Frecuencia de Reloj: 4Mhz
Temporizador de 8 bits y uno de 16 bits
Dos unidades de captura, comparación y PWM
Buses síncronos I2C y SSP
Unidad de comunicaciones serie asíncrona
8 canales A/D de 10 bits
Programación ``in circuit" (ICSP)
Memoria Flash de 8Kb y SRAM de 368 bytes
Memoria EEPROM de 256 bytes

4.1.2.1.2 Grupo motor con control de trayectoria comandado por I2C.

El robot Moway cuenta con un grupo servo-motor. Tiene una parte electrónica encargada de controlar la velocidad de los motores y otra mecánica que da la potencia al Moway para que se pueda desplazar por diferentes entornos.

El grupo servo-motor tiene cuatro funcionalidades, control de velocidad, control de tiempo, control de distancia recorrida, cuentakilómetros general y control de ángulo.

El robot Moway se desplaza cuando el microcontrolador le pasa la señal al grupo servo-motor. Para esta comunicación se utiliza el bus I2C.

4.1.2.1.3 Sensores infrarrojos anticolisión.

Los sensores de infrarrojos anticolisión o sensores detectores de obstáculos se encuentran en la parte delantera de Moway. Los sensores están conectados directamente al microcontrolador. El sensor está formado por una fuente de luz infrarroja y dos receptores colocados en ambos extremos de Moway.

Su funcionamiento consiste en la emisión de un pulso por parte del emisor de luz, si hay un obstáculo el receptor lo capta utilizando una etapa de filtrado y amplificación. Una vez procesada la señal electrónicamente, el PIC puede medirla.

La distancia de alcance en digital es de unos 3 cm y se recomienda un entorno calor para aumentar la posibilidad de reflexión de la luz infrarroja.

4.1.2.1.4 Sensor de intensidad de luz direccional.

Se trata de un sensor de luz colocado en la parte delantera superior del robot. Tiene forma de media luna y su posición le permite localizar la situación de la fuente de luz y actuar en consecuencia.

Este sensor está conectado a un puerto analógico del microcontrolador lo que nos permite conocer con una simple lectura la intensidad de la luz y compararlo con la última lectura.

4.1.2.1.5 Sensores optorreflectivos infrarrojos para el suelo.

Los sensores optorreflectivos infrarrojos también conocidos como sensores de línea se sitúan en la parte inferior delantera del robot. Utilizan la reflexión de luz infrarroja para detectar el tono de la superficie donde se encuentra el robot.

Estos sensores están conectados a dos puertos analógicos del microcontrolador, lo que permite discernir entre diferentes tonos, no sólo líneas blancas sobre fondo negro o al revés.

Su funcionamiento es similar al sensor de obstáculos. Un diodo LED emite luz en el espectro infrarrojo y un receptor de alta sensibilidad detecta la luz reflejada en el suelo. Las superficies claras hacen que la luz se refleje más que en la luz oscura. El receptor pasa la información sobre el reflejo al microcontrolador que a través de su puerto analógico es capaz de saber el color aproximado de la superficie.

4.1.2.1.6 Indicador luminoso superior bicolor.

El indicador luminoso superior bicolor comparte apertura con el sensor de luz. Por lo que se recomienda apagarlo cuando se vaya a capturar el sensor de luz direccional.

Está conectado a dos salidas digitales del microcontrolador lo que permite que este indicador sea bicolor. El color que muestra es verde o rojo, dependiendo la información recibida desde el microcontrolador.

4.1.2.1.7 Leds rojos frontales.

Se encuentran uno en cada lateral de la parte frontal. Sólo se ven cuando se activan ya que se sitúan detrás del filtro infrarrojo delantero (necesario para el sensor de obstáculos). Al estar detrás de ese filtro deben estar apagados cuando se vaya hacer uso del sensor de obstáculos.

Cada sensor está conectado al microcontrolador y según el dato transmitido por el microcontrolador se encienden o apagan.

4.1.2.1.8 Bus de expansión SPI/I2C para tarjetas electrónicas.

El bus de expansión se encuentra en la parte superior del robot, a través de él se conecta el robot Moway con módulos comerciales o con circuitos electrónicos. Permite la conexión de dispositivos I2C y SPI comerciales.

Por otro lado es el bus con el que se conecta el módulo de radio-frecuencias que permite la comunicación entre varios Robots Moway con el ordenador.

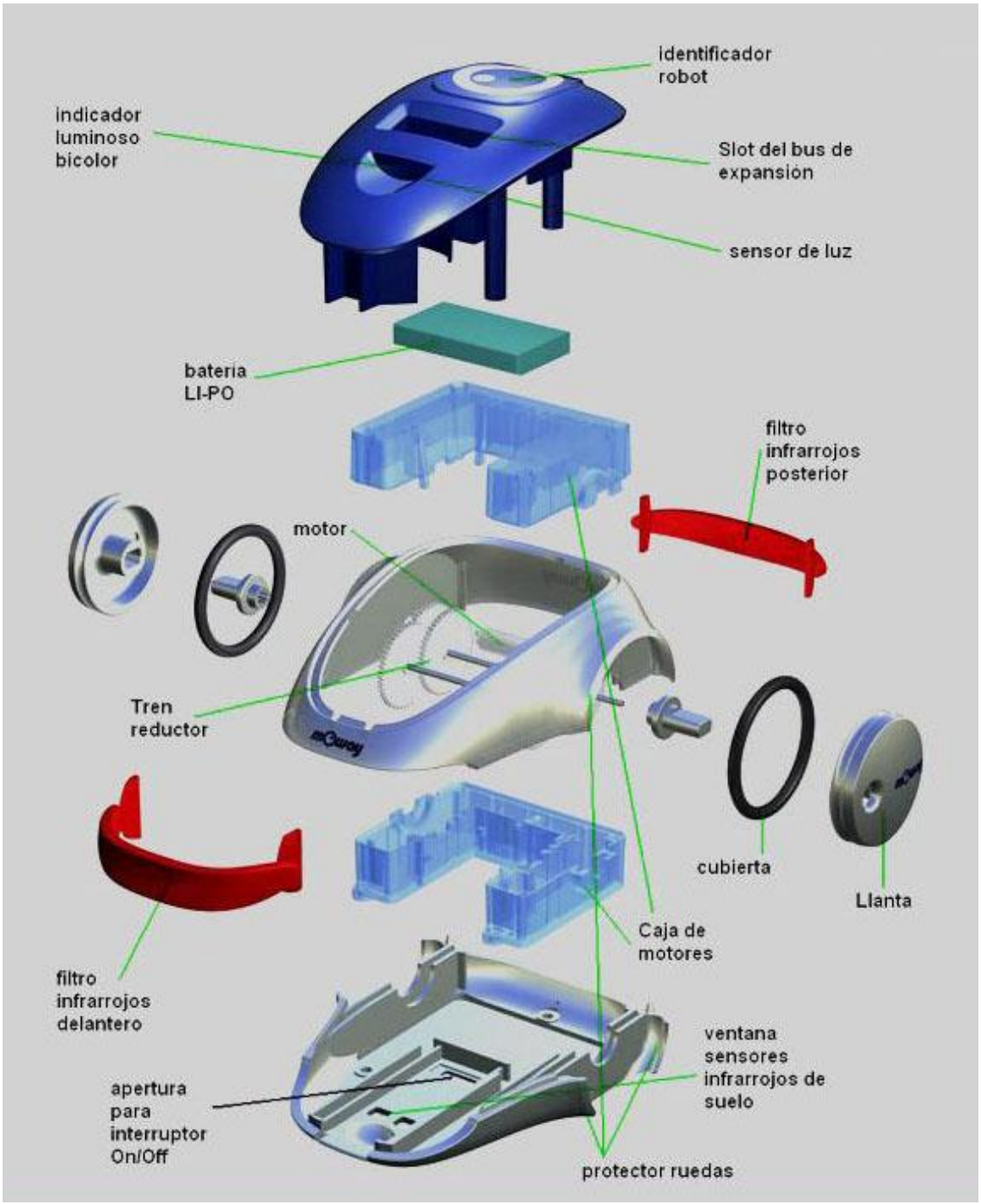
4.1.2.1.9 Batería LI-PO recargable por USB

La batería de Moway se encuentra en el interior, por lo que es necesario desmontar el robot para acceder a ella. Se trata de una pequeña célula LiPo recargable. Su duración depende de la cantidad de sensores activos y de la utilización de sus motores. El tiempo de carga es aproximadamente 2 horas.

La recarga se realiza a través del puerto usb de cualquier ordenador mediante la Base Moway (descrita más adelante). Este tipo de memorias no tiene efecto memoria por lo que no es necesario esperar a que esté totalmente descargada para volver a cargarla.

Es la batería perfecta para este tipo de robots debido a su tamaño, ligereza y flexibilidad.

A continuación se incluye un esquema del robot Moway.



4.1.2.2 Módulos de comunicación RF

El kit de Moway contiene tres módulos de comunicación por radio frecuencia BZI-RF2GH4. El módulo de radio frecuencia está basado en el transceptor nRF24L01 fabricado por "Nordic Semiconductores". La comunicación se realiza a través de un bus SPI.

Las características principales del módulo de radio frecuencia son las siguientes.

Bajo consumo.
Frecuencia de trabajo de 2.4GHz.
Potencia de emisión de 18 a 0dBM (típica de las redes Wifi)
Velocidad de transmisión entre 1 y 2 Mbps.
128 canales de transmisión seleccionables por el bus SPI.

Los módulos de radio frecuencia incorporan además toda la electrónica anexa y una antena. Esto permite que el hardware sea transparente para el usuario final, haciendo su uso mucho más fácil y eficiente.

4.1.2.3 Llave RF-USB

La llave RF-USB o también conocida como Base Moway permite la comunicación del robot a con el ordenador a través de radio frecuencia o conectando la base Moway directamente al robot.

4.1.3 PROGRAMACIÓN

Uno de los objetivos de este proyecto es abarcar los diferentes entornos de desarrollo y programación existentes para la programación de robots. Una de las principales características del robot Moway es la posibilidad de programar los robots en tres entornos distintos.

4.1.3.1 LENGUAJE ENSAMBLADOR

El lenguaje ensamblador es un lenguaje de bajo nivel utilizado para programar computadores, microprocesadores o microcontroladores. Se caracteriza por ser específico para cada hardware, a diferencia de los lenguajes de alto nivel que idealmente son portables.

En los inicios del desarrollo de software se usaba el lenguaje ensamblador, ya que no existían los potentes lenguajes de alto nivel que hoy en día existen. Por ello actualmente el uso de los llamados lenguajes de bajo nivel es casi nulo. Se sigue usando en ciertos ámbitos como la investigación o académicos.

Hoy en día existen potentes compiladores que permiten programar la mayoría de los hardware con lenguajes de alto nivel, como es el caso de los robot Moway, sin embargo el lenguaje ensamblador se sigue utilizando cuando se quiere manipular el hardware directamente, alto rendimiento o los recursos son reducidos y controlados.

Microchip, fabricante de los microcontroladores PIC, desarrolló un software para programar sus microcontroladores, MPLAB IDE. Se trata de un software gratuito para Windows, posee un editor de texto, un compilador y un simulador. MPLAB permite trabajar con diferentes microcontroladores, a la hora de crear un nuevo proyecto, hay que elegir el microcontrolador que se va programar, en el caso de programar los robots Moway, será el microcontrolador PIC16F876. También hay que indicar las librerías que se van a utilizar. Minirobots, empresa fabricante de los robots Moway, ofrece de manera gratuita las librerías necesarias para poder programar sus robots con MPLAB.

4.1.3.1.1 Librería para los sensores Moway para ensamblador

Esta librería recoge todas las funciones encargadas leer la información enviada por los sensores, así como de configurar los puertos de entrada y salida y los indicadores luminosos.

Es de destacar que cada llamada a una función de esta librería ocupa un nivel adicional de la pila, por lo tanto, será necesario que haya al menos dos libres para que no haya errores.

Funciones y variables

SEN_CONFIGURAR

Encargada de configurar los sensores del robot. Configura las entradas y las salidas para manejar los sensores e inicializa las variables. Por lo tanto antes de utilizar los sensores habrá que llamar a esta función para que se configuren.

Pin PIC	I/O	Sensor
PORTA		
RA0	I	Luz
RA1	I	Receptor infrarrojo derecho
RA2	I	Receptor sensor línea derecho
RA3	I	Receptor infrarrojo izquierdo
RA4	O	LED superior rojo
RA5	I	Receptor sensor línea izquierdo
PORTAB		
RB1	O	Transmisor sensor línea
RB2	O	Transmisor infrarrojo
RB4	O	LED inferior derecho
RB6	O	LED superior verde
PORTAC		
RC6	0	LED inferior izquierdo

SEN_LIGHT

Esta función es la encargada de capturar la luz incidente en el sensor de intensidad de luz direccional, el voltaje analógico medido se utilizará para calcular el porcentaje de luz y el dato se copiará en la variable SEN_LIGHT_P.

SEN_OBS_DIG y SEN_OBS_ANALOG

Estas funciones detectan si hay un objeto en la parte delantera derecha o en la parte delantera izquierda. Éstas capturan la información recibida de los sensores de infrarrojos anticolisión. En el caso de la función SEN_OBS_DIG almacenan el valor en el receptor digital y en el caso de la función SEN_OBS_ANALOG en el receptor analógico. Y se copiarán los datos analógicos o digitales a la variables **SEN_OBS_R** (dato captado del sensor de la derecha) y **SEN_OBS_L** (dato captado del sensor de la izquierda).

Estas funciones además devuelven en la variable **SEN_STATUS** si los datos son válidos o no, devolverá un 1 si son válidos y un 0 si no.

SEN_LINE_DIG y SEN_LINE_ANALOG.

Estas funciones informan si el robot está sobre una superficie oscura o no. Para ello captan los datos que transmiten los sensores optorreflectivos infrarrojos para el suelo. Primero se lee el sensor de la derecha, y se almacena el dato, digital o analógico dependiendo de la función, en la variable **SEN_LINE_R**. Posteriormente se lee el sensor de la izquierda, y se almacena el dato en la variable **SEN_LINE_L**.

LED_R_ON

Enciende el diodo LED inferior derecho.

LED_L_ON

Enciende el diodo LED inferior izquierdo.

LED_TOP_RED_ON

Enciende el diodo LED superior rojo.

LED_TOP_GREEN_ON

Enciende el diodo LED superior verde.

LED_R_OFF

Apaga el diodo LED inferior derecho.

LED_L_OFF

Apaga el diodo LED inferior izquierdo.

LED_TOP_RED_OFF

Apaga el diodo LED superior rojo.

LED_TOP_GREEN_OFF

Apaga el diodo LED superior verde.

LED_R_ON_OFF

Parpadeo del diodo LED inferior derecho.

LED_L_ON_OFF

Parpadeo del diodo LED inferior izquierdo.

LED_TOP_RED_ON_OFF

Parpadeo del diodo LED superior rojo.

LED_TOP_GREEN_ON_OFF

Parpadeo del diodo LED superior verde.

4.1.3.1.2 Librería para el sistema motriz de Moway para ensamblador

La librería para el sistema motriz aglutina todas las funciones y variables que permiten controlar el sistema motriz con gran facilidad a la hora de programar los robots.

Si las funciones de la librería para los sensores utilizaban un nivel adicional de la pila, las funciones de esta librería utilizan tres niveles, por lo tanto es necesario que haya como mínimo cuatro niveles libres de la pila de llamadas para que no haya errores.

Estas funciones son las encargadas de enviar los comandos por el I2C al Sistema Motriz. Así el Sistema Motriz se encargará de controlar los motores y el microcontrolador principal se podrá encargar de otras tareas.

Variables

MOT_STATUS

Se trata de un registro que indica el estado del comando. Devuelve información en los dos primeros Bit. En el Bit0 devuelve un 1 si el envío ha sido correcto y un 0 si no ha sido correcto y el Bit1 devuelve un 1 si los datos son correctos y un 0 en caso contrario.

MOT_CON

Es un registro en el que se definen parámetros de los comandos. En el Bit0 se define la dirección, 1 adelante, 0 atrás. En el Bit 1 se define el lado, 1 derecha, 0 izquierda. Y en el Bit2 se define el tipo de comando, 1 tiempo, 0 distancia o ángulo (en MOT_ROT).

MOT_VEL

Variable que almacena la velocidad deseada.

MOT_T_DIST_ANG

Según si el Bit2 del registro **MOT_CON**, el valor de esta variable indicará el tiempo, la distancia o el ángulo.

MOT_CENWHEEL

Esta función almacenará el valor que indique el tipo de rotación sobre el centro o sobre una de las ruedas del robot.

MOT_RAD

Esta función informará a la función **MOT_CUR** del radio.

MOT_RST_COM

Indica el tipo de reset que se desea.

MOT_STATUS_COM

Almacena el tipo de dato que se quiere leer del motor.

MOST_STATUS_DATA_0-1

En estas dos variables se almacena el valor del dato requerido por la función **MOT_FDBCK**.

Funciones

MOT_CONFIG

Al igual que en la liberaría de sensores teníamos una función encargada de configurar los sensores para poder utilizarlos, en la librería para el control de los motores tenemos la función **MOT_CONFIG**, encargada de configurar las entradas y salidas para que el microcontrolador pueda comunicarse con el Sistema Motriz.

Pin PIC	I/O	Sensor
PORTAB		
RB5	I	Indica cuando el motor termina el comando
PORTAC		

RC0	O	SCL del protocolo I2C
RC1	O	SDA del protocolo I2C

MOT_STR

Esta función se utiliza para los desplazamientos en línea recta.

Toma como parámetros de entrada las variables: **MOT_VEL** (Velocidad deseada), **MOT_CON** (Sentido y tipo de comando) y **MOT_T_DIST** (Tiempo, distancia). Si **MOT_T_DIST** es 0, se mantendrá hasta que no se especifique otra orden.

Devuelve el registro **MOT_STATUS**.

MOT_CHA_VEL

Se utiliza para cambiar la velocidad a uno de los motores.

Hay que especificar la velocidad (**MOT_VEL**), sentido y tipo de comando (**MOT_CON**) y el tiempo o distancia a recorrer (**MOT_T_DIST**). Al igual que la función anterior, si **MOT_T_DIST** es igual a 0, el comando se mantendrá hasta que no se especifique otra orden.

Devuelve el registro **MOT_STATUS**.

MOT_ROT

Función para hacer rotar al robot. Es necesario especificar la velocidad (**MOT_VEL**), sentido (**MOT_FWDBACK**), tipo de rotación (**MOT_CENWHEEL**), motor y tipo de comando (**MOT_CON**) y el tiempo o el ángulo a rotar (**MOT_T_ANG**). c

MOT_CUR

La función anterior se utilizaba para rotar al robot, ésta se utilizará para realizar una trayectoria curva. Será necesario especificar la velocidad, sentido, radio (**MOT_RAD**), dirección, tipo de comando y el tiempo o la distancia a recorrer.

El **MOT_RAD** será la velocidad que se le sumará a un motor y se le restará al otro, para que se realice la curva, por lo tanto, la resta no deberá ser menor que 0 y la suma no podrá ser mayor que 100.

Devuelve el registro **MOT_STATUS**.

MOT_STOP

Esta función parará al robot. No tiene parámetros de entrada y devuelve el registro **MOT_STATUS.**

MOT_RST

Esta función se encarga de resetear las variables temporales internas de tiempo, distancia y cuentakilómetros, a través de la variable **MOT_RST_COM**, se pasará que variable queremos resetear y como las funciones anteriores, devolverá el registro **MOT_STATUS**, que nos indicará si la función se ha ejecutado correctamente.

MOT_FDBCK

Esta función sirve para consultar ciertos parámetros del sistema motriz. A través de la variable **STATUS_COM** se indicará que parámetros se quiere consultar. Todas las variables consultadas excepto **STATUS_KM**, devuelven un byte, que se devolverá en **MOT_STATUS_DATA_0**, la variable cuentakilómetros devolverá un segundo byte en **MOT_STATUS_DATA_1**.

4.1.3.1.3 Librería para el módulo de radio frecuencia.

Con esta librería se podrá controlar la comunicación entre el robot y el módulo de radio frecuencia BZI-RF2GH4.

Hay que indicar que para el uso de esta librería será necesario disponer de al menos tres niveles libres de la pila y el "watchlog" desactivado. Además será necesario habilitar el hardware del microcontrolador para utilizar el protocolo SPL. Esto últimos se consigue añadiendo unas líneas de código en la configuración principal del programa

call RF_CONFIG_SPI

Variables

RF_STATUS

Esta variable informa de la situación de la comunicación, es sólo de lectura. Es un registro que nos devuelve la siguiente información en cada uno de sus bits.

Bit7: Sin uso.

Bit6: Muestra si el módulo se ha configurado correctamente.

Bit5: Muestra si el módulo se ha apagado correctamente.

Bit4: Muestra si el módulo se ha encendido correctamente.

Bit3: Muestra si todavía quedan datos por leer.

Bit2: Informa que se han recibido datos correctamente y están accesibles.

Bit1: Muestra si se ha recibido el ACK (confirmación) del receptor tras una transmisión.

Bit0: Muestra si el último envío de datos se ha realizado.

RF_DATA_OUT_0, RF_DATA_OUT_1,…RF_DATA_OUT_7

Son las variables que se envían en cada envío, cada una consta de un byte.

RF_DATA_IN_0, RF_DATA_IN_1,… RF_DATA_IN_7

Son las variables que se reciben en cada envío, en cada recepción se actualizan los 8 bytes.

RF_DIR_OUT

Esta variable almacena la dirección del emisor, consta de un byte.

RF_DIR_IN

Esta variable almacena la dirección del receptor, consta de un byte.

RF_DIR

Esta variable almacena la dirección propia, con la cual se configura el módulo.

RF_CHN

Esta variable almacena el canal, con el cual se configura el módulo.

Funciones

RF_CONFIG

A esta función se le pasa como parámetros de entrada la dirección del dispositivo con que se quiere configurar el módulo y el canal a utilizar para configurar el módulo y devuelve en el Bit 6 de la variable **RF_STATUS**, si la configuración ha sido satisfactoria.

RF_ON

Esta función será la encargada de activar el módulo en modo de escucha.

RF_OFF

Esta función será la encargada de desactivar el módulo y dejarlo en modo de bajo consumo.

RF_SEND

Será la encargada de enviar los 8 bytes de datos a la dirección indicada.

RF_RECEIVE

Será la encargada de recibir los 8 bytes de datos de la dirección indicada.

RF_RECEIVE_INT

Esta función es idéntica a la función **RF_RECEIVE**, pero ésta funciona por interrupción.

RF_INT_EN

Esta función se encarga de habilitar la interrupción externa del microcontrolador.

4.1.3.2 LENGUAJE C

El lenguaje C fue creado en 1971 por *Dennis M. Ritchie*. Fue pensado para la programación de sistemas operativos, especialmente UNIX, pero ha tenido tanto éxito que hoy en día se utiliza además para la programación de aplicaciones.

Se ha tipificado con un lenguaje de medio nivel, incorpora estructuras de los lenguajes de alto nivel pero a su vez permite un control a muy bajo nivel. Los compiladores suelen ofrecer extensiones que permiten la mezcla de lenguaje ensamblador y C. Todo ello hacen que sea un lenguaje muy potente y eficiente.

El lenguaje C contaba con la problemática de la optimización y necesidad de memoria, pero hoy en día los compiladores han mejorado y los microcontroladores han aumentado la capacidad. Así el lenguaje C es uno de los más utilizados para programar los microcontroladores, que además se diseñan optimizados para este lenguaje.

Los programas desarrollados con el lenguaje C tendrán que ser compilados con el compilador CCS. Es un compilador de pago, pero que tiene una versión de evaluación, que permite trabajar con el compilador durante 30 días.

4.1.3.2.1 Librerías

Al igual que para trabajar con lenguaje ensamblador, a la hora de programar los robots Moway , contamos tres librerías, una para los sensores, otra para el sistema motriz y una tercera para el módulo de radio frecuencia. Las variables y funciones especificadas en las librerías para ambos lenguajes son las mismas. A continuación se incluyen unas tablas con el nombre de las variables y funciones de las librerías para C.

Librería para los sensores Moway en C para CSS	
Variables	**Funciones**
SEN_LIGHT_P	void SEN_COFIGURAR()
SEN_LINE_L	void SEN_LIGHT()
SEN_LINE_R	int SEN_OBS_DIG()
SEN_OBS_L	int SEN_OBS_ANALOG()
SEN_OBS_R	void SEN_LINE_DIG()
	void SEN_LINE_ANALOG()
	void LED_R_ON()
	void LED_L_ON()
	void LED_TOP_RED_ON()
	void LED_TOP_GREEN_ON()
	void LED_R_OFF()
	void LED_L_OFF()
	void LED_TOP_RED_OFF()
	void LED_TOP_GREEN_OFF()
	void LED_R_ON_OFF()
	void LED_L_ON_OFF()
	void LED_TOP_RED_ON_OFF()
	void LED_TOP_GREEN_ON_OFF()

Librería para el sistema motriz Moway en C para CSS	
Variables	**Funciones**
MOT_STATUS_DATA_0-1	void MOT_CONFIG()
MOT_FDBCK	int8 MOT_STR (int MOT_VEL,int1 FWDBACK,int1 COMTYPE, int MOT_T_DIST)
	int8 MOT_CHA_VEL(int MOT_VEL, int1 FWDBACK, int1 RL, int1 COMTYPE, int MOT_T_DIST)
	int8 MOT_ROT(int MOT_VEL, int1 FWDBACK, int MOT_CENWHEEL, int1 RL, int1 COMTYPE, int MOT_T_ANG)
	int8 MOT_CUR(int MOT_VEL, int1 FWDBACK, int MOT_RAD, int1 RL, int1 COMTYPE, int MOT_T_DIST)
	int1 MOT_STOP()
	int1 MOT_RST(int8 RST_COM)
	int1 MOT_FDBCK(int8 STATUS_COM)
Librería para el módulo de radio frecuencia Moway en C para CSS	
Variables	**Funciones**
RF_DATA_OUT	int RF_CONFIG(int canal, int dir)
RF_DATA_IN	void RF_CONFIG_SPI()
RF_DIR_OUT	int RF_ON()
RF_DIR_IN	int RF_OFF()
	int RF_SEND()
	int RF_RECEIVE()
	void RF_INT_EN()
	int1 MOT_FDBCK(int8 STATUS_COM)

4.1.3.3 DIAGRAMAS DE FLUJO

Un diagrama de flujo es una representación gráfica de un algoritmo o proceso. Se utiliza en el ámbito de la programación como en otros ámbitos dispares, por ejemplo, en procesos industriales, en economía o en psicología. Los diagramas de flujo hacen que los algoritmos sean más comprensivos y pueden sustituir varias páginas de código.

Minirobots ha diseñado un software, MowayGUI (Moway Graphic User Interfacce), basado en diagramas de flujo para programar los robots Moway. Este software permite que el primer contacto con la robótica sea muy agradable para el usuario, ya que no es necesario tener un avanzado de programación para poder programar al robot.

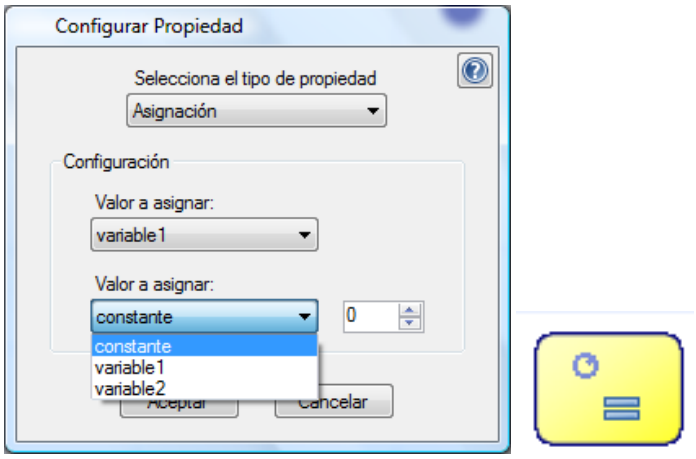
Los elementos que forman los diagramas de flujo son: módulos, condiciones, flechas e inicio y final. A través del MowayGUI se puede crear estos elementos, a continuación se detallan los diferentes tipos de elementos.

4.1.3.3.1 Módulos

Con los módulos se definen las diferentes acciones que realizará el robot cuyas salidas serán incondicionales, como por ejemplo, encender un LED, rotar, realizar un desplazamiento curvo… Los diferentes tipos de módulos son los siguientes.

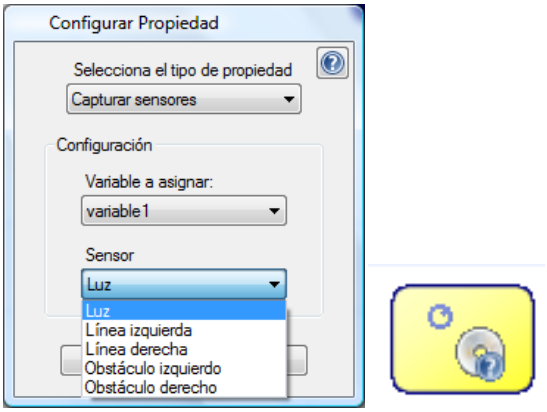
Asignación

El software MowayGUI te permite crear variables, una vez definida una variable, a través del módulo de Asignación, se podrá establecer un valor a una determinada variable, se le podrá asignar un valor constante o el valor de otra variable.



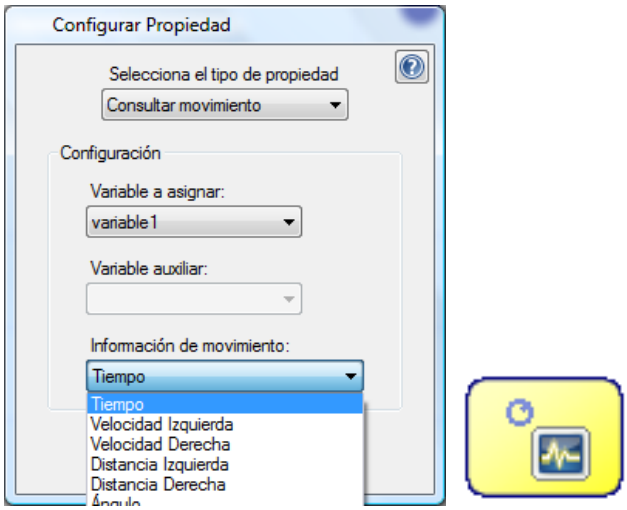
Captura sensores

Este módulo permite asignar a una variable creada, el dato que devuelva un sensor determinado. Muestra una lista con todas las variables creadas y una lista con todos los sensores, hay que elegir una variable y un sensor.



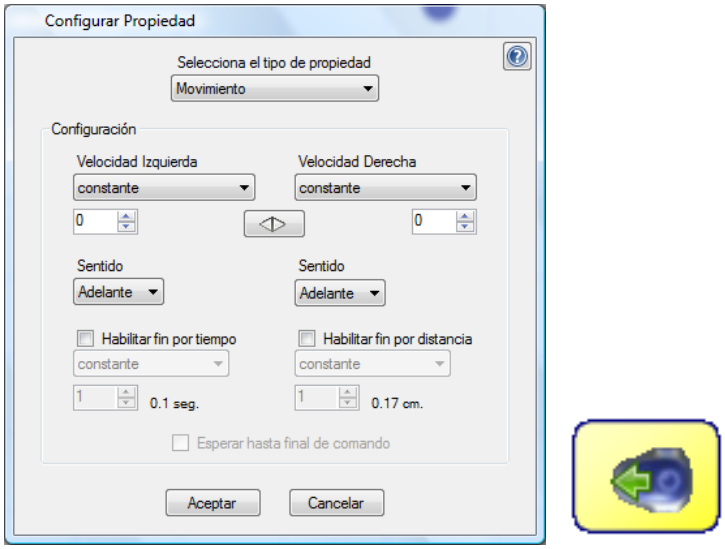
Consultar movimiento

Los robots Moway guardan un histórico de los movimientos, a través de este módulo podremos recuperar alguno de los siguientes datos: tiempo, velocidad izquierda, velocidad derecha, distancia izquierda, distancia derecha, ángulo o cuentakilómetros y asignarlo alguna de las variables creadas.



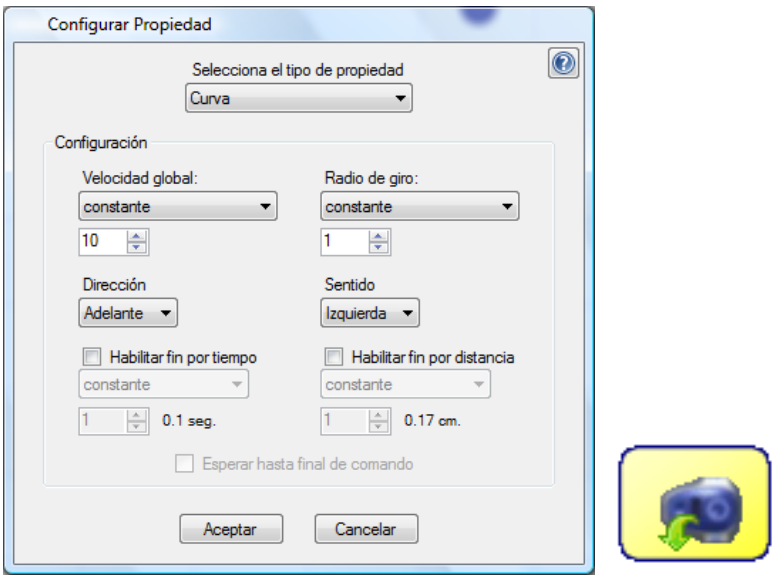
Movimiento

A través del módulo movimiento se puede definir un desplazamiento. En él se podrá elegir la velocidad de cada motor, que podrá ser definido como un valor constante o como el valor de una variable. Se puede definir el sentido, adelante o atrás. Podemos habilitar fin por tiempo o por distancia o no habilitar fin. Tanto la velocidad como el sentido podrán ser iguales o diferentes para cada motor.



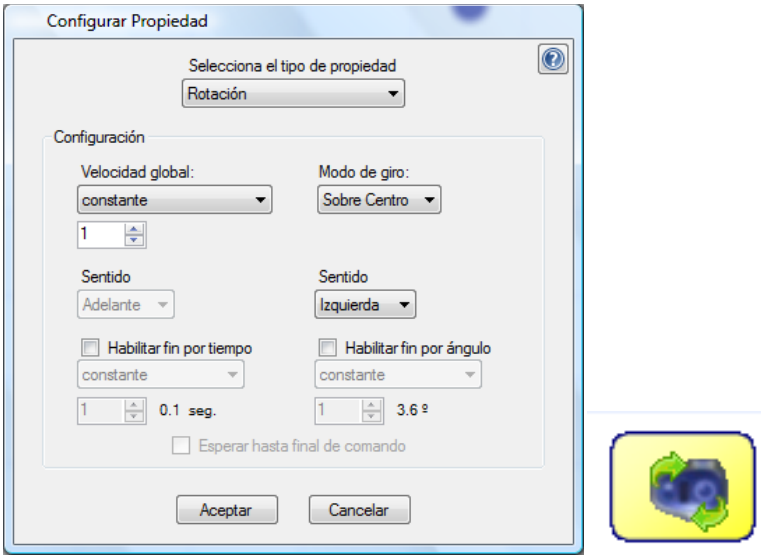
Curva

El módulo Curva permite definir un trayecto curvilíneo. En este módulo se definirá la velocidad con la que se realizará el trayecto, el radio de giro, la dirección y el sentido. La velocidad y el radio de giro podrán ser una constante o el valor asignado a una variable. Además se podrá habilitar un fin por tiempo o por distancia o no habilitar un fin.



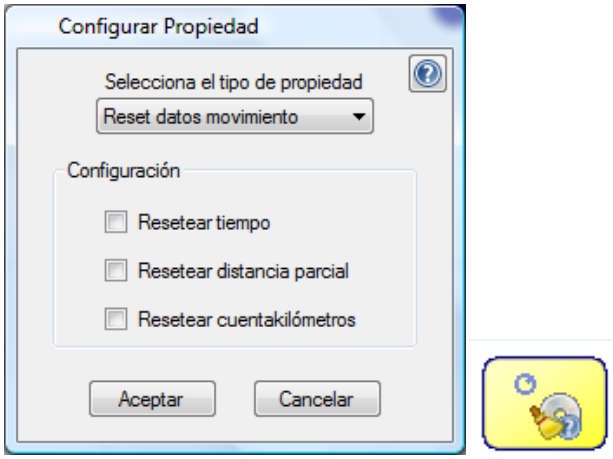
Rotación

A través del módulo rotación se podrá rotar el robot. Para ello se tendrá que definir en el módulo una velocidad, un tipo de giro, un sentido y un fin, por tiempo o por ángulo.



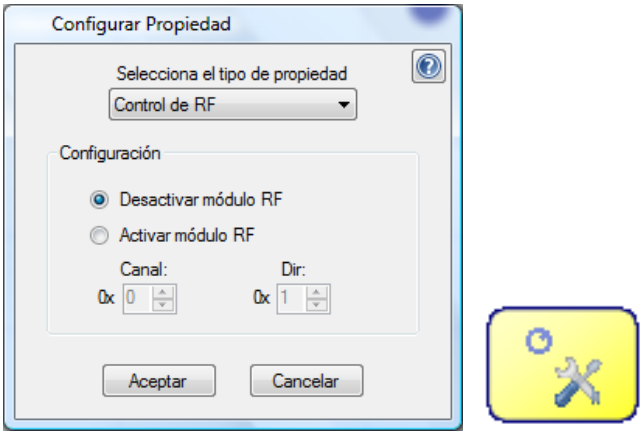
Reset datos movimiento

Como se ha explicado en el módulo de consulta de datos de movimientos, el robot guarda un histórico de los movimientos, a través de este módulo podremos resetear ese histórico. Para ello tendremos que seleccionar que variables queremos resetear.



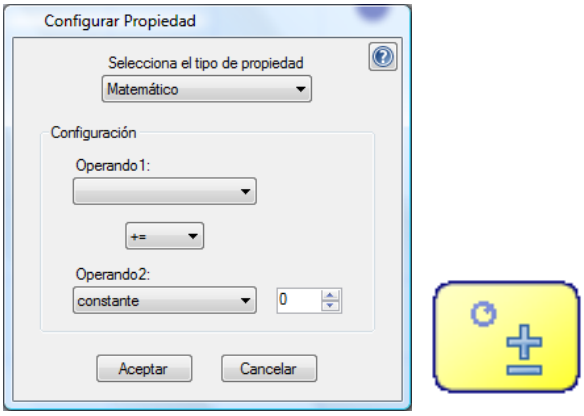
Control de RF

Este módulo permite desactivar, activar y configurar el módulo de radio frecuencia. Para configurar el módulo tendremos que indicarle el canal y la dirección. Habrá que tener en cuenta que para poder comunicarse con otro robot, todos los robots tendrán que estar en un mismo canal.



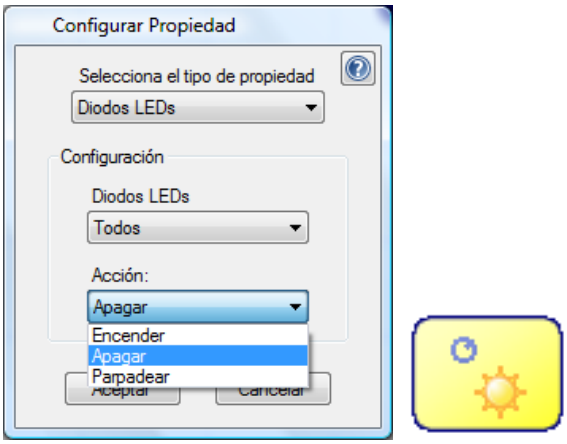
Matemáticos

A través de este módulo podremos modificar el valor de las variables, aumentando o disminuyendo su valor por una constante determinada.



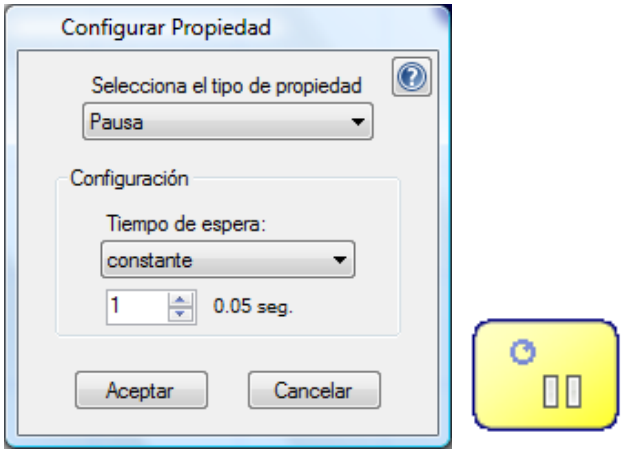
Diodos LEDs

A través de este módulo se podrá controlar los LEDs con los que cuenta el robot Moway , se podrá elegir que todos los LEDs se enciendan, se apaguen o parpadeen. Además se podrá seleccionar un determinado LED para que el módulo sólo le afecte a él.



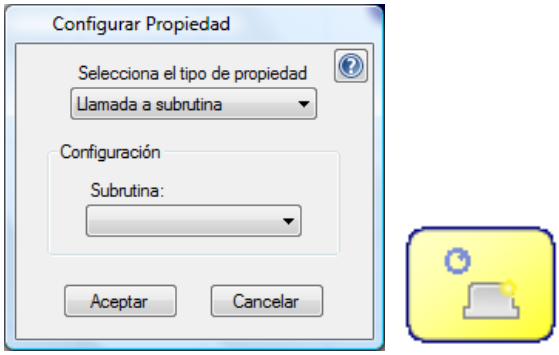
Pausa

Este módulo permite hacer una pausa en el programa, se podrá pausar un tiempo constante o el valor de una variable.



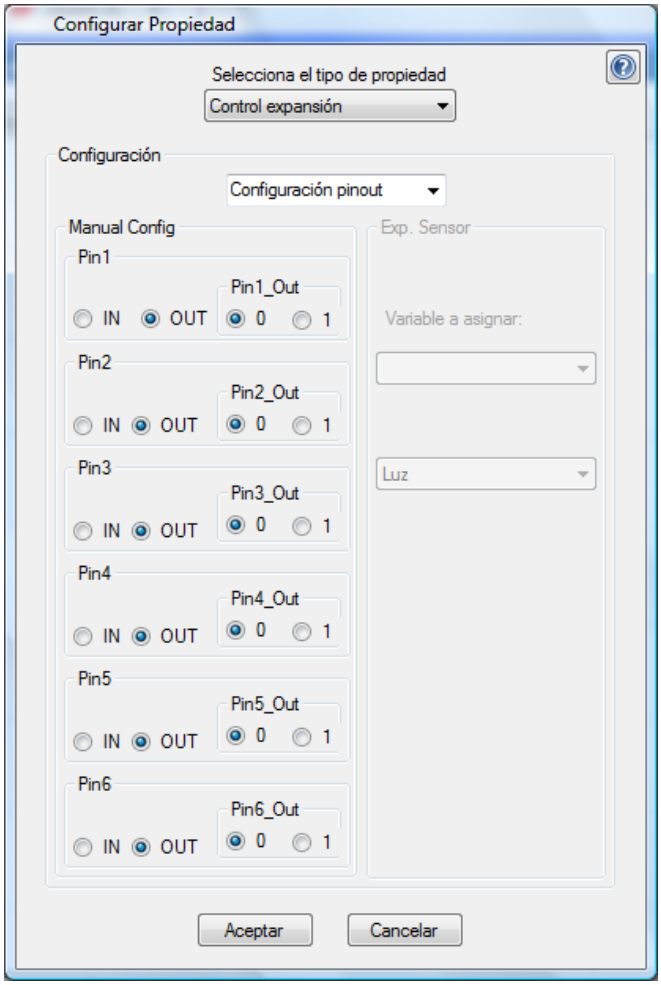
Llamada a subrutina

Este software te permite crear subrutinas, de tal forma que si un código se repite a lo largo de un programa en lugar de escribirlo más de una vez, se pueda llamar a una subrutina que te incluya dicho código. Así quedará el programa más limpio e intuitivo. En el módulo indicaremos la subrutina a la que vamos a llamar.



Control expansión

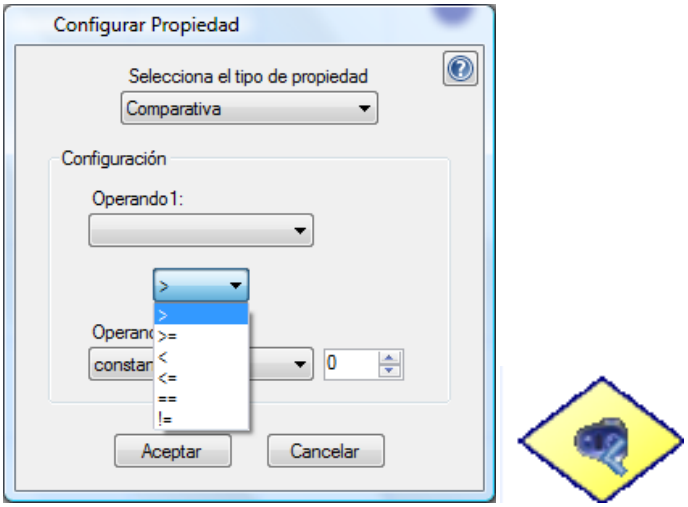
Este módulo permite reconfigurar el control de expansión, se puede utilizar con el módulo hardware de expansión. Sólo se recomienda el uso de este módulo por usuarios avanzados, ya que puede ocasionar daños en el hardware irreversibles.



4.1.3.3.2 Condiciones

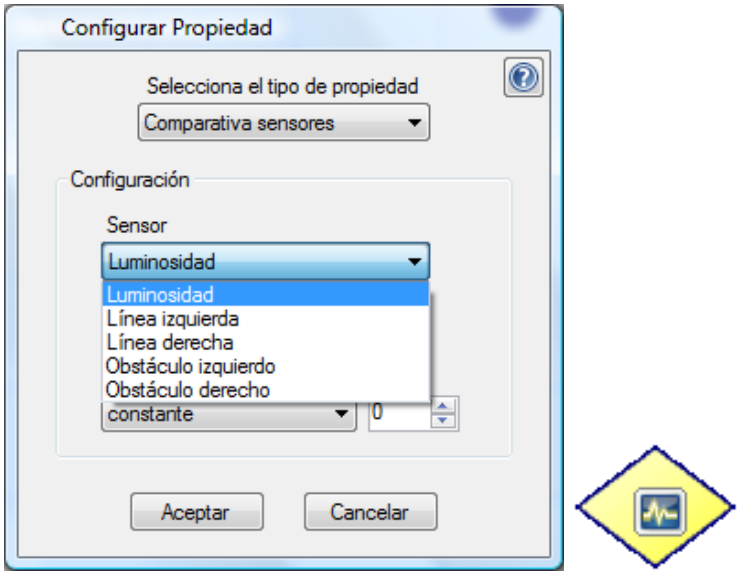
Comparativa

Esta comparativa permite comparar el valor de las variables que se hayan creado en el programa. Se trata de una comparativa matemática, podremos comparar entre dos variables o entre una constante y una variable. Las comparaciones podrán ser mayor, mayor o igual, igual, distinto, menor o igual o menor.



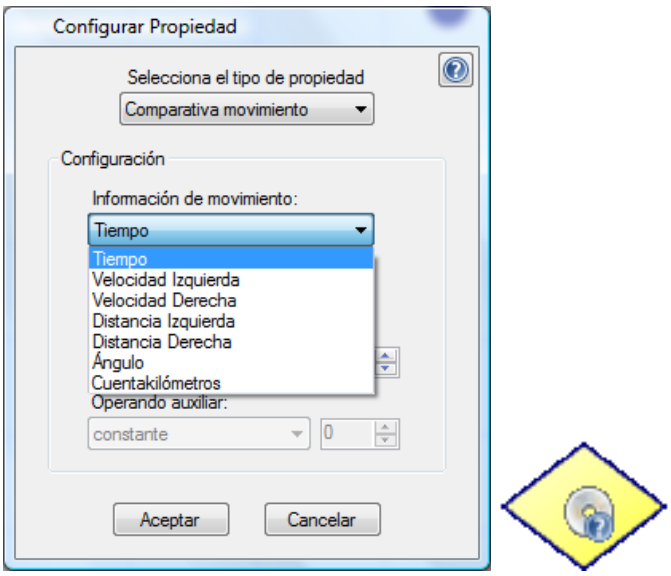
Comparativa sensores

Si la comparativa anterior permitía comparar de forma matemática el valor de una variable con respecto a otra variable o a una constante, esta comparativa se diferencia del anterior en que nos permite comparar el valor captado por un sensor con una variable o una constante. Las comparaciones son las mismas que en la comparativa anterior. Y los sensores a comparar son: sensor de intensidad de luz direccional (Luminosidad), sensor optorreflectivo infrarrojo para el suelo izquierdo (Línea izquierda), sensor optorreflectivo infrarrojo para el suelo derecho (Línea derecha), sensor infrarrojo anticolisión izquierdo (Obstáculo izquierdo) y sensor infrarrojo anticolisión derecho (Obstáculo derecho).



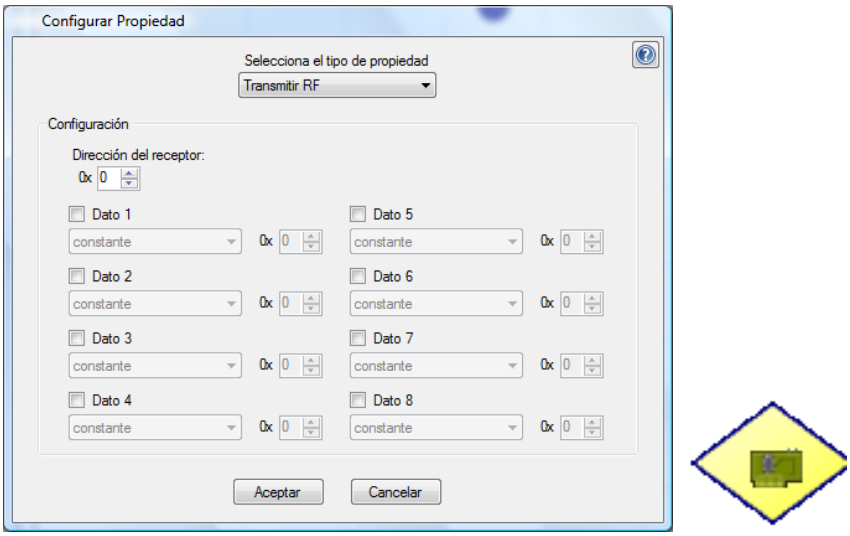
Comparativa movimientos

Esta comparativa se diferencia de los dos anteriores en que permite comparar la información relacionada con el sistema motriz. Así podremos comparar con una constante o una variable: el tiempo en movimiento, la velocidad del motor izquierdo, la velocidad del motor derecho, la distancia recorrida por el motor izquierdo, la distancia recorrida por el motor derecho, el ángulo que se ha girado o el cuentakilómetros.



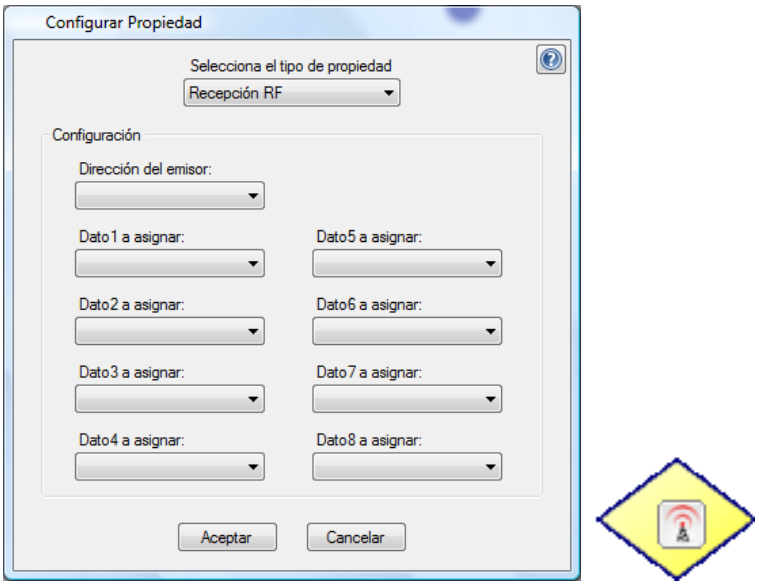
Trasmitir RF

Con esta comparativa se podrá enviar a través del módulo de radio frecuencia uno o más datos. Antes de utilizar esta comparativa se tendrá que configurar el RF del robot, para ello habrá que utilizar el módulo *Control RF*. Se configurará una dirección remitente y se pasará un dato o más, cada dato podrá ser una variable previamente creada e inicializada o un valor constante. Esta comparativa devolverá falso hasta que no reciba la confirmación de que se han recibido los datos por parte del remitente.



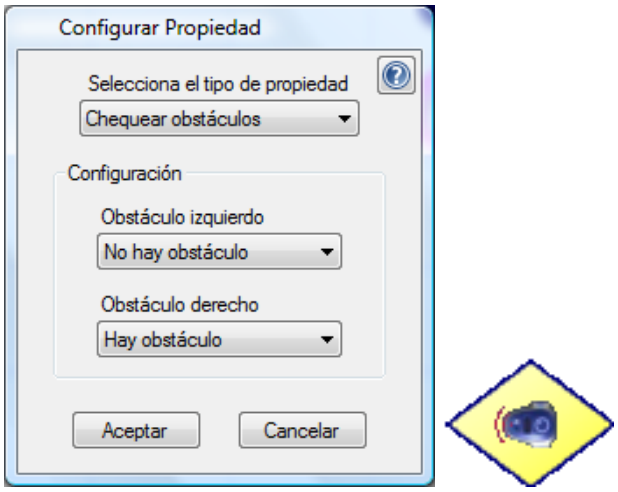
Recepción RF

Esta comparativa permite recibir datos a través del módulo RF, como el anterior, antes de usarlo será necesario configurar el módulo RF. Además se tendrá que crear una variable y asignarle la dirección del emisor y se tendrán que crear tantas variables como datos se quieran guardar.



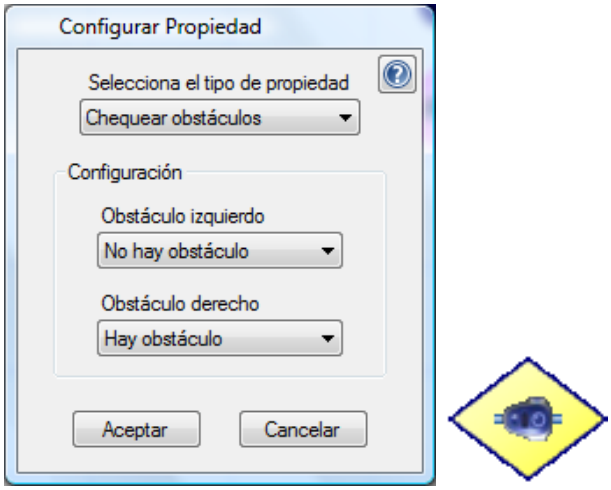
Chequear obstáculos

Esta comparativa chequea si hay obstáculos, se puede configurar para chequear los dos sensores a la vez (derecha o izquierda) o sólo uno de los dos, además se puede chequear si hay obstáculo o no hay obstáculo. Aproximadamente chequeará si hay un obstáculo a 3 centímetros.



Chequear línea

Esta comparativa nos permite comparar la información recogida a través de los sensores de línea, podremos chequear si el sensor derecho detecta línea negra o blanca o no chequearlo y lo mismo con el sensor izquierdo.

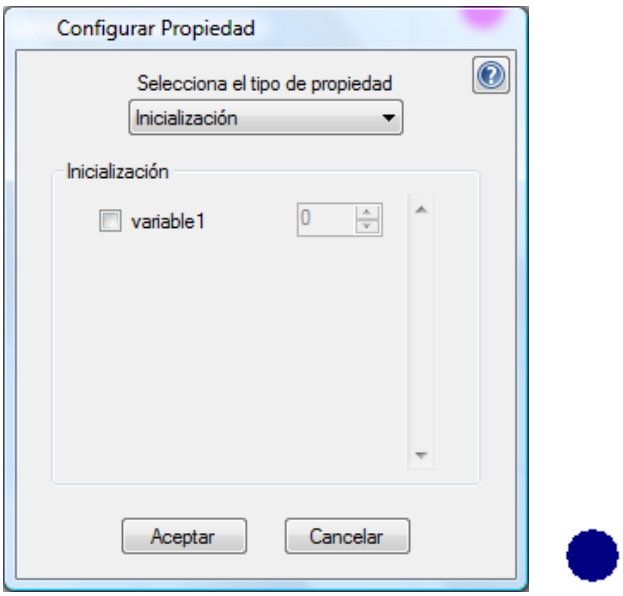


4.1.3.3.3 Flechas

Para formar el diagrama de flujo es necesario unir los módulos y las comparativas, para ello se utilizan las flechas. De cada módulo saldrá una flecha y de cada comparativa saldrán dos. Una cuando el resultado de la comparación sea verdadero y otra cuando sea falso. Las flechas que salgan de comparativas tendrán una etiqueta, V y F que ayudarán a entender el resultado de la comparación. Además el software chequea en todo momento si la flecha está bien posicionada, en casa contrario informa al usuario de ello.

4.1.3.3.4 Inicio y Final

Todo programa tiene que tener un inicio y por ello ***MowayGUI*** inicializa los nuevos programas con el elemento **Inicio**. Además en este elemento se podrán inicializar las variables creadas.



Además se podrá incluir un elemento **Final**. Si se quiere crear un bucle infinito entonces no se incluirá este elemento. Hay que tener en cuenta que si el diagrama llega al elemento **Final** pero no se ha mandado señal de parada a los motores, el robot seguirá en movimiento, por lo que antes de llegar al elemento **Final**, se tendrá que tener en cuenta el estado del robot, LEDs, movimiento…

4.1.3.4 LENGUAJE DE PROGRAMACIÓN ELEGIDO

Tras estudiar los diferentes lenguajes de programación que posibilita el robot ***Moway*** y las necesidades de los programas a desarrollar, se ha decidido trabajar con el diagrama de flujos.

Uno de los objetivos de este proyecto de fin de carrera es trabajar con diferentes entornos de programación, ésta es una de las razonas por las que se ha decido trabajar con un entorno gráfico. Además como se puede concluir de los aparatados anteriores, el software basado en diagrama de flujos es tan potente como los entornos de programación textuales.

4.1.4 SOFTWARE UTILIZADO

Los robots ***Moway*** vienen ya montados, por lo que no se requiere una etapa de diseño y montaje del robot, únicamente es necesaria una etapa de desarrollo de software. Para el desarrollo software como ya se ha explicado se ha utilizado el software **MowayGui**. Para la comunicación con el robot **Moway** se ha utilizado el software **Moway Center**.

4.1.4.1 MOWAYGUI

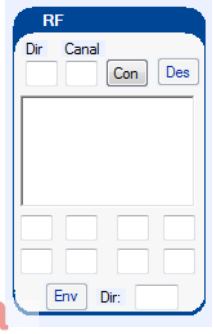
Como ya se ha explicado, la empresa Minirobots ha desarrollado un software que permite la programación de los robots ***Moway*** por medio de diagramas de flujo.

Este software permite grabar el programa desarrollado en el robot y permite configurar el módulo de radiofrecuencia de la llave RF-USB y enviar o recibir datos a través de él.

Grabación de un programa.

Una vez que tenemos creado el programa, haciendo clic en el icono , se podrá guardar el programa en la memoria del robot. Para ello el robot tendrá que estar conectado y encendido a la llave RF-USB. Antes de grabar el programa en el robot, lo guarda y al guardarlo compila el programa y avisa si hay algún error de programación, lo que impide grabar programas con erros de compilación en el error.

Comunicación RF



En la parte izquierda, incluye un módulo para la comunicación RF. Para utilizar este módulo es necesario conectar un módulo de RF en la llave RF-USB. Una vez introducida una dirección y un canal, se puede hacer clic en conectar, "Con" y se establecerá la comunicación. Se podrá recibir como enviar datos, en el cuadro de texto se verán los datos recibidos. Cuando se envíe un dato, el botón de enviar, "Env",

se pondrá verde si el envío ha sido satisfactorio o rojo en caso contrario.

Hay que indicar que mientras esté el módulo de RF conectado, no se podrá grabar ni guardar un programa.

4.1.4.2 MOWAY CENTER

Para la comunicación del robot se ha utilizado el programa **Moway Center**. Permite gestionar la comunicación entre la llave RF-USB y el robot. Se puede descargar de forma gratuita desde la web de **Moway**. Se trata de un software que corre bajo Windows y por ello utiliza el sistema de ventanas de Windows. Está dividido en dos partes:

Parte principal.

Desde la esta parte se puede consultar el estado del robot así como realizar diferentes acciones sobre él. Está dividida en cinco secciones.

Moway Status

Hay tres posibles estados, conectado, apagado y desconectado, en esta sección se informará del estado.

Acciones

Permite interactuar con el robot, está formado por tres acciones, grabación de la memoria de un programa, lectura de la memoria de un programa y borrado de la memoria de programa.

Batería

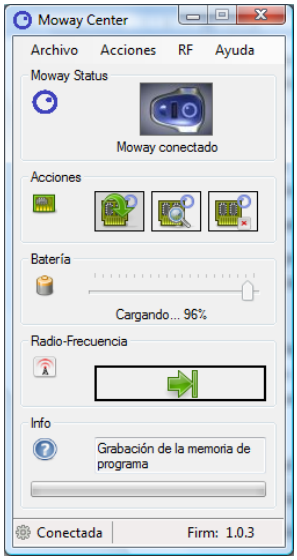
Informa del porcentaje de batería cargada.

Radio-Frecuencia

Permite habilitar la segunda parte, la parte Radio Frecuencia

Info

Muestra la información sobre el software y los procesos.



Radio frecuencia

Esta parte permite la comunicación entre la llave RF-USB y el robot Moway. Para poder utilizar esta parte será necesario colocar un módulo de comunicación RF en la llave RF-USB y en el robot.

Está dividida en cuatro secciones:

Configuración RF

Desde aquí se configurará el módulo de comunicación RF de la llave. Es necesario que tanto el robot como la llave estén configurados en el mismo canal.

Dato a enviar

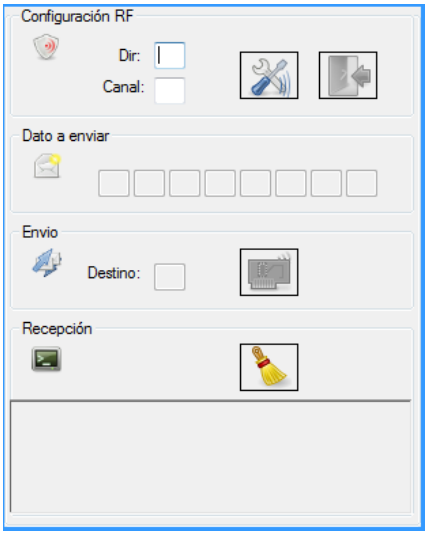
En esta sección se establecerá el dato a enviar al robot.

Envío

En esta sección se indicará la dirección del destinatario y haciendo clic en el icono de la derecha se enviará el dato.

Recepción

En el cuadro de texto de la parte inferior se verán los datos recibidos, haciendo clic en el icono de la escoba se borrarán los datos recibidos.



4.1.5 PRÁCTICAS PARA VER EL FUNCIONAMIENTO.

Para poner en práctica todo lo explicado anteriormente, se han desarrollado una serie de prácticas para ver el funcionamiento de los distintos sensores. Para desarrollar todas ellas se ha decidido utilizar la programación basada en diagramas de flujos, ya que posibilita la realización de estas prácticas de manera clara y eficiente.

4.1.5.1 Sensores infrarrojos anticolisión

Los sensores infrarrojos anticolisión informan si hay obstáculos por delante del robot, tanto por la derecha como por la izquierda.

Para ver el correcto funcionamiento de estos sensores, se introducirá un robot en un laberinto, y el robot tendrá que ser capaz de ir por las calles del laberinto sin chocar con las paredes. El robot comenzará a recorrer el laberinto cuando se le indique a través del módulo de radio frecuencia y parará cuando se le mande parar a través de dicho módulo.

Material utilizado:

- ✓ Un robot Moway.
- ✓ Una llave RF-USB.
- ✓ Dos módulos de comunicación RF: Uno se conectará al robot y otro a la llave RF-USB.
- ✓ Moway Center: Necesario para comunicar al robot el inicio y el fin.
- ✓ MowayGUI: Para programar el robot.

Diseño del programa:

Configurar el módulo de radio frecuencia de la llave RF-USB.

Configurar el módulo de radio frecuencia del robot.

Configurar la dirección del emisor en el robot.

Mandar la señal de inicio.

Recibir la señal de inicio.

Si se ha recibido señal de inicio.

    Encendemos el LED verde superior.

    Si no se ha recibido la señal de fin.

        Si no hay obstáculo.

            Ir hacia adelante hasta que haya obstáculo.

        Si hay obstáculo a la derecha.

            Girar 90º hacía la izquierda.

        Si hay obstáculo a la izquierda.

            Girar 90º hacía la derecha.

        Fin Si.

Si se ha recibido la señal de fin.

Parar el robot.

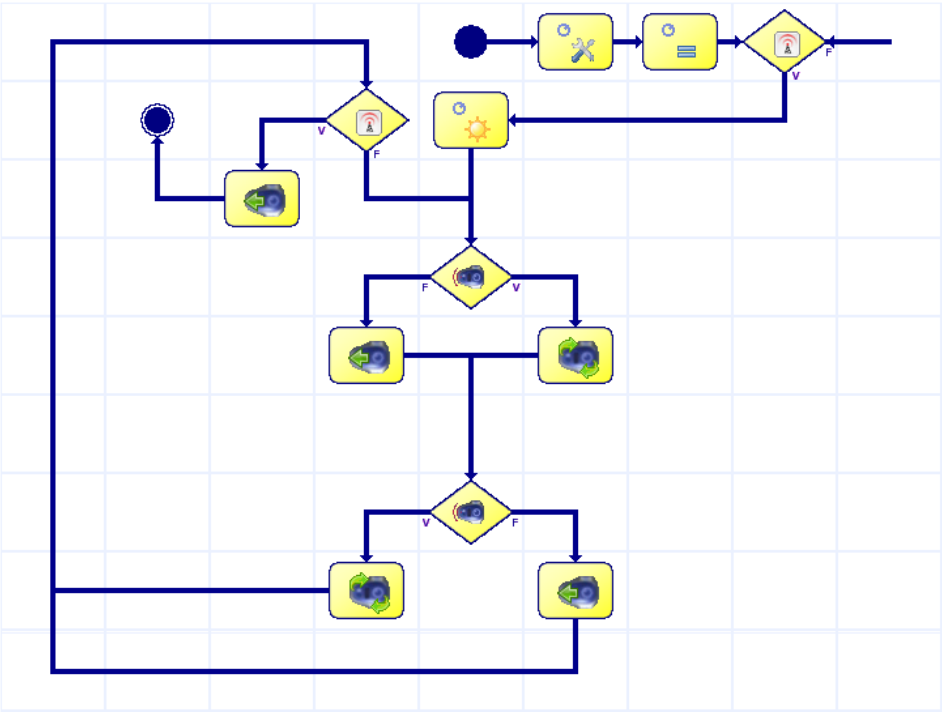
Fin Si.

Si no se ha recibido la señal de inicio.

Volver a recibir datos.

Fin Si.

Programa en MowayGUI



4.1.5.2 Sensor de intensidad de luz direccional

Este sensor está colocado en la parte delantera del robot. Es capaz de medir la intensidad de la luz e identificar la fuente de luz.

Para probar este sensor se colocará el robot en un sitio oscuro, se colocará una linterna en el suelo y el robot tendrá que ser capaz de localizar la fuente de luz y llegar hasta la linterna. Se irá moviendo de lugar la linterna y el robot tendrá que modificar su ubicación. Hay que tener en cuenta que según el tipo de linterna habrá que modificar el umbral de luz que diferenciará si sigue o no la dirección hacía la fuente de luz.

Material utilizado:

- ✓ Un robot Moway.
- ✓ MowayGUI: Para programar el robot.
- ✓ Una linterna.

Diseño del programa

El robot irá girando hacia la izquierda y luego hacia la derecha aumentando el ángulo de rotación y en el caso de que localice la fuente de luz irá hacia ella. Este programa no tendrá fin, por lo que será un bucle infinito.

Inicializar las variables lightValue = 70, rotDir = 0, rotAngle = 0.

Mientras no se apague el robot.

    Si rotAngle es mayor que 120, se le asigna 20.

    Si no es mayor de 120.

    Fin Si.

Si rotDir es 0

    Se le asigna a rotDir 1.

Se manda al sistema motriz rotar a la izquierda.

Si rotDir es 1.

    Se le asina a rotDir 0.

Se manda al sistema motriz rotar a la derecha

    Fin Si.

Si el ángulo rotado es mayor que rotAngle.

Pasamos a la siguiente iteración

Si el ángulo rotado es menor o igual que rotAngle.

Si la luminosidad captada es mayor que lightValue

        Se le manda al robot ir recto.

        Se le asigna a rotAngle 20.

    Si la luminosidad captada es menor o igual que lightValue.

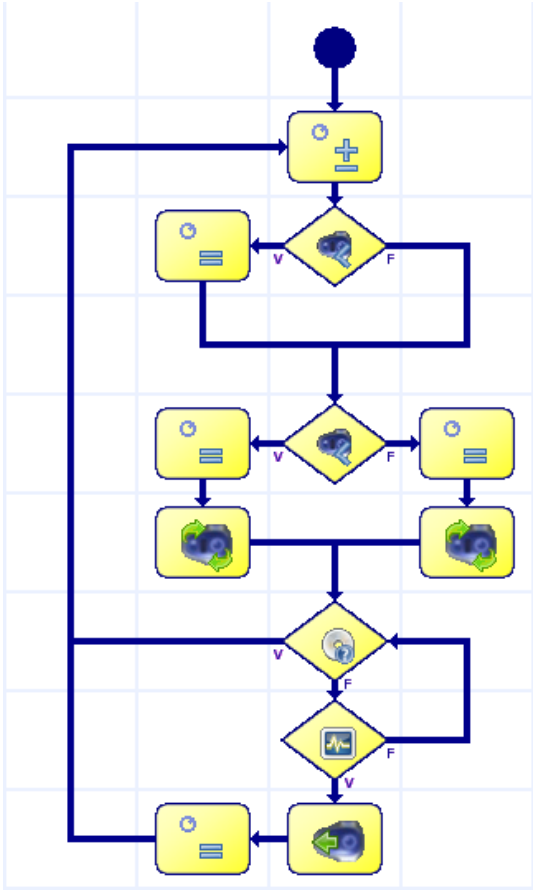
        Seguimos rotando.

    Fin Si.

Fin Si.

Fin Mientras.

Programa en MowayGUI:



4.1.5.3 Sensores optorreflectivos infrarrojos para el suelo

Este sensor está colocado en la parte inferior del robot. Chequean el color de la superficie, distinguiendo entre blanco y negro.

Para ver el funcionamiento de este sensor, se colocará sobre una superficie blanca una cinta aislante negra formando un circuito. El robot tendrá que ser capaz de seguir esta línea sin salirse de ella.

Material utilizado:

- ✓ Un robot Moway.
- ✓ MowayGUI: Para programar el robot.
- ✓ Una superficie blanca y cinta aislante negra

Diseño del programa:

Este programa será un bucle infinito, según la captura de los sensores mandará al sistema motriz una tarea u otra. El contenido del bucle será el siguiente.

Si los sensores devuelven Negro los dos

Vamos hacia adelante.

Si no devuelven los dos negro, si el sensor izquierdo devuelve negro

Hará giros derecha – izquierda aumentando el ángulo de rotación hasta encontrar la línea

Si nos devuelven los dos negros, el sensor izquierdo no devuelve negro y el sensor derecho devuelve negro.

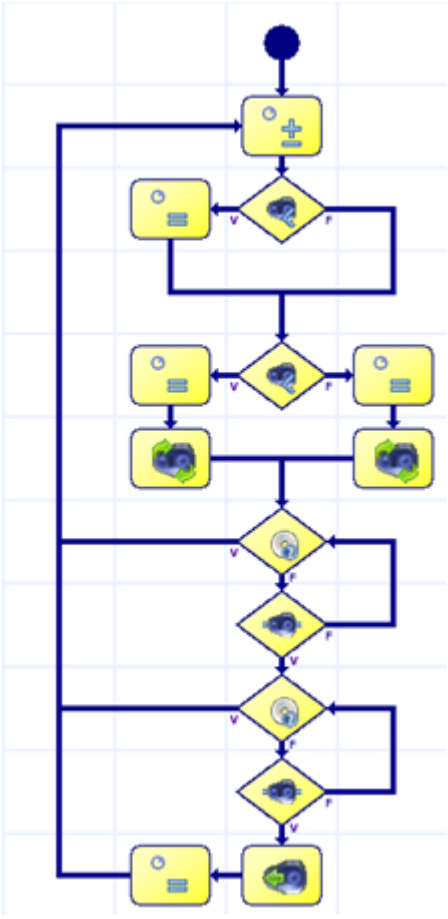
Hará giros derecha – izquierda aumentando el ángulo de rotación hasta encontrar la línea

Si ninguno devuelve negro

No se manda ninguna tarea al sistema motriz, para que siga ejecutando la tarea que se le envío por última vez y se vuelven a chequear los sensores de línea.

Fin Si

Programa en MowayGUI:



4.1.5.4 Módulo RF entre robots.

Como última práctica se va realizar un programa en que dos robots se releven sobre una línea negra. Para ello se van a utilizar los sensores de línea, los sensores de obstáculos, el sensor de luz direccional y la comunicación RF entre dos robots.

Se colocarán dos robots sobre un circuito formado con una pista blanca y una línea negra, uno detrás del otro. El de adelante comenzará a recorrer la línea cuando se le quite el dedo del sensor de luz direccional. Cuando reconozca al otro robot se parará y mandará al otro robot empezar a recorrer la línea.

Material utilizado:

- ✓ Dos robots Moway.
- ✓ Dos módulos RF.
- ✓ MowayGUI.
- ✓ Una pista blanca y cinta aislante negra.

Diseño del programa:

Se realizarán dos programas distintos, uno para el robot que comenzará la carrera (Programa A) y otro para el robot que tome el primer relevo (Programa B).

Programa A

Configurar el módulo RF, dirección 1, canal 1

Si se la luminosidad captada es mayor que 10

Mientras que no se apague el robot

Si hay obstáculos delante.

Se para los motores.

Se manda la señal a la dirección 2

Si se manda correctamente

Se espera a recibir señal, de la dirección 2

Si se recibe señal se sigue la línea

Si no se manda correctamente

Se vuelve a mandar.

Fin Si

Si no hay obstáculos

Se sigue la línea.

Fin Si.

Fin Mientras.

Programa B

Configurar el módulo RF, dirección 2, canal 1

Se espera a recibir señal, de la dirección 1

Mientras que no se apague el robot

Si hay obstáculos delante.

Se para los motores.

Se manda la señal a la dirección 1

Si se manda correctamente

Se espera a recibir señal, de la dirección 1

Si se recibe señal se sigue la línea

Si no se manda correctamente

Se vuelve a mandar.

Fin Si

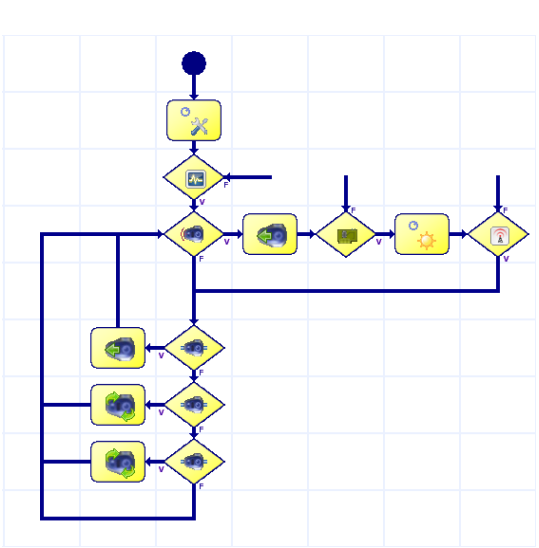
Si no hay obstáculos

Se sigue la línea.

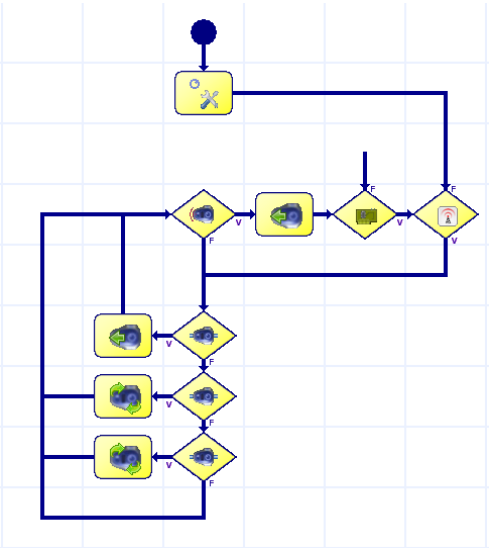
Fin Si.

Fin Mientras.

Programa en MowayGUI:



Programa A



Programa B

4.1.6 BAILARINES

Como ya se ha explicado, esta prueba fue descubierta en el concurso de Deusto. Se trata de una prueba muy atractiva para el público de la competición, ya que ver a los robots bailar resulta bastante divertido. Además uno de los objetivos de este proyecto es trabajar con distintos kit de robots, y esta prueba está diseñada a medida para los robots Moway, ya que los requisitos que deben cumplir los robots participantes en esta prueba son cubiertos con este kit.

4.1.6.1 Análisis de la prueba

En este apartado se va tratar de analizar todos los requisitos que hay que tener en cuenta a la hora de dar una solución competitiva, se comenzará por leer la normativa, a continuación se incluye un breve resumen.

La normativa no marcaba muchas reglas, sólo que el primer robot se pondrá en funcionamiento y conforme se vayan incluyendo el resto de los robots, éstos deberán sincronizarse y continuar con el baile. La organización será la encargada de introducir los robots, así se evitará que los participantes calculen el momento en que van a ir insertando los diferentes robots. Otra de las cosas a tener en cuenta que el ganador será el baile que más guste al público, por lo que habrá que tener un baile vistoso.

De la normativa sacamos el siguiente listado de requisitos:

- ✓ Se necesitarán al menos dos robots.
- ✓ Será necesaria la comunicación entre ellos, para poder sincronizarse.
- ✓ El baile tendrá que ser vistoso.
- ✓ El tiempo de baile tendrá que ser ilimitado, ya que a priori no se conoce cuando se introducirán los robots.

4.1.6.2 Análisis de la solución

En este caso se van a utilizar dos robots, dos robots Moway**.** Por lo tanto necesitaremos dos robots y dos módulos de comunicación a través de radio-frecuencia. Como baile se ha decidido por hacer un baile tipo salsa, en el que los robots bailarán de forma sincronizada. Además se implementará el baile de forma que hasta que no se le indique al robot que ha comenzado el baile que paré, los robots seguirán bailando.

El robot que comience el baile llevará el peso del baile, será el encargado de dirigir el baile al otro robot.

Se implementarán una serie de pasos de baile que el robot tendrá que realizar mientras que no se ponga en marcha el otro robot.

Una vez que los dos robots estén sincronizados, el primer robot será el encargado de mandar a través del módulo de radio-frecuencia al otro robot que tiene que hacer en cada momento.

En el momento que el robot principal reciba la señal de parar, mandará al otro robot parar y se parará el también. Se ha decidido que para mandar la señal de parar se va

utilizar el sensor de luz direccional, cuando se coloque el dedo pulgar en el sensor, el robot tendrá que ser capaz de parar en los siguientes segundos.

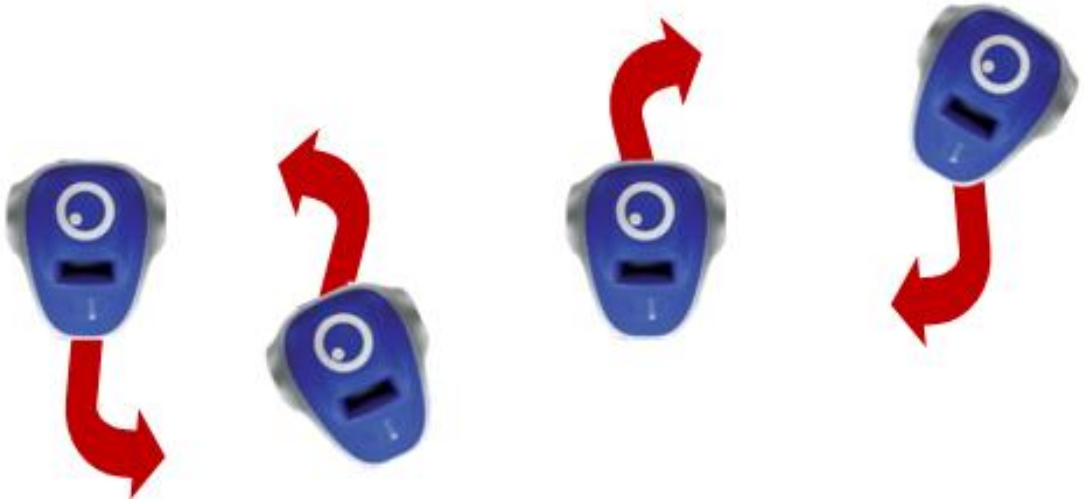
4.1.6.3 Diseño del baile

Para realizar un diseño más claro del baile se ha decidido dividir el baile en bloques. Así tendremos un bloque que incluirá los pasos de baile que el robot realizará hasta que se ponga en marcha el otro robot. Y luego tendremos varios bloques, que formarán los pasos a realizar por un robot u otro. A continuación de forma gráfica se muestran los diferentes bloques. A la hora de hablar de los diferentes robots, se identificará al robot principal como, Robot P y al secundario como, Robot S.

Bloque inicial

El robot principal hará reiterativamente estos pasos hasta que el segundo robot sea introducido.

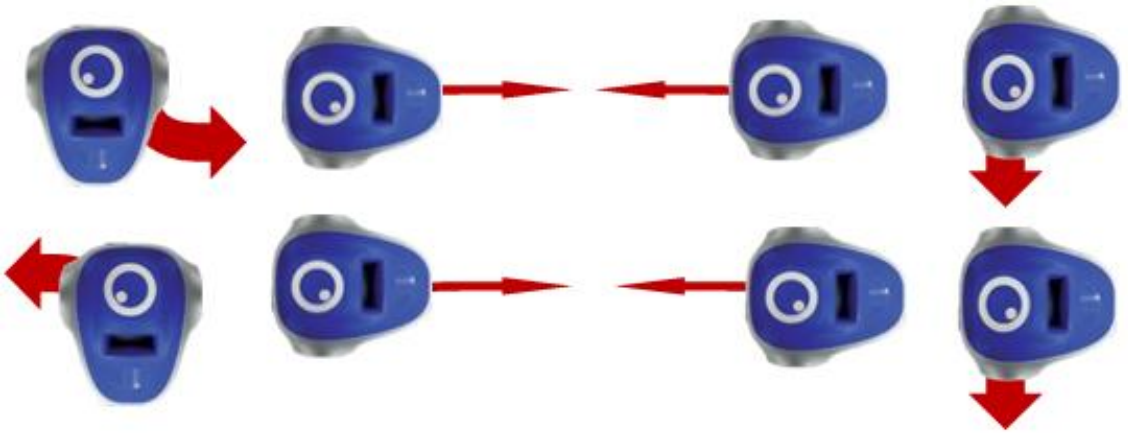
Bloque 1-A



Estos pasos formarán el primer bloque que baile el Robot P cuando se sincronicen los robots.

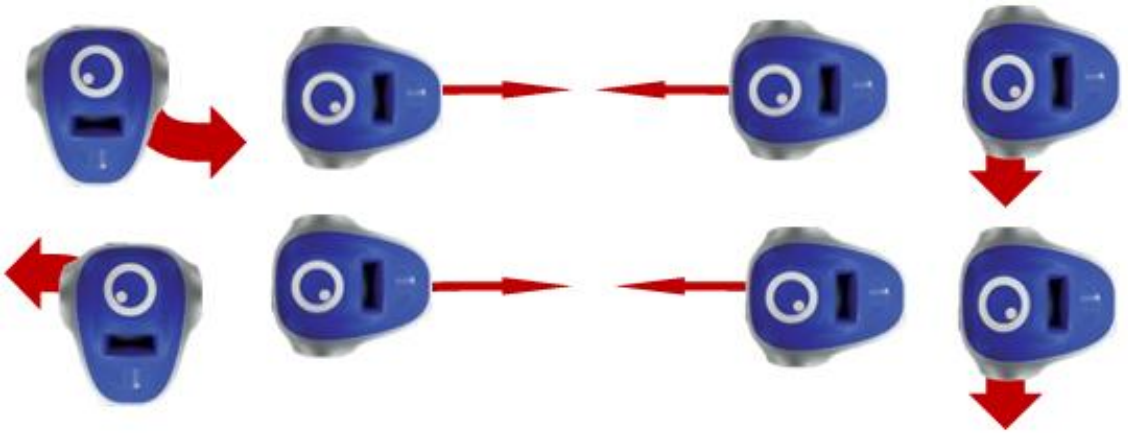
Bloque 1-B

Estos pasos formarán el primer bloque que baile el Robot S cuando se sincronicen los robots.



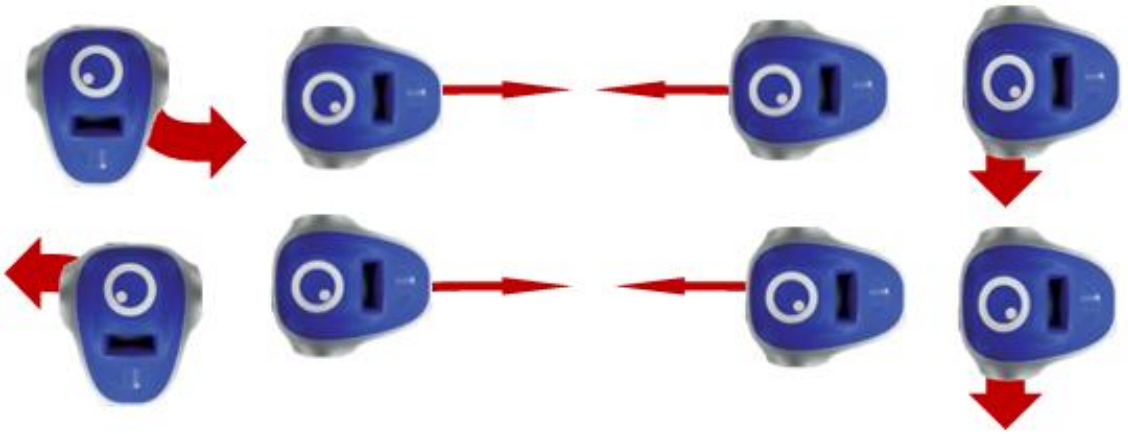
Bloque 2-A

Una vez bailado el bloque 1, los robots pasarán a sus bloques 2, el bloque del Robot P será:



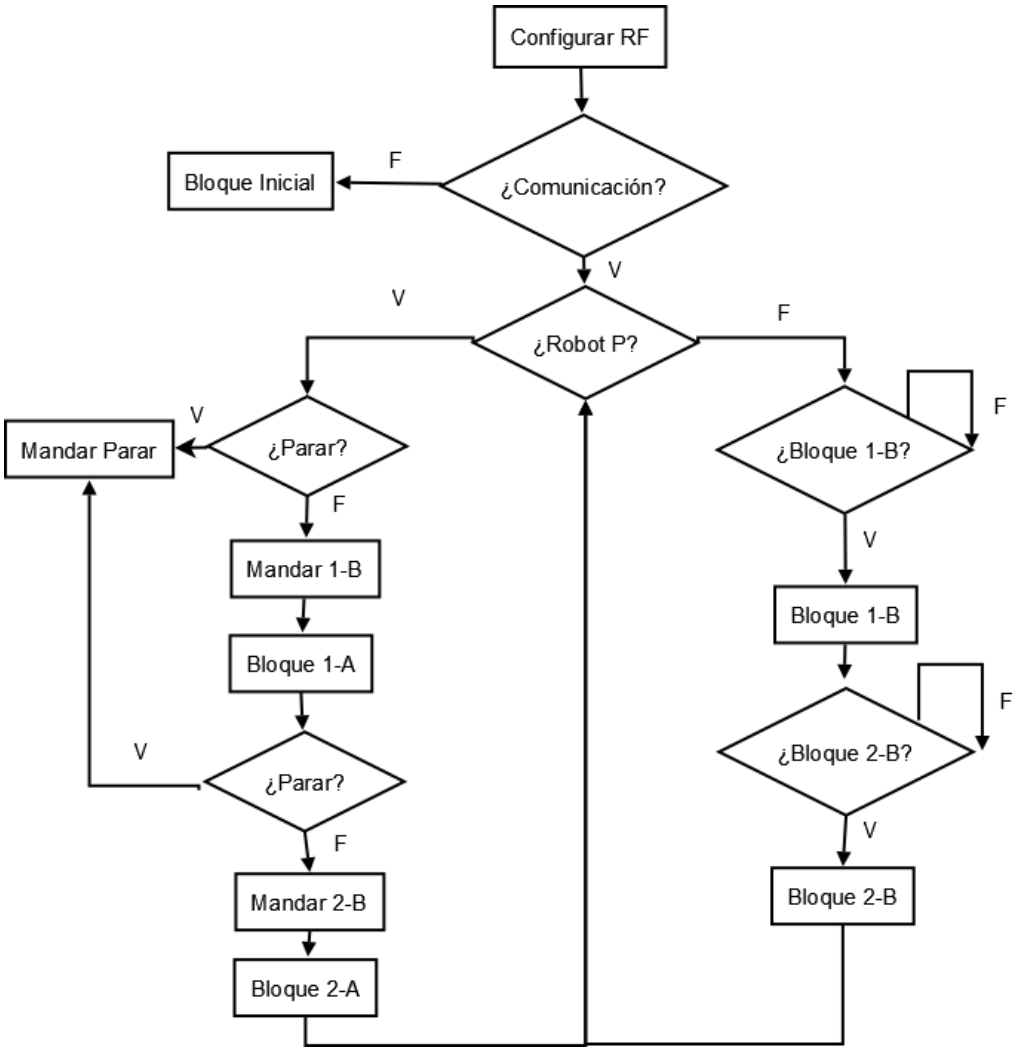
Bloque 2-B

A la vez que el Robot P realiza el Bloque 2-A, el Robot S, realizará los siguientes pasos:



4.1.6.4 Diseño del solución

A través de un diagrama se va presentar el diseño del programa que se implementará para dar solución a esta prueba. A la hora de trabajar, se ha decidido que se implementará utilizando el software ***MowayGUI***, además se hará uso de las subrutinas para tener un programa más limpio. Se implementará una subrutina por cada bloque de baile y en el programa principal se implementará la comunicación con el robot principal y la comunicación entre ellos.

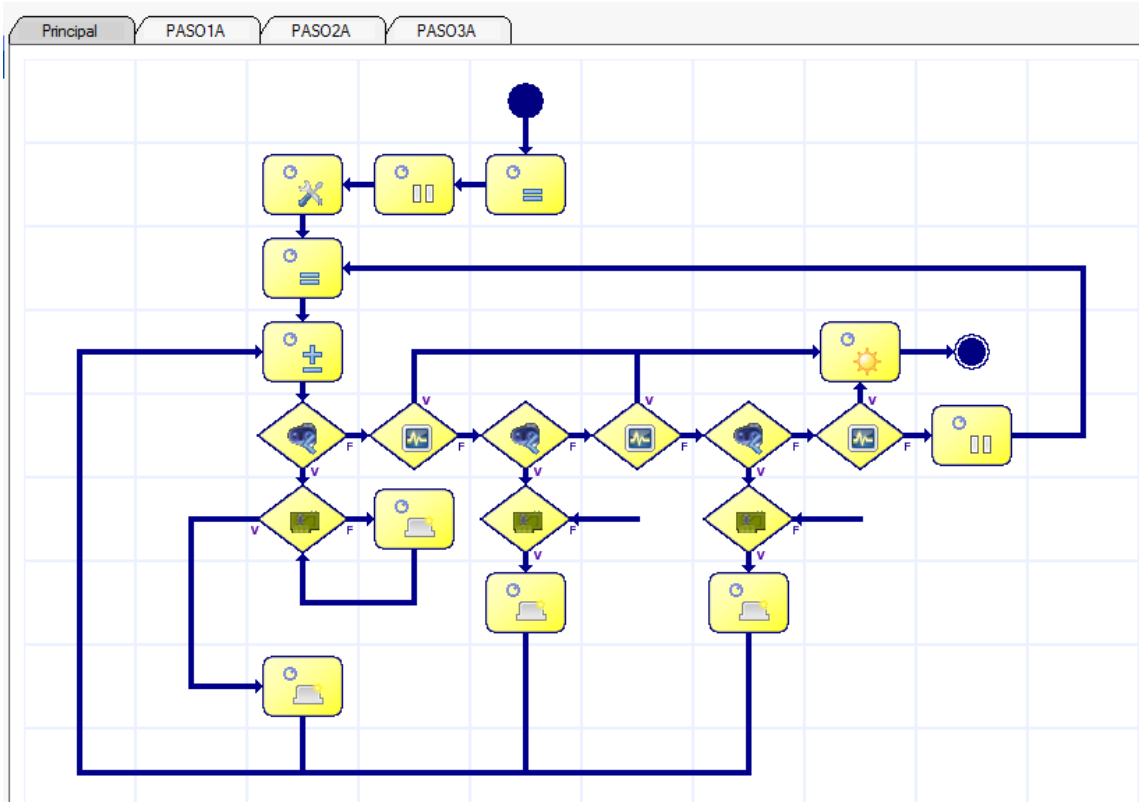


4.1.6.5 Implementación de la solución

Como ya se ha dicho, se va a dividir el baile en bloques, cada bloque será una subrutina, el programa principal será el encargado de gestionar la comunicación y de llamar a la subrutina correspondiente.

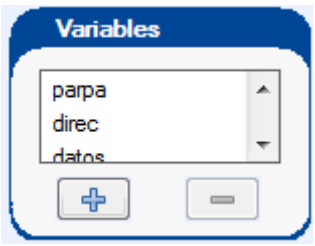
4.1.6.5.1 Implementación del código del robot principal

4.1.6.5.1.1 Programa principal



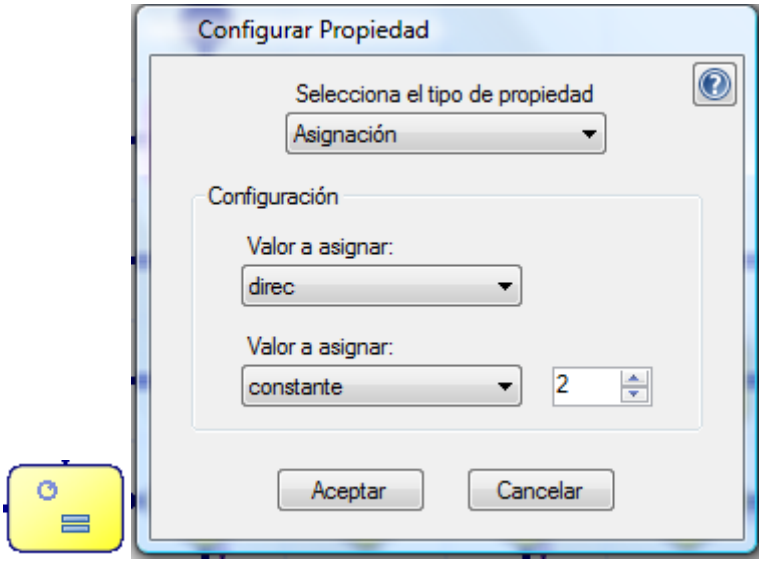
Variables:

- ✓ Variable que almacenará la dirección del robot S, direc.
- ✓ Variable que almacenará el bloque de pasos que toca, datos.
- ✓ Variable que ayudará para que los sensores parpadeen tres veces, parpa

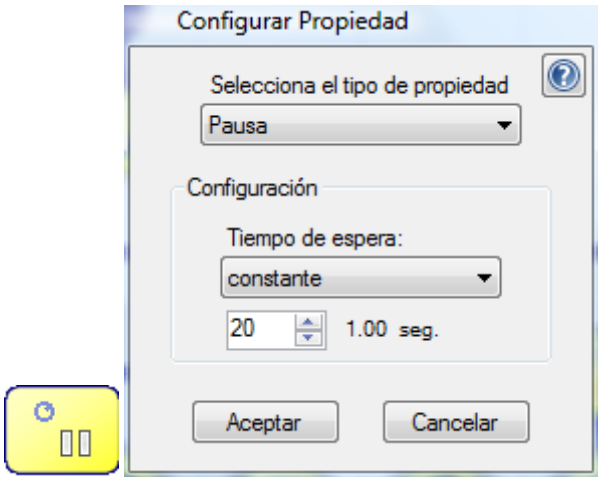


Pasos:

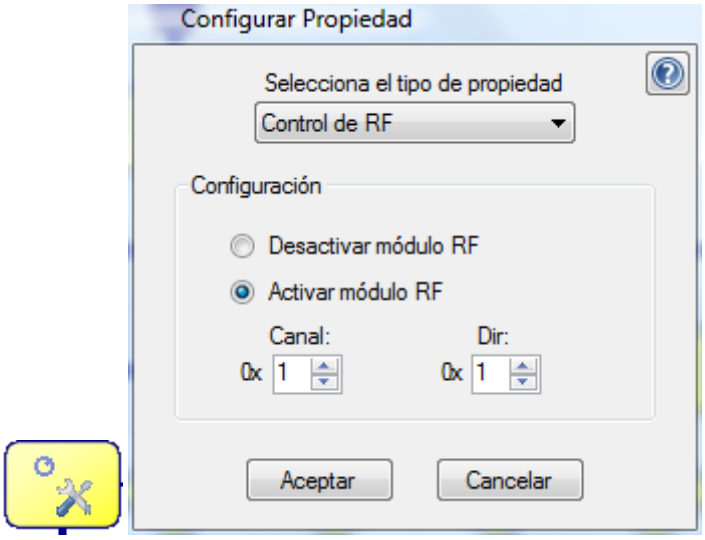
1 Inicializar direc a dos, dirección del Robot S.



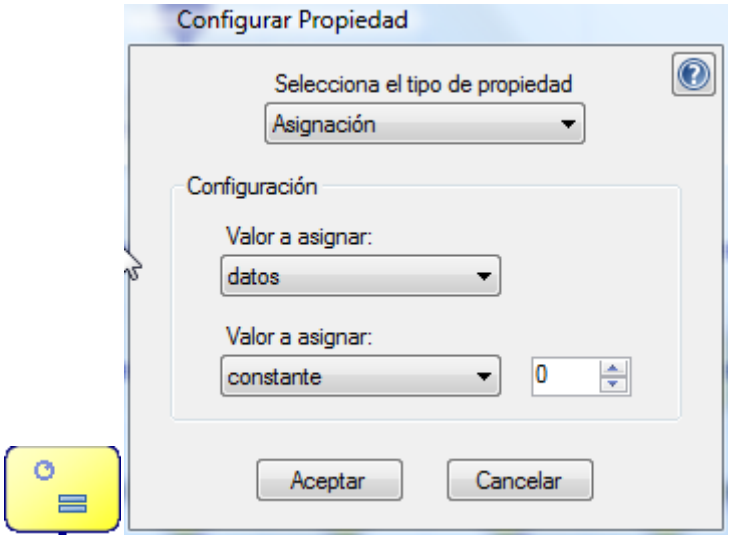
2 Parar veinte segundos antes de comenzar a bailar.



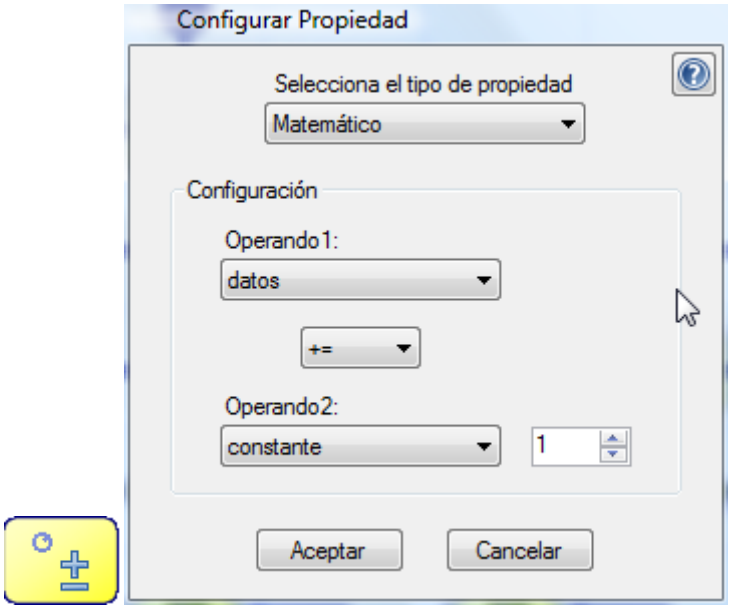
3 Configurar el módulo RF, canal uno, dirección uno.



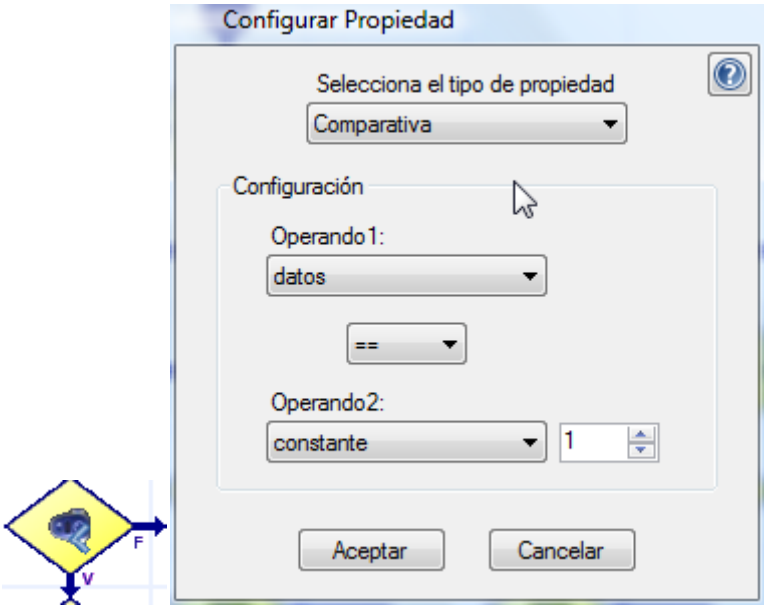
4 Inicializar datos a cero.



5 Sumar uno a datos.

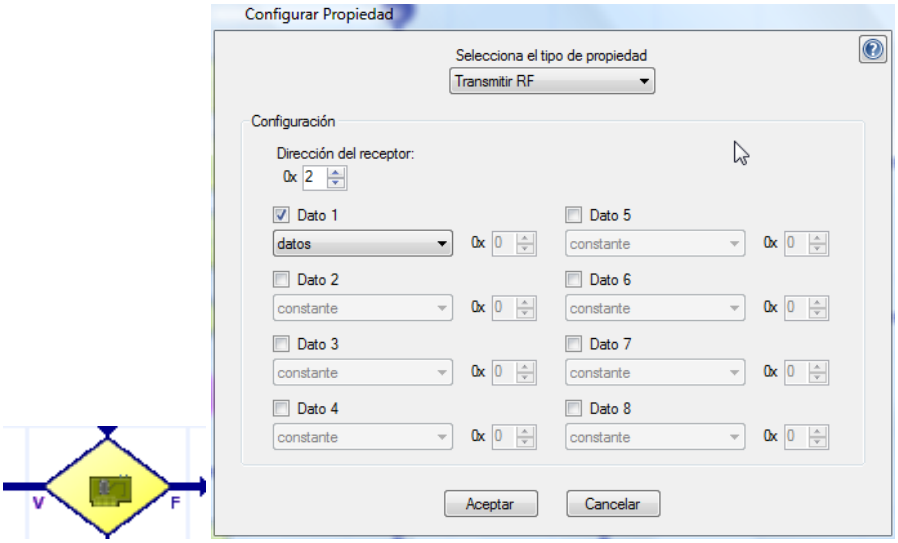


6 Comprobar si datos es igual a uno.



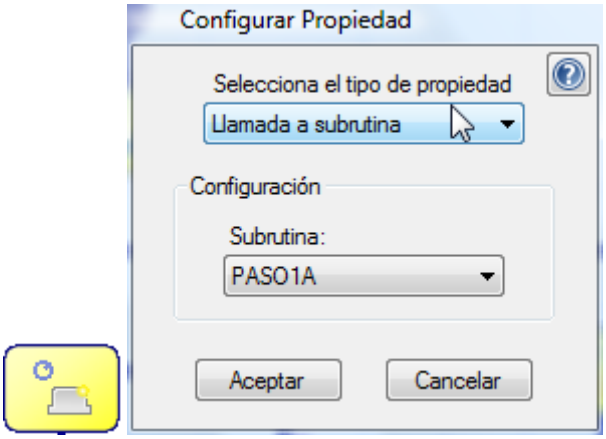
Si datos es igual a 1.

7 Enviar el valor de datos al robot S.



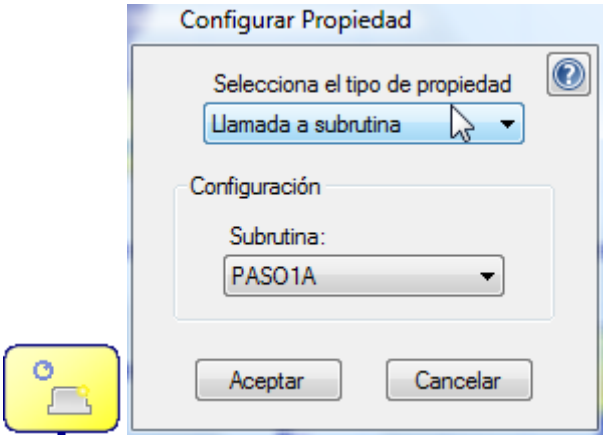
Si el envío no es correcto.

8 Bailar el bloque 1A y volver al paso 7.



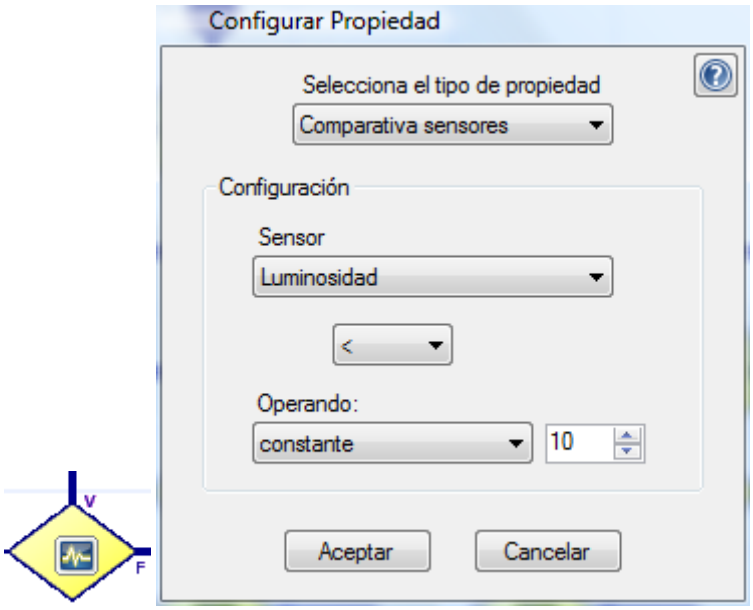
Si el envío es correcto.

9 Bailar el bloque 1A y volver al paso 5.



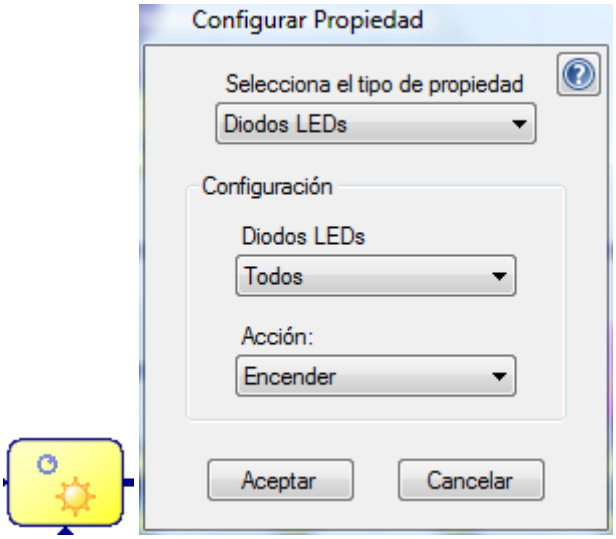
Si no es cierto que datos es 1.

10 Comprobar luz percibida por el sensor de luz.



Si la luminosidad es menor que 10

11 Encender los LEDs



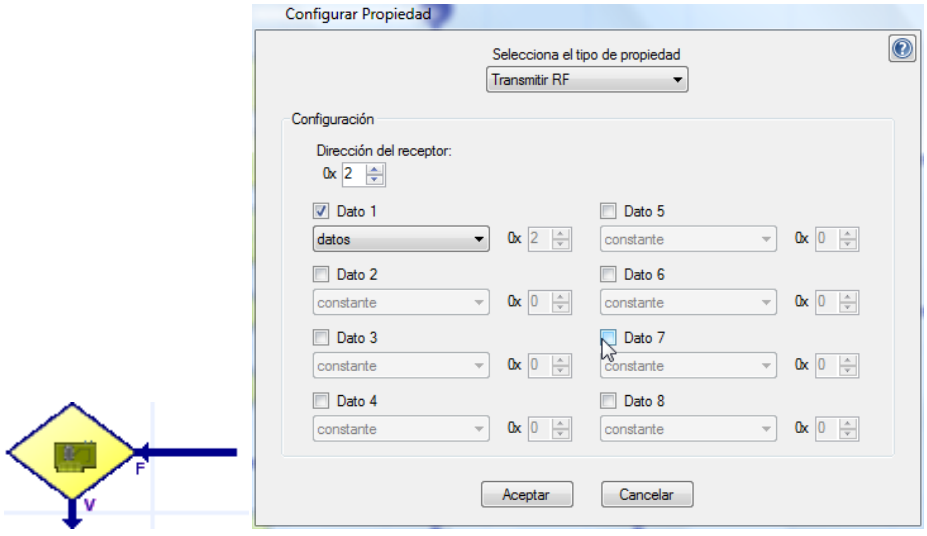
12 Termina la ejecución

Si la luminosidad no es menor que 10

13 Comprobar si datos es igual a dos.

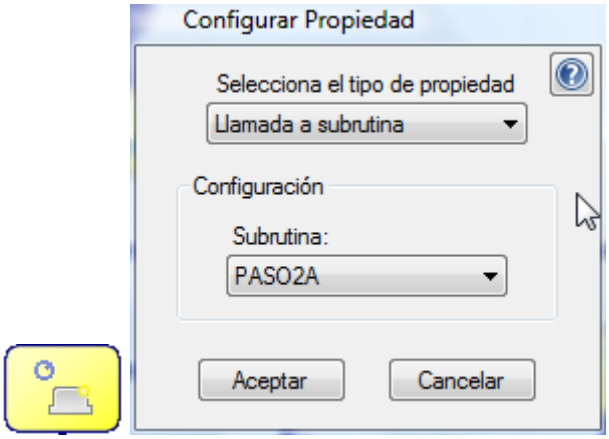
Si es cierto que datos es 2.

14 Enviar el valor de datos al robot S.



Si el envío es correcto.

15 Bailar el bloque 2A y volverá al paso 5.

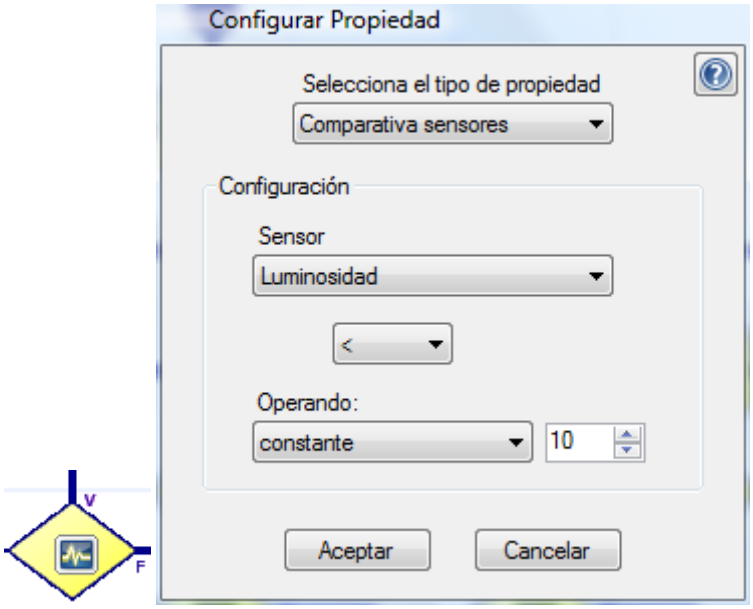


Si el envío no es correcto.

Vuelve al paso 14.

Si datos no es igual a 2

16 Comprobar luz percibida por el sensor de luz.



Si la luminosidad es menor que 10

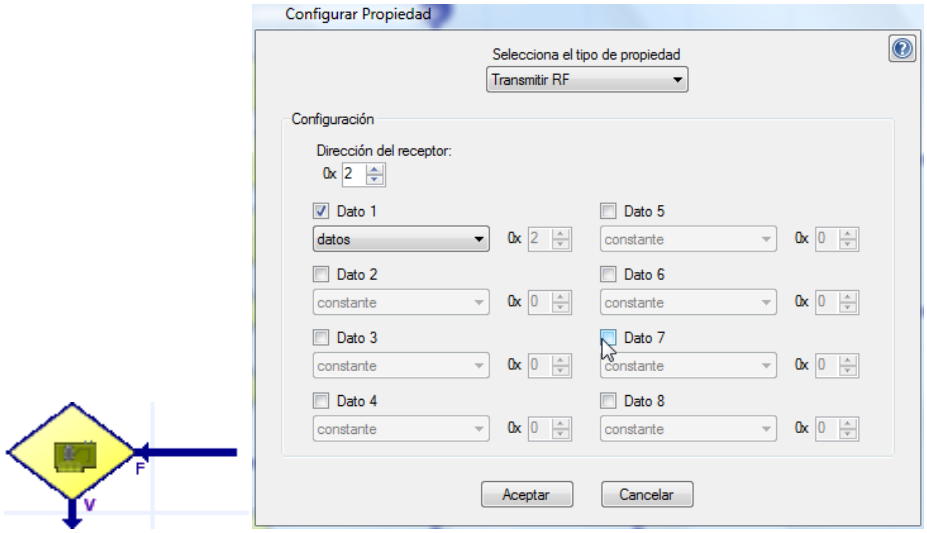
Volver al paso 11 y 12

Si la luminosidad no es menor que 10

17 Comprobar si datos es igual a tres.

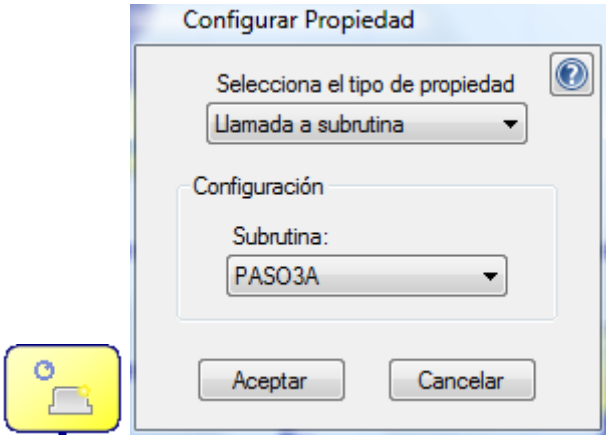
Si es cierto que datos es 3.

18 Enviar el valor de datos al robot S.



Si el envío es correcto.

19 Bailar el bloque 3A y volverá al paso 5.

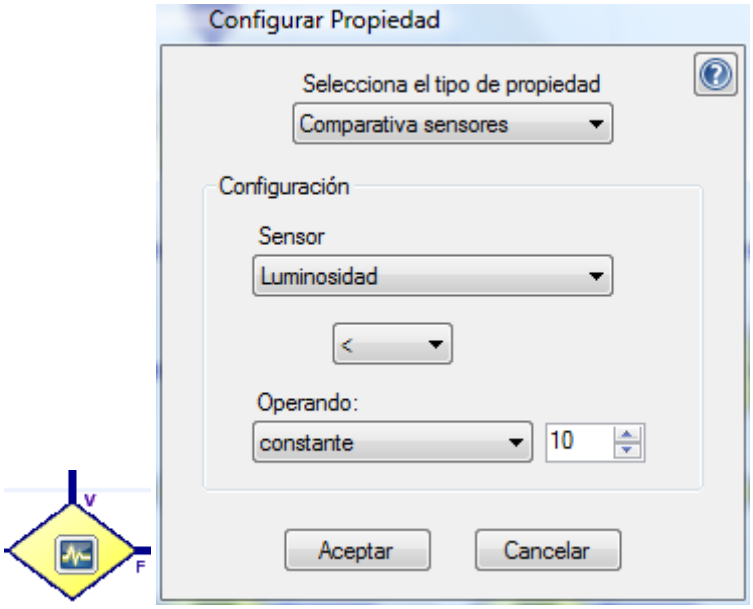


Si el envío no es correcto.

Vuelve al paso 18.

Si datos no es igual a 3.

20 Comprobar luz percibida por el sensor de luz.

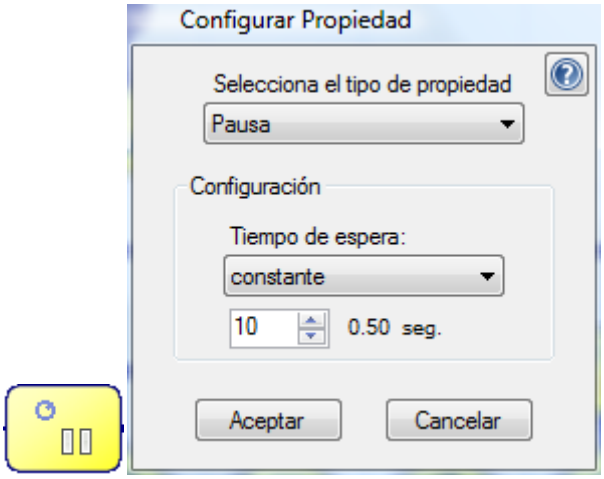


Si la luminosidad es menor que 10

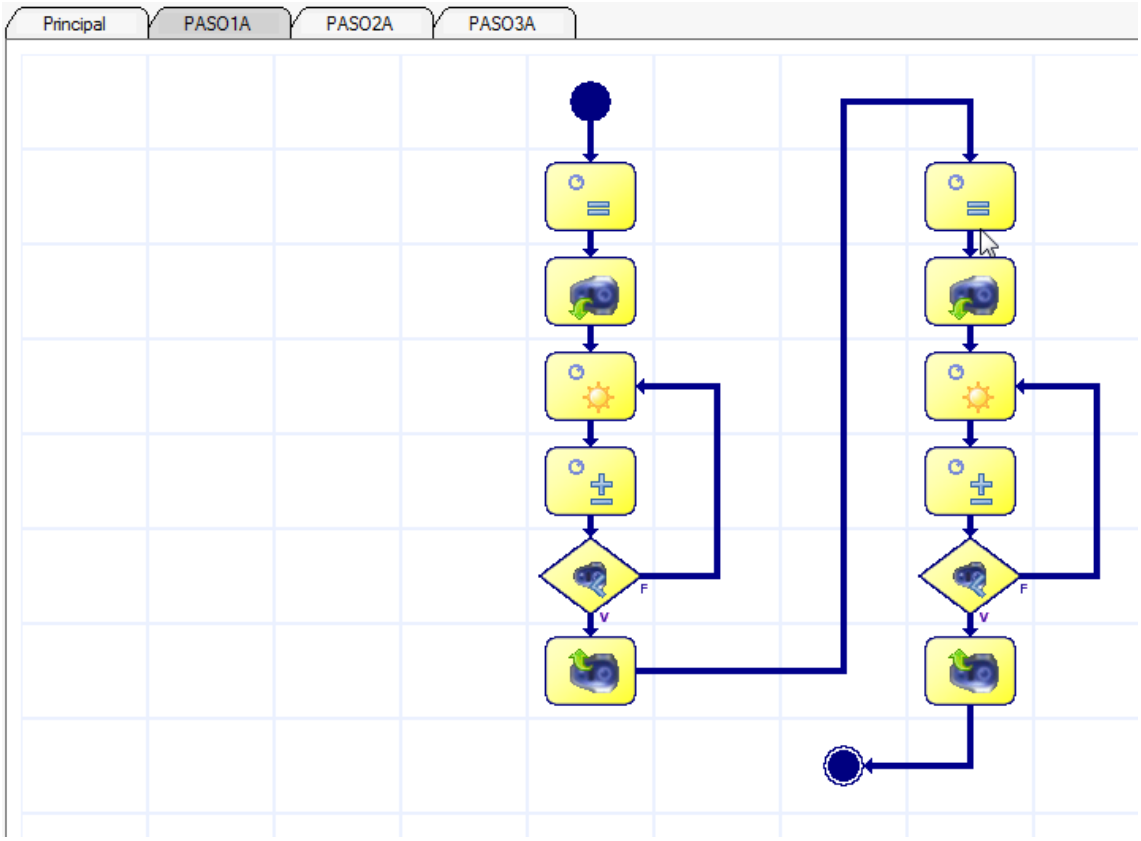
Volver al paso 11 y 12

Si la luminosidad no es menor que 10

21 Parar veinte segundos antes de comenzar a bailar e ir al paso 4.

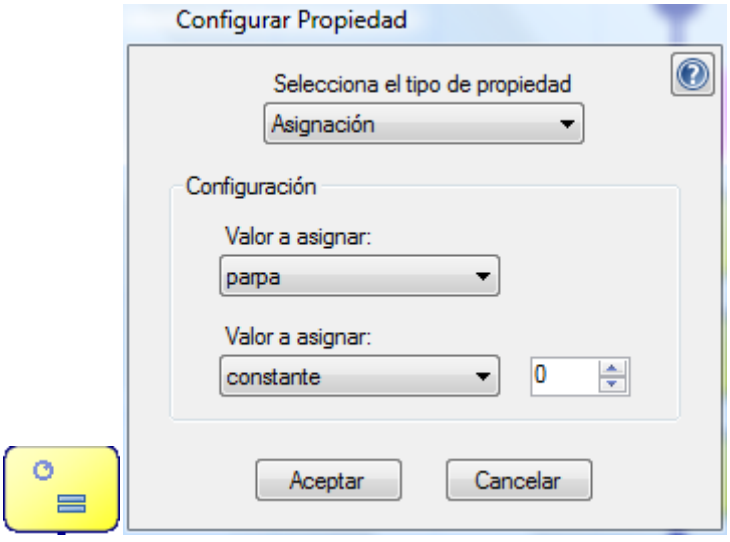


4.1.6.5.1.2 Paso 1A

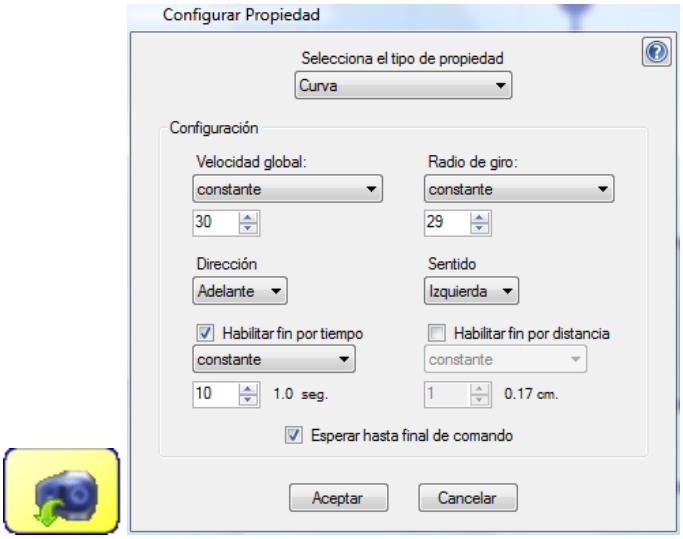


Pasos:

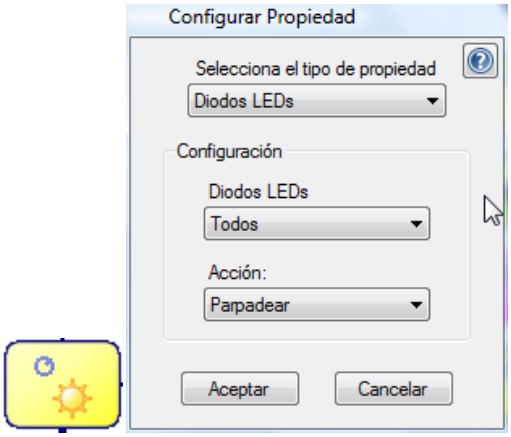
1 Inicializar parpa a cero.



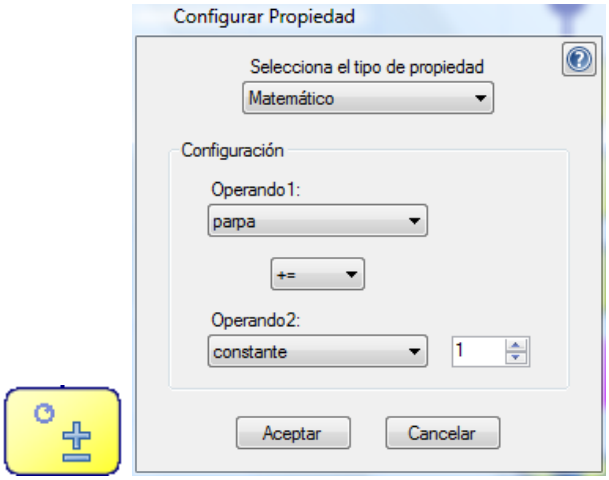
2 Realizar un movimiento curvilíneo hacía la izquierda y adelante.



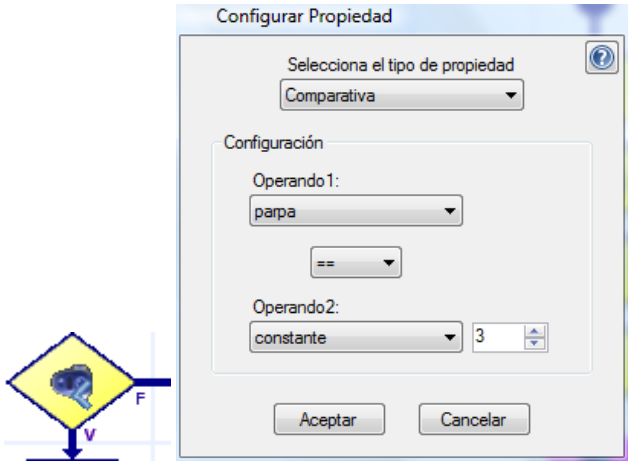
3 Parpadear todos los sensores del robot



4 Sumar uno a la variable parpa



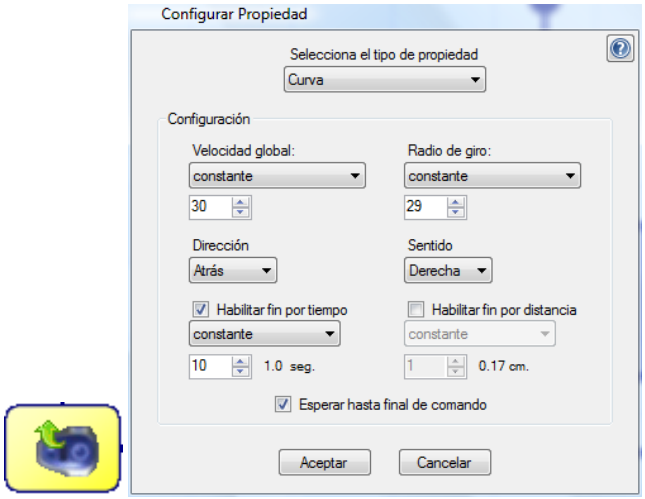
5 Comprobar si parpa es igual a tres.



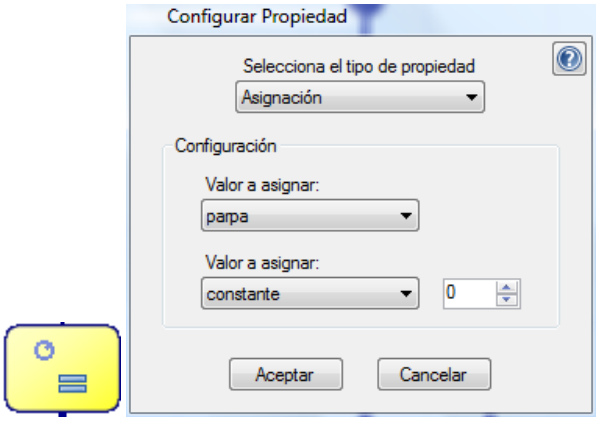
Si no es cierto volver al paso 3.

Si es cierto

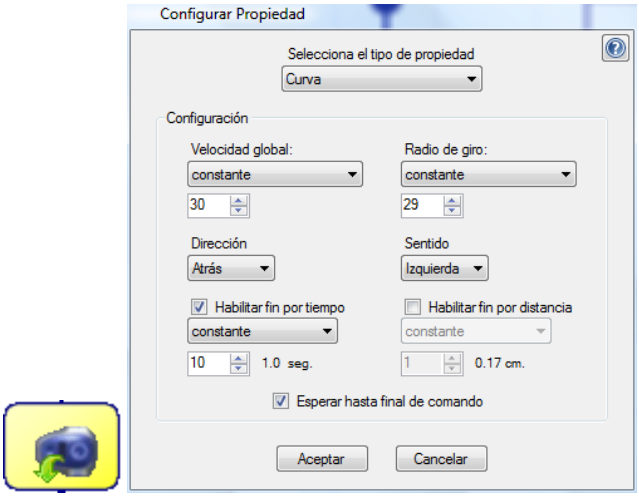
6 Realizar un movimiento curvilíneo hacía la derecha y atrás.



7 Inicializar parpa a cero.

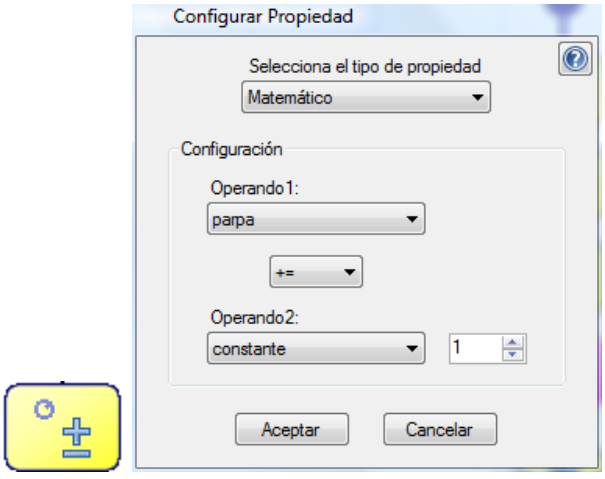


8 Realizar un movimiento curvilíneo hacía la izquierda y atrás.

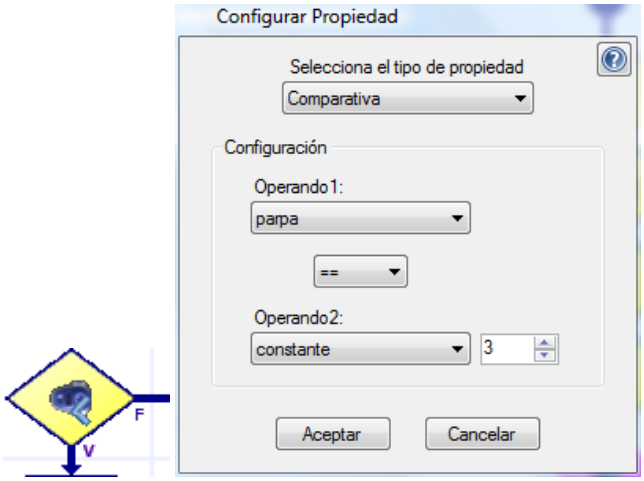


9 Parpadear todos los sensores del robot

10 Sumar uno a la variable parpa.



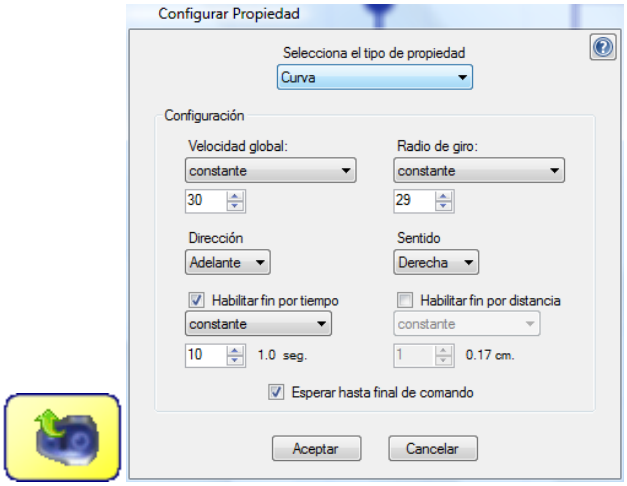
11 Comprobar si parpa es igual a tres



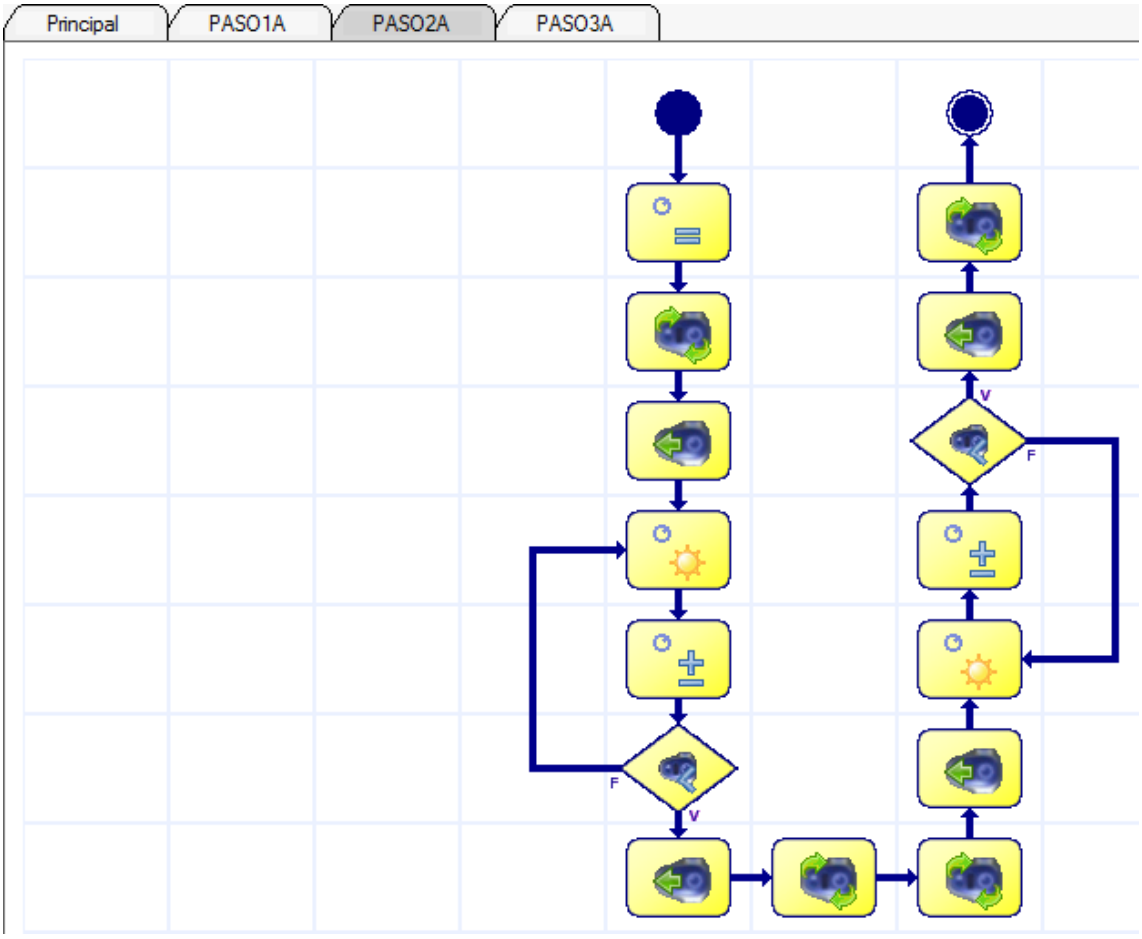
Si no es cierto volver al paso 9.

Si es cierto

12 Realizar un movimiento curvilíneo hacía la derecha y adelante.

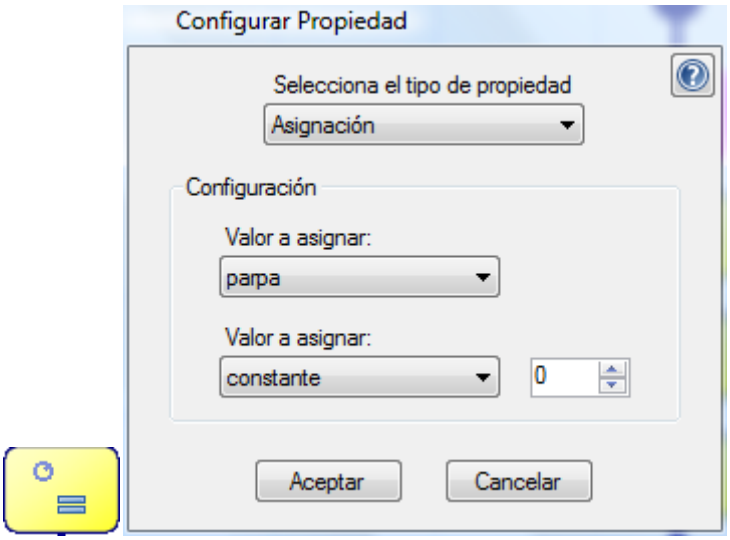


4.1.6.5.1.3 Paso 2A



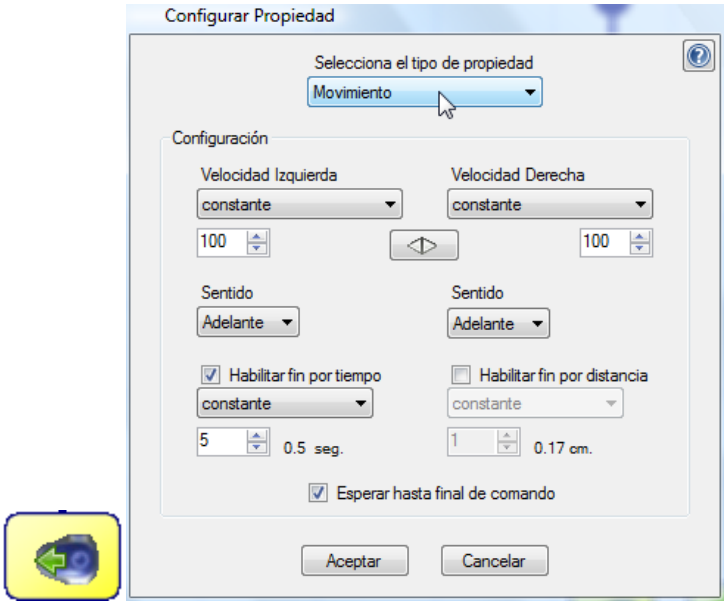
Pasos:

1 Inicializar parpa a cero.



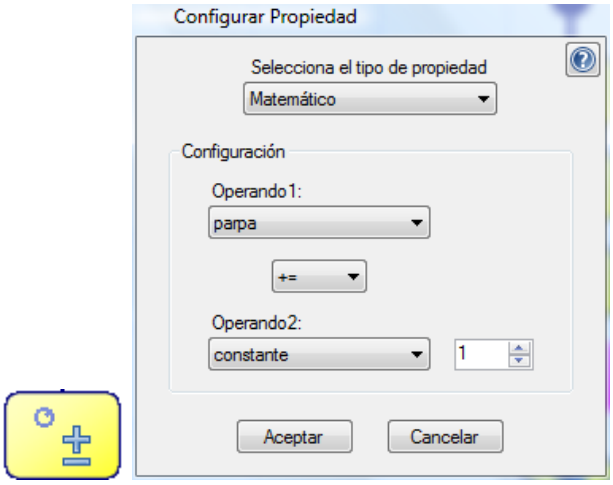
2 Realizar una rotación sobre sí mismo.

3 Realizar un movimiento rectilíneo hacia adelante.

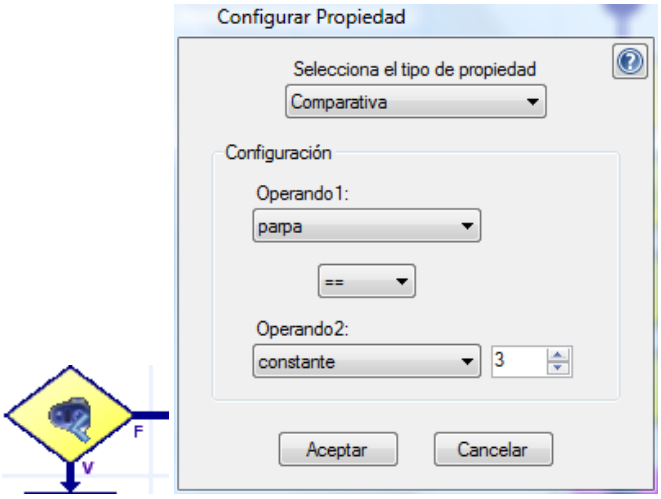


4 Parpadear todos los sensores del robot

5 Sumar uno a la variable parpa.



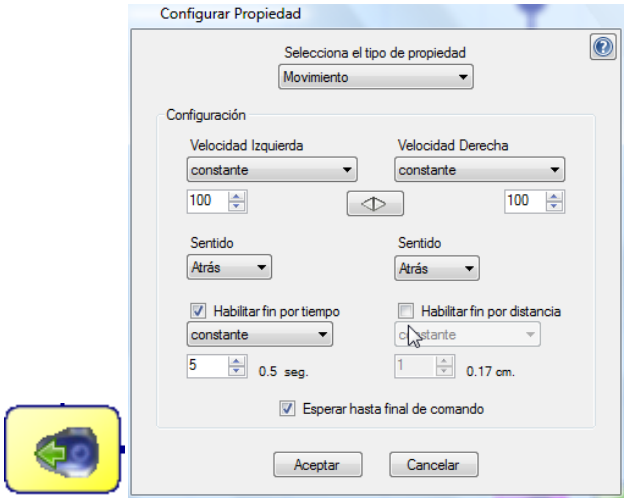
6 Comprobar si parpa es igual a tres



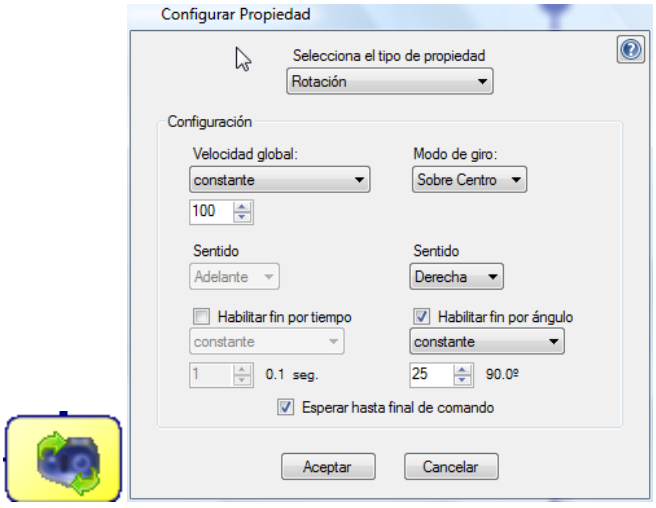
Si no es cierto volver al paso 4.

Si es cierto

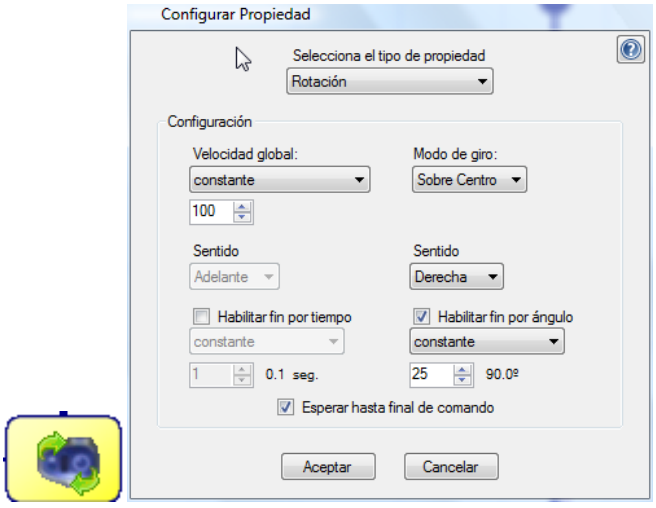
7 Realizar un movimiento rectilíneo hacia atrás.



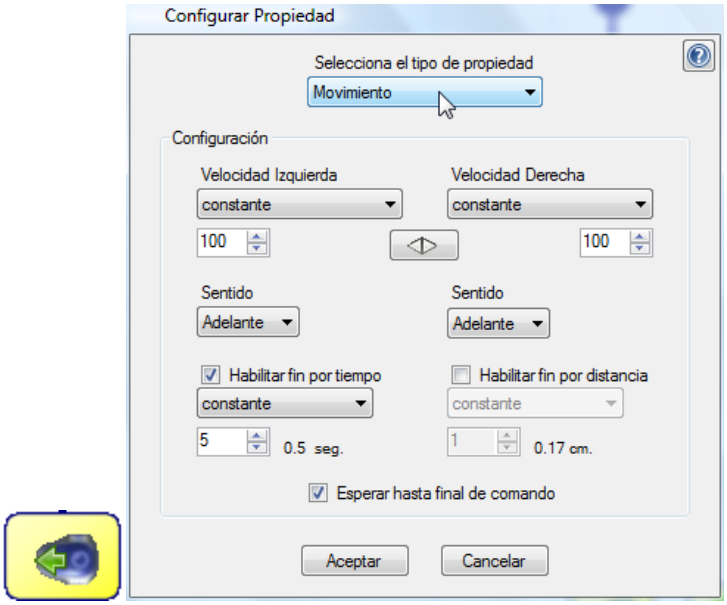
8 Realizar una rotación sobre sí mismo hacia la derecha.



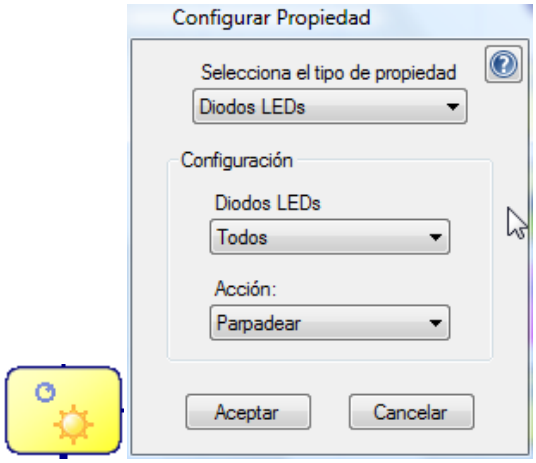
9 Realizar una rotación sobre sí mismo hacia la derecha.



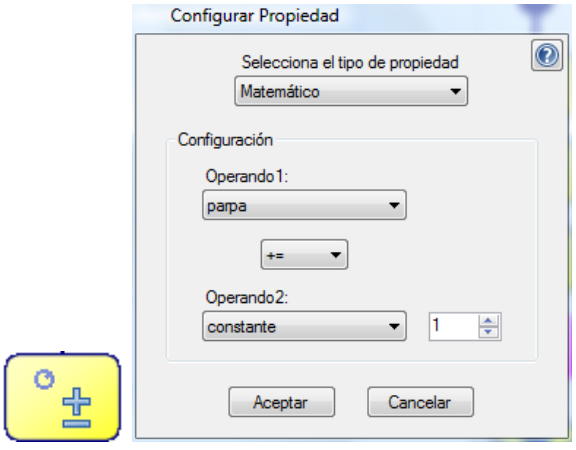
10 Realizar un movimiento rectilíneo hacia adelante.



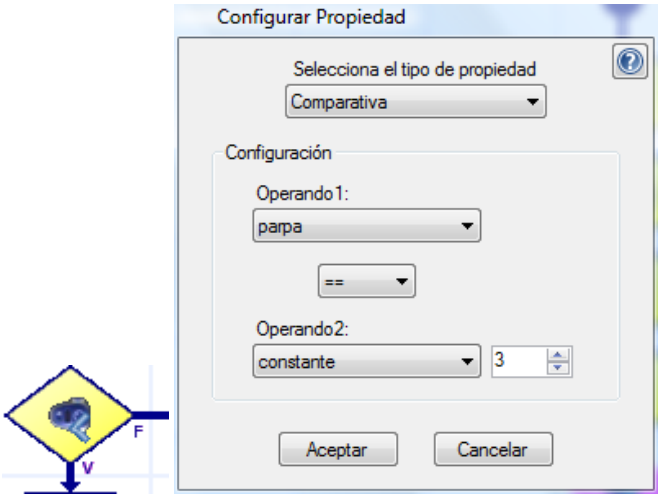
11 Parpadear todos los sensores del robot.



12 Sumar uno a la variable parpa.



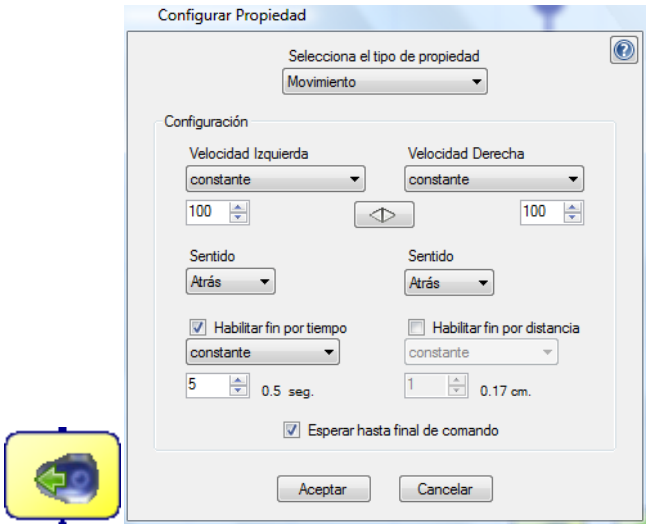
13 Comprobar si parpa es igual a tres



Si no es cierto volver al paso 11.

Si es cierto

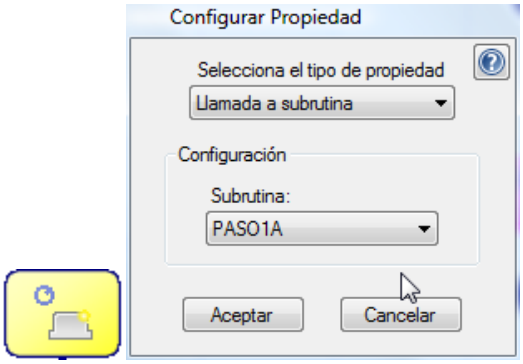
14 Realizar un movimiento rectilíneo hacia atrás.



15 Realizar una rotación sobre sí mismo hacia la izquierda.

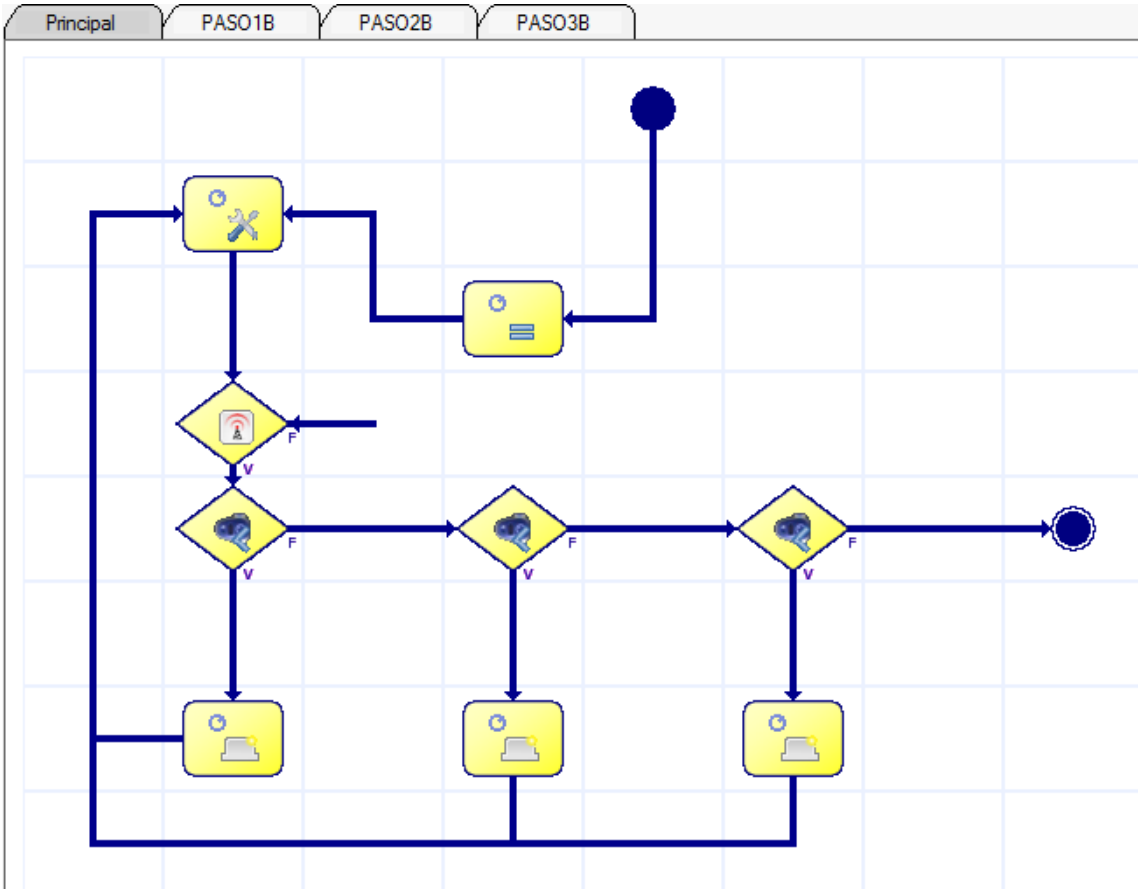
4.1.6.5.1.4 Paso 3A

Consta de un único paso, llamar a la subrutina Paso 1ª



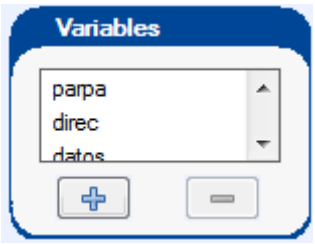
4.1.6.5.2 Implementación del código del robot secundario

4.1.6.5.2.1 Programa principal



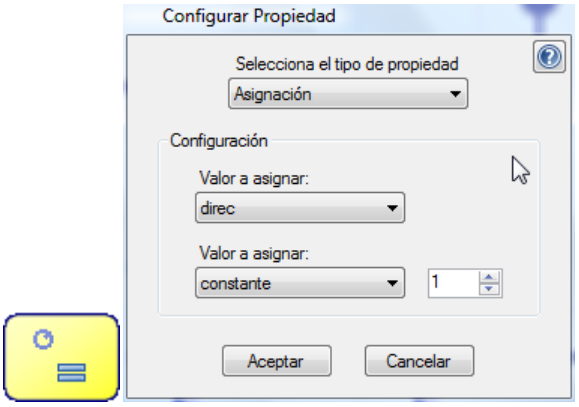
Variables

- ✓ Variable que almacenará la dirección del robot S, direc.
- ✓ Variable que almacenará el bloque de pasos que toca, datos.
- ✓ Variable que ayudará para que los sensores parpadeen tres veces, parpa

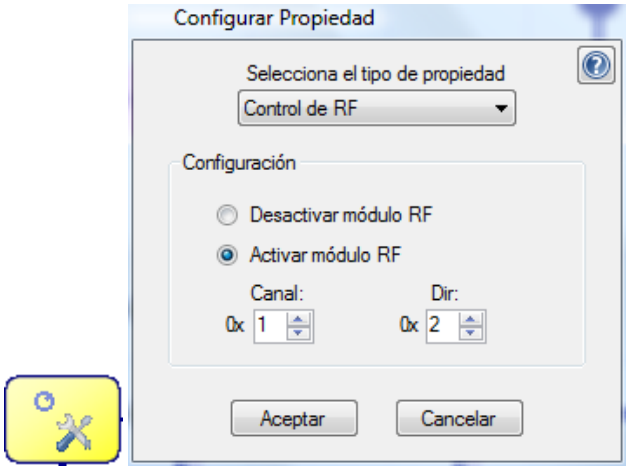


Pasos

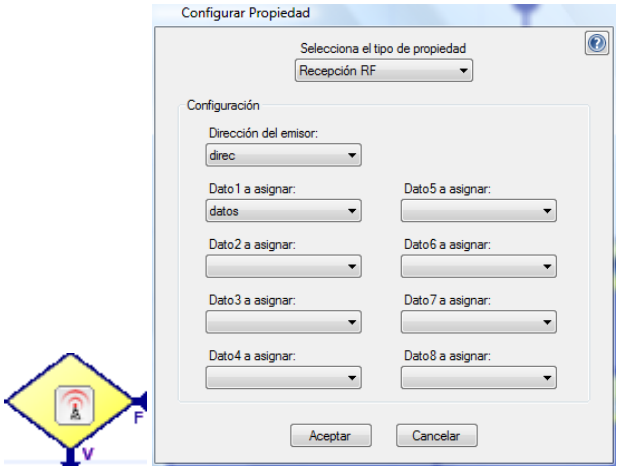
1 Inicializar direc a uno, dirección del Robot P.



2 Configurar el módulo RF, canal uno, dirección dos.



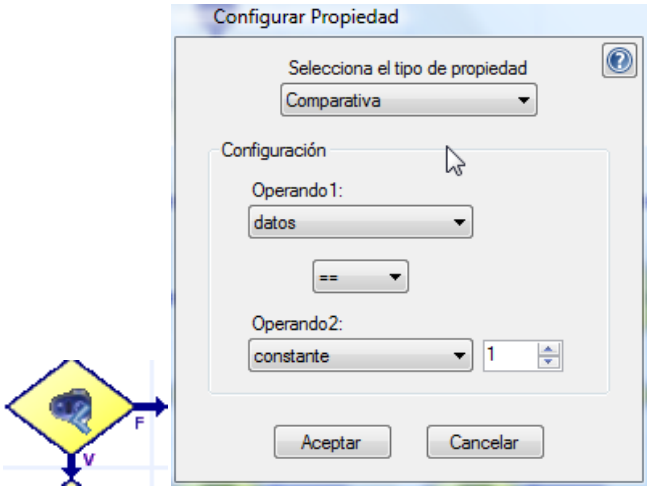
3 Recibir de la dirección direc, el dato 1 y asignarlo a datos.



Si el envío no ha sido correcto, seguir esperando a recibir.

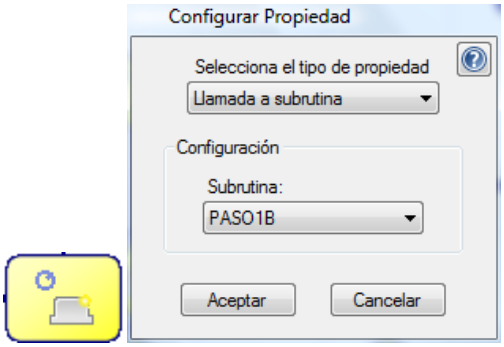
Si el envío ha sido correcto.

4 Comprobar si datos es igual a uno.



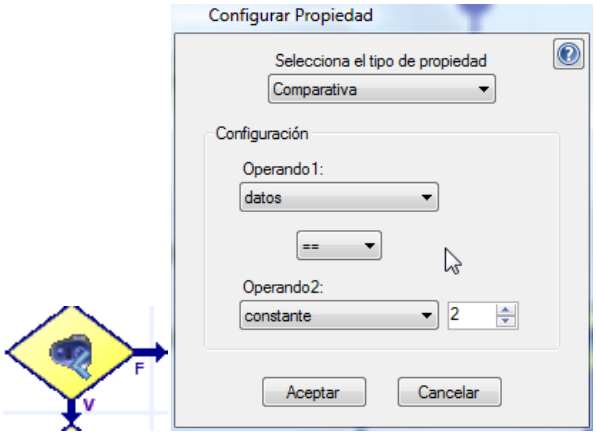
Si datos es igual a 1.

6 Bailar el bloque 1B y volver al paso 2.



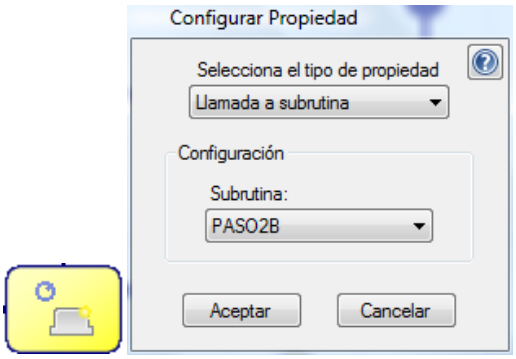
Si datos no es igual a 1.

7 Comprobar si datos es igual a 2



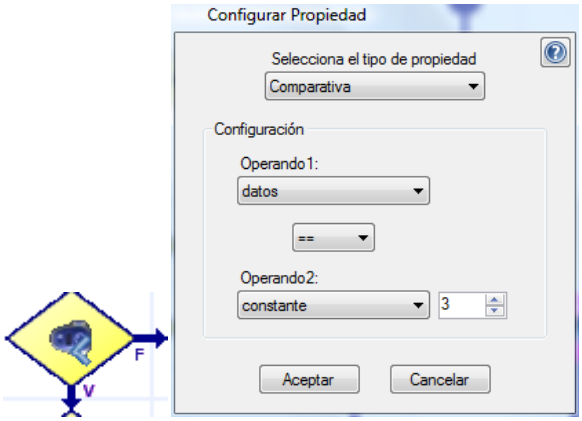
Si datos es igual a 2.

8 Bailar el bloque 2B y volver al paso 2.



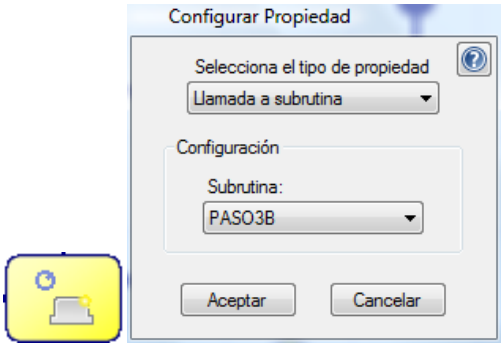
Si datos no es igual a 2.

9 Comprobar si datos es igual a 3.



Si datos es igual a 3.

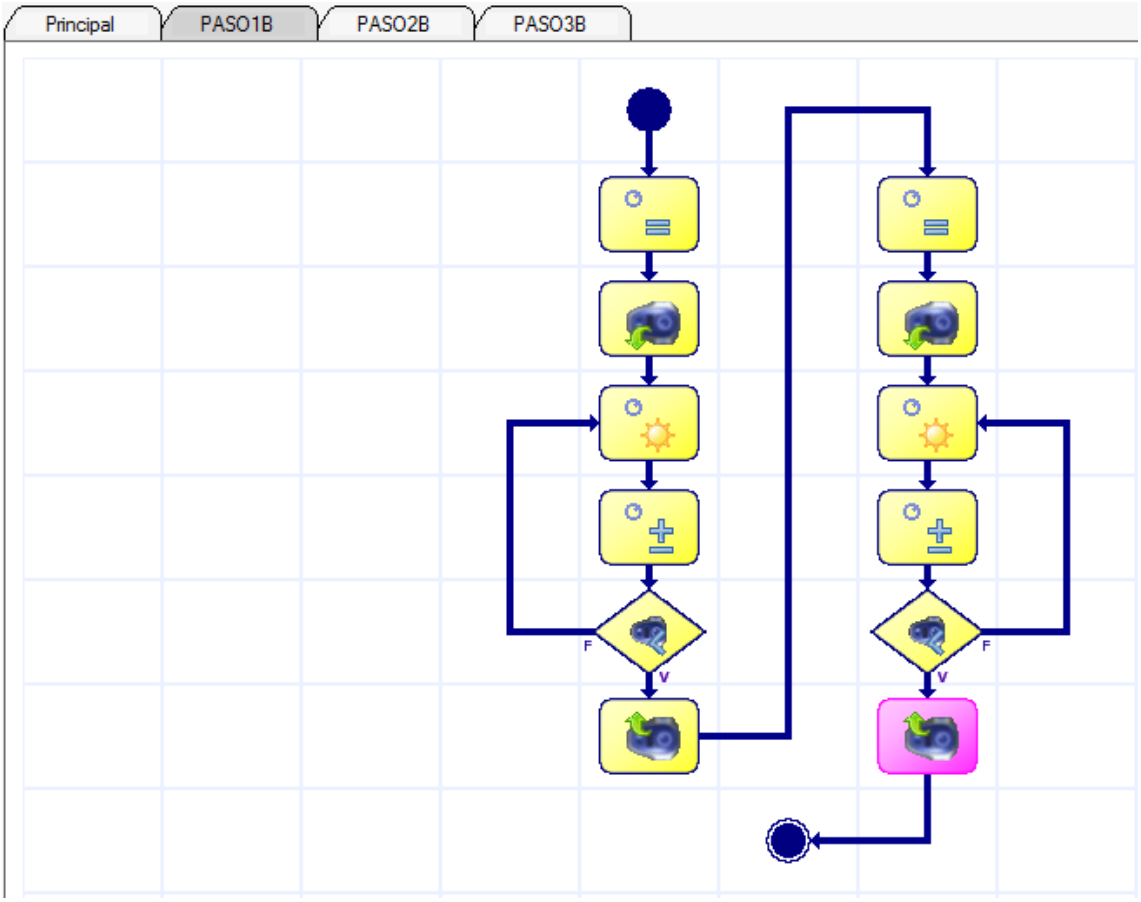
10 Bailar el bloque 3B y volver al paso 3.



Si datos no es igual a 3.

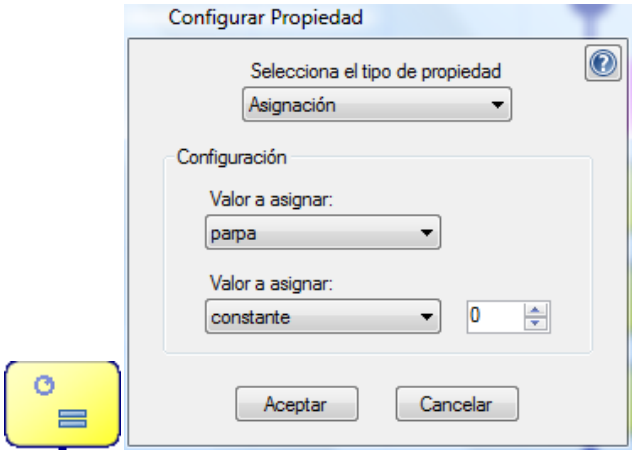
11 Terminar.

4.1.6.5.2.2 Paso 1B

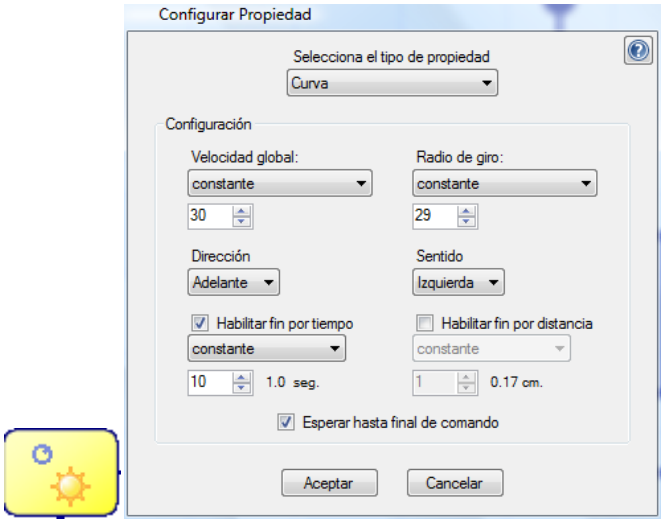


Pasos:

1 Inicializar parpa a cero.

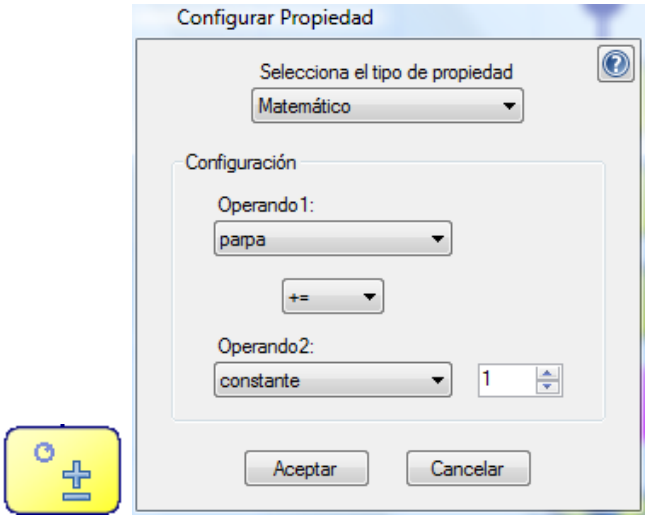


2 Realizar un movimiento curvilíneo hacía la izquierda y adelante.

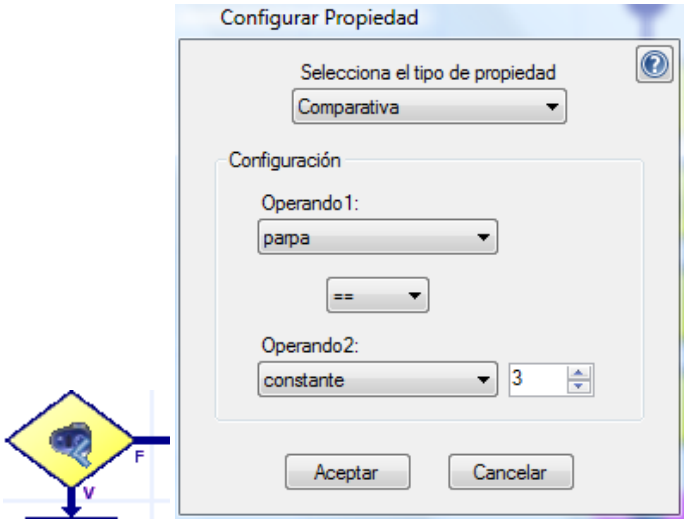


3 Parpadear todos los sensores del robot

4 Sumar uno a la variable parpa



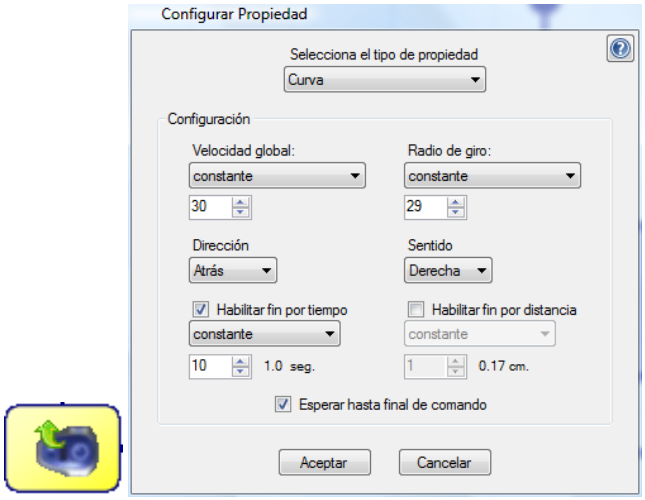
5 Comprobar si parpa es igual a tres.



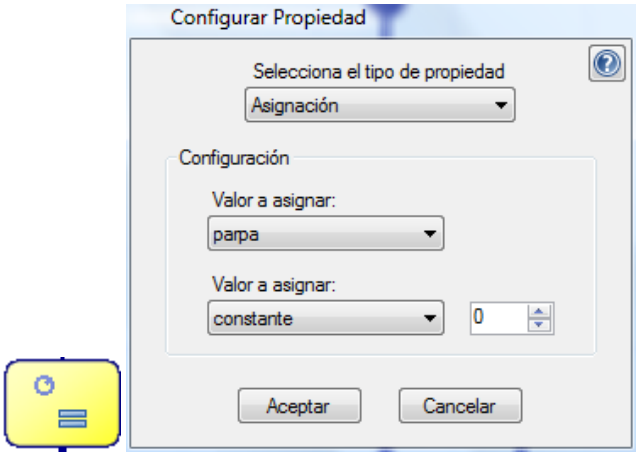
Si no es cierto volver al paso 3.

Si es cierto

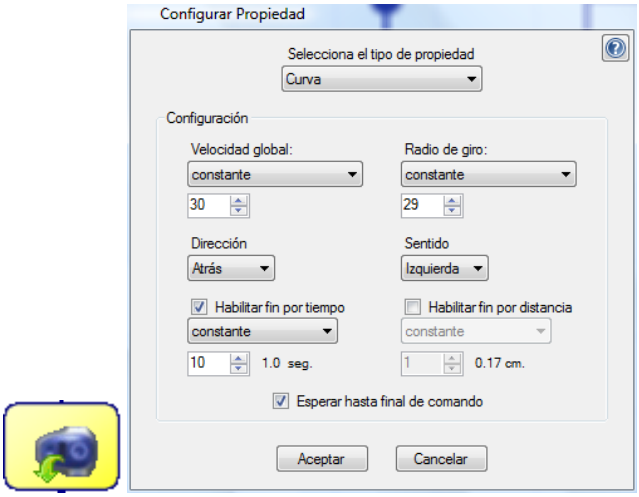
6 Realizar un movimiento curvilíneo hacía la derecha y atrás.



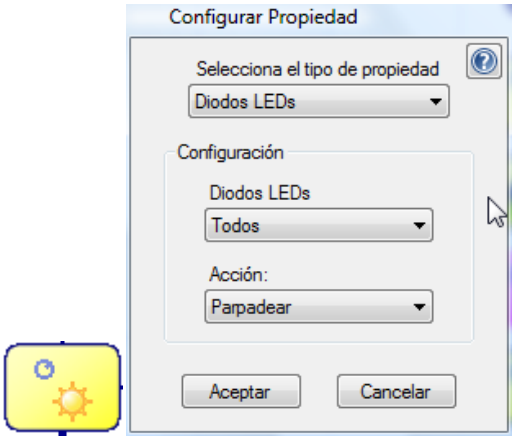
7 Inicializar parpa a cero.



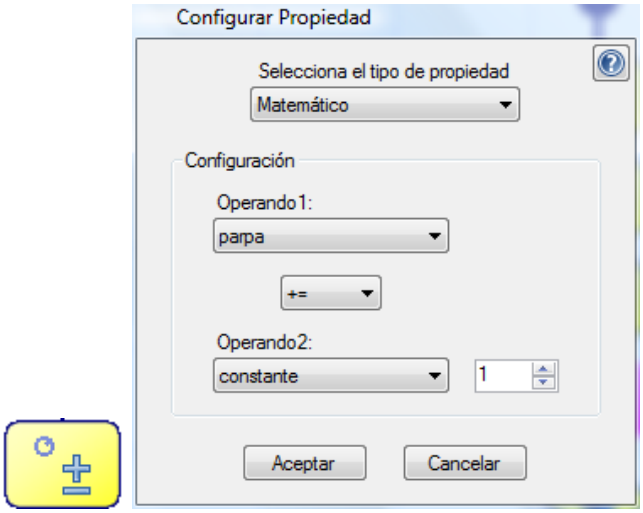
8 Realizar un movimiento curvilíneo hacía la izquierda y atrás.



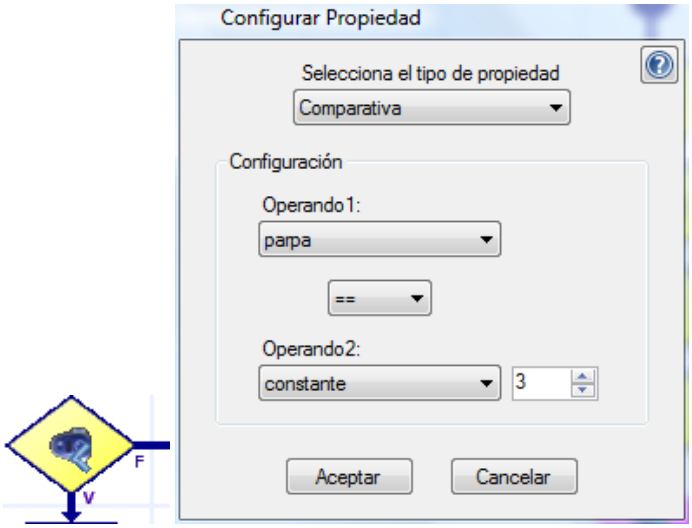
9 Parpadear todos los sensores del robot



10 Sumar uno a la variable parpa.



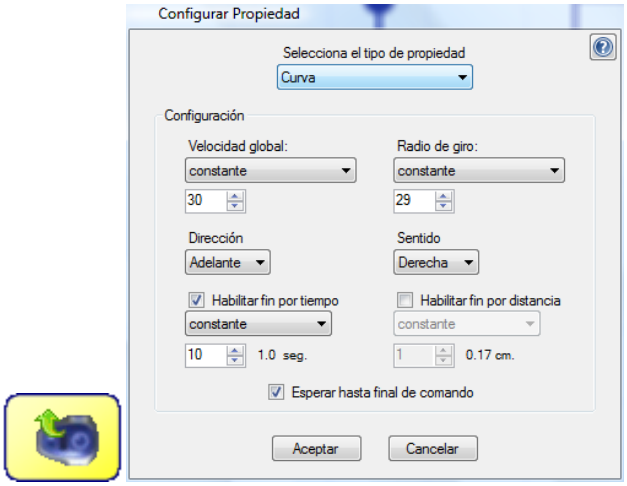
11 Comprobar si parpa es igual a tres



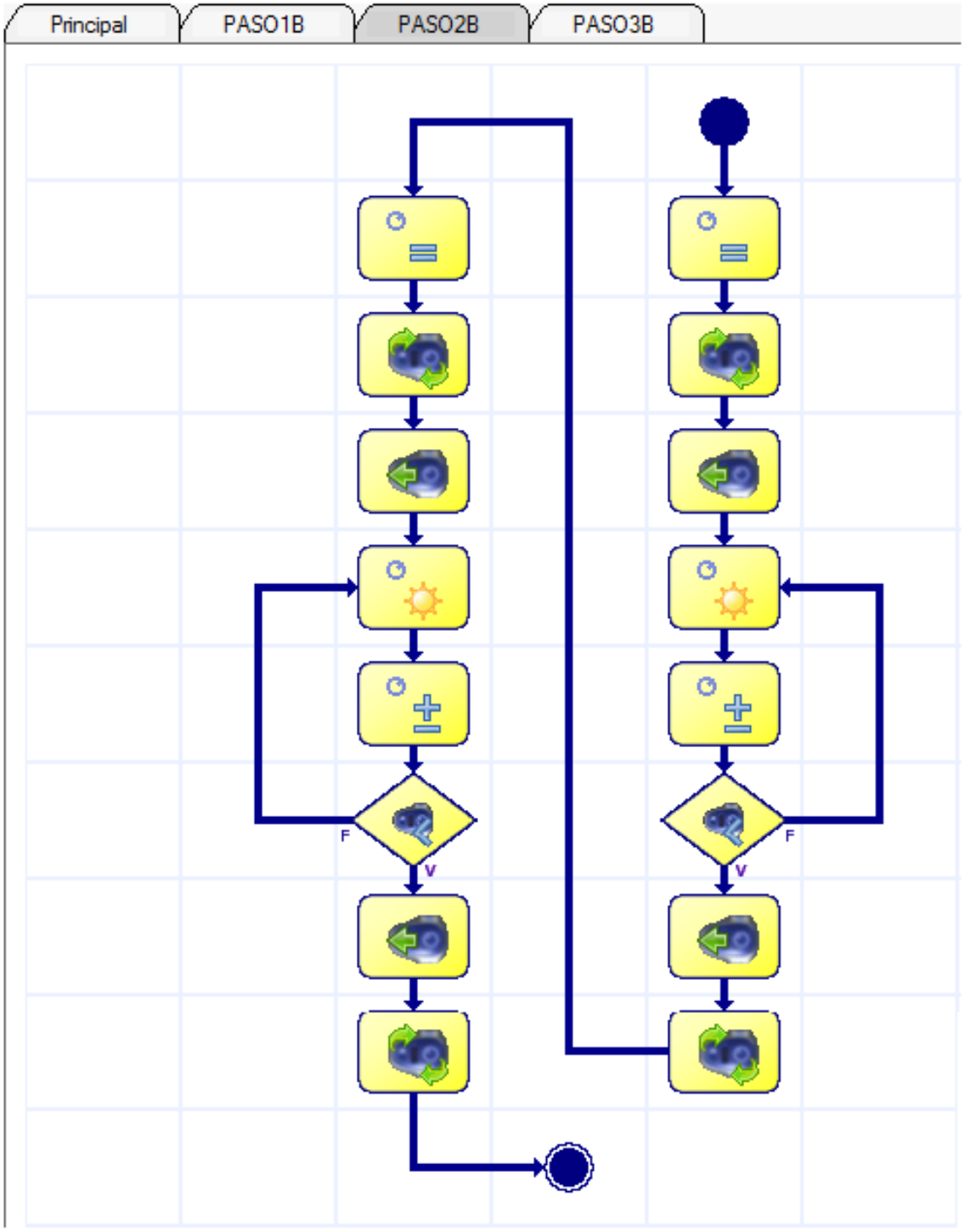
Si no es cierto volver al paso 9.

Si es cierto

12 Realizar un movimiento curvilíneo hacía la derecha y adelante.

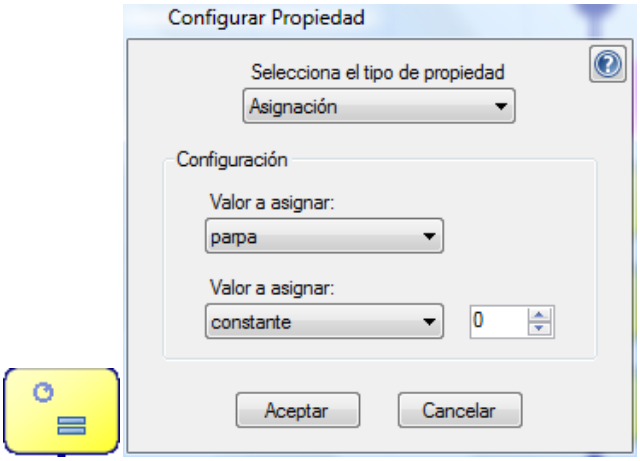


4.1.6.5.2.3 Paso 2B

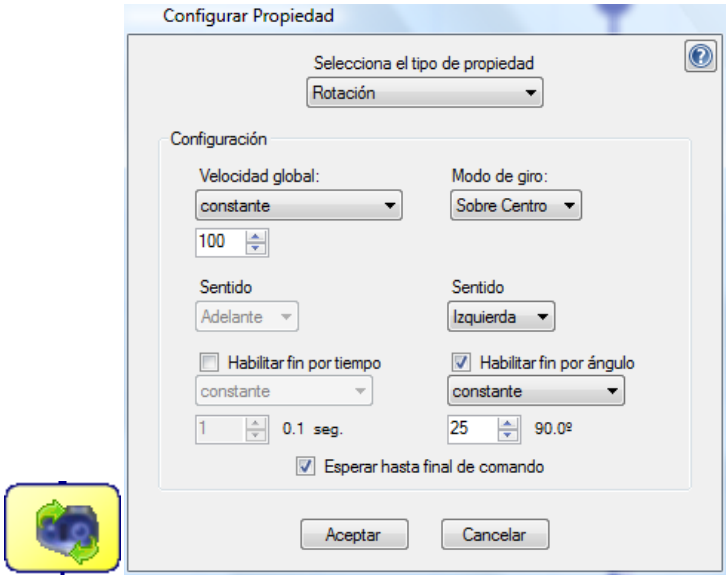


Pasos:

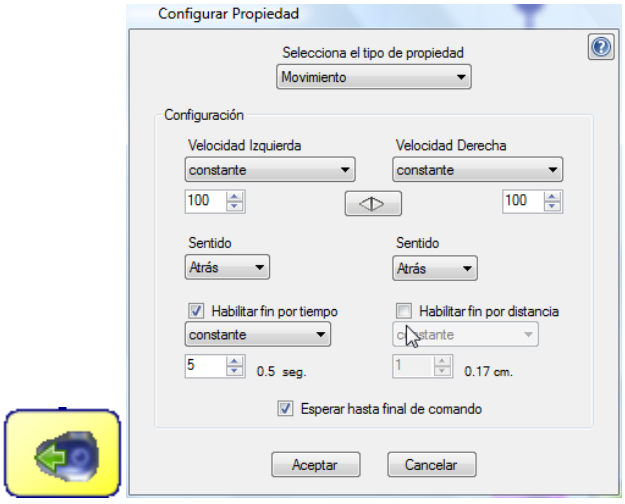
1 Inicializar parpa a cero.



2 Realizar una rotación sobre sí mismo.

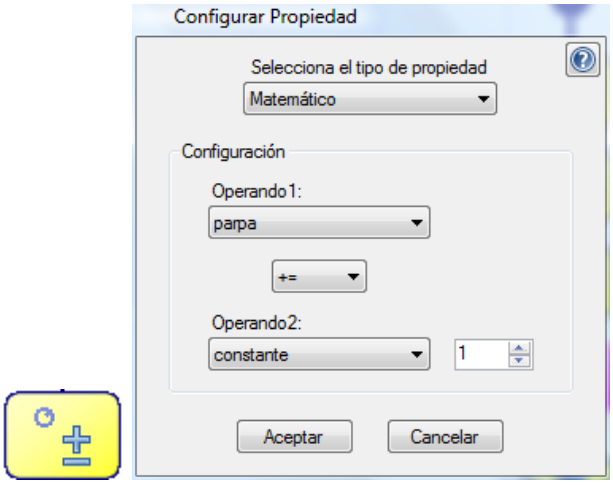


3 Realizar un movimiento rectilíneo hacia atrás.

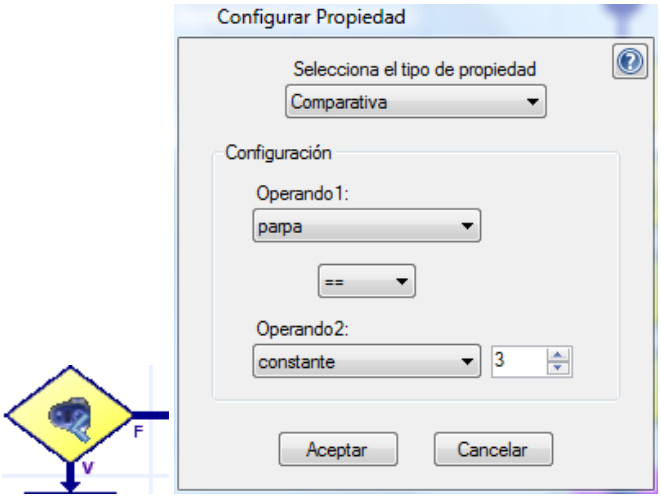


4 Parpadear todos los sensores del robot

5 Sumar uno a la variable parpa.



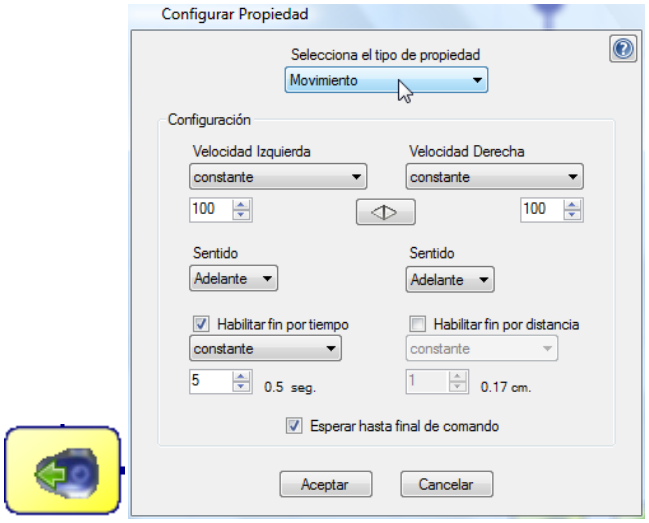
6 Comprobar si parpa es igual a tres



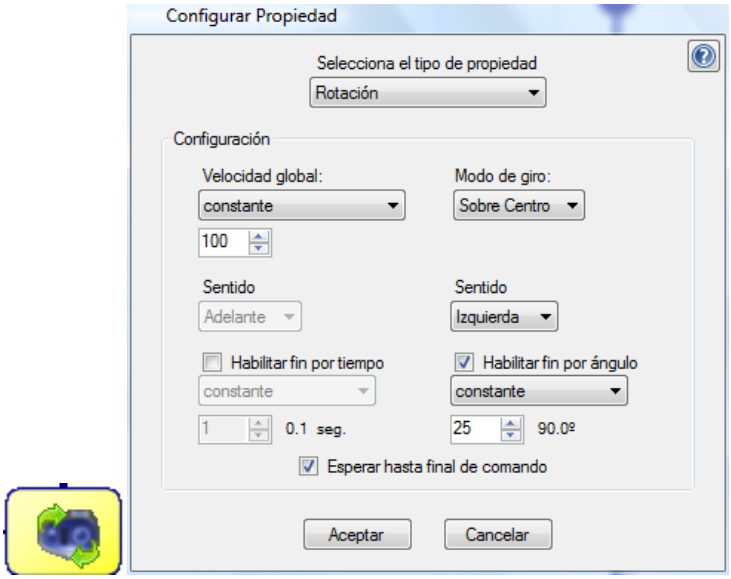
Si no es cierto volver al paso 4.

Si es cierto

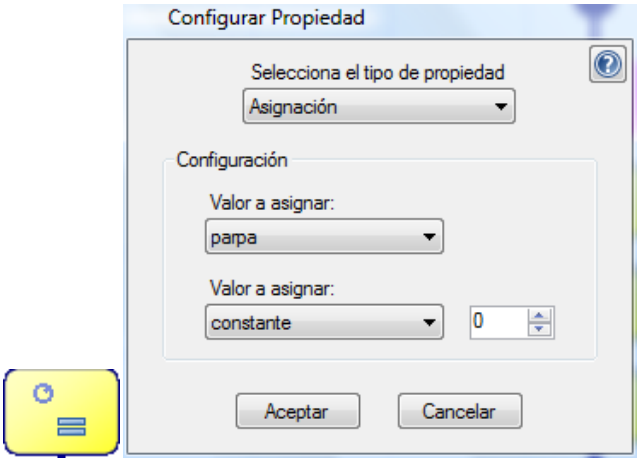
7 Realizar un movimiento rectilíneo hacia adelante.



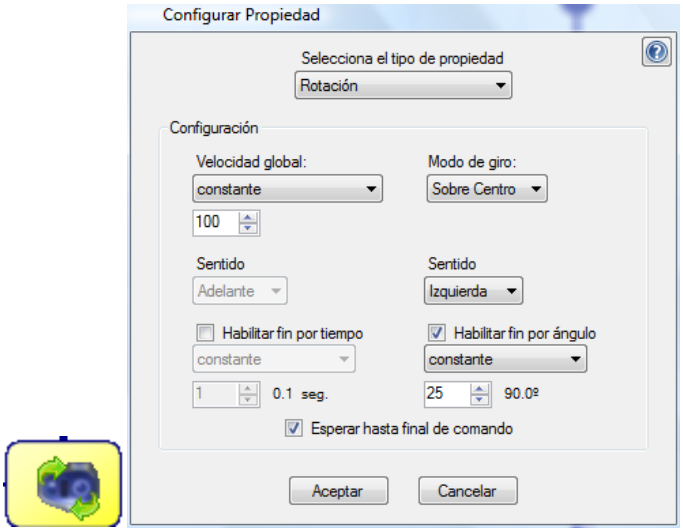
8 Realizar una rotación sobre sí mismo hacia la izquierda.



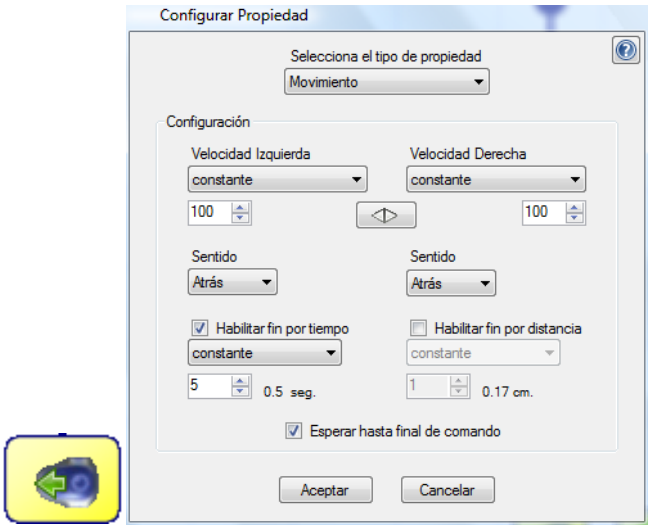
9 Inicializar parpa a cero.



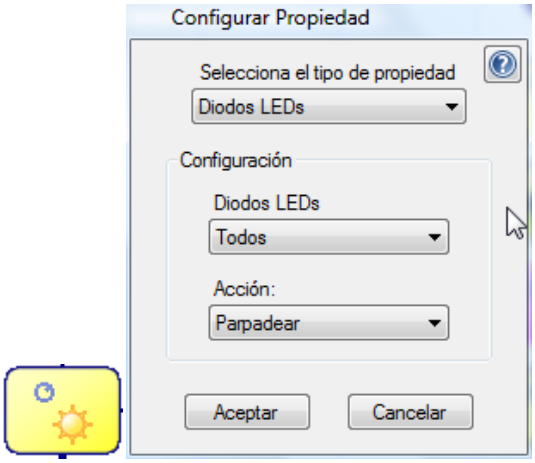
10 Realizar una rotación sobre sí mismo hacia la izquierda.



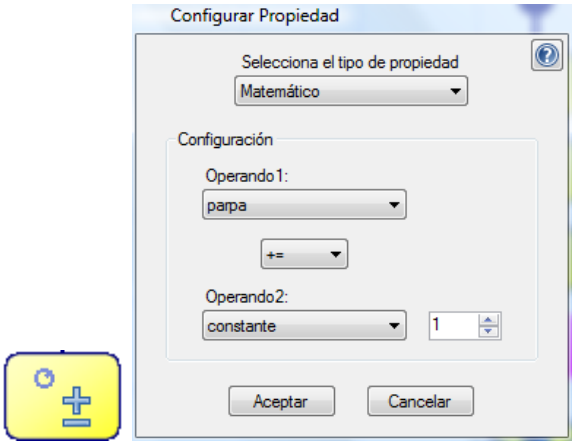
11 Realizar un movimiento rectilíneo hacia atrás.



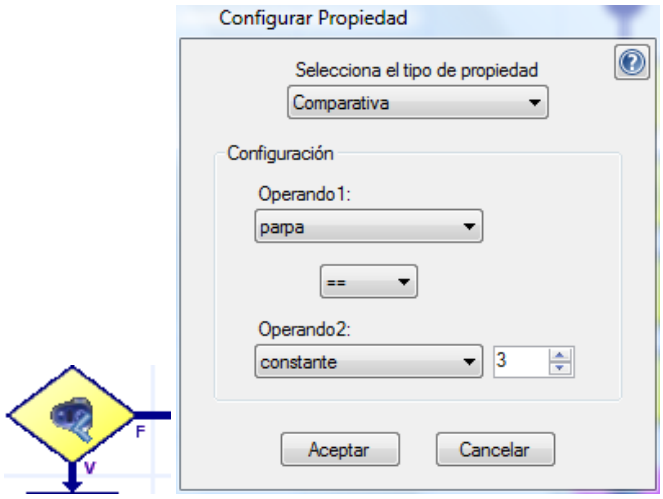
11 Parpadear todos los sensores del robot.



12 Sumar uno a la variable parpa.



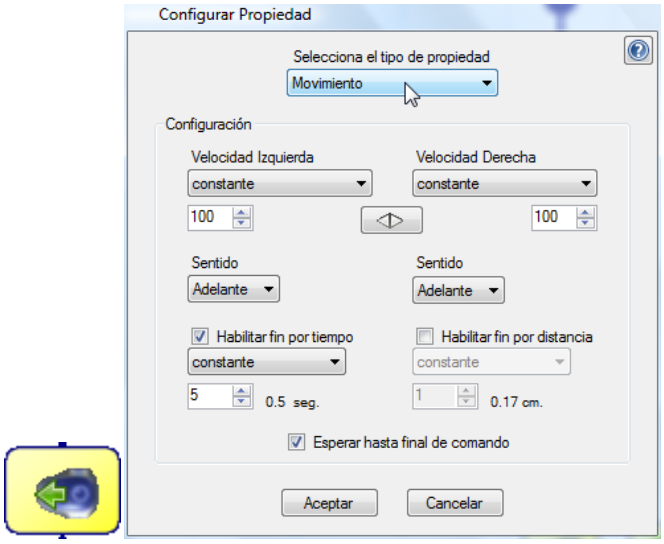
13 Comprobar si parpa es igual a tres



Si no es cierto volver al paso 11.

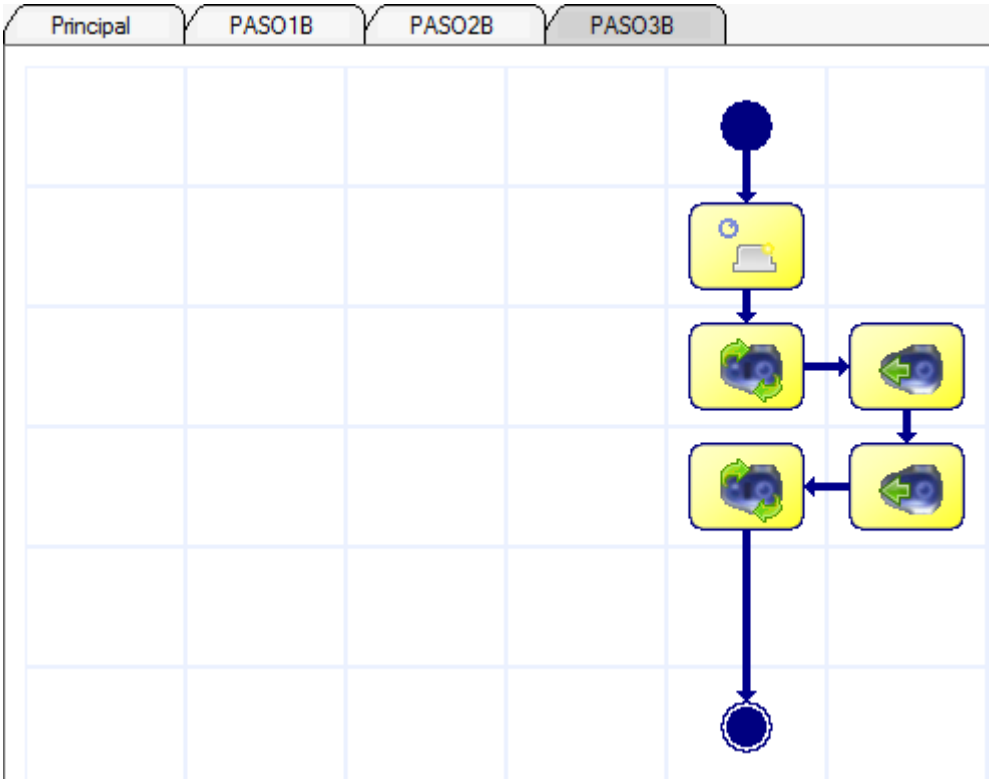
Si es cierto

14 Realizar un movimiento rectilíneo hacia adelante.



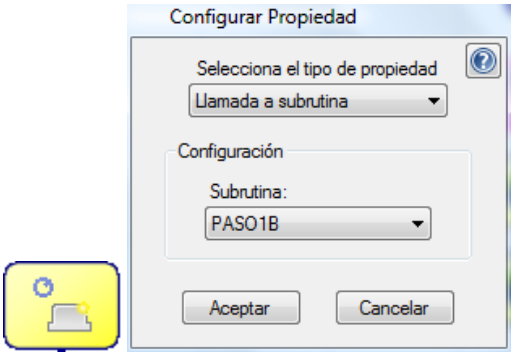
15 Realizar una rotación sobre sí mismo hacia la derecha.

4.1.6.5.2.4 Paso 3B

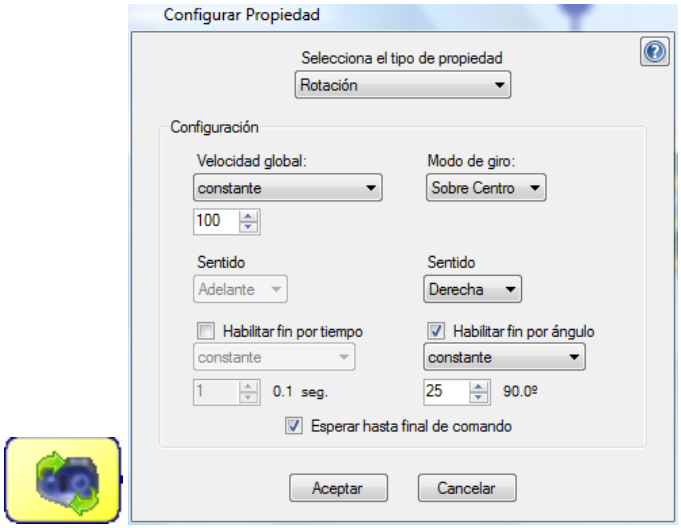


Pasos:

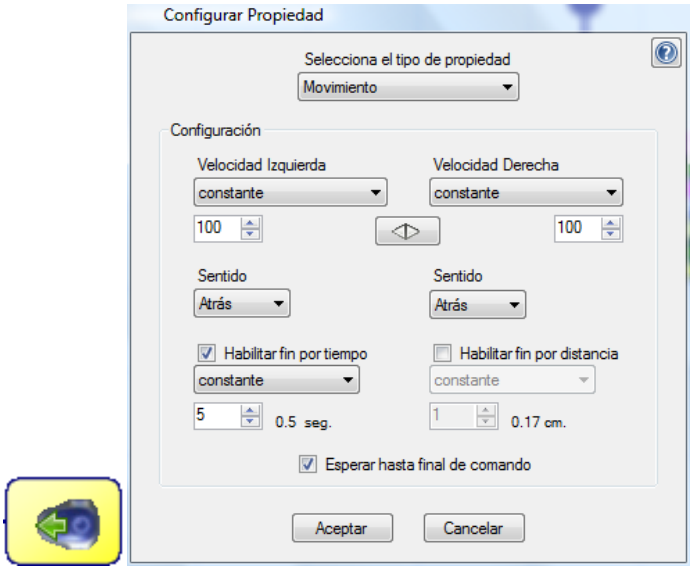
1 Bailar el Bloque 2B



2 Realizar una rotación sobre sí mismo hacía la derecha

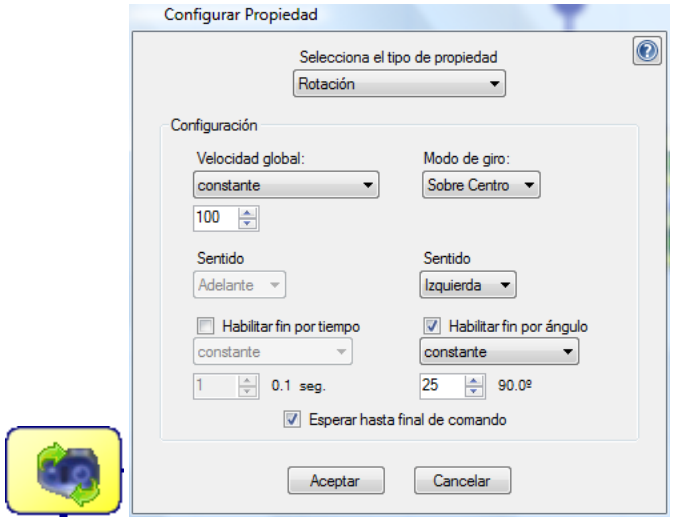


3 Realizar un movimiento rectilíneo hacía atrás.



4 Realizar un movimiento rectilíneo hacía adelante.

5 Realizar una rotación sobre sí mismo hacía la izquierda.



4.2 LEGO MINDSTORMS NXT

LEGO Mindstorms es un juego de robótica para niños fabricado por la empresa LEGO, el cual posee elementos básicos de las teorías robóticas, como la unión de piezas y la programación de acciones, en forma interactiva. Este robot fue comercializado por primera vez en septiembre de 1998.

Los productos LEGO están siempre orientados a los niños, este producto se diseño orientado a los niños entre 10 y 14 años, de ahí el diseño del producto. Se trata de un producto robusto y asequible, en el que los niños pueden montar sus propios robots fácilmente. Sin embargo, hoy en día, este producto es utilizado tanto por niños como en centros universitarios, ya que es una herramienta muy útil para la enseñanza de programación y electrónica.

4.2.1 LEGO

LEGO es una empresa de juguetes danesa, conocida mundialmente por sus bloques de plástico. Su nombre viene de la frase del danés “leg godt”, que significa “juega bien”. Hasta 1949, LEGO, se dedicó casi exclusivamente a los juguetes de madera, más tarde se pasó al plástico y comenzó en el desarrollo y venta de sus famosos bloques.

En la década de 1960 se empezó a introducir los bloques de plástico como herramienta educativa, ya que se consideraban una gran ayuda para los educadores gracias a su capacidad de desarrollar las habilidades creativas y resolución de problemas en niños. En 1980 LEGO, inauguró su departamento de productos educativos para expandir sus productos educativos.

Desde sus comienzos en la producción de ladrillos de plástico, LEGO, ha lanzado miles de juegos con distintos motivos, nuevas piezas son lanzadas constantemente, aumentando cada vez la versatilidad de los juguetes LEGO. Existen piezas que pueden ser programadas con un ordenador personal para desempeñar procedimientos completos, que se venden bajo el nombre LEGO Mindstorms, con el que se va trabajar en este proyecto.

4.2.2 KIT DE LEGO

Esta herramienta nació en tiempos complicados para LEGO. La empresa firmó un acuerdo con el grupo de epistemología y aprendizaje del MIT, LEGO financiaba las investigaciones de dicho grupo sobre cómo aprenden los niños y a cambio podría lanzar al mercado nuevos productos con sus ideas sin tener que pagar regalías al MIT.

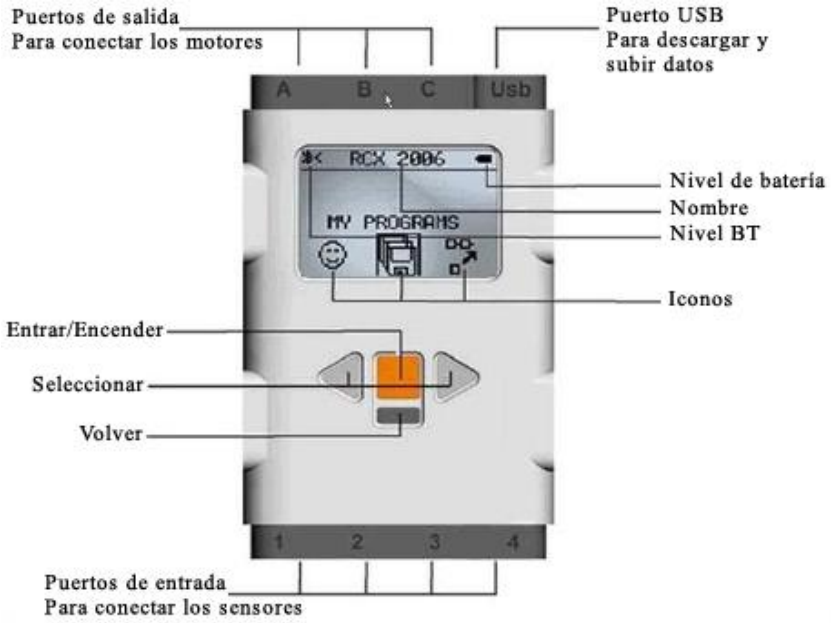
Seymeour Papert, mentor del grupo del MIT, era un matemático interesado en estudiar la relación entre la ciencia, la adquisición del conocimiento y el desarrollo de la mente infantil. De él surge el nombre, Mindstorms, que proviene de un libro suyo MindStorms: Children, Computers and Powerful Ideas.

Antes del LEGO Mindstorms, hubo otros productos, así en 1986 LEGO comercializó el Lego TC Logo, tuvo éxito, pero era muy limitado. En 1993 se lanzó el producto Control Lab, más cercano a LEGO Mindstorms.

LEGO Mindstorms propone una solución ideal para la enseñanza de la robótica en todos niveles educativos, el hardware, software y recursos, están hechos a la medida para su uso en las aulas. Los estudiantes aprenden a diseñar, programar y controlar totalmente las funciones de los robots que llevan a cabo tareas automatizadas.

4.2.2.1 BLOQUE NXT

Es el cerebro del robot Mindstorms. Es el ladrillo controlador inteligente de LEGO que permite al robot Mindstorms cobrar vida y ejecutar diferentes operaciones.



4.2.2.1.1.1 Microcontrolador

El microcontrolador que posee es un ARM7 de 32 bits, que incluye 256 Kb de memoria Flash y 64 Kb de RAM externa, la cual a diferencia del bloque RCX, posee mayores capacidades de ejecución de programas, evitando que los procesos inherentes de varios paquetes de datos colisionen y produzcan errores y un posible error en la ejecución del software. Su presentación es similar al Hitachi H8 ya que se encuentra en el circuito impreso del bloque, junto a la memoria FLASH.

4.2.2.1.1.2 Puertos de salida

En la parte superior del bloque NXT se encuentran los puertos de salida, hay tres, los puertos A, B y C. Estos puertos permiten la comunicación con los motores y partes móviles.

4.2.2.1.1.3 Puertos para sensores

El bloque NXT tiene cuatro puertos de entrada, los puertos 1, 2, 3 y 4, permiten la comunicación con los sensores. Se encuentran en la parte inferior del bloque.

4.2.2.1.1.4 Puerto USB

El puerto USB permite la comunicación entre el robot y un ordenador a través de un cable USB. Se utiliza para descargar los programas del ordenador al NXT (o para cargar los datos del robot al ordenador). También se puede usar la conexión de Bluetooth para cargar y descargar.

4.2.2.1.1.5 Otros elementos del bloque

A través del visualizador NXT y los botones se puede navegar por el menú del bloque NXT, a través de él se pueden configurar algunos parámetros, navegar por los programas grabados en el robot, ejecutarlos o borrarlos.

Por último, el bloque NXT cuenta con un altavoz, que permite emitir sonidos durante la ejecución de un programa.

4.2.2.2 SENSOR DE CONTACTO

El sensor de contacto permite detectar si el bloque que lo posee ha colisionado o no con algún objeto que se encuentre en su trayectoria inmediata. Al tocar una superficie, una pequeña cabeza externa se contrae, permitiendo que una pieza dentro del bloque cierre un circuito eléctrico. En este caso, si la presión supera una medida estándar de 450, se considera que el sensor está presionado, de otro modo, se considera que está sin presión.

4.2.2.3 SENSOR DE SONIDO

El sensor de sonido puede detectar los decibelios [dB] y el decibelio A [dBA] a la vez.

El dBA: detectando el decibelio A, la sensibilidad del sensor se adapta a la sensibilidad del oído humano. En otras palabras, a los sonidos que los oídos humanos pueden escuchar.

Los dB: detectando los decibelios estándar, todos los sonidos son medidos con la misma sensibilidad. De este modo, hay sonidos que pueden ser incluidos y que sean demasiado altos o bajos para que el oído humano los pueda oír.

El sensor de sonido puede medir la presión del sonido en niveles superiores a 90 dB. El nivel de la presión del sonido es extremadamente complicado, así que el Mindstorms NXT visualiza el porcentaje de la presión del sonido leída por el sensor.

- ✓ 4-5 % es como una sala en silencio.
- ✓ 5-10 % podría ser que alguien está diciendo algo a distancia.
- ✓ 10-30 % es una conversación normal cerrada con el sensor o un reproductor de música a ni volumen normal.
- ✓ 30-100% mucha gente gritando o música reproducida a un volumen alto.

4.2.2.4 SENSOR DE LUZ

El sensor de luz es uno de los dos sensores que dan al robot la visión (el sensor de ultrasonidos es el otro). Este sensor permite al robot distinguir entre claro y oscuro. Puede leer la intensidad de la luz en una habitación y medir la intensidad del colorido de las superficies.

El sensor de luz permite tomar una muestra de luz mediante un bloque modificado que en un extremo trae un conductor eléctrico y por el otro una cámara oscura que capta luces entre los rangos 0,6 a 760lux. Este valor lo considera como un porcentaje, el cual es procesado por el bloque lógico, obteniendo un porcentaje aproximado de luminosidad.

4.2.2.5 SENSOR DE ULTRASONIDOS

El sensor de ultrasonidos es el otro sensor que da al robot la visión, éste permite al robot ver y detectar objetos. También permite con su uso que el robot evite obstáculos, permite medir y tener la sensación de la distancia y detectar movimientos.

El sensor de ultrasonidos mide la distancia en centímetros y en pulgadas. Este sensor es capaz de detectar objetos que se encuentren desde 0 a 255 cm, con una precisión relativa de +/- 3 cm.

Mediante el principio del eco, el sensor es capaz de recibir la información de los distintos objetos que se encuentren en el campo de detección. El sensor funciona mejor cuando las señales ultrasónicas provienen de objetos grandes, planos o superficies duras. Los objetos pequeños, curvos o suaves, como pelotas, pueden ser muy difíciles de detectar.

4.2.2.6 OTROS SENSORES

Los sensores que se han descrito son los sensores que incluyen el kit de LEGO Mindstorms pero existen muchos más sensores que se pueden adquirir por separado. A continuación se citan los más conocidos:

- ✓ **Sensor buscador de infrarrojos**, permite detectar fuentes de luz infrarroja y determinar su dirección.
- ✓ **Sensor electro-óptico detector de proximidad NXT**, detecta con precisión milimétrica objetos y pequeños cambios en la distancia.
- ✓ **Sensor brújula**, puede medir el campo magnético de la tierra y calcula una dirección magnética con la que guiar el sistema de dirección del robot.
- ✓ **Sensor de color**, permite que el robot distinga no solo entre blanco y negro, sino todo un abanico de colores.
- ✓ **Sensor Gyro**, devuelve el número de grados por segundo y la dirección de rotación. Mide +/- 360° por segundo.
- ✓ **Sensor de aceleración – inclinación**, contiene un acelerómetro de 3 ejes que mide la aceleración en los 3 ejes, x, y y z.

4.2.2.7 SERVO MOTORES

El modelo NXT usa servo motores, los cuales permiten la detección de giros de la rueda, indicando los giros completos o medios giros, que es controlado por el software. En el kit se incluyen tres motores.

Los motores son independientes al bloque, lo que entrega movilidad al sistema dinámico según las necesidades de construcción. Además se pueden programar para que giren en un sentido un número determinado de vueltas o a una velocidad determinada, incluso es posible controlar un giro en valores de precisión de un grado, haciendo muy sencillo que el robot trace una línea recta.

Este motor tiene un **sensor de rotación** incorporado. Este permite controlar los movimientos del robot de manera precisa. El sensor de rotación mide las rotaciones del robot en grados o rotaciones enteras. Una rotación es igual a 360 grados, si se quiere hacer que el motor gire 190 grados, la salida será media vuelta. Este sensor permite también establecer diferentes velocidades en los motores.

4.2.2.8 PIEZAS

El LEGO Mindstorms, a diferencia de algunas de los juegos que vende LEGO , trae algunas piezas extras que permiten entregar flexibilidad y movimiento al robot que se esté construyendo.

Para clasificar las piezas, se sugiere una clasificación entre las piezas móviles, flexibles y de fijación, las cuales son las que incluye el LEGO Mindstorms para desarrollar cualquier robot en especial.

4.2.2.8.1 Piezas móviles

Las piezas móviles que dispone LEGO Mindstorms se centran principalmente en la rotación de bloque, para lograr que las ruedas se muevan en un movimiento circular con respecto al bloque completo. Estas piezas móviles se pueden clasificar en dos:

- ✓ Pieza de rotación, permite rotar un bloque de LEGO con respecto a otro, siendo hueco en el centro del mismo, y con la patas de conexión; lo cual permite añadir más piezas en la parte superior del bloque de rotación.
- ✓ Pieza de giro, a diferencia de la pieza de rotación, la pieza de giro permite girar un bloque en el espacio, permitiendo una simulación de ojos de un robot. Esta pieza no posee una utilidad real, pero sirve de adorno para el robot.

4.2.2.8.2 Piezas flexibles

Las piezas flexibles permiten recrear una articulación de un sistema articulado, donde se requiere que el robot deba realizar un movimiento no rígido en forma específica, como el brazo robot o el brazo clasificador de piezas. Las piezas flexibles por lo general son tubos de plástico capaces de conectarse con dos bloques que no se encuentren separados a una distancia mayor de 4 cm

4.2.2.8.3 Piezas de fijación

Las piezas de fijación, son aquellas que sirven para fijar los ejes de rotación producidos por las piezas de rotación, lo cual implica que son usadas en el centro de las ruedas que posee el LEGO. Por lo general, son tubos de 0.5 mm de diámetro el cual se puede poner en la punta de una barra que actúa como eje central de la rueda, fijando que la misma no se salga durante la ejecución de un programa.

4.2.2.9 RUEDAS

Existen distintas versiones de ruedas, que vienen desde las llantas más anchas, que permiten mayor estabilidad y velocidad; hasta las ruedas más pequeñas que permiten el movimiento del robot en zonas más pequeñas. Se incluye además una cinta que simula el efecto de oruga que poseen los tanques, para que el usuario pueda crear tanques, o una cinta transportadora de objetos.

4.2.3 CONSTRUCCIÓN

La construcción del robot se basa en la unión de bloques de plástico, característicos de LEGO, junto con piezas plegables y algunas piezas que permiten la rotación de ruedas o piezas. El modelo se debe centrar en el bloque programable, ya que este bloque provee la energía necesaria para el movimiento del robot creado. Además, se pueden fijar los sensores que se adjuntan en el kit, para que sean útiles en el desarrollo del robot.

Una de las principales estrategias de construcción se basa en determinar el tipo de robot y si el software proporcionado sirve para construir el robot. Una vez determinado esto, se puede comenzar a construir siempre en bloques de función, como por ejemplo, ensamblar las ruedas a un eje o los sensores en las bases que puedan ser útiles.

Los fallos más comunes en la construcción se debe a la falta de movilidad de las piezas o que los sensores no detectan los valores correctos o simplemente no se mueve. Las estrategias de solución son variadas, pero a la larga, sugieren seguir el camino del rehacer el proceso de pensar y armar y luego ejecutar, es por esto último, que LEGO es un juego de robótica que desarrolla la lógica.

4.2.4 PROGRAMACIÓN

Cuando LEGO puso en el mercado la primera versión de LEGO Mindstorms, difícilmente podía imaginar hasta dónde se iba a abrir el abanico de herramientas de programación disponibles. Una comunidad comenzó a crecer en Internet desarrollando ideas y herramientas al margen de LEGO. Una vez que la empresa fue consciente que ofrecer a la comunidad la información necesaria para desarrollar nuevas herramientas haría crecer su potencialidad, publicaron documentación para desarrollar nuevas herramientas de programación.

Las herramientas de programación disponibles son numerosas, así que cualquiera encontrará una adaptada a sus necesidades o intereses, incluso podrá crear una nueva si su nivel de conocimiento se lo permite. Dentro de éstas herramientas a continuación se van a describir las herramientas gratuitas, NXC y LeJOs, y el software que se incluye en el kit educativo, NXT-G.

4.2.4.1 NXC

NXC, **Not eXactly C**, es un lenguaje de programación de alto nivel utilizado para programar el bloque NXT. NXC es un lenguaje de alto nivel, similar a C, construido sobre el compilador del NBC que es el lenguaje ensamblador de la máquina. Es un lenguaje adaptado al bloque NXT, por ello existen muchas limitaciones que derivan de las limitaciones del NXT.

NXC es un software libre desarrollado bajo la licencia MPL (Mozilla Public License). Se puede obtener toda información necesaria a través de su web:

http://bricxcc.sourceforge.net/nxc/

Para facilitar la programación con este lenguaje, existe un software, Bricx. Esta herramienta integra las librerías necesarias para programar el bloque NXT, funciona como editor y compilador y además permite la comunicación entre el ordenador y el robot, facilitando en gran medida la programación.

En la página oficial de NXC se puede encontrar toda la información actualizada, a continuación se describe a groso modo la sintaxis y funciones de este lenguaje.

4.2.4.1.1 Reglas léxicas

NXC al igual que C y C++ es **sensible a mayúsculas y minúsculas**, por lo que el identificador “xYz” no es igual que “XYZ”.

Los **comentarios** comienzan con /* y acaban con */ y pueden ser de una o más líneas. Los comentarios de una sola línea pueden también escribirse comenzando con // y terminando con una nueva línea.

Los **espacios en blanco**, **tabuladores** y **saltos de línea** sólo se utilizan para hacer el programa más legible, pero no afectan al significado del programa. Si algún operador está formado por más de un carácter, no se podrá introducir ningún espacio en blanco entre sus caracteres.

Las **constantes numéricas** pueden escribirse tanto en decimal como en hexadecimal. Las constantes hexadecimales tendrán que comenzar con 0x o 0X.

Los **nombres de las variables**, tareas, funciones o estructuras deberán comenzar por una letra mayúscula, minúscula o guión bajo. El resto de caracteres podrán estos caracteres más números. Como en todos los lenguajes hay una serie de identificadores reservados, que se detallan a continuación.

__RETURN__	char	long	sub
__RETVAL__	const	mutex	switch
__STRRETVAL__	continue	priority	task
__TMPBYTE__	default	repeat	true
__TMPWORD__	do	return	typedef
__TMPLONG__	else	safecall	unsigned
abs	false	short	unti
asm	for	sign	void
bool	goto	start	while
break	if	stop	
byte	inline	string	
case	int	struct	

4.2.4.1.2 Estructura del programa

Un programa NXC está compuesto por bloques de códigos y variables. Hay dos tipos de bloques de códigos, tareas y funciones. Cada tipo de bloque de código tiene sus propias características, pero tienen una misma estructura.

4.2.4.1.2.1 Tareas

El bloque NXT soporta la ejecución multi-hilo, una tarea en NXC se corresponde con un hilo en NXT. Las tareas se definen utilizando la palabra reservada “task”:

```
task nombre()
{
        //El código de la tarea se coloca aquí
}
```

Todos los programas tienen que tener por lo menos una tarea llamada “main”. El programa comenzará con esa tarea. El número máximo de tareas es 256.

Las tareas pueden iniciarse con las instrucciones “start” y pararse con “stop”. Además el API de NXC dispone de un comando para parar todas las tareas en ejecución “StopAllTasks”. Además una tarea puede pararse por sí misma, utilizando la función “ExitTo” o al llegar al final de su código.

4.2.4.1.2.2 Funciones

A veces es conveniente agrupar unas cuantas instrucciones en una función, así podrán ser ejecutadas cuando se necesite. NXC soporta funciones con argumentos y con valores devueltos. La sintaxis para definir funciones es:

```
[safecall] [inline] tipo_valor_devuelto nombre(lista_de_argumentos)
{
    //cuerpo de la función
}
```

El tipo de valor devuelto tiene que ser es tipo de dato que va a devolver la función, si no va devolver nada, se especificará el valor devuelto como "void".

Puede no pasarse ningún argumento o pasar uno o más argumentos. Los argumentos se definirán por el tipo seguido del nombre, los múltiples argumentos se separarán por comas. NXC soporta el paso de argumentos por valor, por valor constante, pro referencia o por referencia constante.

Las funciones opcionalmente pueden marcarse como funciones en línea, es decir, cada vez que se llame a la función, el código de la función se incluirá en el programa, esto puedo provocar un tamaño de código excesivo. Si no se marcan como en línea, automáticamente le bloque NXT creará una subrutina y las llamadas a la función se convertirán en llamadas a la subrutina. El total de subrutinas (funciones no marcadas como en línea) y el número de tareas deber ser como máximo 256.

Otro parámetro opcional es "safecall", las funciones marcadas con esta palabra reservada no podrán ser llamadas al mismo tiempo por más de un hilo. Es decir, si la función está en ejecución y es llamada por otro hilo, éste deberá esperar a que termine la primera ejecución.

4.2.4.1.3 Variables

Todas las variables en NXC son de los siguientes tipos:

Type Name	Information
Bool	8 bit unsigned
Byte, unsigned char	8 bit unsigned
Char	8 bit signed
Unsigned int	16 bit unsigned
Short, int	16 bit signed
Unsigned long	32 bit unsigned
Long	32 bit signed
Mutex	Tipo especial para uso exclusivo de código
String	Lista de byte
Struct	Tipos de estructuras defindas por el usuario
Arrays	Listas de cualquier tipo

Las variables se definen precedidas por el tipo. Se pueden definir más de una variable a la vez del mismo tipo, el nombre de las variables se separará por comas. La definición se terminará con punto y coma. Opcionalmente se podrán inicializar cada variable usando el igual.

Las **variables globales** se declaran fuera de cualquier bloque de código, incluso de la tarea "main", en el ámbito del programa. Las **variables locales** se declaran dentro de la tarea o función en la que se vaya a utilizar.

Las **estructuras** se definen de la misma forma que en C. Después de definir una estructura, se podrán crear variables de ese tipo o podrán utilizarse para declarar otras estructuras. Los miembros de las estructuras son accesibles usando el punto.

```
struct nombre()
{
        //Elementos de las estructura
}
```

NXC soporta **listas**, las listas son declaradas como cualquier tipo, pero abriendo y cerrando corchetes al final del nombre de la variable. Se pueden declarar listas multinivel. El máximo de niveles en NXC son 4.

Las listas globales de un nivel pueden inicializarse en el momento en el que se declaran usando la siguiente sintaxis:

```
int X[] = {1, 2, 3, 4}
```

Se puede acceder a los elementos de la lista por la posición que ocupan, siendo el primero el índice 0, el segundo el 1, etc.

Para inicializar las listas locales o las listas multinivel es necesario utilizar la función "ArrayInit". A continuación se demuestra como inicializar listas multinivel.

```
int matriz [][];
int vector[];
ArrayInit(vector, 0, 10); // 10 ceros en vector
ArrayInit(matriz, vector, 10) //10 vectotres en matriz
```

NXC también soporta inicializar el tamaño de las listas, tanto globales como locales. El compilador automáticamente generará la lista y la inicializa a ceros.

4.2.4.1.4 Sentencias

El cuerpo de los bloques de código (tareas o funciones) estan compuestos por sentencias o instrucciones. Las sentencias son terminadas siempre con punto y coma.

La **declaración de variables** es una sentencia, ya se ha explicado en el apartado anterior como declarar variables.

La **asignación** es otro tipo de sentencias, una vez declaradas las variabels pueden ser asignadas al valor de una expresión:

variable operador expresión.

Los diferentes operadores son los siguientes:

Operator	**Action**
=	Asignar a la variable el valor de la expresión
+=	Añadir la expresión a la variable
-=	Sustraer la expresión a la variable
*=	Multiplicar la variable por la expresión
/=	Dividir la variable por la expresión
%=	Asignar a la variable el resto de dividir por la expresión
&=	Comparar bit a bit y asignar a la variable los bits que estén en la variable y en la expresión.
|=	Comparar bit a bit y asignar a la variable los bits que estén en la variable o en la expresión.
^=	Comparar bit a bit y asignar a la variable los bits que estén o en la variable o en la expresión.
||=	Asignar a la variable el valor absoluto de la expresión
+-=	Asignar al variable la señal (-1,+1,0) de la expresión
>>=	Desplazamiento derecho de la variable por la expresión
<<=	Desplazamiento izquierdo de la variable por la expresión

Una **estructura de control** es una sentencia compuesta. Si es una lista de sentencias estás van encerradas con corchetes. {lista de sentencias}. A continuación se incluye una tabla con la sintaxis de las diferentes estructuras de control.

if (condición) consecuencia else alternativa

while (condición) cuerpo

do cuerpo while (condición)

for (sentencia; condició; sentencia) cuerpo

repeat (expresión) cuerpo

switch (expresión) { case expresión constante: cuerpo break; default cuerpo break;}

until (condición) (se suele utilizar sin sentencias en el cuerpo)

goto etiqueta; (en el código se pueden poner etiquetas a las sentencias y con la sentencia goto saltar a la sentencia con la etiqueta indicada)

La sentencia **asm** se utiliza para definir alguna llamada al API de NXC. La sintaxis es: asm { una o más línea de código de ensamblador }. Esta sentencia simplemente emite el cuerpo de la sentencia como código NBC, es decir, en lenguaje de ensamblador. Se utiliza para optimizar el código y que se ejecute lo más rápido posible.

Exiten otras sentencias como llamadas a otras funciones “nombre (argumentos);”, iniciar tareas “start(tarea);” o parar “stop(tarea);”, ajustar la prioridad de las tareas “priority tarea, nueva prioridad;”, en los bucles salir del bucle “break;” o pasar a la

siguiente iteración “continue;”, en las funciones devolver algún valor “return [expresión];”, entre otras.

4.2.4.1.5 Expresiones

El tipo más primitivo de expresiones son los valores. Para formar expresiones más complejas se utilizar varios operadores. NXC sólo deja construir expresiones con dos tipos de valores, constantes numéricas y variables.

Hay dos valores especiales definidos, **true** y **false**. El valor **false** es cero, mientras que el valor **true** es uno. El resultado de operadores relacionales (p.e. <), cuando el resultado sea falso el valor será 0, en otro caso 1.

La mayoría de los operadores se pueden utilizar sólo en la evaluación de expresiones constantes. A continuación se listan los operadores ordenados de forma decreciente por prioridad.

Operador	Descripción	Asociación	Restricción	Ejemplo
abs() sign()	Valor absoluto Señal del operando	Ninguna Ninguna		abs(x) sign(x)
++,--	Pos-incremento, pos-decremento	Izquierda	Sólo variables	x++
- ~ !	Negación Negación por bit Negación lógica	Derecha Derecha Derecha		-x ~123 !x
*, /, %	Multiplicación, división, módulo	Izquierda		x * y
+, -	Suma y resta	Izquierda		x + y
<<, >>	Desplazamiento izquierdo y derecho	Izquierda		x << 4
<, >, <=, >=	Operadores relacionales	Izquierda		x < y
==, !=	Igual, distinto	Izquierda		x == y
&	Bit a bit AND	Izquierda		x & y
^	Bit a bit XOR	Izquierda		x ^ y
|	Bit a bit OR	Izquierda		x | y
&&	AND lógico	Izquierda		x && y
||	OR lógico	Izquierda		x || y
?:	Valor condicional	Ninguna		x==1 ? y : z

La comparación de dos expresiones forma una condición. Una condición puede ser negada con el operador de negación o dos condiciones pueden ser unidas con los operadores AND o OR.

4.2.4.1.6 Directivas

Existen dos directivas que es importante conocer, la directiva **#include** que permite incluir otros ficheros que serán necesarios para la ejecución del programa y la directiva #**define** que es usada para redefinir macros.

```
#include "foo.h"
```

```
#define foo(x) do { bar(x); \
                    baz(x); }
```

4.2.4.1.7 NXC API

El API de NXC define un conjunto de constantes, funciones, valores u macros para dar acceso a las diferentes capacidades del bloque NXT como los sensores, salidas y comunicaciones.

4.2.4.1.7.1 Características generales

Funciones de Tiempo	
Función	**Descripción**
Wait(time)	Crea una tarea de dormir durante el tiempo indicado (milisegundos)
CurrentTick()	Devuelve el valor del sistema de tiempo actual en milisegundos.
FirstTick()	Devuelve el valor del sistema de tiempo en el momento que comenzó a correr el programa en milisengundos.
SleepTime()	Devuelve el número de minutos que el NXT se mantendrá encendido antes de apagarse automáticamente.
SleepTimer()	Devuelve el número de minutos que quedan para apagarse automáticamente.
ResetSleepTimer()	Reinicia el tiempo que queda para apagarse automáticamente.
SetSleepTime(minutos)	Define el tiempo de espera en minutos para apagar el NXT automáticamente.
SleepNow()	Apagar el NXT inmediatamente.
SetSleepTimer(minutos)	Definir el tiempo que tiene que pasar para que el NXT se apague automáticamente.
Funciones de control del programa	
Función	**Descripción**
Stop(bvalor)	Parar de correr el programa si bvalor es verdadero.
StopAllTasks()	Parar todas las tareas que en momento están corriendo.
StartTask(task)	Iniciar la tarea especificada.
StopTask(task)	Parar la tarea especificada.
Acquire(mutex)	Hacer una variable exclusiva.
Release(mutex)	Quitar a una variable la exclusividad.
Precedes(task1, task2,…)	Especificar las tareas que se ejecutarán cuando termine la tarea actual.
Follows(task1,task2,…)	Especificar que la tarea que se ejecutará cuando termine las tareas indicadas.
ExitTo(task)	Terminar inmediatamente la ejecución de la tarea actual y empezar la ejecución de

	la tarea especificada.
Funciones para los strings	
Función	**Descripción**
StrToNum(str)	Convertir un string en un valor numérico.
StrLen(str)	Longitud del string especificado.
StrIndex(str,idx)	Valor numérico del carácter de la posición indicada.
NumToStr(value)	Convertir a string el valor numérico pasado.
FormatNum(fmrstr,value)	Formatear el string usando el formato y el valor pasado.
StrCat(str1,str2,…)	Concatenar los string especificados.
SubStr(string,idx,len)	Sub-string comenzado en la posición idx y de longitud len.
StrReplace(string, idx, newStr)	String con la parte del string sustituida (empezando por la posición idx) con el contenido indicado.
Flattern(value)	Byte como string del valor especificado.
FlatternVar(anytype)	Byte como string de la variable especificada.
UnflatternVar(string, anytype)	Convertir el contenido del string en bytes devolviendo su tipo original.
ByteArrayToStr(arr)	Convertir la lista en un string añadiendo un elemento nulo al final de los elementos de la lista. (lista de una dimensión)
ByteArrayToStrEx(arr, out str)	Convertir la lista en un string añadiendo un elemento nulo al final de los elementos de la lista. (lista de una dimensión)
Funciones para los listas	
Funciones	**Descripción**
StrToByteArray(str, out arr)	Convertir el string en una lista eliminado la terminación nula al final del string.
ArrayLen(array)	Longitud de la lista
ArrayInit(array, value, count)	Inicializar la lista de count elementos con el valor value.
ArraySubset(out aout, asrc, idx, len)	Copiar el substring comenzando en el índice idx, de longitud len.
ArrayBuild (out aout, src1[,src2,…])	Construir un Nuevo array con los elementos indicados.
Funciones numéricas	
Función	**Descripción**
Random(n)	Número aleatorio entre 0 y n.
Random()	Número aleatorio.
Sqrt(x)	Raíz cuadrada de x.
Sin(degrees)	Seno de degrees.
Cos(degrees)	Coseno de degrees.
Asin(value)	Arcoseno de value.

Acos(value)	Arcocoseno de value.
Bcd2dec(bcdValue)	Convertir de binario a decimal
Funciones para la pantalla	
Función	**Descripción**
SysDrawText(DrawTextType & args)	Sirve para dibujar un texto en la pantalla del NXT.
SysDrawPoint(DrawPointType & args)	Sirve para dibujar un pixel en la pantalla del NXT.
SysDrawLine(DrawLineType & args)	Sirve para dibujar una línea en la pantalla del NXT.
SysDrawCircle(DrawCircleType & args)	Sirve para dibujar un círculo en la pantalla del NXT.
SysDrawRect(DrawRectType & args)	Sirve para dibujar un rectángulo en la pantalla del NXT.
SysDrawGraphic(DrawGraphicType & args)	Sirve para dibujar una imagen gráfica en la pantalla del NXT.
SysSetScreenMode(SetScreenModeType & args)	Establecer el modo de pantalla de la pantalla del NXT.
SysSoundPlayFile(SoundPlayFileType & args)	Reproducir un fichero de sonido dado.
SysSoundPlayTone(SoundPlayToneType & args)	Reproducir un tono dado por los parámetros.
SysSoundGetState(SoundGetStateType & args)	Recuperar información sobre el módulo de sonido.
SysSoundSetState(SoundSetStateType & args)	Establecer la configuración del módulo de sonido.
SysReadButton(ReadButtonType & args)	Leer el estado del botón.
SysRandomNumber(RandomNumberType & args)	Obtener un número aletorio.
SysGetStartTick(GetStartTickType & args)	Obtener el valor del tick de tiempo en el que el programa comenzó a ejecutarse.
SysKeepAlive(KeepAliveType & args)	Reiniciar el tiempo del apagado automático.
SysFileOpenWrite(FileOpenType & args)	Crear un fichero para escribir en él.
SysFileOpenAppend(FileOpenType & args)	Abrir un fichero para escribir en él.
SysFileOpenRead(FileOpenType & args)	Abrir un fichero para leer de él.
SysFileRead(FileReadWriteType & args)	Leer de un fichero.
SysFileWrite(FileReadWriteType & args)	Escribir en un fichero.
SysFileClose(FileCloseType & args)	Cerrar un fichero.
SysFileResolveHandle(FileResolveHandleType & args)	Obtener el handle del fichero en uso.
SysFileRename(FileRenameType & args)	Renombrar un fichero.
SysFileDelete(FileDeleteType & args)	Borrar un fichero.
SysCommLSWrite(CommLSWriteType & args)	Escribir en un sensor I2C.
SysCommLSCheckStatus(CommLSCheckStatusType & args)	Chequear el estado de un sensor I2C.
SysCommLSRead(CommLSReadType & args)	Leer de un sensor I2C.

SysMessageWrite(MessageWriteType & args)	Escribir un mensaje a una lista (correo)
SysMessageRead(MessageReadType & args)	Leer un mensaje de una lista (correo)
SysCommBTWrite(CommBTWriteType & args)	Escribir a una conexión Bluetooh
SysCommBTCheckStatus(CommBTCheck StatusType & args)	Chequear el estado de una conexión Bluetooh.
SysIOMapRead(IOMapReadType & args)	Leer datos del firmware del módulo IOMap
SysIOMapWrite(IOMapWriteType & args)	Escribir datos del firmware del módulo IOMap.
SysIOMapReadByID(IOMapReadByIDTyp e & args)	Leer datos del firmware del módulo IOMap
SysIOMapWriteByID(IOMapWriteByIDTy pe & args)	Escribir datos del firmware del módulo IOMap.
SysDisplayExecuteFunction(DisplayExecut eFunctionType & args)	Ejecutar directamente la función de dibujo.
SysCommExecuteFunction(CommExecuteF unctionType & args)	Ejecutar directamente la función de comunicación
SysLoaderExecuteFunction(LoaderExecute FunctionType & args)	Ejecutar directamente la función del módulo de carga.
SysCall(funcID, args)	Macro genérica para llamar a funciones del sistema.

4.2.4.1.7.2 Módulo de entrada

El módulo de entrada controla todas las entradas de los sensores excepto de los sensores digitales I2C. Hay cuatro sensores, internamente numerados del 0 a 3. Externamente numerados del 1 al 4. Los puertos de los sensores son definidos con los nombres S1, S2, S3 y S4, en el NBC los nombres de los puertos son IN_1, IN_2, IN_3 y IN_4.

El valor de los sensores se define en SENSOR_1, SENSOR_2, SENSOR_3 Y SENSOR_4. Estos nombres pueden usarse en los programas para leer el valor actual del sensor.

Los puertos de los sensores del NXT son capaces de interactuar con distintos sensores. Para configurar el tipo de sensor se utiliza la función SetSensorType. Hay 12 tipos de sensores y el treceavo es el que indica que no hay un sensor configurado.

Tipo de sensor	**Significado**
SENSOR_TYPE_NONE	Sensor no configurado
SENSOR_TYPE_TOUCH	Sensor de contacto
SENSOR_TYPE_TEMPERATURE	Sensor de temperatura
SENSOR_TYPE_LIGHT	Sensor de luz
SENSOR_TYPE_ROTATION	Sensor de rotación
SENSOR_TYPE_LIGHT_ACTIVE	Sensor de luz con luz
SENSOR_TYPE_LIGHT_INACTIVE	Sensor de luz sin luz
SENSOR_TYPE_SOUND_DB	Sensor de sonido en la escala dB
SENSOR_TYPE_SOUND_DBA	Sensor de sonido en la escala dbA

SENSOR_TYPE_CUSTOM	Sensor personalizado
SENSOR_TYPE_LOWSPEED	Sensor digital I2C
SENSOR_TYPE_LOWSPEED_9V	Sensor digital I2C (9V)
SENSOR_TYPE_HIGHSPEED	Sensor de alta velocidad

El bloque NXT permite configurar los sensores en diferentes modos. El modo de los sensores determina como será procesado el valor obtenido del sensor. Con la función SetSensorMode se define le modo del sensor.

Modo del sensor	**Significado**
SENSOR_MODE_RAW	Valor bruto de 0 a 1023
SENSOR_MODE_BOOL	Valor booleano (0 o 1)
SENSOR_MODE_EDGE	Cuenta el número de transiciones booleanas.
SENSOR_MODE_PULSE	Cuenta el número de períodos booleanos.
SENSOR_MODE_PERCENT	Valor de 0 a 100
SENSOR_MODE_FAHRENHEIT	Grados Fahrenheit
SENSOR_MODE_CELSIUS	Grados Celsius
SENSOR_MODE_ROTATION	Rotación (16 ticks por revolución)

Para poder trabajar con los sensores en el API de NXC se definen la siguiente lista de funciones.

Función	**Descripción**
SetSensor(port, const configuration)	Establecer el tipo y el modo del sensor.
SetSensorType(port, const type)	Establecer el tipo del sensor.
SetSensorMode(port, const mode)	Establecer el modo del sensor.
SetSensorLight(port)	Configurar el sensor de un puerto como sensor de luz.
SetSensorSound(port)	Configurar el sensor de un puerto como sensor de sonido.
SetSensorTouch(port)	Configurar el sensor de un puerto como sensor de contacto.
SetSensorLowspeed(port)	Configurar el sensor de un puerto como sensor digital I2C.
SetInput(port, const field, value)	Configurar el tipo, el modo y el puerto.
ClearSensor(const port)	Limpiar el valor del sensor.
ResetSensor(port)	Reiniciar el valor del sensor.
SetCustomSensorZeroOffset(const p, value)	Configurar el offset 0 del sensor.
SetCustomSensorPercentFullScale(const p, value)	Configurar la escala entera de porcentajes.
SetCustomSensorActiveStatus(const p, value)	Configurar el valor de activación del sensor.
SetSensorDigiPinsDirection(const p, value)	Configurar la dirección de los pines digitales.
SetSensorDigiPinsStatus(const p, value)	Configurar el valor de los estados de los pines digitales.
SetSensorDigiPinsOutputLevel(const p, value)	Configurar el valor del nivel de salida de los pines digitales.
Sensor(n)	Lectura del sensor.

SensorUS(n)	Lectura de un sensor de ultrasonidos.
SensorType(n)	Tipo de sensor configurado.
SensorMode(n)	Modo del sensor configurado.
SensorRaw(n)	Valor bruto del sensor.
SensorNormalized(n)	Valor normalizado del sensor.
SensorScaled(n)	Valor de escala del sensor.
SensorInvalid(n)	Valor del flag de dato inválido.
SensorBoolean(const n)	Valor booleano del sensor.
GetInput(n, const field)	Valor del campo indicado del sensor.
CustomSensorZeroOffset(const p)	Valor configurado como offset 0.
CustomSensorPercentFullScale(const p)	Valor configurado como escala de porcentaje entera.
CustomSensorActiveStatus(const p)	Valor configurado como activación del sensor.
SensorDigiPinsDirection(const p)	Dirección de los pines digitales.
SensorDigiPinsStatus(const p)	Valor de los estados de los pines digitales.
SensorDigiPinsOutputLevel(const p)	Valor del nivel de salida de los pines digitales.

4.2.4.1.7.3 Módulo de salida

El módulo de salida del NXT abarca las salidas del motor. Dependiendo de la función la salida o las salidas deben de ser una constate o una variable que contenga un valor apropiado de puerto de salida. Las constantes OUT_A, OUT_B y OUT_C se usan para identificar los tres puertos de salida. A su vez, NXC API tiene definidas las constantes para hacer referencia a la combinación de los puertos, OUT_AB, OUT_AC, OUT_BC y OUT_ABC.

El nivel de potencia puede estar en el rango de 0 (mínimo) a 100 (máximo). Los niveles de potencia negativos invierten la dirección de rotación.

Cada salida tiene varios campos que definen el estado actual del puerto de salida. Estos campos son los siguientes.

Campo	**Tipo**	**Acceso**	**Rango**	**Descripción**
UpdateFlags	ubyte	Lectura/escritura	0,255	Flag de los bits
OutputMode	ubyte	Lectura/escritura	0,255	Modo de la salida
Power	sbyte	Lectura/escritura	-100,100	Especifica el nivel de potencia.
ActualSpeed	sbyte	Lectura	-100,100	Porcentaje de potencia aplicada a la salida
TachoCount	slong	Lectura	Rango long	Valor del contador de la posición interna
TachoLimit	ulong	Lectura/Escritura	Rango long	Número de grados que el motor puede rotar.
RunState	ubyte	Lectura/Escritura	0,255	Estado de ejecución
TurnRatio	sbyte	Lectura/Escritura	-100/100	Valor proporcional de giro
RegMode	ubyte	Lectura/Escritura	0,255	Modo de regulación del uso

Overload	ubyte	Lectura	0,1	Indica si el motor está funcionando más despacio de lo esperado
RegPValue	ubyte	Lectura/Escritura	0,255	Término proporcional usado
RegIValue	ubyte	Lectura/Escritura	0,255	Término integral usado
RegDValue	ubyte	Lectura/Escritura	0,255	Término derivado usado
BlockTachoCount	slong	Lectura	Rango long	El valor de la posición del bloque relacional
RotationCount	slong	Lectura	Rango long	Contador de la posición relativa al programa.

Controlar las salidas es una de las características de los programas, existen un gran número de funciones que permiten trabajar fácilmente con las salidas. Éstas se definen en la siguiente lista. Las versiones Ex de las funciones se usan para reiniciar constantes.

Función	**Descripción**
Off(outputs)	Apagar la salida especificada
OffEx(outputs, const reset)	
Coast(outputs)	Poner en punto muerto
CoastEx(outputs, const reset)	
Float(outputs)	Float es un alias de Coast.
OnFwd(outputs, pwr)	Arrancar el motor hacia adelante y con potencia pwr
OnFwdEx(outputs, pwr, const reset)	
OnRev(outputs, pwr)	Arrancar el motor hacia atrás y con potencia pwr
OnRevEx(outputs, pwr, const reset)	
OnFwdReg(outputs, pwr, regmode)	Arrancar el motor hacia adelante usando el modo de regulación especificado.
OnFwdRegEx(outputs, pwr, regmode, const reset)	
OnRevReg(outputs, pwr, regmode)	Arrancar el motor hacia atrás usando el modo de regulación especificado.
OnRevRegEx(outputs, pwr, regmode, const reset)	
OnFwdSync(outputs, pwr, turnpct)	Arrancar los motores hacia adelante regulando la sincronización con la relación de arranque.
OnFwdSyncEx(outputs, pwr, turnpct, const reset)	
OnRevSync(outputs, pwr, turnpct)	Arrancar los motores hacia atrás regulando la sincronización con la relación de arranque.
OnRevSyncEx(outputs, pwr, turnpct, const reset)	
RotateMotor(outputs, pwr, angle)	Arrancar el motor hacia adelante los grados especificados.
RotateMotorPID(outputs, pwr, angle, p, i, d)	
RotateMotorPID(outputs, pwr, angle, p, i, d)	Arrancar el motor hacía adelante los grados especificados, indicando los factores proporcional, integral y derivado
RotateMotorExPID(outputs, pwr, angle, turnpct, sync, stop, p, i, d)	
ResetTachoCount(outputs)	Reiniciar el contador de punto puerto.
ResetBlockTachoCount(outputs)	Reiniciar el contador de punto muerto del bloque.

ResetRotationCount(outputs)	Reiniciar el contador de rotación
ResetAllTachoCounts(outputs)	Reiniciar todos los putos muertos
SetOutput(outputs, const field1, val1, …, const fieldN, valN)	Reiniciar los campos especificados
GetOutput(output, const field)	Obtener el valor del campo especificado.
MotorMode(output)	Obtener el modo del motor.
MotorPower(output)	Obtener el nivel de potencia.
MotorActualSpeed(output)	Obtener la velocidad actual.
MotorTachoCount(output)	Obtener el contador del punto muerto.
MotorTachoLimit(output)	Obtener el límite del punto muerto.
MotorRunState(output)	Obtener el estado.
MotorTurnRatio(output)	Obtener la relación de giro.
MotorRegulation(output)	Obtener la regulación.
MotorRegPValue(output)	Obtener el valor PID proporcional.
MotorRegIValue(output)	Obtener el valor PID integral.
MotorRegDValue(output)	Obtener el valor PID derivado.
MotorBlockTachoCount(output)	Obtener el punto muerto del bloque
MotorRotationCount(output)	Obtener el contador de rotación.
MotorRotationCount(output)	Obtener el pulso del motor con la frecuencia de modulación.
SetMotorPwnFreq(val)	Establecer el pulso del motor con la frecuencia de modulación.

4.2.4.1.7.4 Módulo de sonido

El módulo de sonido abarca todas las características relacionadas con la salida de sonidos. El NXT soporta reproducir tonos básicos así como dos tipos de ficheros diferentes, SoundModuleName (“Sound.mod”) y SoundModuleID (0x00080001)

Los ficheros de sonido tienen que ser ficheros wav. Los ficheros de melodías tienen que ser del tipo MIDI. Cuando un sonido o un fichero se están reproduciendo, la ejecución del programa no termine a que termine la reproducción. La ejecución se pude parar utilizando la función Wait().

El API de NXC define una serie de funciones para trabajar con los tonos y los sonidos. Al igual que en el módulo de salidas, las versiones Ex implican el reinicio de alguna constante.

Función	**Descripción**
PlayTone(frequency, duration)	Reproducir un tono simple de la frecuencia y duración especificada.
PlayToneEx(frequency, duration, volume, bLoop)	Reproducir un tono simple de la frecuencia, duración y volumen especificados.
PlayFile(filename)	Reproducir el fichero especificado.
PlayFileEx(filename, volume, bLoop)	Reproducir el fichero y con el volumen especificado.
SoundFlags()	Devuelve los flags actuales de sonido.
SetSoundFlags(n)	Establecer los flags actuales de sonido.

SoundState()	Estado actual del módulo de sonido.
SetSoundModuleState(n)	Establecer el estado actual del sonido.
SoundMode()	Devuelve el modo actual del sonido.
SetSoundMode(n)	Establecer el modo actual del sonido.
SoundFrequency()	Devuelve la frecuencia actual del sonido.
SetSoundFrequency(n)	Establecer la frecuencia actual del sonido.
SoundDuration()	Devuelve la duración del sonido.
SetSoundDuration(n)	Establecer la duración del sonido.
SoundSampleRate()	Obtener la tasa actual del sonido.
SetSoundSampleRate(n)	Establecer la tasa actual del sonido.
SoundVolume()	Obtener el volumen actual.
SetSoundVolume(n)	Establecer el volumen actual.
StopSound()	Parar la reproducción.

4.2.4.1.7.5 Módulo de pantalla

El módulo de pantalla del NXT soporta dibujar en la pantalla del NXT puntos, líneas, rectángulos y círculos, así como ficheros de iconos gráficos, textos y números.

La pantalla tiene su origen (0,0) en la esquina inferior, izquierda. En el API del NXT se definen ocho líneas para poder dibujar texto o números, las constantes que hacen referencia a las líneas son: LCD_LINE1, LCD_LINE2, LCD_LINE3, LCD_LINE4, LCD_LINE5, LCD_LINE6, LCD_LINE7, LCD_LINE8.

Las funciones que posibilitan interactuar con la pantalla son las siguientes:

Funciones	**Descripción**
NumOut(x, y, value, clear = false)	Dibuja un número en la posición especificada.
TextOut(x, y, msg, clear = false)	Dibuja un texto en la posición especificada.
GraphicOut(x, y, filename, clear = false)	Dibuja un gráfico en la posición especificada.
GraphicOutEx(x, y, filename, vars, clear = false)	
CircleOut(x, y, radius, clear = false)	Dibuja un círculo en la posición especificada.
LineOut(x1, y1, x2, y2, clear = false)	Dibuja una línea en la posición especificada.
PointOut(x, y, clear = false)	Dibuja un punto en la posición especificada.
RectOut(x, y, width, height, clear = false)	Dibuja un rectángulo en la posición especificada.
ResetScreen()	Reiniciar la pantalla.
ClearScreen()	Limpiar la pantalla a pantalla en blanco.
DisplayFlags()	Mostrar los flags actuales de pantalla.
SetDisplayFlags(n)	Establecer los flagas actuales de pantalla.
DisplayEraseMask()	Devolver la máscara de borrado de la pantalla.
SetDisplayEraseMask(n)	Establecer la máscara de borrado de la pantalla.
DisplayUpdateMask()	Devolver la máscara de actualización de la pantalla.
SetDisplayUpdateMask(n)	Establecer la máscara de actualización de la pantalla.
DisplayDisplay()	Devolver la dirección de memoria de la pantalla.

SetDisplayDisplay(n)	Establecer la dirección de memoria de la pantalla.
DisplayTextLinesCenterFlags()	Devolver los flags actuales del centro de las líneas de texto.
SetDisplayTextLinesCenterFlags(n)	Establecer los flags actuales del centro de las líneas de texto.
GetDisplayNormal(x, line, count, data)	Leer la "cuenta" de bytes de la memoria de la pantalla normal.
SetDisplayNormal(x, line, count, data)	Establecer la "cuenta" de bytes de la memoria de la pantalla normal.
GetDisplayPopup(x, line, count, data)	Leer la "cuenta" de bytes de la memoria del popup de la pantalla.
SetDisplayPopup(x, line, count, data)	Establecer la "cuenta" de bytes de la memoria del popup de la pantalla.

4.2.4.1.7.6 Módulo de botones

El módulo de botones de NXT abarca todo el soporte para los cuatro botones del bloque NXT. Para el uso de los botones se han definido unas constantes.

Botón	**Descripción**
BTN1, BTNEXIT	Botón volver/apagar.
BTN2, BTNRIGHT	Botón seleccionar derecho.
BTN3, BTNLEFT	Botón seleccionar izquierdo.
BTN4, BTNCENTER	Botón encender/entrar.

Para trabajar con el módulo de botones se han definido las siguientes funciones.

Función	**Descripción**
ButtonCount(btn,reset)	Devuelve el número de veces que se ha presionado el botón.
ButtonPressed(btn,reset)	Devuelve si el botón ha sido presionado.
ReadButtonEx(btn, reset, out pressed, out count)	Lee el botón especificado

4.2.4.2 NXJ

leJOS NXJ, pronunciado como la palabra lejos, es una pequeña máquina virtual java. En 2006 fue importado al bloque NXT. Todas las clases son especificadas en el API del NXJ así como herramientas usadas para subir el código al bloque NXT.

Esta herramienta ofrece un lenguaje orientado a objetos (Java), hilos, listas, recursividad, sincronización, excepciontes, tipos Java incluidos float, logn y string, así como muchas de las clases de java.lang, java.utl y java.io.

leJOS es un proyecto de código abierto, alojado al igual que el NXC en repositorio sourceforge. El auto original de de la pequeña máquina virtual java fue Jose Solorzano.

http://lejos.sourceforge.net/

Para programar con este lenguaje se utilizan los típicos programas para programar Java, como Netbeans o Eclipse. Los programas una vez compilados se suben a través del Bluetooh al bloque NXT.

A continuación se describen brevemente los paquetes propios de leJOS utilizar para programar los robots LEGO Mindstorms con este lenguaje.

4.2.4.2.1lejos.addon.gps

El paquete lejos.addon.gps proporciona el análisis del GPS. En muchos casos se puede utilizar el paquete javax.microedition.location, con el API java estándar para obtener los datos GPS. A parte de las clases que se describen a continuación, este paquete ofrece más datos, como información de los satélites.

Interfaz	
GPSListener	Esta es la interfaz para gestionar los eventos del GPS.
Clases	
GGASentence	Administrar sentencias GGA, esencial para fijar datos en 3D y para la precisión de los datos
GPS	Administrar los datos recibido desde un dispositivo GPS.
GSASentence	Administrar sentencias GSA (DOP y satélites activos)
GSVSentence	Administrar sentencias GSV (vista de satélites GPS)
NMEASentence	Administrar todas las sentencias NMEA
RMCSentence	Administrar todas las sentencias RMC
Satellite	Modelar los datos extraídos de las sentencias NMEA GSV
SimpleGPS	Administrar los datos recibido desde un dispositivo GPS.
VTGSentence	Administra las sentenicas VTG recibidas de un receptor NMEA GPS.

4.2.4.2.2lejos.addon.keyboard

Soporte para teclados Bluetooh SPP.

Interfaz	
keyListener	Esta interfaz es para las clases que desea recibir eventos de teclado.

Clases	
Keyboard	Esta clase sólo trabaja con teclados SPP, no con los teclados estándar HID.
KeyEvent	Contiene datos de un evento del teclado.

4.2.4.2.3lejos.charset

Una simple API de conjunto de caracteres par leJOS.

Interfaz
CharsetDecoder
CharsetEncoder
Clases
Latin1Decoder
Latin1Encoder
UTF8Decoder
UTF8Encoder

4.2.4.2.4lejos.geom

La robótica soporta formas geométricas usando coordenadas float, para ello se utilizan:

Clases	
Line	Representa una línea y soporta el cálculo del punto de intersección de dos segmentos.
Point	Punto con coordenadas float usado en la navegación.

4.2.4.2.5lejos.io

Clases del paquete java.io soportados por leJOS.

Clases	
LejosInputStreamReader	
LejosOutputStreamWriter	Conjunto de carácter abstractos de un solo byte escritos en Outputstream

4.2.4.2.6lejos.nxt

Paquete que controla el acceso a los sensores y motores NXT, entre otros.

Interfaces	
ADSensorPort	Una abstracción de un puerto que soporta sensores analógicos y digitales.
BasicMotorPort	Una abstracción de un puerto de motor que soporte los tipos motor RCX.
BasicSensorPort	Una abstracción de un puerto que soporte configuración y tipos y modos de los sensores.
ButtonListener	Una abstracción de los eventos de botones recibidos.
I2CPort	Una abstracción de un puerto que soporta sensores I2C.
LegacySensorPort	Una abstracción de un puerto en modo heredado de sensores RCX.

ListenerCaller	Inferfaz para las llamadas de los eventos leJOS.
SensorConstans	Constantes usadas como tipos y modos de sensores.
SensorPortListener	Interfaz para monitorizar los cambios en el valor de los sensores analógicos o digitales.
TachoMotorPort	Una abstracción de puertos motores que soportan motores NXC con tacómetros.
Clases	
Battery	Proporciona acceso a la batería.
Button	Abstracción de un botón NXT.
ColorLightSensor	Driver del sensor de color de LEGO.
Flash	Acceso de lectura y escritura a memoria flash.
I2CSensor	Clase abstracta que implementa los métodos comunes para todos los sensores I2C.
LCD	Salida de texto y gráficos en la pantalla.
LCDOutputStream	Un stream de salida simple que implementa la salida por consola.
LightSensor	Esta clase obtiene las lecturas obtenidas del sensor de luz de LEGO.
Motor	Abstracción de un motor NXT.
MotorPort	Abstracción de un puerto de un motor NXT.
NXT	Abstracción del dispositivo local NXT.
Poll	Proporciona aceso al bloque de eventos del NXT.
SensorPort	Abstracción de un puerto de entrada del NXT.
Settings	Ajustes persistentes del leJOS NXJ.
Sound	Rutinas de sonido NXT.
SoundSensor	Abstracción e un sensor de sonido.
SystemSettings	Esta clase está diseñada para usar con otras clases para leer los ajustes persistentes.
TouchSensor	Abstracción de un sensor de contacto.
UltrasonicSensor	Abstracción de un sensor de ultrasonidos.
VM	Esta clase proporciona acceso a una serie de estructuras internas de la máquina virtual leJOS.

4.2.4.2.7lejos.nxt.addon

Paquete para la gestión de partes de terceros y sensores heredados de RCS, motores y otros equipos no incluidos en el kit LEGO NXT.

Clases	
ColorSensor	HiTechnic color sensor.
CompassSensor	Abstracción de un HiTechnic o sensor de brújula
EOPD	Sensor EOPD de HiTEchnics
GyroSensor	Sensor Gyro de HiTechnic
IRLink	IRLink de HiTechnic
IRSeeker	Sensor IRSeeker de HiTechnic
LDCMotor	Abstracción para administrar cualquier motro DC.
LMotor	Abstracción genérica para gestionar los servos RC y motor DC.
LSC	Gestionar los dispositivos LSC.

LServo	Abstracción para modelar cualquier servo RC.
MSC	Administrar los dispositivos MSC8.
MServo	Abstracción para modelas cualquier RC Servo.
NXTCam	NXTCam de Mindsensors.

4.2.4.2.8lejos.nxt.comm

Conjunto de clases que gestionan las comunicaciones NXT.

Clases	
Bluetooh	Proporciona la comunicación Bluetooh
BTConnection	Proporciona una comunicación a través del la conexión Bluetooh.
LCP	Implenta el protocolo de comunicación de LEGO.
LCPBTResponder	Soporte para comandos LCP sobre Bluetooh.
LCPResponder	Sopoerta comandos LCP.
NXTCommConnector	Interfaz estándar para conectar/parar durante uan conexión.
NXTCommDevice	Clase básica para el dispositivo de comunicaciones.
NXTInputStream	Extensión de InputStream para el Bluetooh.
NXTOutputStream	Implementa un OutputStream para el Bluetooh.
RConsole	Proporciona un acceso de bajo nivel al hardware RS485.
RS845Connection	Objeto de conexión para conexiones con RS485.
USB	Acceso de bajo nivel al USB.
USBConnection	Proporciona una conexión USB.

4.2.4.2.9lejos.nxt.debug

Depuración de las clases de hilos

Clases	
DebugInterface	Proporciona la interfaz principal de la capacidad de depuración de LeJOS.
DebugMonitor	Monitor simple de depuración que se pude ejecutar de forma pararela a un programa NXJ.

4.2.4.2.10 lejos.nxt.rcxcomm

Emulación de clases de comunicación con RCX.

Interfaces	
Opcode	Constantes Opcode
Clases	
LLC	Emula la clase RCS LLC usando la clase RCXLink.
LLCHandler	Implementación del protocolo del paquete LLC.
LLCRealibleHandler	Garantiza la entrega fiable usando las sumas de

	comprobación, ACK, y un número de secuencia de un solo bit.
PacketHandler	Paquete de controlador
RCXAbstractPort	Proporciona una interfaz similar al java.net.Socket
RCXPort	Proporciona una interfaz similar al java.net.Socket.
Serial	Emula la clase RCX Serial.

4.2.4.2.11 lejos.nxt.remote

Acceso remoto al NXT usando el Bluetooh.

Interfaces	
ErrorMessages	Mensajes de error capturados después de la llamada al bloque NXT.
NXTCommRequest	Interfaz para todas las clases que implementan las comunicaciones de bajo nivel.
NXTProtocol	Constantes del protocolo de comunicación LEGO.
Clases	
AsciizCodec	Métodos para codifica y decodificar ASCII
DeviceInfo	Representa un acceso remoto a NXT vía LCP.
FileInfo	Estructura que da información sobre un fichero leJOS NXT.
FirmwareInfo	Información firmware para un acceso remoto vía LCP.
InputValues	Acceso vía LCP a los valores del sensor de entrada del NXT remoto.
NXTComm	Inicialización de comunicación a un NXT remoto.
NXTCommand	Envía peticiones LCP al NXT y recibe réplicas.
OutputState	Contenedor para la posesión de los valores del estado de la salida.
RemoteBattery	Lectura de la batería del NXT remoto.
RemoteMotor	Clase Motor.
RemoteMotorPort	Soporta la conexión de un motor al NXT remoto.
RemoteNXT	Proporciona un API similar al API de leJOS para el acceso a motores, sensores,…
RemoteSensorPort	Usando LCP, emula un puerto sensor.

4.2.4.2.12 lejos.robotics

Interfaces abstractas hardware para el paquete de robótica.

Interfaces	
ColorDetector	
DCMotor	Interfaz para controlar el motor DC sin codificador.
DirectionFinder	Abstracción para la brújula y otros dispositivos.
ElevationPlatform	
Encoder	Abstracción para la construcción del tacómetro.
LampLightDetector	Interfaz para el sensor de luz.

LightDetector	Implementación de una plataforma independiente para los sensores.
MoveListener	Cualquier clase que quería actualizar por un evento.
MovementProvider	Deberán ser aplicados por un piloto que proporciona un movimiento parcial a una postura cuando se le solicite.
RangeFinder	Abstracción de un sensor de telémetro que devuelve la distancia al objeto más cercano
RangeScanner	Abstracción de un rango único de exploración del sensor, plataforma giratoria con un telémetro, o un robot completo,
RotationPlatform	Una plataforma para la rotación de un sensor y elevando el ángulo.
Tachometer	Abstracción para un tacómetro.
TachoMotor	Interfaz para motores codificados sin límite de movimiento.
TachoMotorListener	NOTA: Puede que se desee tener un detector que avise cuando la rotación arbitraria se haya completado.
Clases	
Colors	Valores para los colores estándar de LEGO.
ExtendedPlatform	
MotorEvent	
Movement	Modelos de un movimiento realizado por un piloto.
Pose	Representa la localización y ángulo de dirección del robot.
RangeReading	Representan una lectura única escala.
RangeReadings	Representan un conjunto de rangos de lectura.
SimplePlataform	

4.2.4.2.13 lejos.robotics.localization

Soporte de localización

Interfaces	
PoseProvider	Proporciona las coordenadas y dirección.
Clases	
MCLParticle	Representa una partícula para el algoritmo de filtrado de partículas.
MCLParticleSet	Representa un conjunto de partículas para el algoritmo de filtrado de partículas.
MCLPoseProvider	
TachoLocalizar	Una extensión abstracta del TachoNavigator

4.2.4.2.14 lejos.robotics.mapping

Una nueva propuesta incompleta para la navegación

Interfaces

RangeMap	Interfaz que permite determinar el rango de búsqueda en un mapa.
Clases	
LineMap	Un mapa de una habitación o de un espacio cerrado, representado por segmentos de línea.

4.2.4.2.15 lejos.robotics.navigation

Interfaces	
Pilot	Conjunto común de funciones para ser utilizadas por las clases de navegación.
Clases	
CompassPilot	Un piloto que realiza un seguimiento de la dirección con un sensor brújula.
SimpleNavigator	Seguimiento de la posición del robot.
TachoPilo	Software abstracto del mecanismo pilot de un robot NXT.

4.2.4.2.16 lejos.robotics.proposal

Interfaces	
ArcPilot	Un piloto de mejora de que es capaz de viajar en los arcos.
ArcRotatePilot	Crea un conjunto de puntos conectados por líneas rectas que conducen de un lugar a otro sin chocar con geometría asignada.
BasicPilot	Esta clase guía al piloto hacía el destino.
PathFinder	
PoseController	
RotatePilot	
Clases	
ArcPoseController	Dirigir a un piloto de las coordenadas actuales a un conjunto de coordenadas destino.
CarefulDifferentialPilot	
DeadReckonerPoseProvider	Seguimiento de coordenadas mediante navegación por estimación, mediante el control de los movimientos de piloto.
DifferentialPilot	
GPSPoseProvider	Probablemente trabaja con datos javax.
MapPathFinder	Buscador de rutas que tiene un mapa y un rango de lecturas tomadas.
SimplePathFinder	Buscador de rutas muy simple.
SteeringPilot	Mecanismo de dirección similar a un coche.
UpdateablePose	Representación de la posición de un robot.
WayPoint	
Clases	
DestinationUnreachable Exception	

4.2.4.2.17 lejos.util

Clases de utilidad

Interfaces	
TimerListener	Atender los eventos del tiempo.
Clases	
Assertion	Clase que se utiliza en la depuración para probar las afirmaciones.
ButtonCounter	Clase para los datos de entrada usando el teclado NXT.
Datalogger	Almacena los valores transmitidos vía Bluetooh o USB.
DebugMessages	Para testear los programas y mostrar mensajes.
Delay	Colección de rutinas de retraseo.
KalmanFilter	
LUDecomposition	Descomposición LU.
Matrix	
Stopwatch	Tiempo de seguimiento (en milisegundos)
TextMenu	Mostrar lista de elementos
Timer	Objeto Timer.

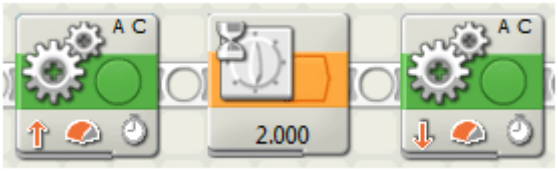
4.2.4.3 NXT-G

NXT-G es un software basado en iconos, lo que permite a los estudiantes programar en un instante. Está basado en el motor LABVIEW de National Instruments [28]. Se trata de un software parecido al MowayGUI, ambos son dos software creados a medida para estos kits de robótica.

A continuación se muestran pequeños programas para mostrar cómo se puede programar un robot LEGO Mindstorms utilizando este software.

4.2.4.3.1 Movimientos

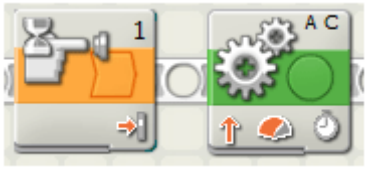
Se programa el robot para que se mueva hacia adelante, espere y se mueva hacia atrás.



Para realizar este programa se necesita un robot NXT con dos motores, un motor conectado al puerto A y otro motor conectado al puerto C.

4.2.4.3.2 Capturar el estado de los sensores

Este programa esperará a que se presione el sensor de contacta para mandar al robot moverse hacia adelante.



Para realizar este programa se necesita conectar un motor al puerto A, otro motor al puerto C y un sensor de contacto al puerto 1.

4.2.4.3.3 Bucles

Este programa repetirá de manera infinita el programa anterior.

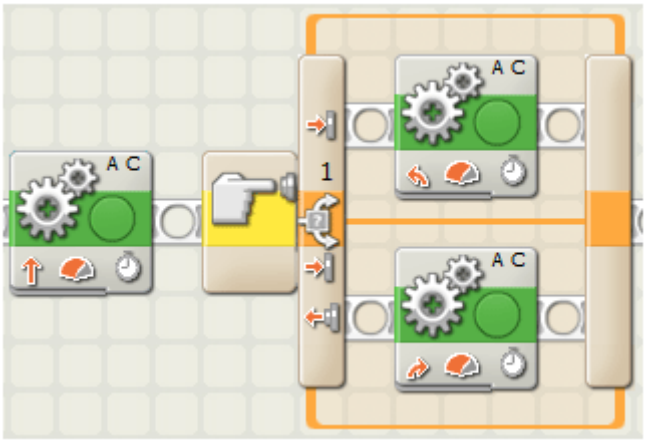
4.2.4.3.4 Sensor de luz

Este programa se moverá hasta que el robot encuentre una línea negra.

Para realizar este programa s necesita conectar un motor al puerto A, otro motor al puerto C y un sensor de luz al puerto 3. Además se necesita un cinta aislante negra para dibujar una línea.

4.2.4.3.5 Switch

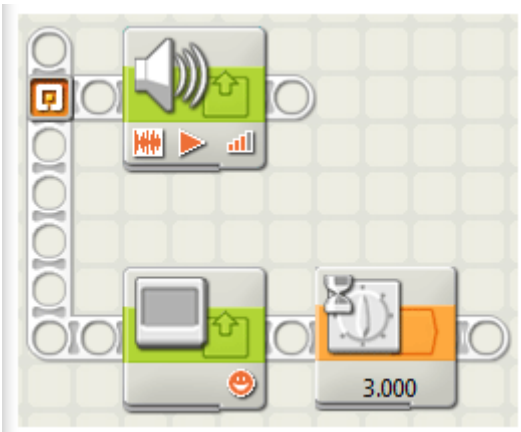
El robot comenzará con un movimiento hacia adelante, si el sensor de contacto está presionado, el robot girará hacia la izquierda, si se libera el sensor de contacto, el robot girará hacía la derecha.



Para el desarrollo de este programa se necesita conectar al robot un motor al puerto A y otro motor al puerto C, y un sensor de contacto al puerto 1.

4.2.4.3.6 División de tareas

Este programa se puede ver cómo crear dos tareas.



4.2.4.4 LENGUAJE DE PROGRAMACIÓN ELEGIDO

Tras analizar estos tres lenguajes de programación, NXC, NXJ y NXT-G, el lenguaje elegido para preparar los robots LEGO para las pruebas robóticas es el lenguaje basado en C, NXC.

Uno de los objetivos de este proyecto es trabajar con diferentes entornos de programación. Con la herramienta Moway se ha trabajado con un entorno gráfico o icónico y con LEGO se ha trabajado con un lenguaje textual. Además tanto la prueba de velocistas como de rastreadores, exigen una programación más exhaustiva y el lenguaje NXT-G resultaba demasiado básico para programar soluciones competitivas.

La documentación existente sobre el lenguaje NXC y la comunidad de NXC es mayor que en NXJ, de ahí que el aprendizaje del lenguaje NXC sea más fácil. Además los conocimientos previos de lenguaje de programación como C son mayores que Java, por lo tanto, el lenguaje NXC se adapta mejor a las necesidades y de ahí que sea con el que se va trabajar.

4.2.5 SOFTWARE UTILIZADO

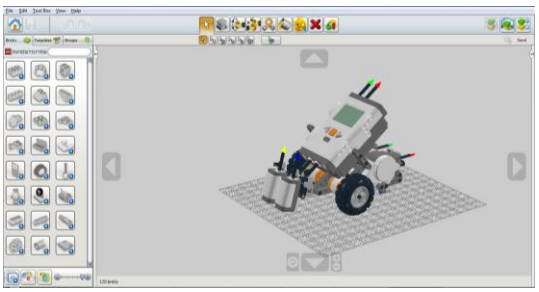
La preparación de robots con el kit LEGO Mindstorm requiere dos tareas principales, montaje del robot y programación del robot. Cada una dos tareas con sus respectivas fases.

Para el diseño del montaje existen varias herramientas, en este proyecto se ha elegido el software **Lego Digital Designer**, LDD, que permite diseñar en 3D el robot y crear un manual para su montaje. Para la programación de los robots con NXC, se puede utilizar cualquier editor y compilador, en este proyecto se ha trabajado con el software **Bricx**, se trata de un editor que incorpora un compilador y un protocolo de comunicación con el robot, permitiendo la creación del programa, la compilación y la subida al robot.

4.2.5.1 LDD

Se trata de un software de LEGO que permite crear de manera virtual cualquier cosa. Está divido en tres secciones, Factory, donde están disponibles todas las piezas clásicas. Mindstorms, que incluye las piezas del set del mismo nombre e incluye entre otras cosas, motores, sensores y varios tipos de engranajes. Y finalmente Creator, el popular set que incluye todo tipo de vehículos y cosas raras como dinosaurios y castillos.

El programa se divide en dos partes, la barra de herramientas situada a la izquierda, con las piezas que se pueden ir incorporando al diseño y la parte de la derecha, el lienzo 3D en el que se ve el montaje.



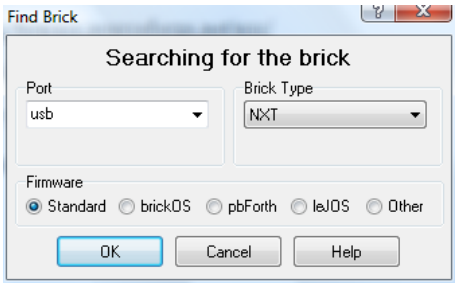
En la parte derecha superior hay tres iconos, se corresponden con los tres modos, el modo construcción que es la vista que se ve en la imagen anterior, el modo vista, que se ve el robot integrado en un paisaje y el modo manual de construcción.

Este software es una buena herramienta para diseñar un robot, pero cuenta con alguna limitación: no existen todas las piezas que viene en el kit educacional o el manual de montaje a veces es complicado de seguir.

4.2.5.2 Bricx Command Center

Bricx Command Center es un software gratuito que se puede descargar desde la web del proyecto NXC (**http://bricxcc.sourceforge.net/nxc/**). Como ya se ha dicho, facilita la vida del programador con el editor de texto con diferentes colores, compilador incluido y con módulo para la comunicación con el robot.

Al iniciar el programa aparece una ventana en la cual se puede configurar el puerto y el tipo de robot que se va conectar. Si el robot está conectado, en el caso del robot NXT, habrá que seleccionar el puerto USB y el tipo de bloque NXT, una vez seleccionado haciendo clic en el botón OK se entra en el software. Si el robot no está conectado al iniciar el Bricx, para indicar el puerto en el que está conectado, habrá que hacer clic sobre "find brick"  y configurar de nuevo la misma ventana que al principio.



Una vez configurado estas opciones, ya se podrá escribir un nuevo programa. Para ello se crea un nuevo archivo y se guarda con extensión .nxc (importante paso).

Una vez escrito, el primer paso es compilar el programa, para ello bien se selecciona la opción "compile program" o se puls F5. Si el programa está bien escribo, no se devolverá ningún mensaje de error y tan solo quedará descárgalo al NXT.

Para descargar el programa al robot, o bien se selecciona "Download" o se pulsa F6. Para este paso es necesario tener el robot conectado.

4.2.6 VELOCISTAS

La prueba de velocistas es un clásico de las competiciones robóticas. Las normativas pueden variar ligeramente de unos concursos a otros, en este caso, el objetivo es preparar un robot competitivo siguiendo la normativa de la competición Robolid en la que se participó.

El concurso consiste en una carrera de persecución entre dos robots en una pista cerrada, comenzando en puntos opuestos y avanzando en el mismo sentido.

4.2.6.1 Análisis de la prueba

Tras leer la normativa del concurso se han obtenido una serie de restricciones que el robot debe cumplir para poder participar en la prueba.

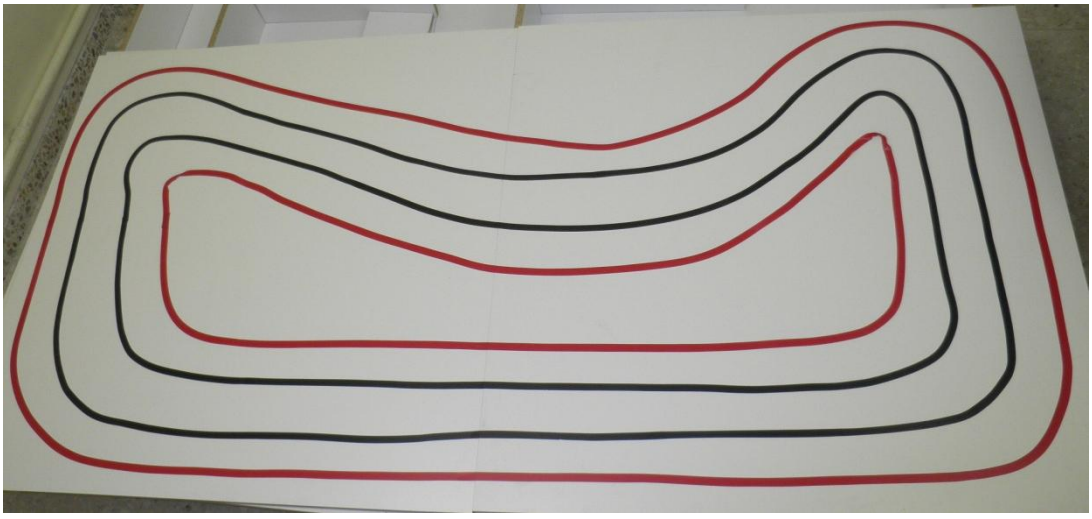
En cuanto al diseño, el robot tiene que tener unas **dimensiones** máximas de 20 centímetros de ancho y 30 de largo. Tiene que ser completamente autónomos.

En cuanto a la solución, el robot tendrá que ser capaz de seguir la **calle de 15±5cm**, delimitada por dos líneas oscuras de 2±0.5cm de anchura cada una, sobre una superficie clara. Para ello podrá seguir una de las dos líneas o ir entre ellas.

Se establecen un límite interior y exterior que el robot no podrá alcanzar. Estos límites se encuentran a 15±5cm de las líneas oscuras. En el caso de alcanzar estas zonas el robot quedará eliminado.

La pista podrá tener curvas con un radio máximo de 40±5cm, en diferentes sentidos. Por lo tanto el robot tendrá que ser capaz de trazar estas curvas.

Las pistas pueden variar, para comprobar el buen funcionamiento del robot se ha utilizado la siguiente pista:



4.2.6.2 Diseño de la solución

En esta prueba el robot se va enfrentar a una serie de dificultades. La primera y más evidente es ser capaz de seguir la calle. La segunda es detectar la línea izquierda que delimita la calle. Y la tercera es detectar la línea derecha. Por lo tanto se han establecido tres estados para el robot:

- ✓ Estado 1. Seguir calle.
- ✓ Estado 2. Detectar línea izda.
- ✓ Estado 3. Detectar línea dcha.

El estado inicial o de arranque será el de seguir calle (1). La percepción que el robot tiene de la pista es a través de los dos sensores de luz que se traduce en porcentaje de luz. Para trabajar con estos porcentajes se define una función que con la lectura de estos sensores indicará si el robot se encuentra sobre la calle o sobre una línea, en el caso de que se encuentre sobre una línea, la función indicará si se trata de la línea izquierda o derecha.

En el momento que se detecte que el robot está sobre la línea derecha el robot tendrá que ser capaz de girar hacia la izquierda hasta dejar de estar sobre la línea. En el caso de que el robot esté sobre la línea izquierda, el robot tendrá que ser capaz de girar hacia la derecha hasta dejar de estar sobre la línea. Si resumimos los eventos comentados tendremos.

- ✓ Evento A. Sobre la calle.
- ✓ Evento B. Sobre línea izda.
- ✓ Evento C. Sobre línea derecha.

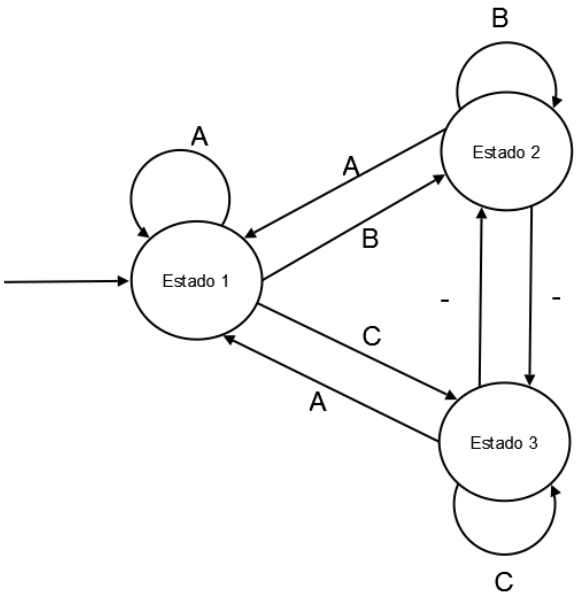
Por último se obtienen las acciones a realizar en cada estado.

- ✓ En el estado 1. Ir hacia adelante.
- ✓ En el estado 2. Girar hacia la derecha.
- ✓ En el estado 3. Girar hacia la izquierda.

En la siguiente tabla se definen las transiciones de estados en función de los eventos.

Evento/Estado	1	2	3
A	1	1	1
B	2	2	-
C	3	-	3

A continuación se muestra gráficamente las transiciones de estados:



4.2.6.3 Diseño del robot

Una vez analizada y diseñada la solución, se decidió trabajar con el kit de LEGO Mindstorms, para ello será necesario contar con:

- ✓ Un bloque NXT.
- ✓ Dos servo motores de Mindstorms
- ✓ Dos sensores de luz de Mindstorms
- ✓ Dos ruedas LEGO de tamaño máximo
- ✓ Una rueda loca que de soporte pero no dificulte los giros.

Además se ha tenido en cuenta las dimensiones máximas, intentando hace un robot lo más acoplado posible, el resultado ha sido un robot de 13cm de ancho y 20cm de largo.

El robot tiene en la parte delantera dos sensores de luz que son los informadores de la situación del robot, el robot está sobre la superficie blanca o está sobre alguna de las líneas negras, estos sensores están conectados a los puertos 1 y 4, el sensor situado a la izquierda se conecta al puerto 4 y el sensor de la derecha al puerto 1.

En la parte lateral se sitúan los servomotores, el motor izquierdo conectado al puerto C y el derecho al A.

A priori parece lo más inteligente utilizar las ruedas de máximo tamaño, ya que por cada rotación el avance será mayor cuanto mayor radio tengan las ruedas. Se han realizado una serie de test que indican que las ruedas de máximo tamaño son muy difíciles de controlar en los giros y por lo tanto el robot pierde mucha velocidad en ellos. Por lo que sale más rentable conectar a los motores A y C ruedas de tamaño intermedio.

Para terminar, el robot tendrá en la parte de atrás una rueda, conocida como rueda loca, ya que su única función es dar soporte al robot, se girará de forma libre, por lo tanto no será un impedimento para los giros del robot.

Para diseñar y montar el robot se ha utilizado la herramienta LEGO Digital Designer que permite diseñar de forma virtual el robot y crear un manual para su posterior montaje. Se incluye en el ANEXO 1, un manual de referencia para montar el robot.

4.2.6.4 Implementación de la solución

En el ANEXO 2 se incluye el código fuente utilizado para esta prueba. A continuación se explica dicho código.

El programa de velocistas consiste en una tarea principal que realiza una lectura de los sensores, se procesa y se calcula el nuevo estado en función de la entrada y del estado actual, y por último se actúa.

Se definen dos constantes, una para definir la velocidad con la que el robot avanzará hacia adelante cuando se encuentre sobre la calle, “Speed”. Y otra constante, “GiroSpeed”, que define la velocidad de giro que se utilizará para los giros cuando el robot alcance alguna de las líneas que delimitan la calle.

Se define una variable global de tipo entero, “Negro”, que servirá para guardar en ella el valor captado por el sensor de luz al estar sobre una línea negra delimitadora.

Para calcular el estado en el que se encuentra el robot se utiliza una función, “calcular_estado”. Esta función calcula el estado en el que se encuentra el robot, para ello se le pasa la lectura actual de los sensores de luz.

En el programa principal se define una variable, “estado”, que almacenará el estado devuelvo por la función “calcular_estado”.

Se configuran los sensores, en este programa se configuran los sensores de luz con el foco de luz activo y en modo RAW, es decir el valor obtenido estará en el rango 300 a 800, siendo 300 blanco absoluto y 800 negro absoluto. Se utiliza este modo y no el de porcentajes para tener más precisión.

Para poder adaptarse a las condiciones de la pista, se ha decidido inicializar la variable “Negro” en el momento en el que vaya a comenzar la carrera. Para ello se pide que se coloque el sensor 4 sobre la línea negra y se pulse el botón de encender/entrar para capturar el valor de dicho sensor. Una vez guardado en la variable “Negro”, se pide que el robot se coloque en la posición de inicio de la carrera. En el momento que se vuelva a presionar el botón de encender/entrar, el robot iniciará la carrera.

Durante la carrera el robot pasará por los tres estados definidos, seguir calle (estado1), detectar línea izquierda (estado 2) y detectar línea derecha (estado 3). En el programa se han etiquetado tres sentencias con estas etiquetas, de tal forma que el paso de un estado a otro se realizará a través de la función goto.

Al comenzar la carrera la primera sentencia de ejecutarse será la etiquetada con estado1, esta sentencia manda a los motores ir hacia adelante a la velocidad definida en la constante “Speed” y con los motores sincronizados.

Cuando el robot detecte una línea, cambiará de estado y el programa irá a la sentencia etiquetada con el nombre del estado, es decir, estado2 o estado3, el robot girará hasta volver al estado 1 e ir a la sentencia estado 1.

Con las etiquetas se establece un bucle infinito, por lo tanto será necesario pulsar el botón de volver para parar la ejecución.

4.2.7 RASTREADORES

4.2.7.1 Análisis de la prueba

Tras leer la normativa del concurso se han obtenido una serie de restricciones que el robot debe cumplir para poder participar en la prueba.

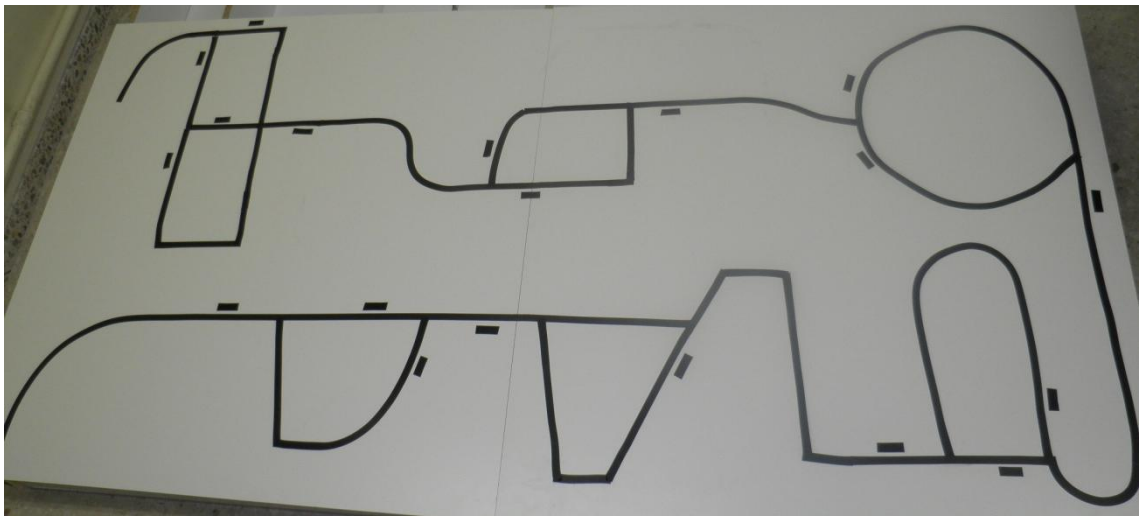
En cuanto al diseño del robot, las restricciones del robot son las mismas que para la prueba de velocistas.

En cuanto a la solución, el robot tendrá que ser capaz de seguir la línea negra de 2±0.5cm de anchura sobre una superficie clara.

El robot tendrá que ser capaz de leer las marcas de bifurcación, éstas se encontrarán a una distancia entre 10 y 15cm de la bifurcación, será una línea de 2±0.5cm de anchura y 5±1cm de longitud, separadas entre 1 y 2cm de la línea principal. Las marcas se encontrarán a ambos lados de la línea.

La pista podrá tener curvas de cualquier ángulo, así como giros poligonales nunca menores de 90º. Por lo que el robot tendrá que ser capaz de trazar tanto las curvas como los giros poligonales.

Las pistas pueden variar, el robot desarrollado para esta prueba tuvo que ser capaz de seguir la siguiente pista.



4.2.7.2 Diseño de la solución

Las dificultades a las que se presenta el robot son más complejas que en la prueba de velocistas. La primera dificultad es ser capaz de seguir la línea, otra dificultad es la de ser capaz de detectar una bifurcación y tomar el camino correcto. También tendrá que ser capaz de detectar ángulos rectos.

Si convertimos estás dificultades en estados, el robot en una prueba de rastreadores pasará por cuatro estados.

- ✓ Estado 1. Seguimiento de línea.
- ✓ Estado 2. Detección de bifurcación.

- ✓ Estado 3. Tomar bifurcación.
- ✓ Estado 4. Ángulo recto.
- ✓ Estado 5. Fuera de la línea mientras se seguía la línea.
- ✓ Estado 6. Fuera de la línea mientras se localiza una marca de bifurcación.
- ✓ Estado 7. Fuera de la línea mientras se toma la bifurcación.

Las acciones que se realizarán en cada uno de estos estados son:

- ✓ En el estado 1. Seguimiento de línea.
- ✓ En el estado 2. Seguimiento de línea.
- ✓ En el estado 3. Seguimiento de línea.
- ✓ En el estado 4. Girar hasta encontrar la línea perpendicular.
- ✓ En el estado 5. Encontrar la línea.
- ✓ En el estado 6. Encontrar la línea.
- ✓ En el estado 7. Encontrar la línea.

El estado inicial es el estado 1. El robot seguirá en este estado hasta que detecte una marca de bifurcación. En este evento hay que tener especial cuidado, ya que habrá que diferenciar entre marcas de bifurcación, ángulos rectos e incluso manchas. Para ello crearemos tres eventos, marca esporádica, marca de bifurcación y ángulo recto.

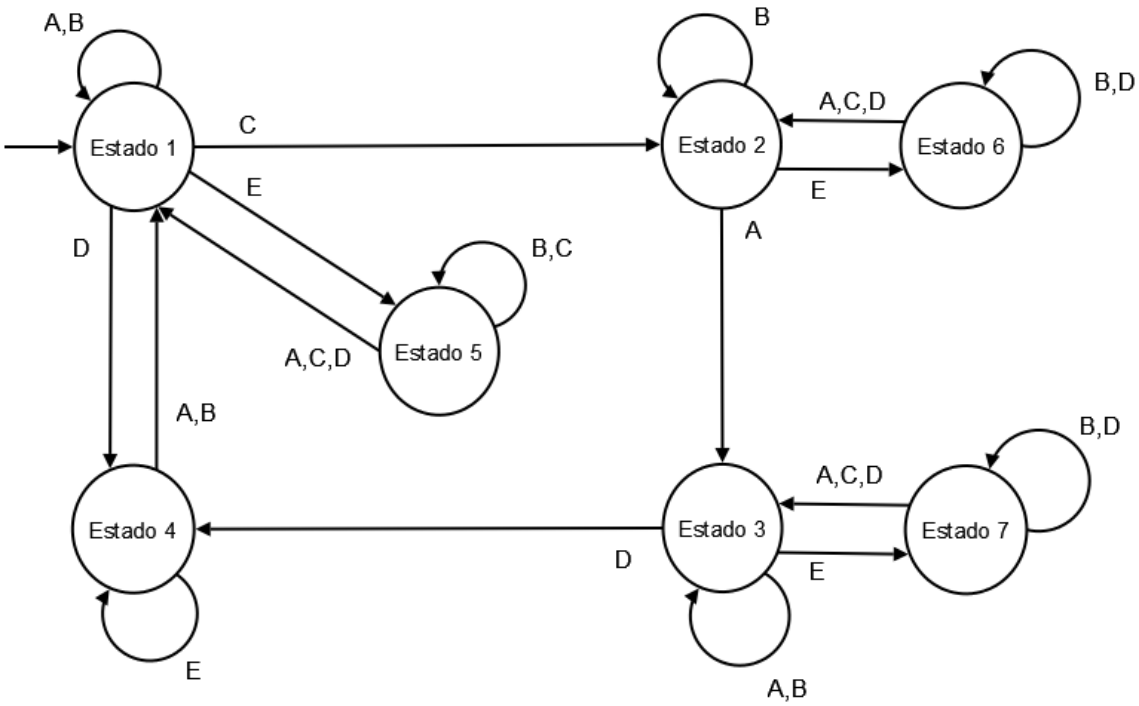
Para diferencias entre estas marcas esporádicas, marcas de bifurcación y ángulo recto, se tomarán 3 medidas de los sensores de luz. Para separar una medidas de otras se utilizará el sensor de rotación.

Los eventos quedan como se expone a continuación:

- ✓ Evento A. Línea normal.
- ✓ Evento B. Marca esporádica.
- ✓ Evento C. Marca de bifurcación.
- ✓ Evento D. Detección de ángulo recto.
- ✓ Evento E. Detección de blanco.

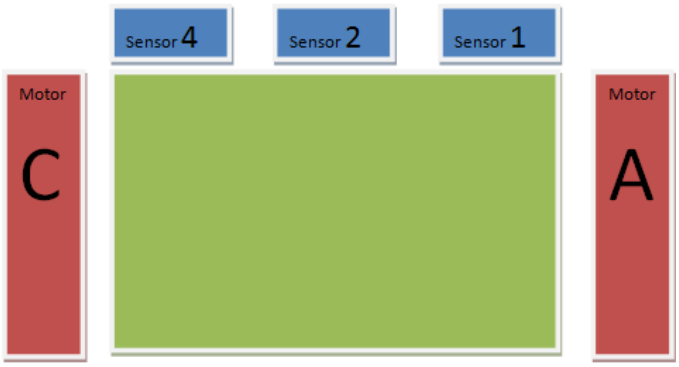
La máquina de estados y su tabla de transiciones:

Evento/Estado	**1**	**2**	**3**	**4**	**5**	**6**	**7**
A	1	3	3	1	1	2	3
B	1	2	3	1	5	6	7
C	2	-	-	-	1	2	3
D	4	-	4	-	1	2	3
E	5	6	7	4	5	6	7



4.2.7.3 Diseño del robot

Una vez analizada y diseñada la solución, se ha decidido trabajar con el mismo robot que para la prueba de velocistas, añadiendo un nuevo sensor de luz, y conectarlo al puerto 3.



4.2.7.4 Implementación de la solución

En el ANEXO 3 se incluye el código fuente utilizado para esta prueba. A continuación se explica dicho código.

Al igual que en la prueba de velocistas, el programa consiste en una tarea principal que se encargará de procesar los datos recogidos de los sensores y calcular el evento y cambiar de estado y por último actuar.

Se utiliza una variable, "Negro" para almacenar el valor capturado al iniciar el programa con el sensor del medio sobre la línea. Se utiliza otra variable "Bifur", para almacenar el sentido de la bifurcación, derecha (dcha) o izquierda (izda).

Para calcular el evento se utiliza una función "calcular_evento", a esta función se le pasa como argumentos los valores recogidos de los sensores. Se comprueba si los sensores de los laterales leen negro o no, en el caso de que lean se repetirá de nuevo la medición para descartar que se trate de un hueco esporádico (mancha), si se sigue leyendo negro por ese sensor, se volverá hacer una lectura para diferenciar ángulos rectos de marcas de bifurcación. Para esto se utiliza el sensor de rotación, se reinicia el contador de rotaciones del motor A y cuando la cuenta llegue a 20 se volverá a obtener una lectura para descartar hueco esporádico, cuando la cuenta sea igual a 30 se leerá de nuevo para diferencias ángulos rectos de marcas de bifurcación.

Además se definen tres funciones, "girar_izda" y "girar_dcha" que se utilizarán para tomar bifurcaciones y ángulos rectos, y "encontrar" que se utilizará para volver a la línea cuando el robot se salga de ella.

Al igual que en la prueba de velocistas, se utilizarán los sensores en modo RAW y al comenzar la prueba se tomará una medida de la captura del sensor sobre la línea para evitar problemas con la pista.

Durante la ejecución del programa, el robot pasará por todos los estados, para ello se usan las etiquetas, para moverse de un estado a otro dependiendo del evento que se produzca. De esta forma se crea un bucle infinito.

En cada estado, se creará un bucle en el que se saldrá cuando se produzca un evento que tenga como consecuencia una transición de estado.

Tras realizar distintas pruebas, se ha decidido establecer una velocidad de movimiento baja, penalizando el tiempo que tardará el robot en realizar el trayecto pero aumentando la fiabilidad en la toma de bifurcaciones, la velocidad establecida es 30 de 100.

5 CONCLUSIONES

En la realización de este proyecto se han llevado a cabo varias etapas, una primera etapa en la cual se estudiaron los diferentes entornos de trabajo y diferentes kits de robótica, una segunda etapa en la que se estudiaron las diferentes competiciones robóticas y las distintas pruebas de dichas competiciones y una etapa final que consiste en la preparación de robots competitivos con los kits de robótica y para unas determinadas pruebas.

Durante la primera etapa se descubrió que existen varios kits de robótica orientados para personas que se quieran iniciar en la robótica. En este proyecto como ya se ha dicho se ha trabajado con el kit de Moway y con el kit Mindstorms de LEGO.

El kit de Moway es muy completo, ya que en el robot vienen incorporado varios sensores que permiten que este robot se pueda adaptar a diferentes competiciones, además el módulo de radio frecuencia le da una calidad extra que otros kits carecen. Además el software Moway GUI permite comenzar a trabajar de forma rápida con estos robots y realizar pequeños programas muy interesantes como son las prácticas que se han presentado anteriormente.

Sin embargo se trata de un robot muy limitado en cuanto a hardware, que impide volcar en él programas pesados debido a su poca memoria y además su sistema motriz no es muy potente. Existen varias pruebas en las cuales el robot de Moway podría participar como velocistas o laberinto. Pero en el primer caso la velocidad alcanzada por el robot no es muy alta, impidiendo que el robot sea competitivo. En el caso del laberinto, los sensores antiobstáculos son perfectos para este tipo de pruebas, pero el límite de memoria hace que sea imposible realizar un programa que resuelva la prueba.

Para la prueba de bailarines, este robot encaja perfectamente, por doble motivo, por un lado por su agilidad al ser un robot muy pequeño y por otro lado por su módulo de comunicación, que permite realizar una comunicación de manera muy sencilla.

El kit de Mindstorms de LEGO se parece al de Moway en la cantidad de sensores que incluye. El kit de LEGO es flexible ya que permite diseñar cualquier robot móvil (o no) y por ello se puede adaptar a varias pruebas. Además es un kit muy extendido, por lo tanto hay una gran comunidad detrás de él, investigando sobre el diseño y programación. En este proyecto se ha trabajado con el lenguaje de programación NXC, creado bajo una comunidad de usuarios.

El principal problema de los robots de LEGO es su falta de agilidad, siendo bastante brusco en sus movimientos. Pese a adaptarse perfectamente a pruebas como rastreadores o velocistas se trata de un robot al que le cuestan los giros, por lo que queda penalizado en el momento que haya que girar. Además el robot tiene algunos problemas a la hora de realizar movimientos rectilíneo sin desviaciones.

Durante la elaboración de este proyecto se ha participado en varias competiciones robóticas en distintos papeles, First Lego League (FLL) (como juez de mesa), Deustobot y Robolid (como concursante) y en el European Open Day on Educational

Robotics en la UPNA (en un stand). En ellas he podido comprobar diferentes competencias (desde un punto de vista educativo) que se pueden trabajar en este tipo de competiciones, tanto en el papel de participante como en el papel de organizadora (juez de mesa en la FLL).

Algunas de las competencias trabajadas son la autonomía, en este proyecto fin de carrera se han adquirido nuevos conocimientos o técnicas de manera autónoma. A la hora de participar en las competiciones se ha tenido que mostrar el trabajo realizado y explicar las características del robot tanto a jurado como a otros concursantes, trabajando la competencia de la comunicación. Se ha tenido que realizar un proyecto informático llevando a cabo los principios y metodologías propias de la ingeniería.

Los resultados obtenidos durante la elaboración de este proyecto fin de carrera se pueden catalogar como positivos. Se ha conseguido trabajar y preparar robots con diferentes entornos de programación. Se han trabajado distintas pruebas de las competiciones y se ha podido ir a varias competiciones robóticas.

Los resultados de la competición de Robolid fueron muy positivos y sobre todo enriquecedores, el nivel de la competición era bastante alto y el robot de LEGO era el único robot preparado con un kit (comercial).

En la prueba de velocistas se demostró que la fiabilidad del robot era máxima, no llegando a tocar en ningún momento las líneas rojas que delimitan la pista y que suponen la descalificación, por ello pese a ser menos rápido que otros robots, el robot pudo realizar una clasificación positiva.

En cuanto a la prueba de rastreadores, prueba de mayor nivel, el robot volvió a demostrar ser más fiable que la media de robots, reconociendo la gran mayoría de las marcas de bifurcación y tomando con éxito las bifurcaciones. El gran problema que presenta el robot en este tipo de pruebas son las circunferencias, ya que para el robot reconocer las marcas de bifurcación en una curva le resulta bastante complejo.

Se han realizado test de pruebas con diferentes tipos de pistas y se ha llegado a la conclusión de que el robot de LEGO no tendría problemas en reconocer ángulos rectos y marcas de bifurcación en líneas rectas, pero sí cuando la marca de bifurcación está en curva. Para este tipo de marcas se necesitaría un robot más pequeño y más manejable.

Este proyecto fin de carrera ha significado a nivel personal un reto, de cómo plasmar los conocimientos adquiridos durante los cursos en el mundo de la robótica. Hasta este proyecto se habían realizado un gran número de créditos prácticos, muchos de ellos de programación en diferentes entornos, pero siempre se trataba de software simulado en un PC. Con este proyecto fin de carrera, se ha desarrollado software que interactúa con hardware y se ven los resultados. Implementar un software que luego se refleja en el comportamiento del robot viendo así los resultados reales es mucho más motivador que las simulaciones. Tras la elaboración de este proyecto creo que la robótica sería una gran herramienta para las clases prácticas que se reciben. La experiencia en las competiciones robóticas ha sido muy enriquecedora, es un encuentro de expertos informáticos que nos dieron muy buenas ideas para mejorar nuestros robots.

Además tras la participación en la competición de Deusto y la organización del "I Encuentro Europeo sobre robótica educativa", se nos invito a participar en un programa de radio y en uno de televisión, siendo una experiencia muy enriquecedora.

6 VIDEO DE LOS RESULTADOS

Se han subido a la web de UPNATV una serie de vídeos con los resultados obtenidos tanto en las pruebas de Moway como en las pruebas de Bailarines, Rastreadores y Velocistas, a continuación se incluyen los enlaces:

- ✓ Sensores infrarrojos anticolisión:

  http://upnatv.unavarra.es/es/pub/anticolisionmoway

- ✓ Sensores de intensidad de luz direccional Moway:

  http://upnatv.unavarra.es/es/pub/luzdireccionalmoway

- ✓ Sensores optorreflectivos infrarrojos para el suelo Moway:

  http://upnatv.unavarra.es/es/pub/velocistasmoway

- ✓ Módulo de radio frecuencia entre robots Moway:

  http://upnatv.unavarra.es/es/pub/relevosmoway

- ✓ Bailarines Moway:

  http://upnatv.unavarra.es/es/pub/bailarinesmoway

- ✓ Velocistas LEGO:

  http://upnatv.unavarra.es/es/pub/velocistasmoway

- ✓ Velocistas LEGO vs Moway:

  http://upnatv.unavarra.es/es/pub/velocistaslegovsmoway

- ✓ Rastreadores LEGO:

7 REFERENCIAS

[1] Arquitas de Tarento, Siglo IV antes de Cristo, "*La paloma*". Ave mecánica que funcionaba con vapor.

[2] Hoffman, 1817, "*El Coco*". Cuento en el que aparece una mujer que parece una muñeca mecánica.

[3] Isaac Asimov, "*Circulo Vicioso (Runaround)*". Relato en el que dos astronautas y un robot son enviados a una estación minera de Mercurio. Los astronautas envían a un pozo de selenio al robot, ya que es capaz de soportar grandes temperaturas, pero no son conscientes de que la fuente de Selenio contiene algún tipo de peligro para el robot. El robot se encuentra en un estado que en los seres humanos se llamaría "embriaguez", y no hace caso a las órdenes de los astronautas (incumpliendo la segunda ley). Así un astronauta se expone a las altas temperaturas, poniéndose en peligro, lo que hace reaccionar al robot, la primera regla provocó que el robot salvará al astronauta y así salir del estado de "embriaguez" y poder culminar con éxito la misión.

[4] Isaac Asimov, "*Yo, Robot*". Colección de relatos que plantean diferentes situaciones a las que tendrán que enfrentarse distintos especialistas en robótica y en las que se plantean paradojas e ingeniosos ejercicios intelectuales que indagan sobre la situación del hombre actual en el universo tecnológico.

[5] Julio Pastor Mendoza, profesor de la Universidad de Alcalá y experto de las competiciones robóticas, "*Participación en competiciones internacionales de robots como forma de potenciar habilidades profesionales básicas"*.

[6] Julio Pastor Mendoza, profesor de la Universidad de Alcalá y experto de las competiciones robóticas, "*Evaluación de las mejoras en la formación en aptitudes y competencias de los estudiantes de ingeniería que participan en competiciones de robots móviles autónomos*".

8 ENLACES DE INTERÉS

"Leyes de la robótica" http://es.wikipedia.org/wiki/Tres_leyes_de_la_rob%C3%B3tica

"Competiciones robóticas" http://www.roboticadeservicios.com/

"Microbot educacional MOWAY", http://www.iberobotics.com/shop/product_info.php?products_id=28

"*Prácticas MOWAY*"

http://www.moway-robot.com/index.php?option=com_content&task=blogcategory&id=52&Itemid=203

"*Manual de usuario MOWAY*"
http://www.bizintekinnova.com/moway/Manual%20usuario%20mOway%202.1.0.pdf

"*NXJ API*", http://lejos.sourceforge.net/nxj.php

"*Proyecto Robocompetences*", http://asimov.depeca.uah.es/robocompetences/

"*NXC Tutorial*", http://bricxcc.sourceforge.net/nbc/nxcdoc/NXC_tutorial.pdf

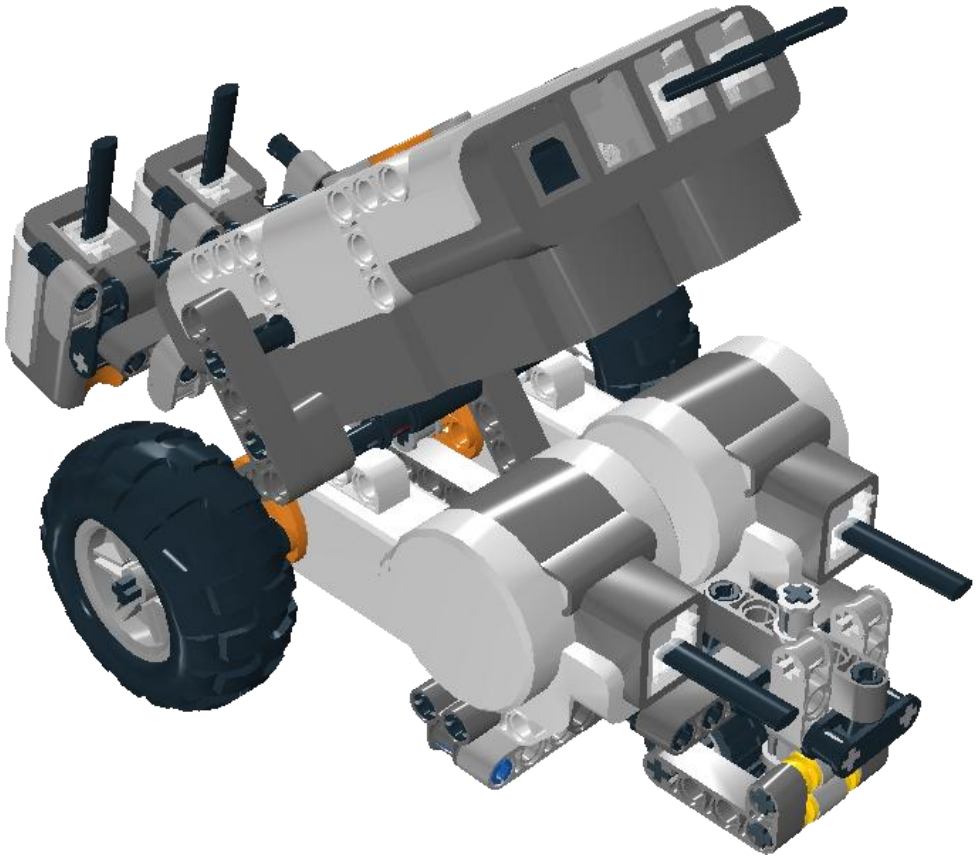
"*NXC Guide",* http://bricxcc.sourceforge.net/nbc/nxcdoc/NXC_Guide.pdf

"Programación NXT", http://www.lrobotikas.net

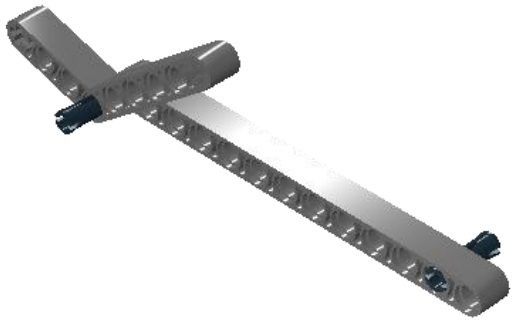
ANEXO 1 MANUAL DE MONTAJE DEL ROBOT NXT

LEGO Digital Designer — Building Instructions

Model Name: **velocista**
Number of Bricks: **120**



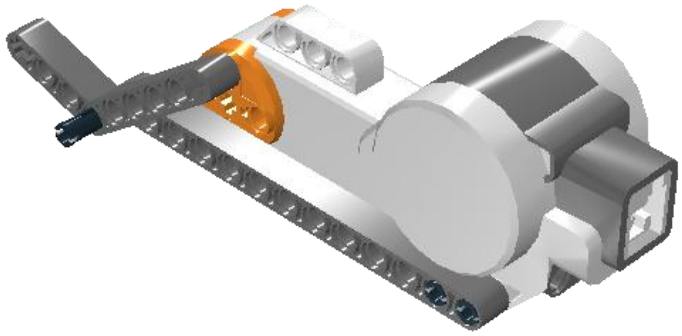
Step 1



Step 2

1 x

1 x

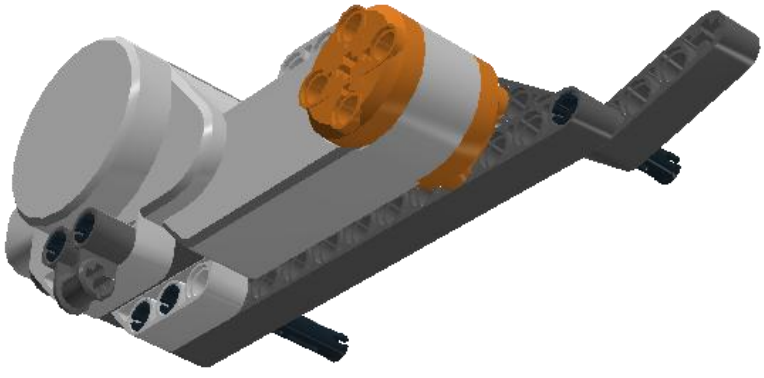


Step 3

2 x

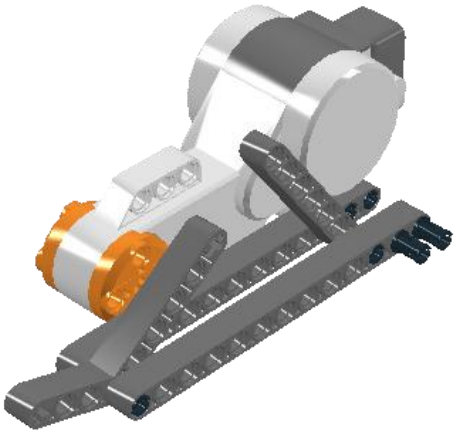
1 x

1 x



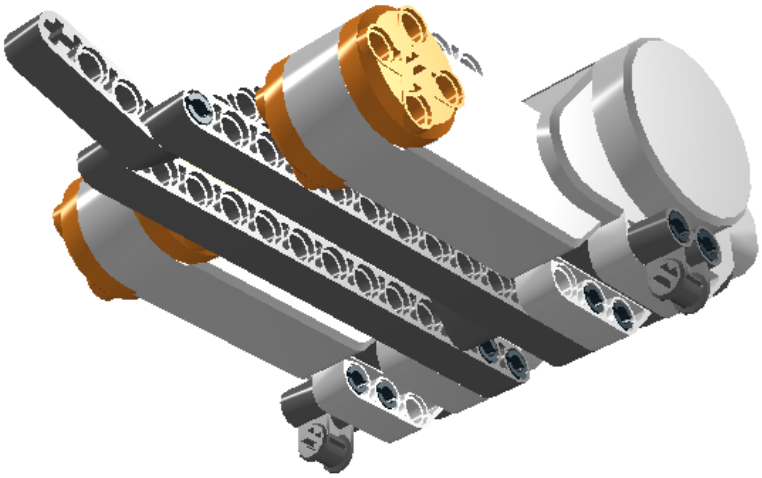
Step 4

1 x 1 x 2 x



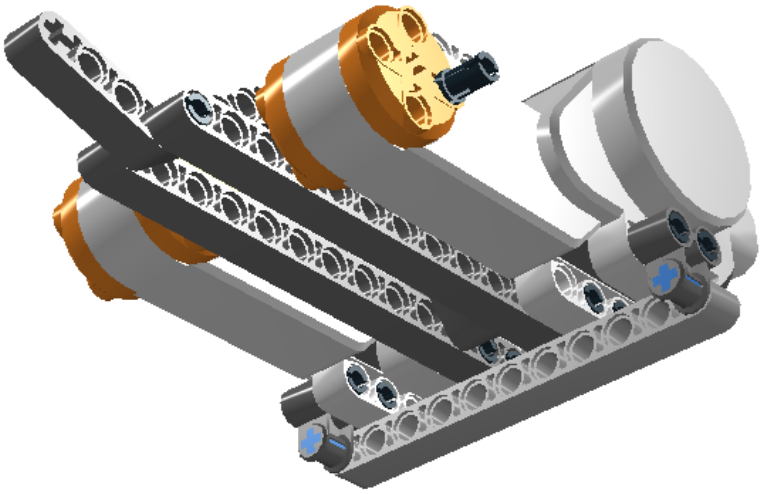
Step 5

1 x 2 x 1 x



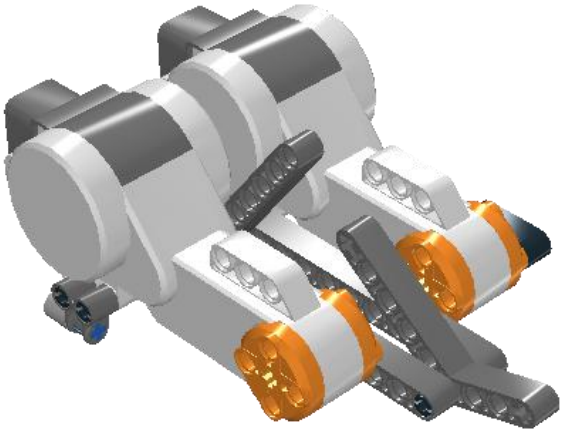
Step 6

1 x 1x 2x



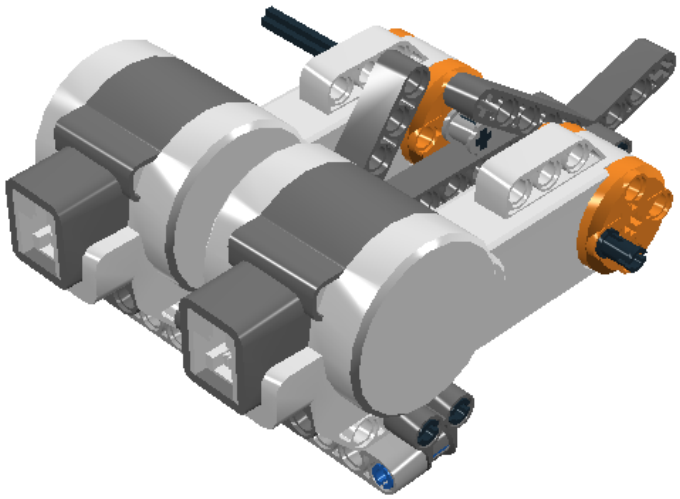
Step 7

1 x 1 x



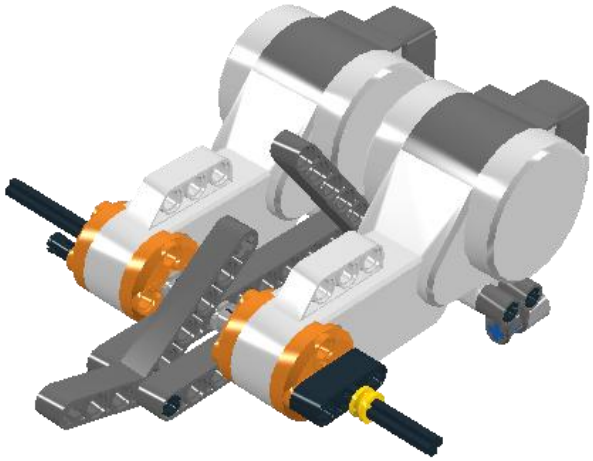
Step 8

1 x 1 x 1 x



Step 9

1 x 1 x 1 x 1 x

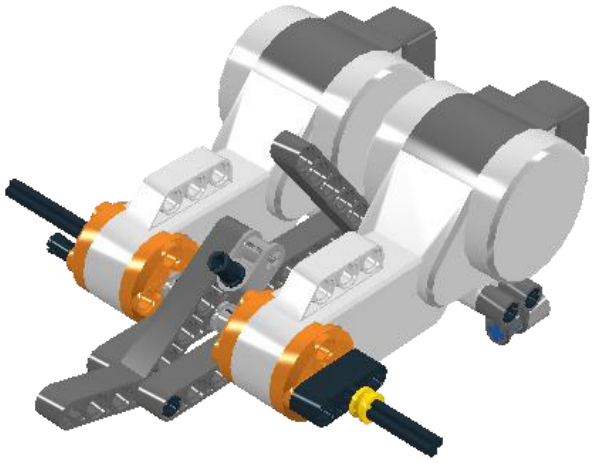


Step 10

1 x 1 x

Step 11

1 x 1 x

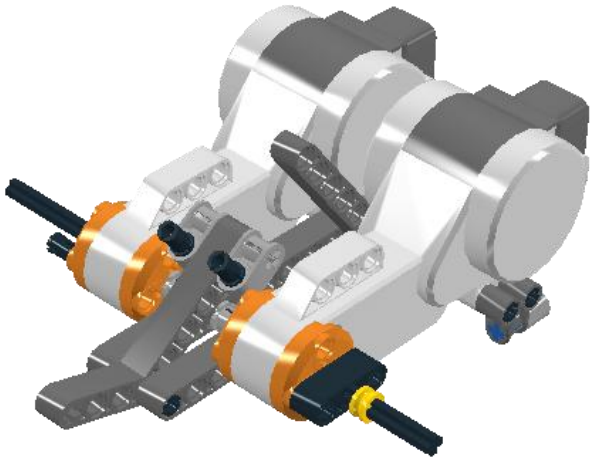


Step 12

1 x 1 x

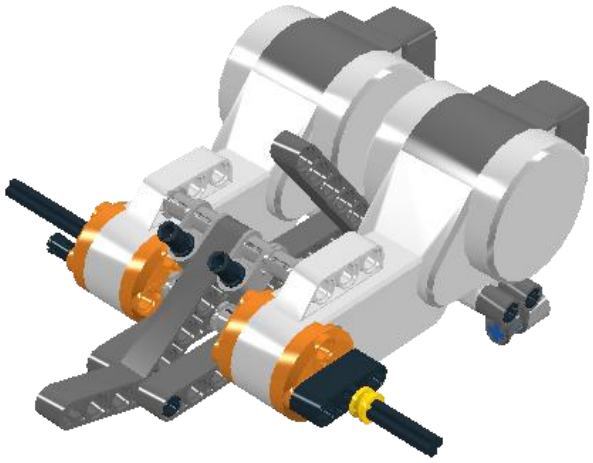
Step 13

1 x 1 x

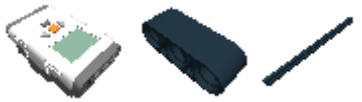
Step 14

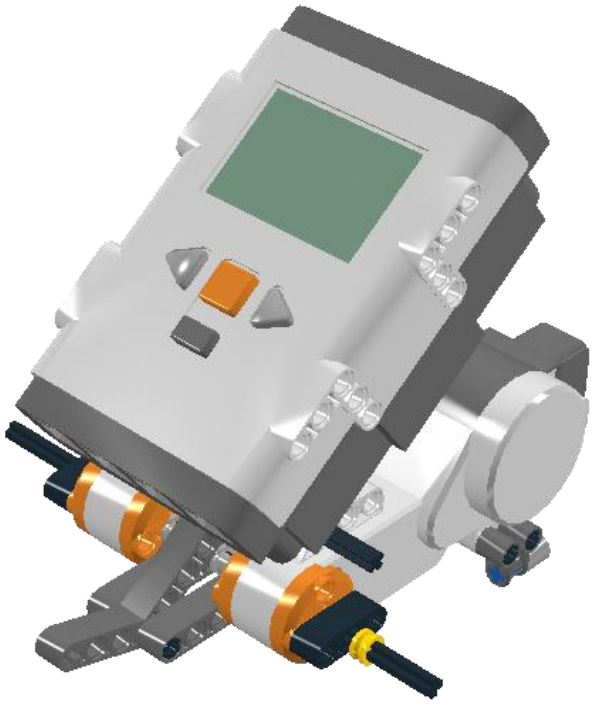
2 x



Step 15

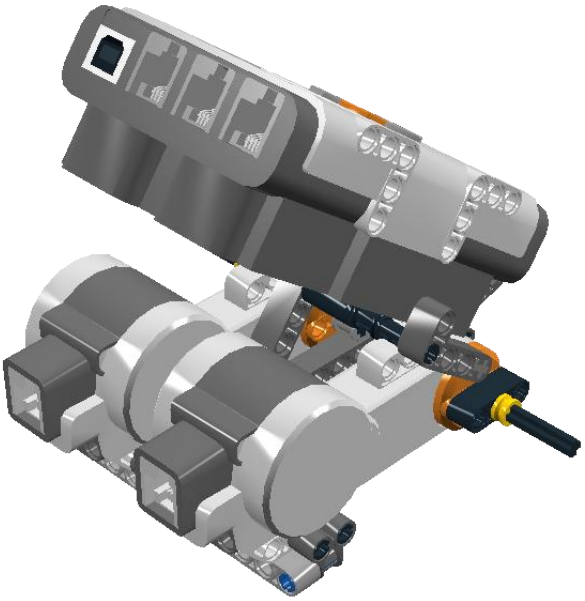
1 x 1 x 1 x





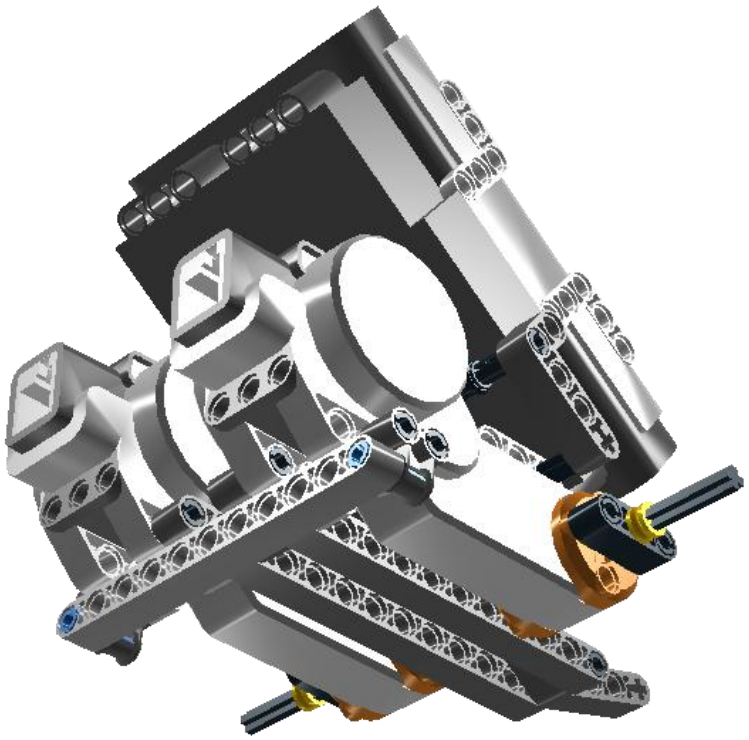
Step 16

1 x 1 x 1 x 1 x



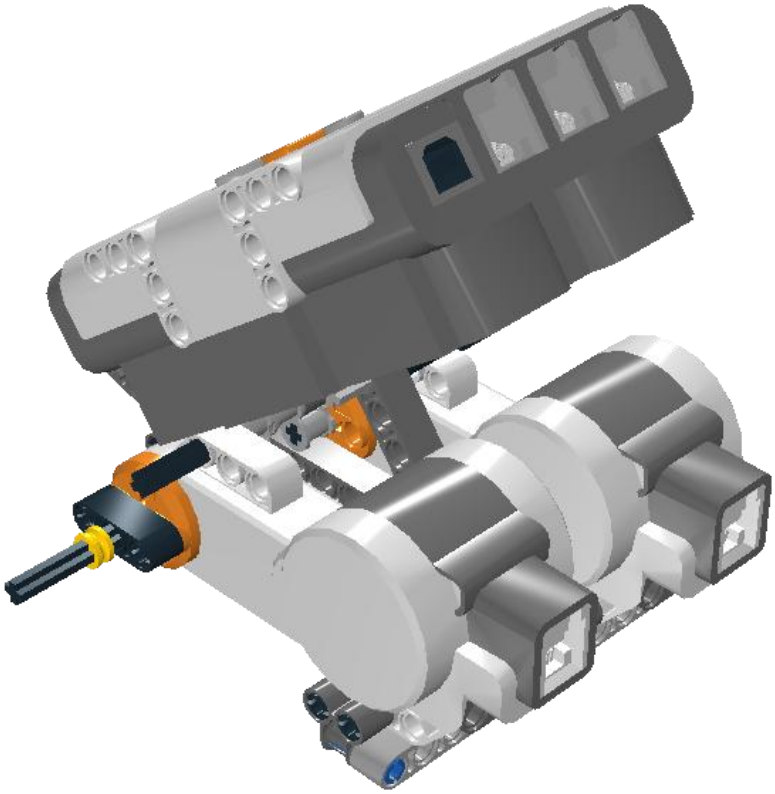
Step 17

1 x



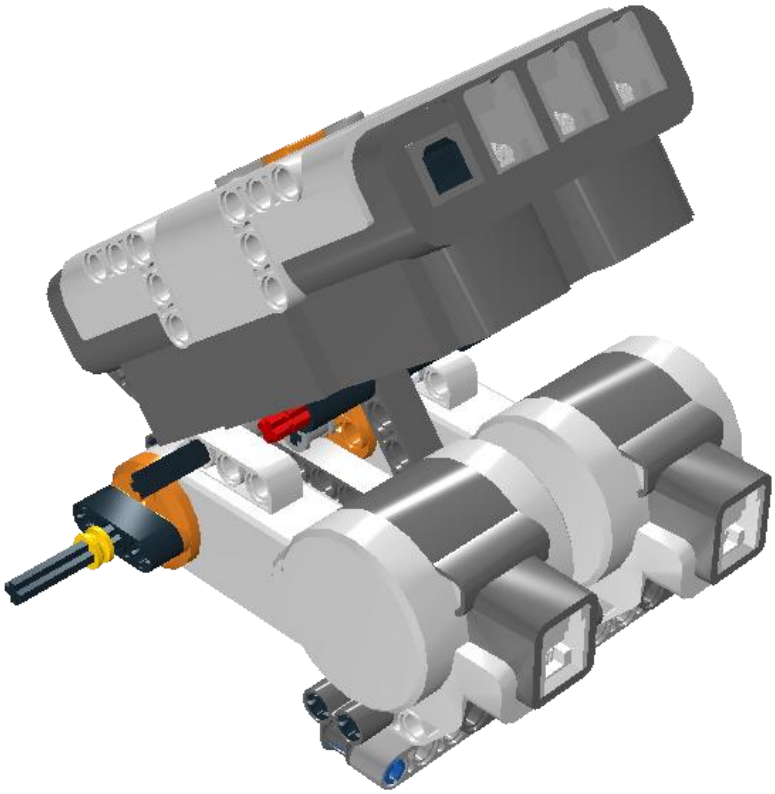
Step 18

1 x



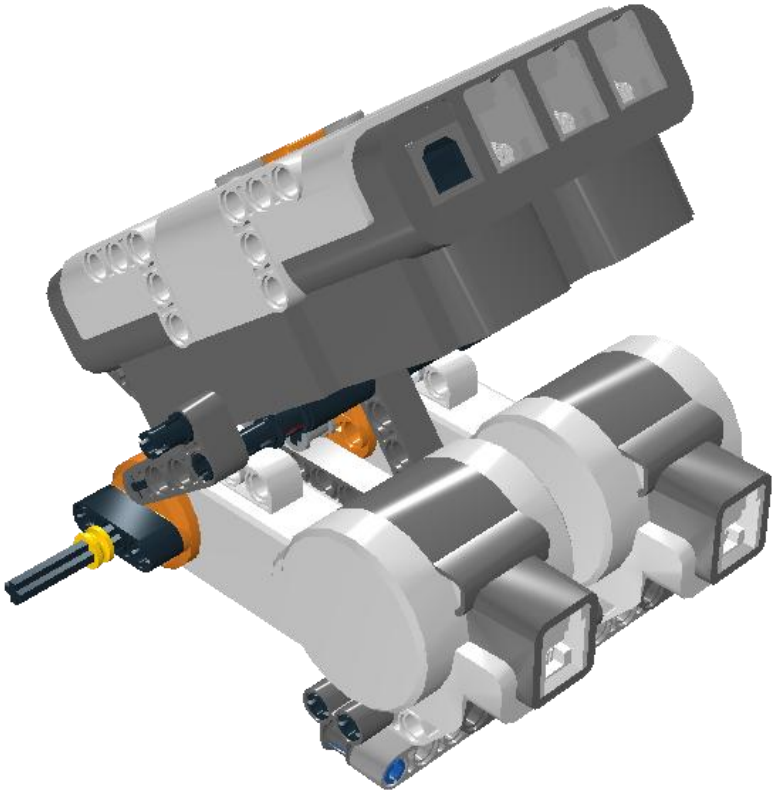
Step 19

1 x 1 x



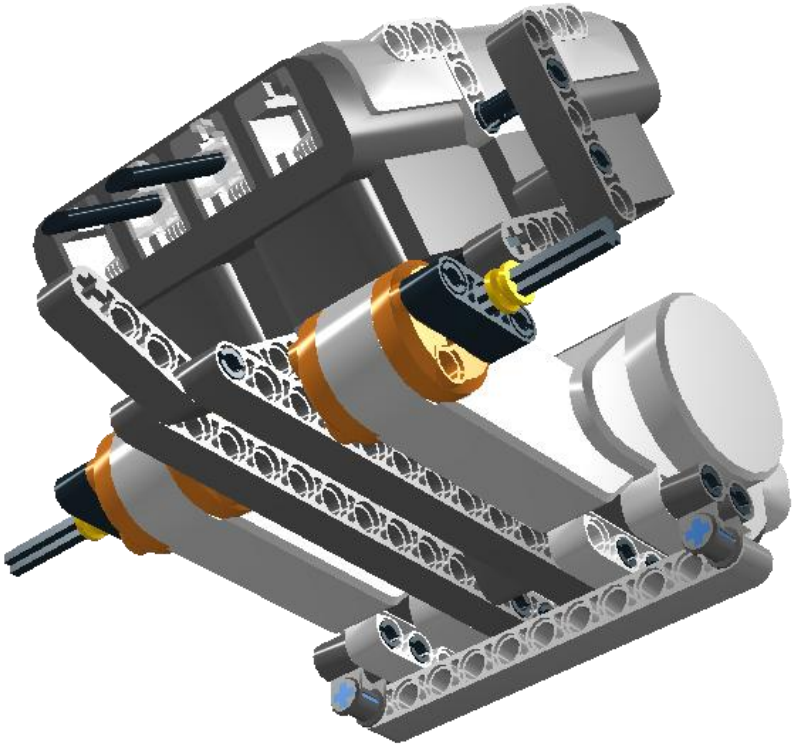
Step 20

1 x 1 x 1 x



Step 21

2 x 1 x 1 x



Step 22

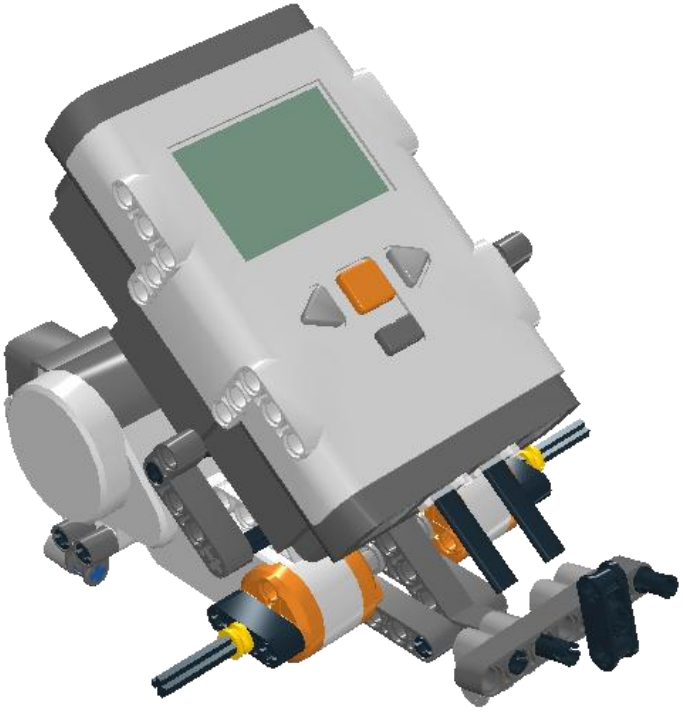
1 x 3 x

Step 23

2 x 1 x 1 x

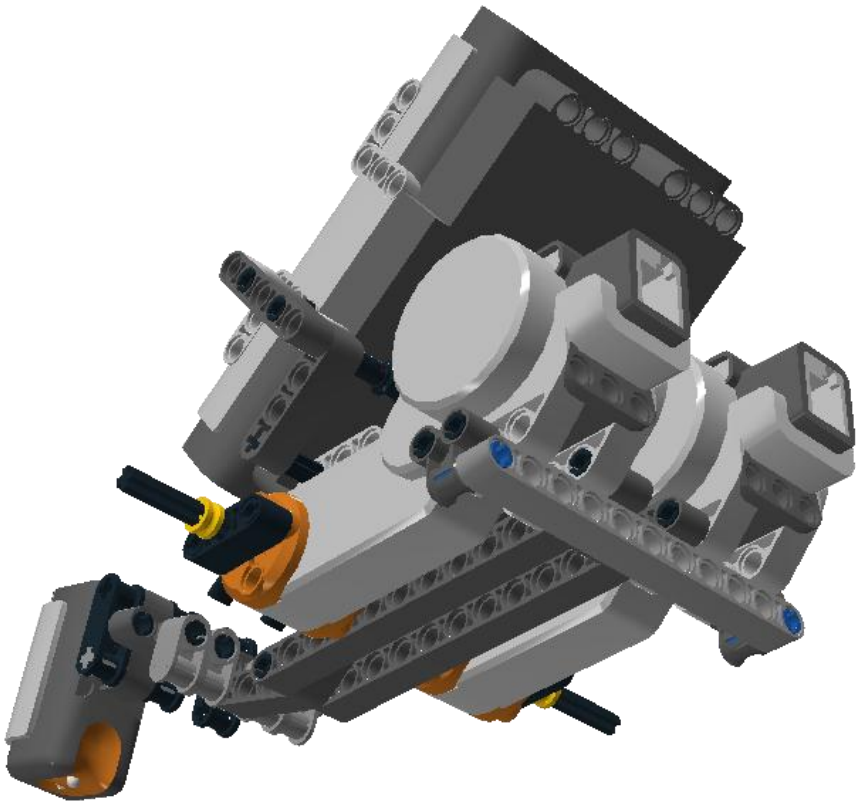
Step 24

1 x 1 x 1 x 1 x



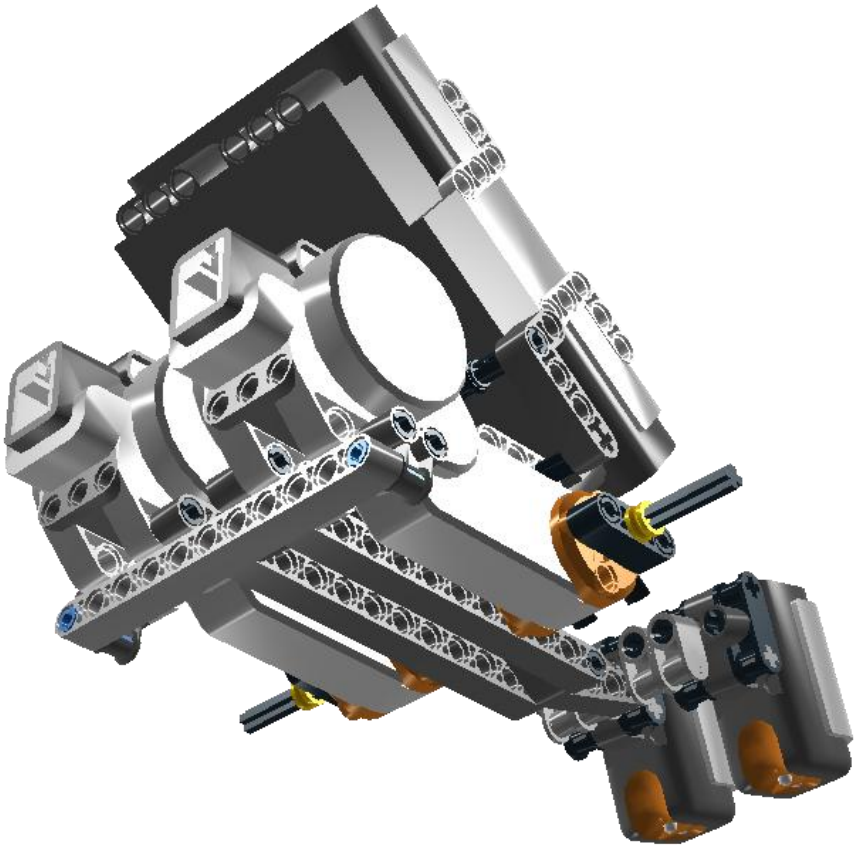
Step 25

1 x 1 x 2 x



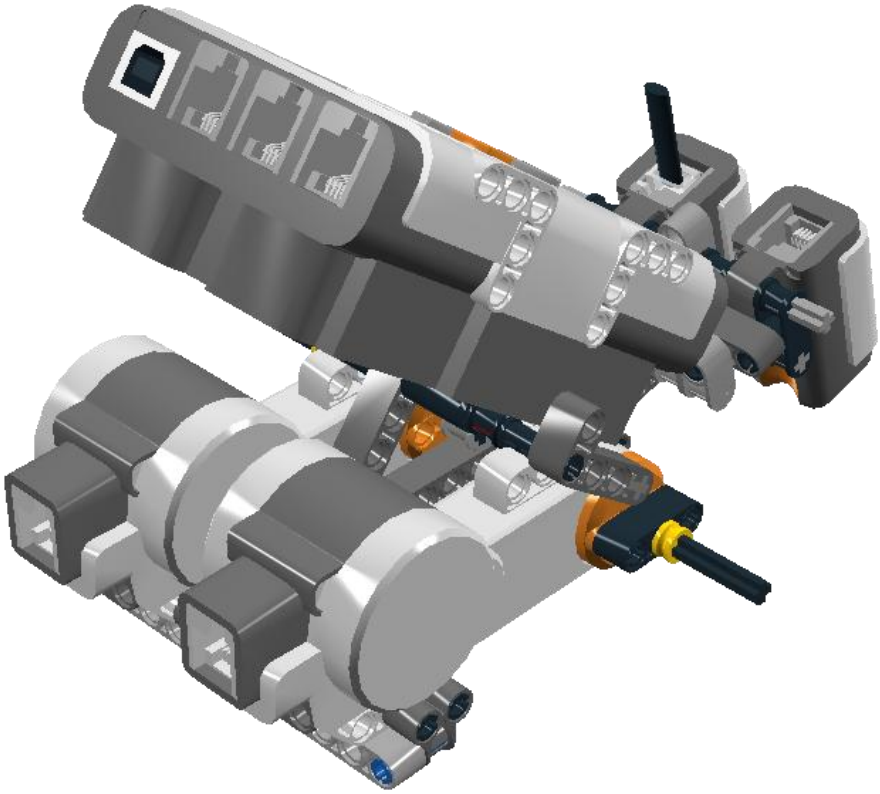
Step 26

1 x 1 x 1 x 1 x



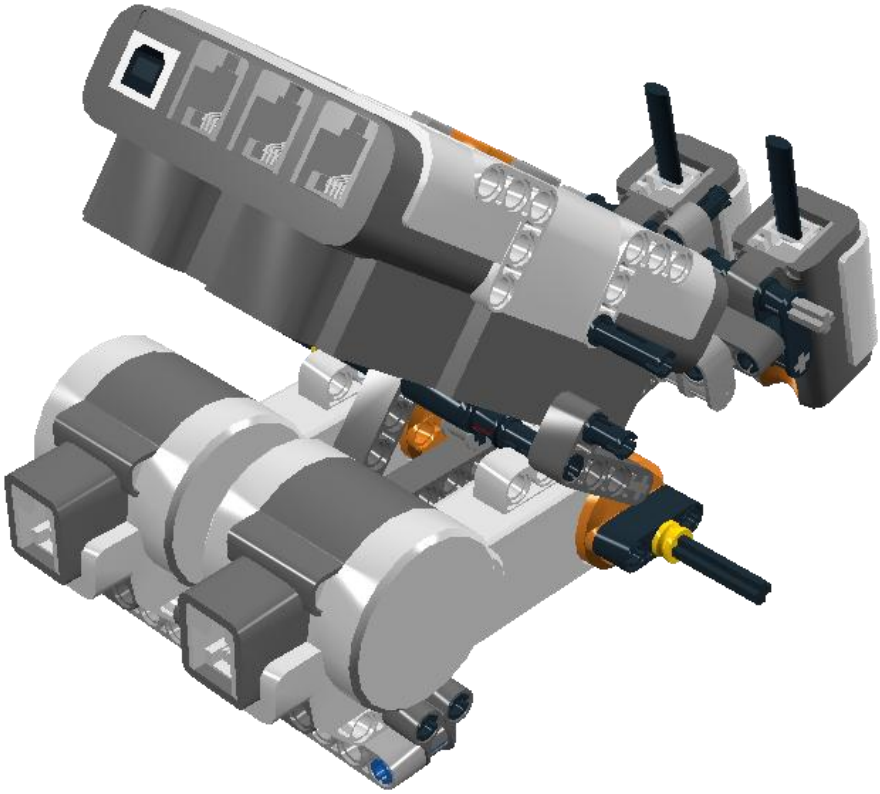
Step 27

1 x 1 x 1 x



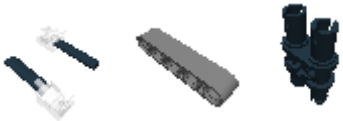
Step 28

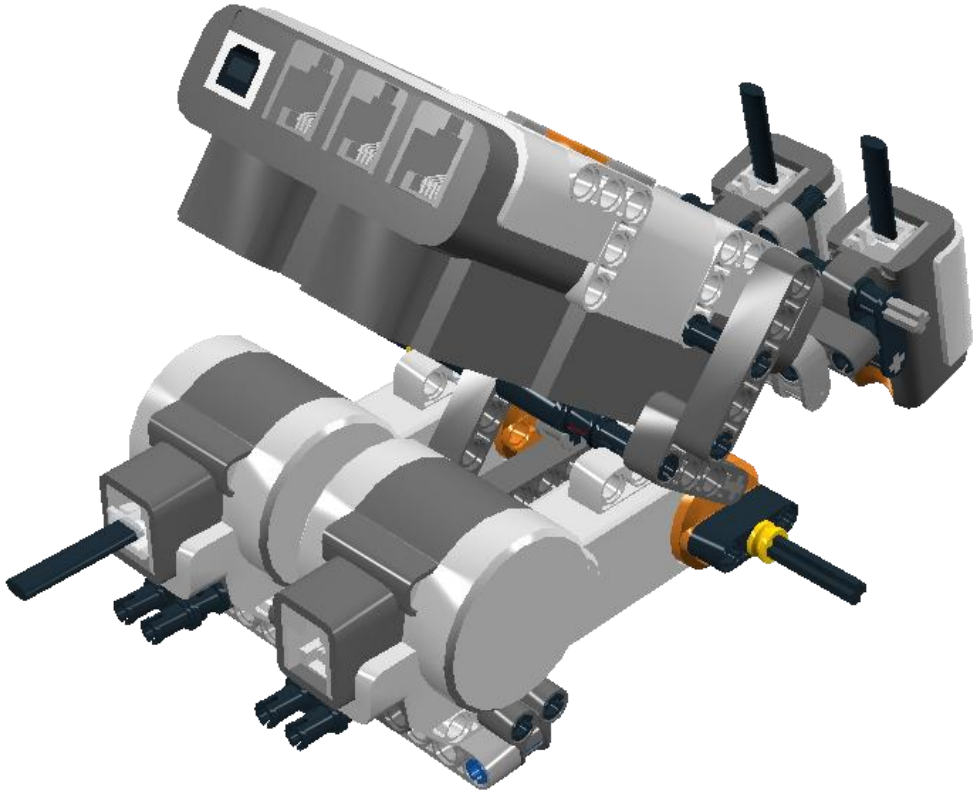
1 x 2x 1 x



Step 29

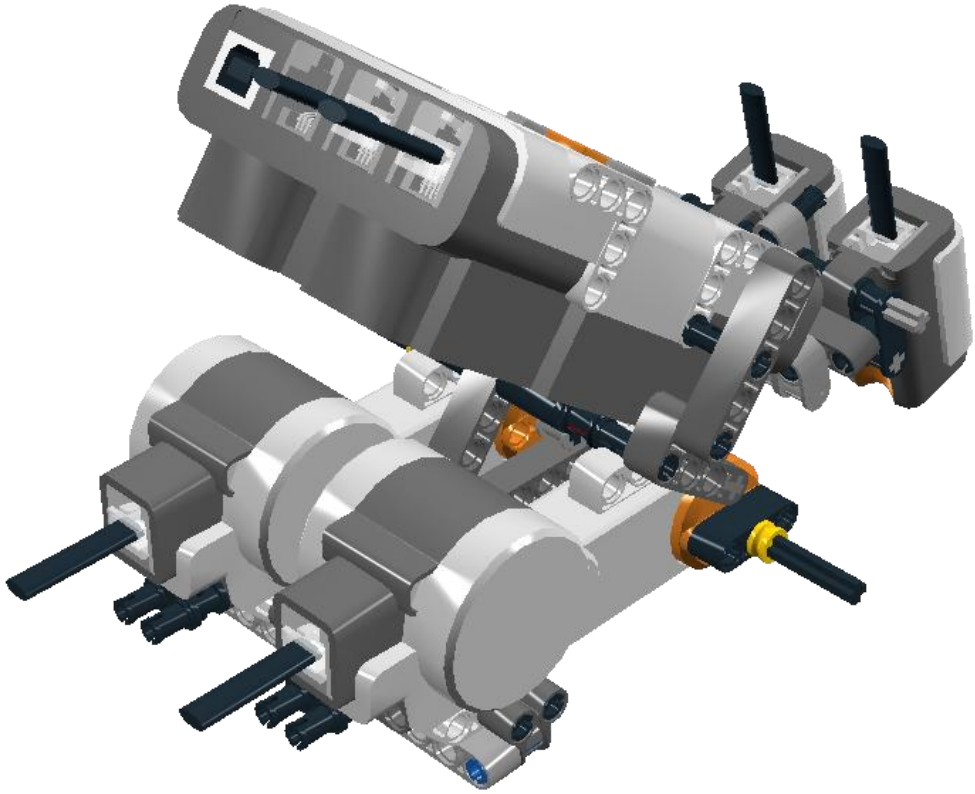
1 x 1 x 2 x





Step 30

2 x

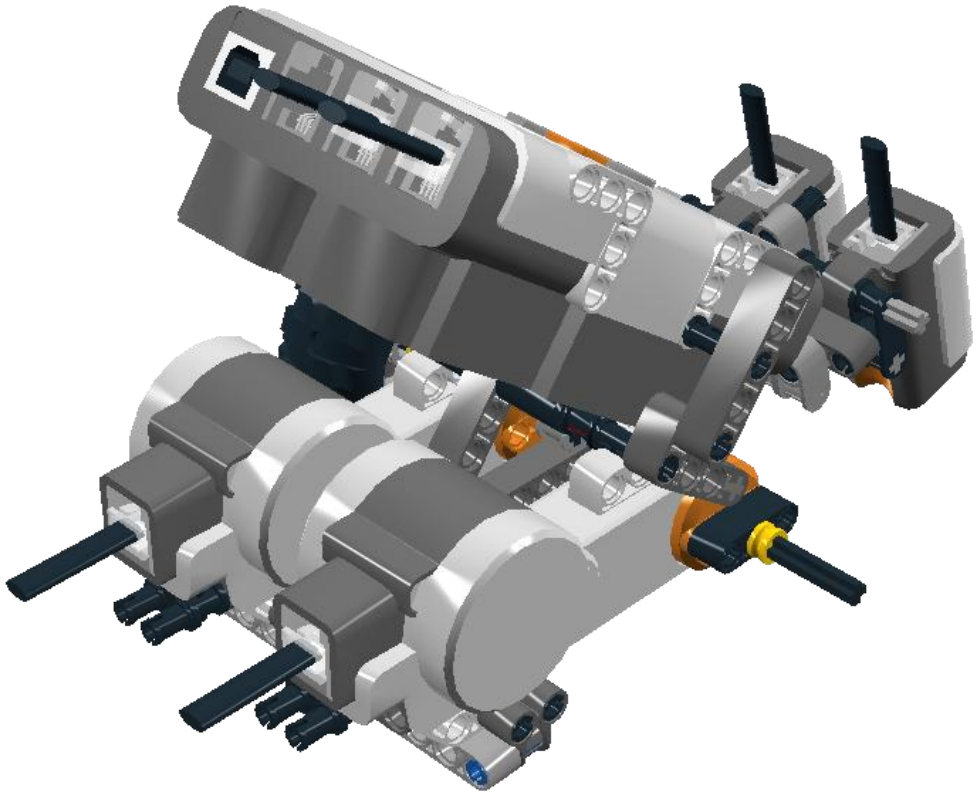


Step 31

1 x 1 x

Step 32

1 x 1 x

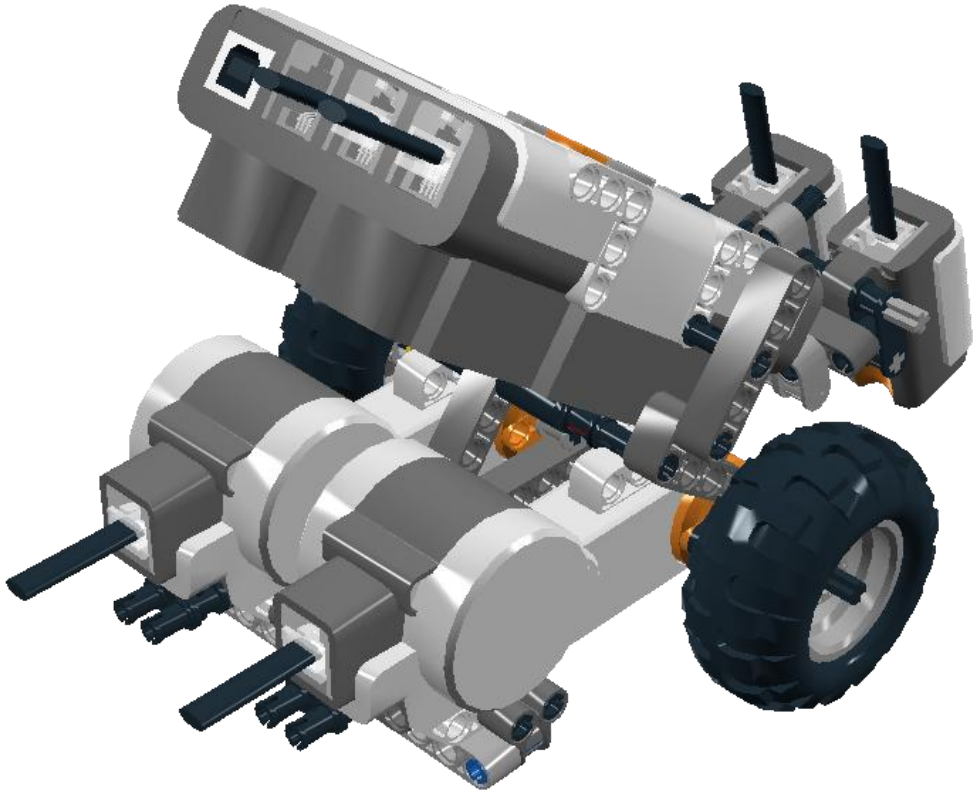


Step 33

1 x 1 x

Step 34

1 x 1 x

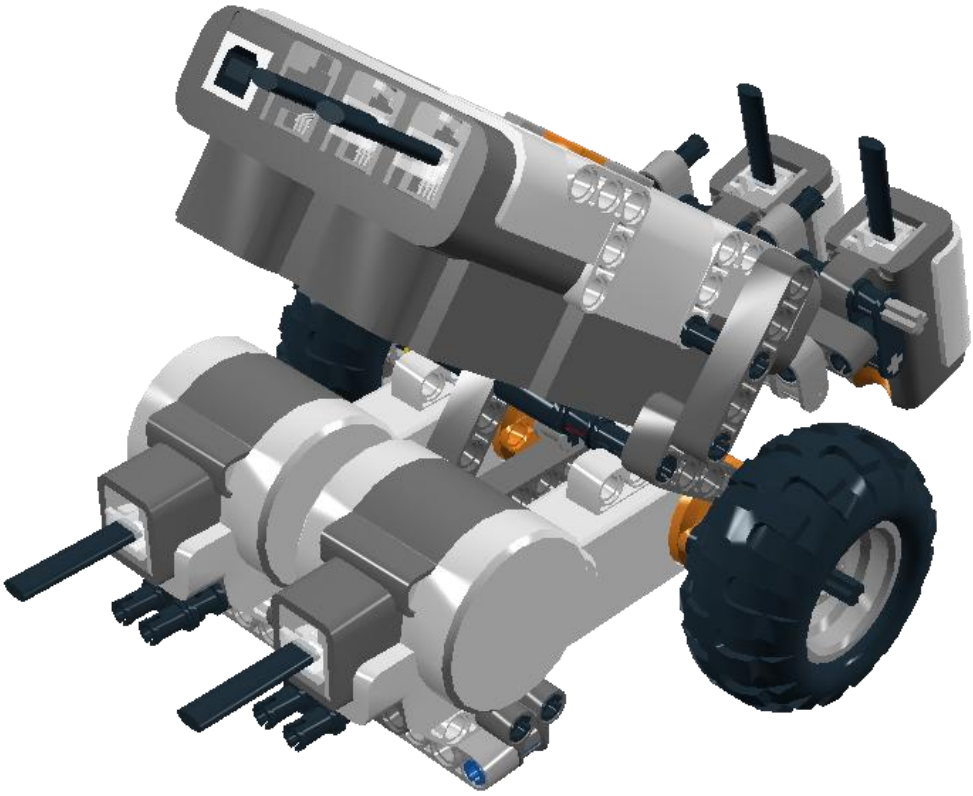


Step 35

1 x 1 x


Step 36

1 x 1 x

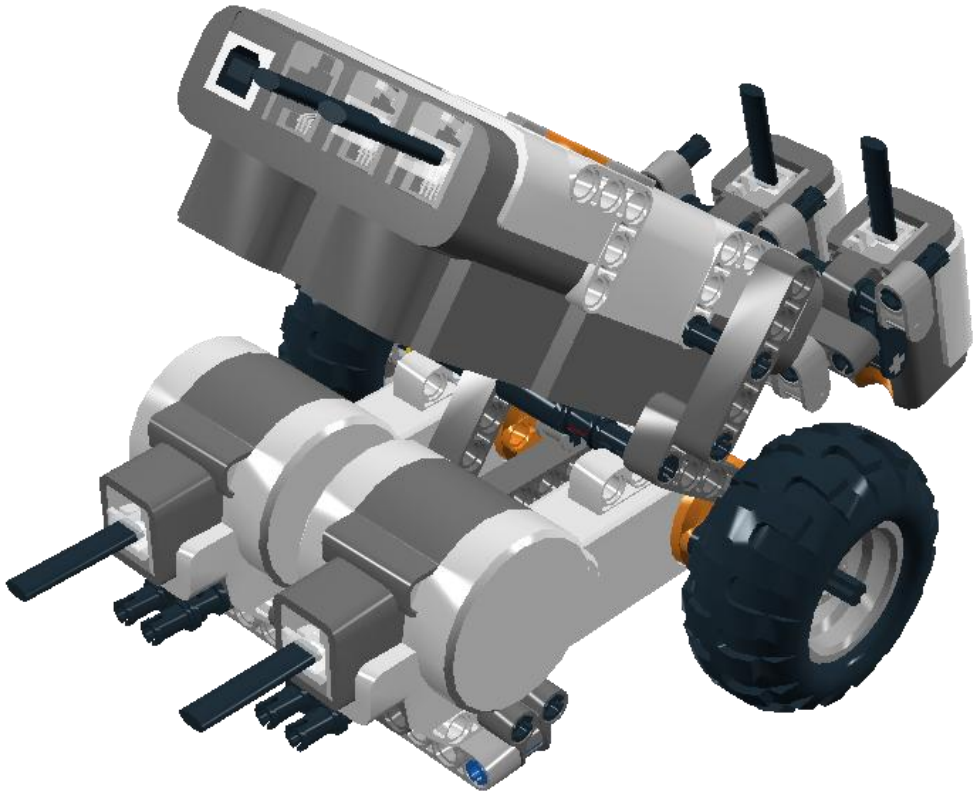



Step 37

1 x 1 x

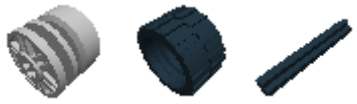

Step 38

1 x 1 x

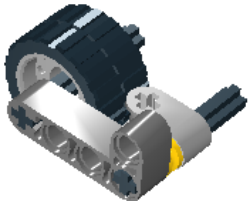
Step 39

1 x 1 x 1 x



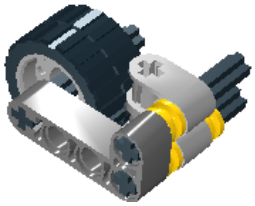
Step 40

1 x 1 x 1 x 1 x



Step 41

2x 1 x 1 x

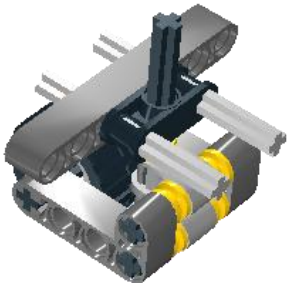


Step 42

1 x 2 x 1 x

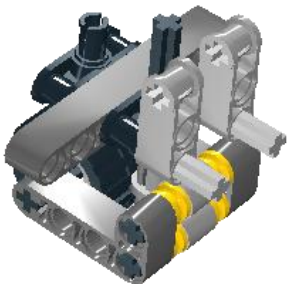
Step 43

1 x 2 x 1 x



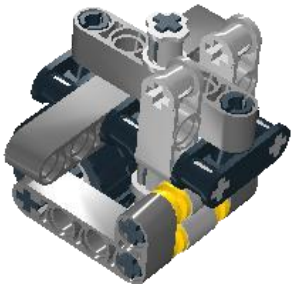
Step 44

1 x 1 x 2 x

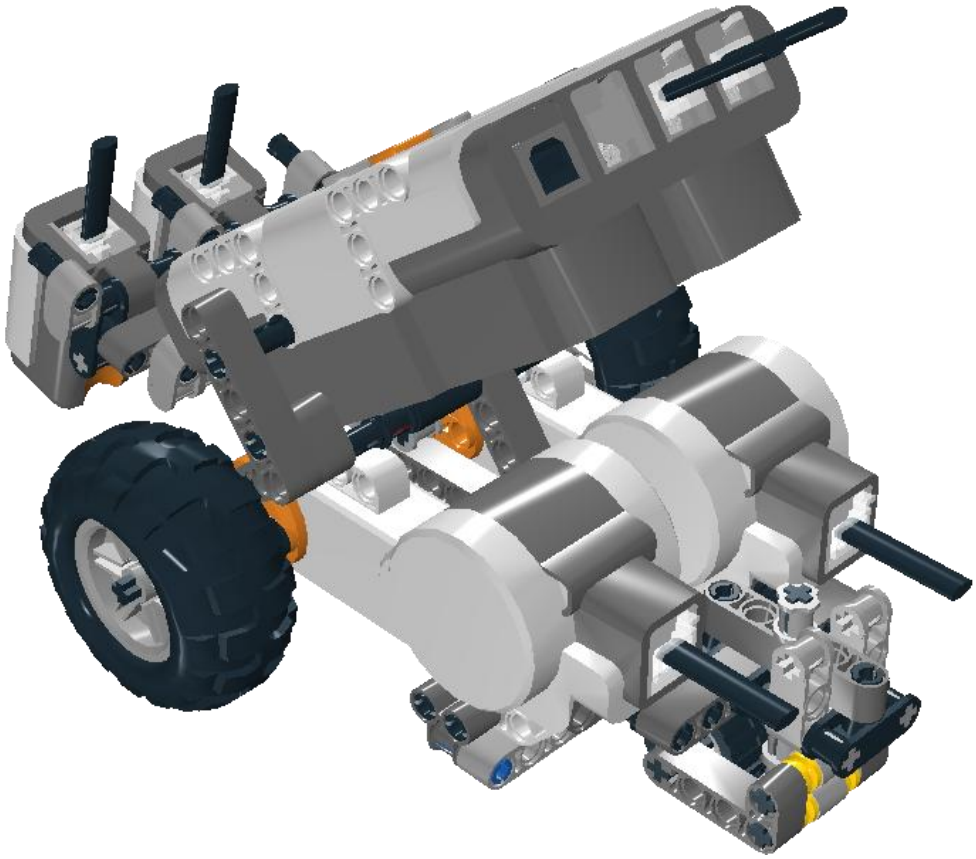


Step 45

1 x 1 x 1 x 1 x



Step 46



LEGO Digital Designer website

ANEXO 2 CÓDIGO FUENTE VELOCISTA

```
/*
Proyecto Fin de Carrera
"DISEÑO, PROGRAMACIÓN Y USO DE MINIROBOTS PARA COMPETICIONES ROBÓTICAS"
Autor: Gloria Martínez Magallón
Tutor: Alfredo Pina Calafi
*/
//CONSTANTES
#define Speed 90
#define GiroSpeed  40
#define GiroTime 200
//VARIABLE GLOBAL
int Negro;
//FUNCIÓN PARA CALCULAR EL ESTADO
int calcular_estado(int izdo, int dcho){
   int bizdo, bdcho;
   bizdo = (izdo > (Negro - 50) ? 1 : 0);
   bdcho = (dcho > (Negro - 50) ? 2 : 0);
   return 1 + bizdo + bdcho;
}
task main(){
    //Variables
    int estado;
    int boton;
    //Inicializar tipo y modo de sensor
    SetSensorType(IN_4, IN_TYPE_LIGHT_ACTIVE ); //Sensor izdo
    SetSensorMode(IN_4, IN_MODE_RAW);
    SetSensorType(IN_1, IN_TYPE_LIGHT_ACTIVE ); //Sensor dcho
    SetSensorMode(IN_1, IN_MODE_RAW);
    //Inicializar variables
    TextOut(5, LCD_LINE2, "Colocar el robot sobre", true);
    TextOut(5, LCD_LINE3, "la línea izda", false);
    boton = ButtonCount(BTN4,false);
    until (ButtonCount(BTN4,false)==boton+1);
    Negro = SensorRaw(IN_4);
    //Colocar el robot para empezar la carrera y esperar a que se indique el
    //comienzo
    TextOut(5, LCD_LINE2, "Colocar el robot sobre", true);
    TextOut(5, LCD_LINE3, "la posicion inicial", false);
    until (ButtonCount(BTN4,false)==boton+2);
    //Ir hacía delante con la velocidad definida y los motores sincronizados
    estado1:
          OnFwdReg(OUT_AC, Speed, OUT_REGMODE_SYNC);
    estado = calcular_estado(SensorRaw(IN_4), SensorRaw(IN_1));
    //Mientras no alcance ninguna de las líneas delimitadores
    while (estado==1){
      estado = calcular_estado(SensorRaw(IN_4), SensorRaw(IN_1));
    }
```

```
    //Comprobamos que línea ha alcanzado y vamos a su código.
    switch (estado)
    {
      case 2 :
        goto estado2;
        break;
      case 3 :
        goto estado3;
    }
    //Giramos hacía la derecha hasta que deje de la línea negra
    estado2:
        OnFwd(OUT_C, GiroSpeed+5);
        OnFwd(OUT_A, -GiroSpeed);
        Wait(GiroTime);
        until(estado==1){
            estado = calcular_estado(SensorRaw(IN_4), SensorRaw(IN_1));
        }
        goto estado1;
    //Giramos hacía la izquierda
    estado3:
        OnFwd(OUT_C, -GiroSpeed);
        OnFwd(OUT_A, GiroSpeed+5);
        Wait(GiroTime);
        until(estado==1){
            estado = calcular_estado(SensorRaw(IN_4), SensorRaw(IN_1));
        }
        goto estado1;
}
```

ANEXO 3 CÓDIGO FUENTE RASTREADORES

```
/*
Proyecto Fin de Carrera
"DISEÑO, PROGRAMACIÓN Y USO DE MINIROBOTS PARA COMPETICIONES ROBÓTICAS"
Autor: Gloria Martínez Magallón
Tutor: Alfredo Pina Calafi
*/
//VARIABLES GLOBALES
#define VOL 3
int Speed = 30;
long Temporizador = 3000;

int SpeedGiro = 20;
//VARIABLE GLOBAL
int Negro;
string Bifur = "nohay";
int Estado = 1;
string Evento;
//FUNCIONES
string calcular_evento(int s4,int s2,int s1){
    long time;
    if (s2 >= Negro){
      if (s4 >= Negro){
          ResetRotationCount(OUT_A);
          until (MotorRotationCount(OUT_A) > 20);
          if (SensorRaw(IN_4) >= Negro){
            until (MotorRotationCount(OUT_A) > 30);
            if (SensorRaw(IN_4) >= Negro && Bifur=="nohay"){
              until (SensorRaw(IN_4) < Negro);
              Bifur = "izda";
              TextOut(5, LCD_LINE3, "Evento C", false);
              return "C";
            }else{
              if (Bifur == "dcha"){
                TextOut(5, LCD_LINE3, "Evento A", false);
                Bifur = "nohay";
                until(SensorRaw(IN_4) < Negro);
                return "A";
              }
              Bifur = "izda";
              TextOut(5, LCD_LINE3, "Evento D", false);
              return "D";
            }
          }else{
            TextOut(5, LCD_LINE3, "Evento B", false);
            return "B";
          }
      }else if (s1 >= Negro){
```

```
            ResetRotationCount(OUT_A);
            until (MotorRotationCount(OUT_A) > 20);
            if (SensorRaw(IN_1) >= Negro){
              until (MotorRotationCount(OUT_A) > 30);
              if (SensorRaw(IN_1) >= Negro && Bifur=="nohay"){
                until (SensorRaw(IN_1) < Negro);
                Bifur = "dcha";
                TextOut(5, LCD_LINE3, "Evento C", false);
                return "C";
              }else{
                if (Bifur == "izda"){
                  TextOut(5, LCD_LINE3, "Evento A", false);
                  Bifur = "nohay";
                  until(SensorRaw(IN_1) < Negro);
                  return "A";
                }
                Bifur = "dcha";
                TextOut(5, LCD_LINE3, "Evento D", false);
                return "D";
              }
            }else{
              TextOut(5, LCD_LINE3, "Evento B", false);
              return "B";
            }
          }else{
              TextOut(5, LCD_LINE3, "Evento A", false);
              return "A";
          }
        }else{
           TextOut(5, LCD_LINE3, "Evento E", false);
          return "E";
        }
}
void girar_izda(){
       OnFwd(OUT_C, -SpeedGiro);
       OnFwd(OUT_A, SpeedGiro+5);
}
void girar_dcha(){
       OnFwd(OUT_A, -SpeedGiro);
       OnFwd(OUT_C, SpeedGiro+5);
}
void encontrar(){
     bool aux = true;
     long tem = 100;
     long time =  0;
     until (SensorRaw(IN_2) >= Negro){
       time =  CurrentTick();
       tem = tem * 2;
       if (aux){ girar_izda(); aux = false;}
       else{ girar_dcha(); aux = true;}
```

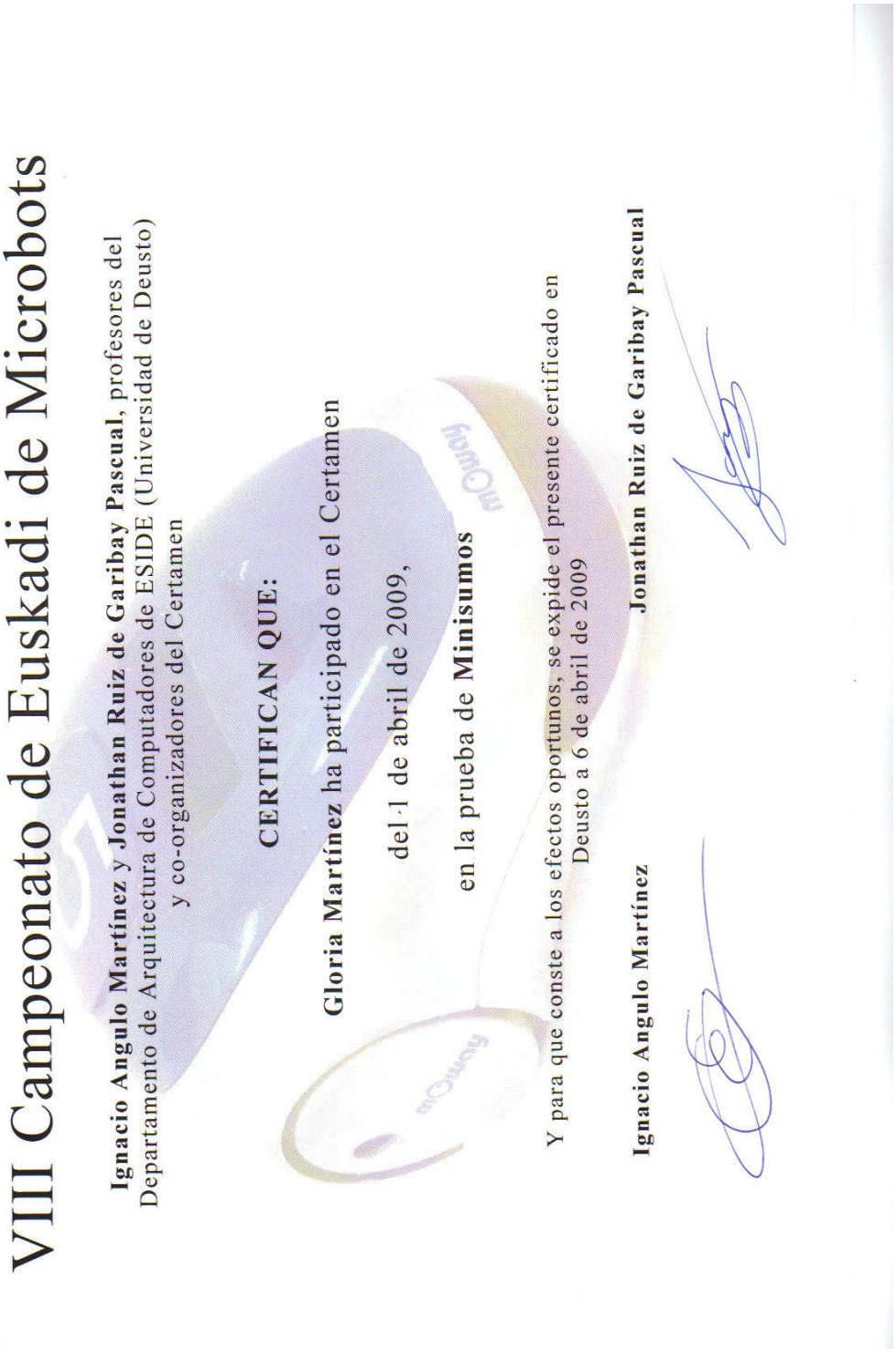
```
      until (SensorRaw(IN_2) >= Negro || (CurrentTick() - time) > tem);
     }
     OnFwdReg(OUT_AC, Speed, OUT_REGMODE_SYNC);
}
task main(){
    int boton;
    //Inicializar tipo y modo de sensor
    SetSensorType(IN_4, IN_TYPE_LIGHT_ACTIVE ); //Sensor izdo
    SetSensorMode(IN_4, IN_MODE_RAW);
    SetSensorType(IN_2, IN_TYPE_LIGHT_ACTIVE ); //Sensor centro
    SetSensorMode(IN_2, IN_MODE_RAW);
    SetSensorType(IN_1, IN_TYPE_LIGHT_ACTIVE ); //Sensor dcho
    SetSensorMode(IN_1, IN_MODE_RAW);
    //Inicializar variables
    TextOut(5, LCD_LINE2, "Colocar el robot sobre la línea", true);
    boton = ButtonCount(BTN4,false);
    until (ButtonCount(BTN4,false)==boton+1);
    Negro = SensorRaw(IN_2)-50;
    NumOut(5, LCD_LINE2, Negro, true);
    NumOut(5, LCD_LINE3, SensorRaw(IN_1), false);
    NumOut(5, LCD_LINE1, SensorRaw(IN_4), false);
     until (ButtonCount(BTN4,false)==boton+2);
    //Colocar el robot para empezar la carrera y esperar a que se indique el
    //comienzo
    //TextOut(5, LCD_LINE2, "Colocar el robot en la posición de carrera", true);
    //until (ButtonPressed(BTN4, true));
    TextOut(5, LCD_LINE2, "", true);
    //Ir hacía delante con la velocidad definida y los motores sincronizados
    estado1:
      TextOut(5, LCD_LINE2, "Estado 1", false);
      //Mientras no alcance ninguna de las líneas delimitadores
      until (Evento=="C"|| Evento=="D"||Evento=="E"){
         OnFwdReg(OUT_AC, Speed, OUT_REGMODE_SYNC);
         Evento = calcular_evento(SensorRaw(IN_4),
SensorRaw(IN_2),SensorRaw(IN_1));
      }
      if (Evento == "C") goto estado2;
      else if (Evento == "D") goto estado4;
      else if (Evento == "E") goto estado5;
    estado2:
      TextOut(5, LCD_LINE2, "Estado 2", false);
      until (Evento=="A" || Evento=="E"){
          OnFwdReg(OUT_AC, Speed, OUT_REGMODE_SYNC);
          Evento = calcular_evento(SensorRaw(IN_4),
SensorRaw(IN_2),SensorRaw(IN_1));
      }
      if (Evento == "A") goto estado3;
      else if (Evento == "E") goto estado6;
    estado3:
      TextOut(5, LCD_LINE2, "Estado 3", false);
```

```
      until (Evento=="D" || Evento=="E" || (Evento=="A" && Bifur=="nohay")){
         OnFwdReg(OUT_AC, Speed, OUT_REGMODE_SYNC);
         Evento = calcular_evento(SensorRaw(IN_4),
SensorRaw(IN_2),SensorRaw(IN_1));
      }
      if(Evento == "D") goto estado4;
      else if (Evento == "E") goto estado7;
    estado4:
      PlayTone(300, 100);
      TextOut(5, LCD_LINE2, "Estado 4", false);
      until(Evento=="A" || Evento=="B"){
       OnFwdReg(OUT_AC, Speed, OUT_REGMODE_SYNC);
       Wait(200);
       if (Bifur=="dcha") girar_dcha();
       else girar_izda();
       until(SensorRaw(IN_2) < Negro);
       until(SensorRaw(IN_2) >=Negro);
       Wait(50);
       Evento = calcular_evento(SensorRaw(IN_4),
SensorRaw(IN_2),SensorRaw(IN_1));
      }
      Bifur="nohay";
      goto estado1;
    estado5:
      TextOut(5, LCD_LINE2, "Estado 5", false);
      until (Evento == "A" || Evento == "C" || Evento =="D" || Evento == "B"){
         encontrar();
         Evento = calcular_evento(SensorRaw(IN_4),
SensorRaw(IN_2),SensorRaw(IN_1));
      }
      goto estado1;
    estado6:
      TextOut(5, LCD_LINE2, "Estado 6", false);
      until (Evento == "A" || Evento == "C" || Evento =="D"){
         encontrar();
         Evento = calcular_evento(SensorRaw(IN_4),
SensorRaw(IN_2),SensorRaw(IN_1));
      }
      goto estado2;
   estado7:
      TextOut(5, LCD_LINE2, "Estado 7", false);
      until (Evento == "A" || Evento == "C" || Evento =="D"){
         encontrar();
         Evento = calcular_evento(SensorRaw(IN_4),
SensorRaw(IN_2),SensorRaw(IN_1));
      }
      goto estado3;

   TextOut(5, LCD_LINE2, "Error", true);
```

```
    TextOut(5, LCD_LINE2, Evento, true);
    until(1==2);
}
```

ANEXO 4 CERTIFICADO VII CAMPEONATO DE EUSKADI DE MINIROBOTS



VIII Campeonato de Euskadi de Microbots

Ignacio Angulo Martínez y **Jonathan Ruiz de Garibay Pascual**, profesores del Departamento de Arquitectura de Computadores de ESIDE (Universidad de Deusto) y co-organizadores del Certamen

CERTIFICAN QUE:

Gloria Martínez ha participado en el Certamen

del 1 de abril de 2009,

en la prueba de **Minisumos**

Y para que conste a los efectos oportunos, se expide el presente certificado en Deusto a 6 de abril de 2009

Ignacio Angulo Martínez

Jonathan Ruiz de Garibay Pascual

ANEXO 5 CERTIFICADO ROBOLID '09

la

Asociación de Microbótica

de la

Universidad de Valladolid

CERTIFICA QUE D/Dª.:

Gloria Martínez Magallón, con DNI: 78752170 D

Ha participado en el **certamen de robótica**

ROBOLID '09

los días 15 y 16 de Abril de 2009 , en las Escuelas de Ingenieros Informáticos y de Telecomunicación e Industriales de Valladolid

Fdo. David del Río Nevado. Coordinador

Valladolid, 16 de Abril de 2009

DISEÑO, PROGRAMACIÓN Y USO DE MINIROBOTS PARA COMPETICIONES ROBÓTICAS

INGENIERO TÉCNICO EN INFORMÁTICA DE GESTIÓN

Gloria Martínez Magallón - 20 de Abril 2011

ETAPAS DEL PROYECTO FIN DE CARRERA

- Primera Fase:
  - Estudio de las competiciones y pruebas robóticas.
- Segunda Fase:
  - Estudio de los kits de robótica.
- Tercera Fase:
  - Diseño, montaje y programación de robots para competiciones robóticas

Competencias robóticas

- Nacionales
  - DeustoBot
  - Robolid
  - Alcabot - Hispabot
  - MadridBot
  - CosmoBot
- Internacionales
  - Eurobot
  - Robocup

Competencias robóticas - DeustoBot

- Organizado por el grupo de robótica de la Universidad de Deusto.
- Pruebas
  - Sumo.
  - Mini – sumo.
  - Rastreadores.
  - Bailarines.

Competencias robóticas - Robolid

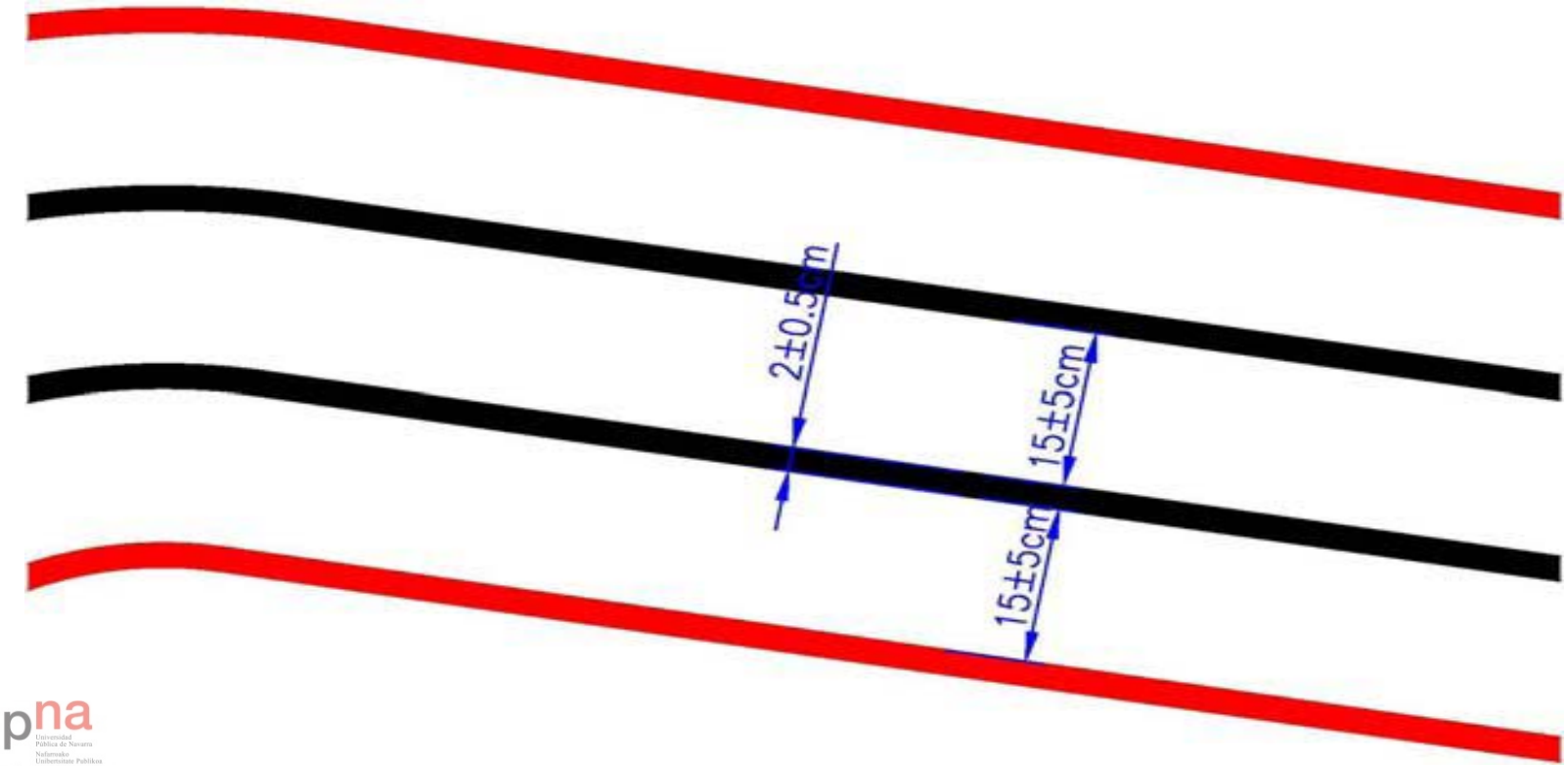
- Organizado por AMUVA.
- Pruebas
  - Sumo.
  - Velocistas.
  - Rastreadores.
  - Programación.

Competencias robóticas - Velocistas

- Objetivo
  - Conseguir velocidades altas de marcha en un recorrido perfectamente preestablecido.
- Características de los robots.
  - 20 cm de ancho y 30 cm de largo.
  - Completamente autónomos.

Competencias robóticas - Velocistas

- Pista de velocistas

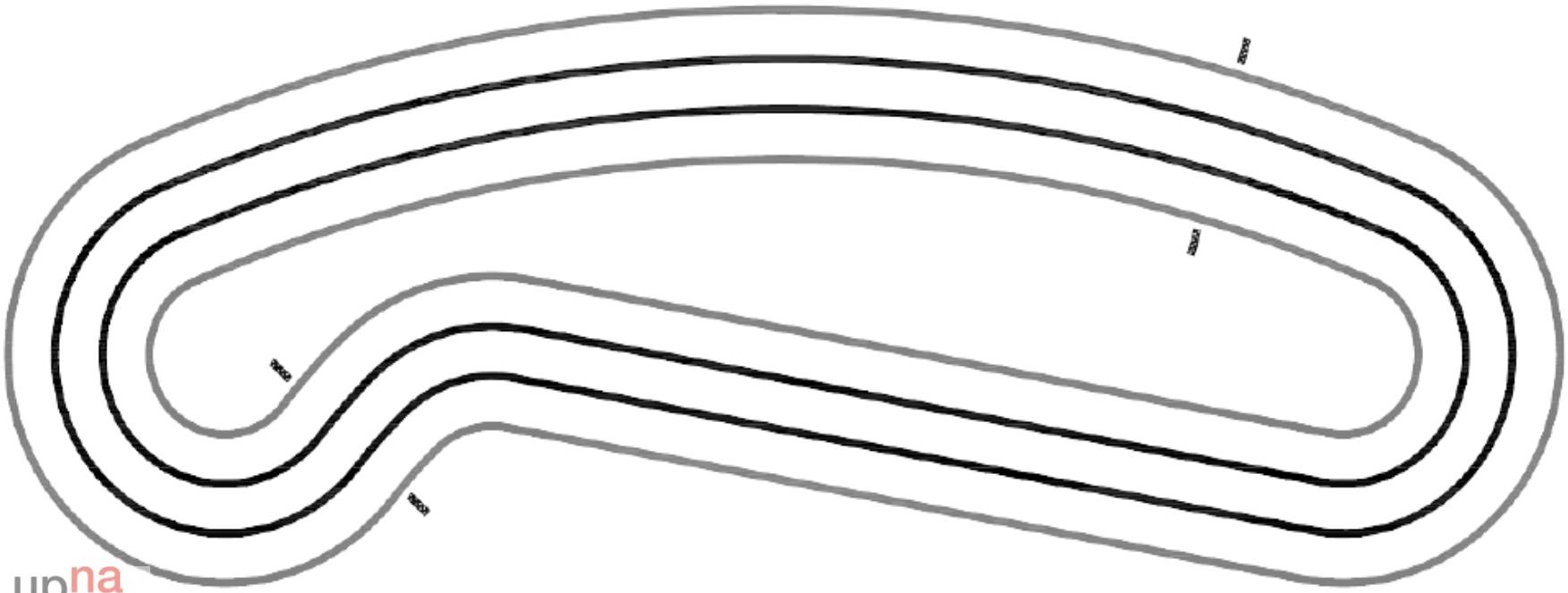


Competencias robóticas - Velocistas

- Sistema de puntuación
  - Ronda de clasificación: Competición individual.
    - Los 8 mejores pasarán a la siguiente ronda.
  - Rondas eliminatorias y finales: Competición por parejas.
    - Al mejor de tres mangas.

Competencias robóticas - Velocistas

- Ejemplo de pista

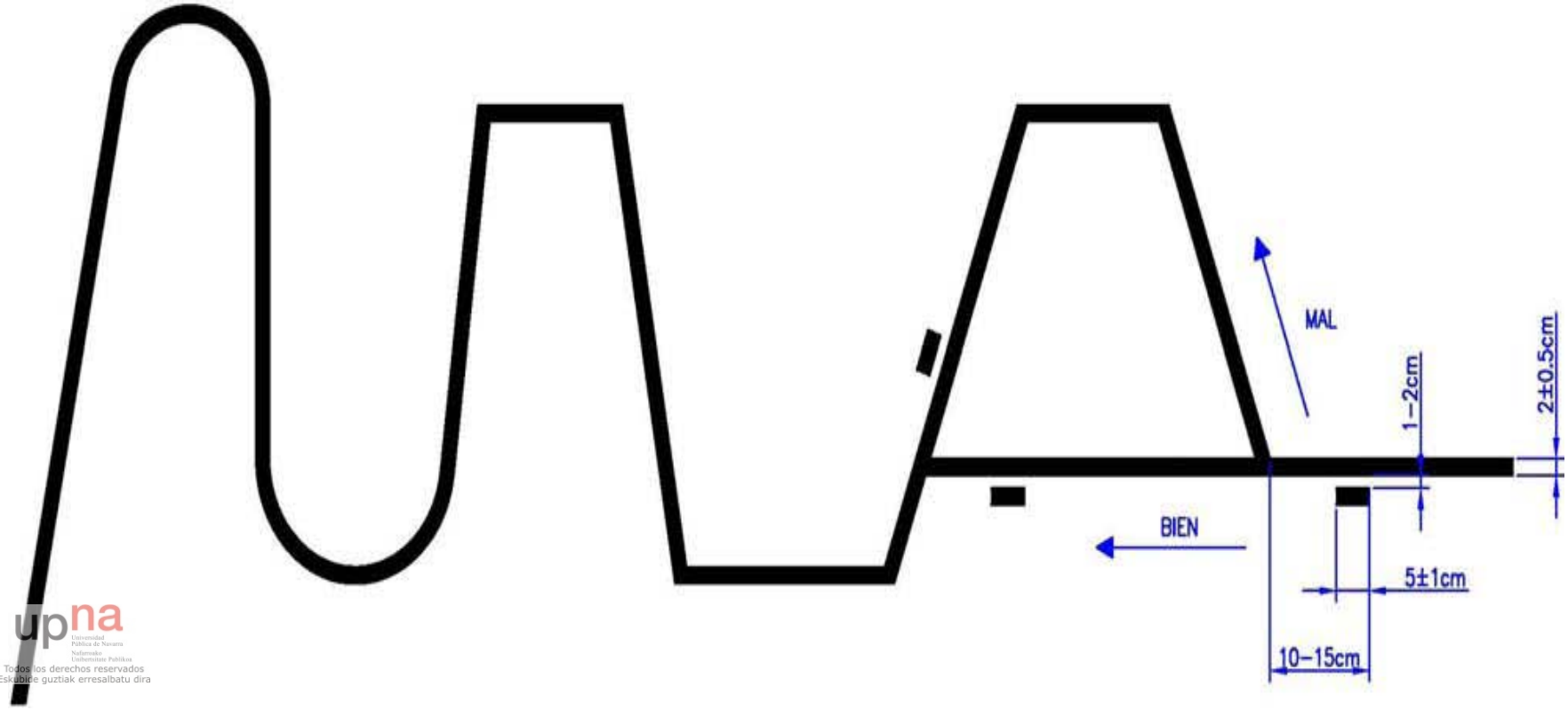


Competencias robóticas - Rastreadores

- Objetivo
  - Recorrer un camino sinuoso, previamente establecido, en el menor tiempo posible.
- Características de los robots.
  - 20 cm de ancho y 30 cm de largo.
  - Completamente autónomos.

Competencias robóticas - Rastreadores

- Pista de rastreadores

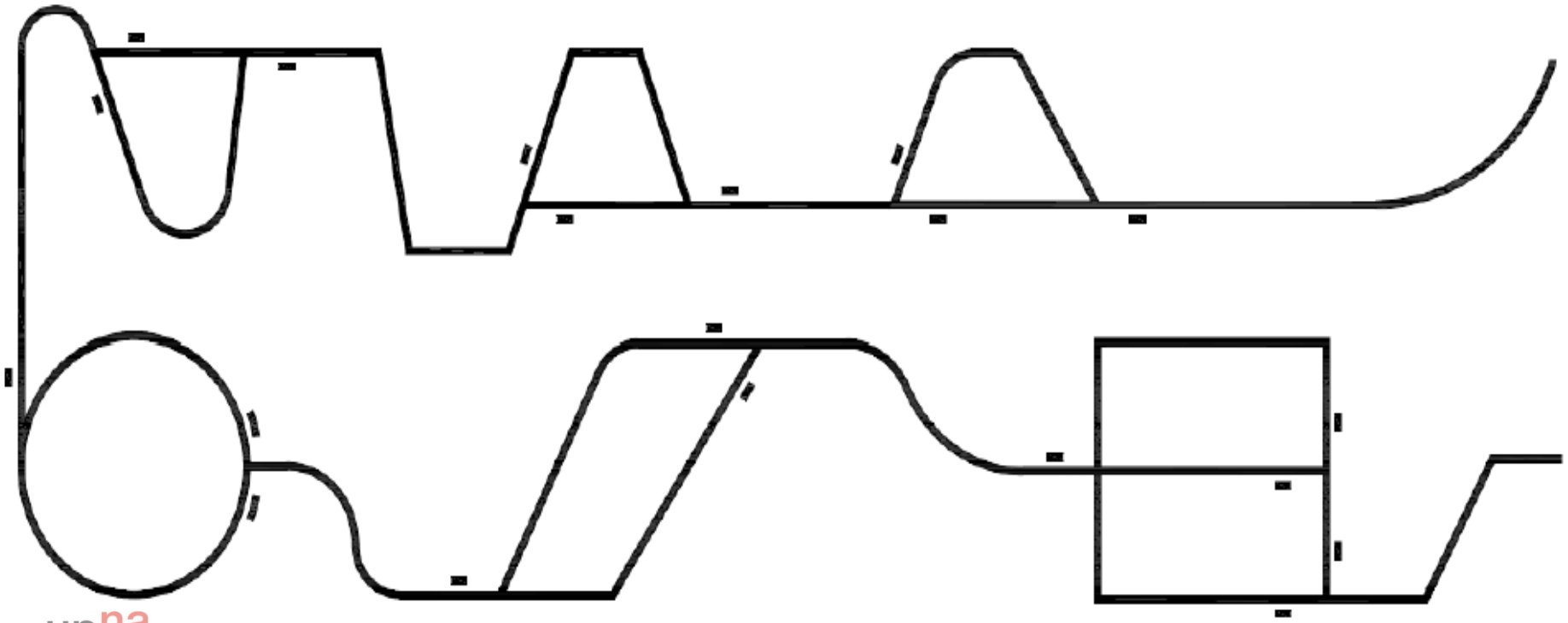


Competencias robóticas - Rastreadores

- Sistema de puntuación
  - Menor número de errores cometidos y menor tiempo.
  - Pista dividida en cuatro tramos, 25 puntos por tramos superado sin errores.
  - Penalización:
    - 10 puntos por tomar una bifurcación por el camino no indicado.

Competencias robóticas - Rastreadores

- Ejemplo de pista



Kits de robótica

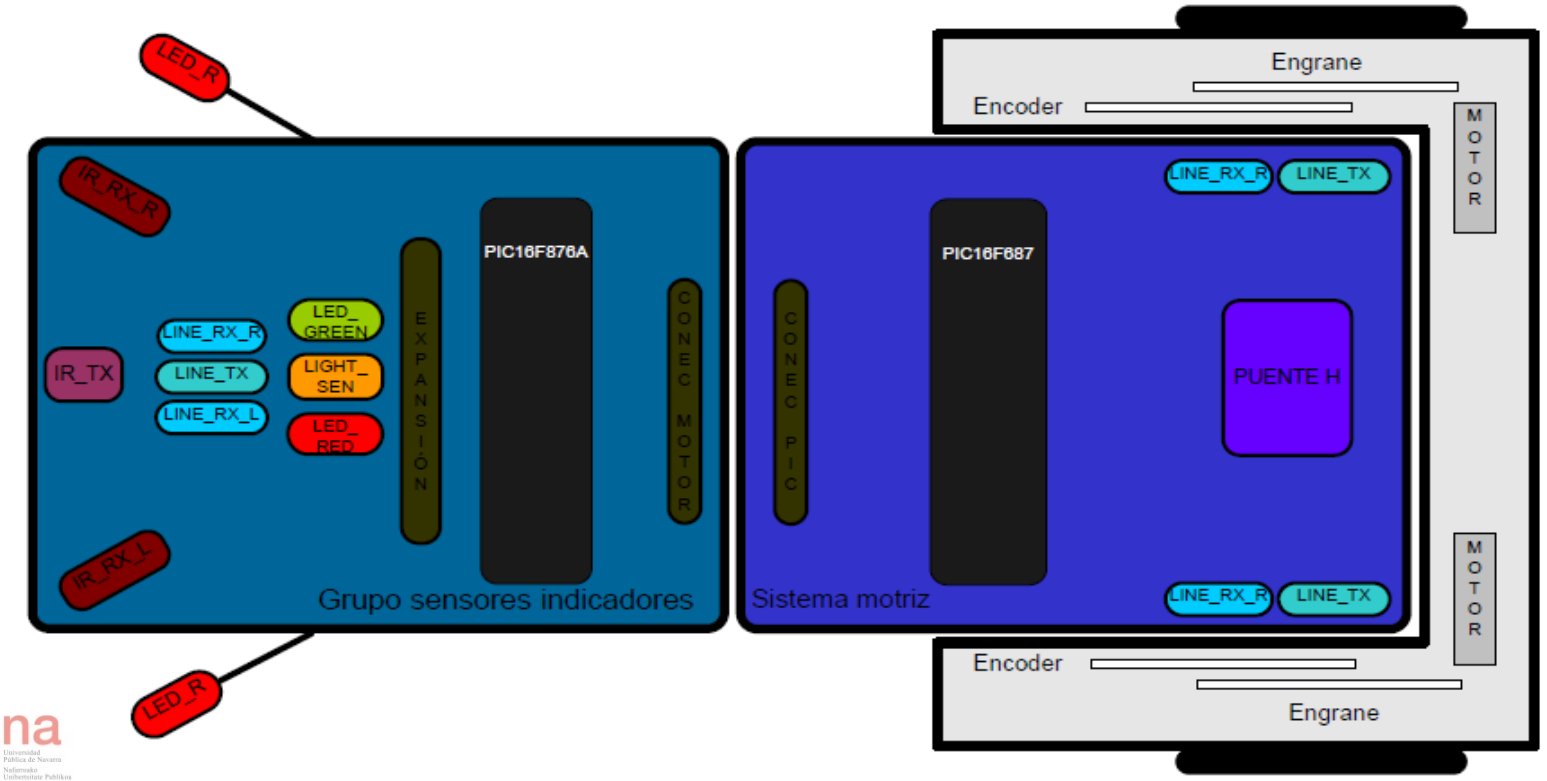
- Moway
- LEGO Mindstorms

Kits de robótica - Moway

- Herramienta utilizada en diferentes ámbitos.
  - Robótica, investigación, industria…
- Desarrollada por Minirobots (Barakaldo)
  - 2008, “Premio Barakaldo Empresa Innovadora”

Kits de robótica - Moway

- Robot Moway.



Kits de robótica - Moway

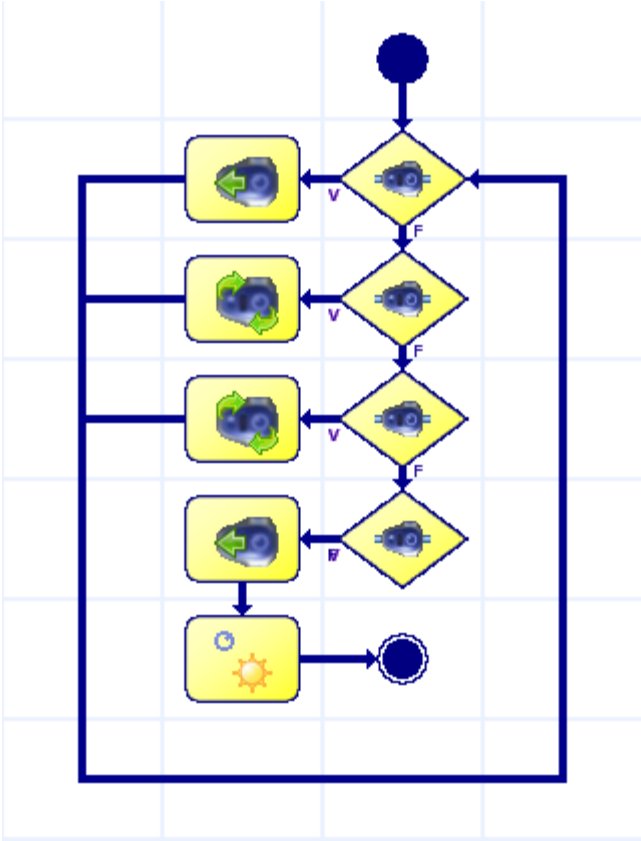
- Módulos de comunicación por radio frecuencia.

Kits de robótica - Moway

- Programación.
  - Lenguaje Ensamblador.
  - Lenguaje C.
  - Diagramas de Flujo

Kits de robótica - Moway

- Diagramas de flujo: MowayGUI.



Kits de robótica - Moway

- Prácticas para ver el funcionamiento.
  - Sensores infrarrojos anticolisión:

Kits de robótica - Moway

- Prácticas para ver el funcionamiento.
  - Sensores de intensidad de luz direccional:

Kits de robótica - Moway

- Prácticas para ver el funcionamiento.
  - Sensores optorreflectivos infrarrojos :

Kits de robótica - Moway

- Prácticas para ver el funcionamiento.
  - Módulo RF entre robots :

Prueba “Bailarines” - Moway

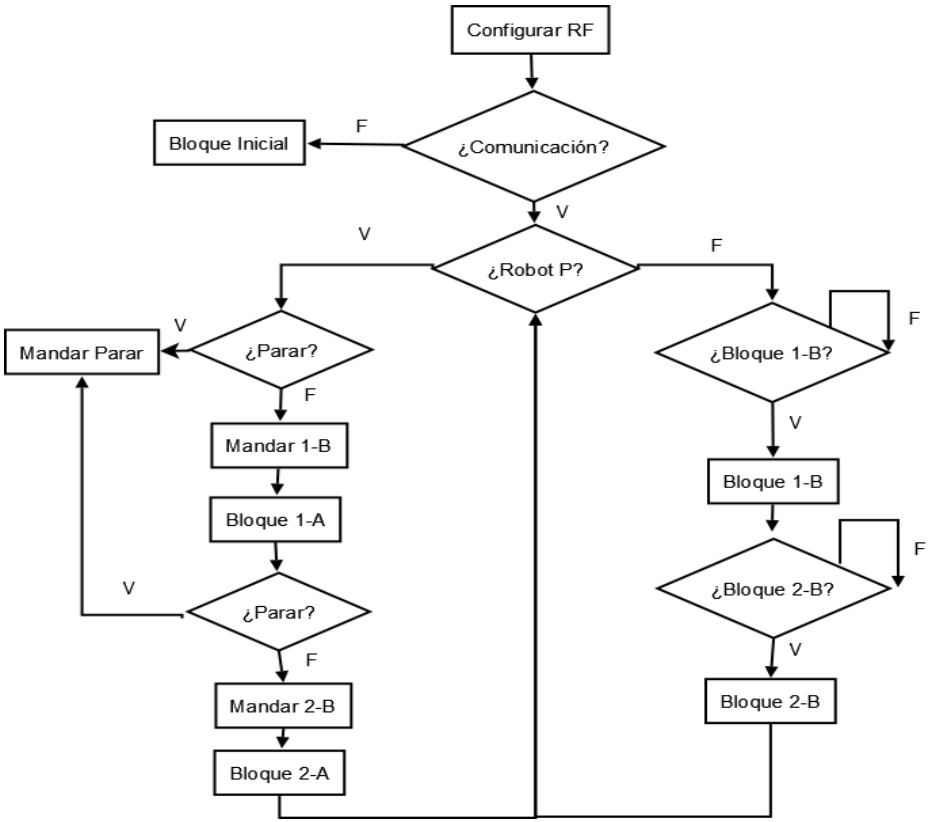
- Análisis de la prueba:
  - Se necesitan al menos 2 robots.
  - Será necesaria la comunicación entre los robots.
  - El tiempo de baile tendrá que ser ilimitado.

Prueba “Bailarines” - Moway

- Análisis de la solución:
  - Se utilizarán dos robots Moway y dos módulos RF.
  - Uno de los robots, llevará el peso del baile.
  - Además se ha añadido una señal de parar.

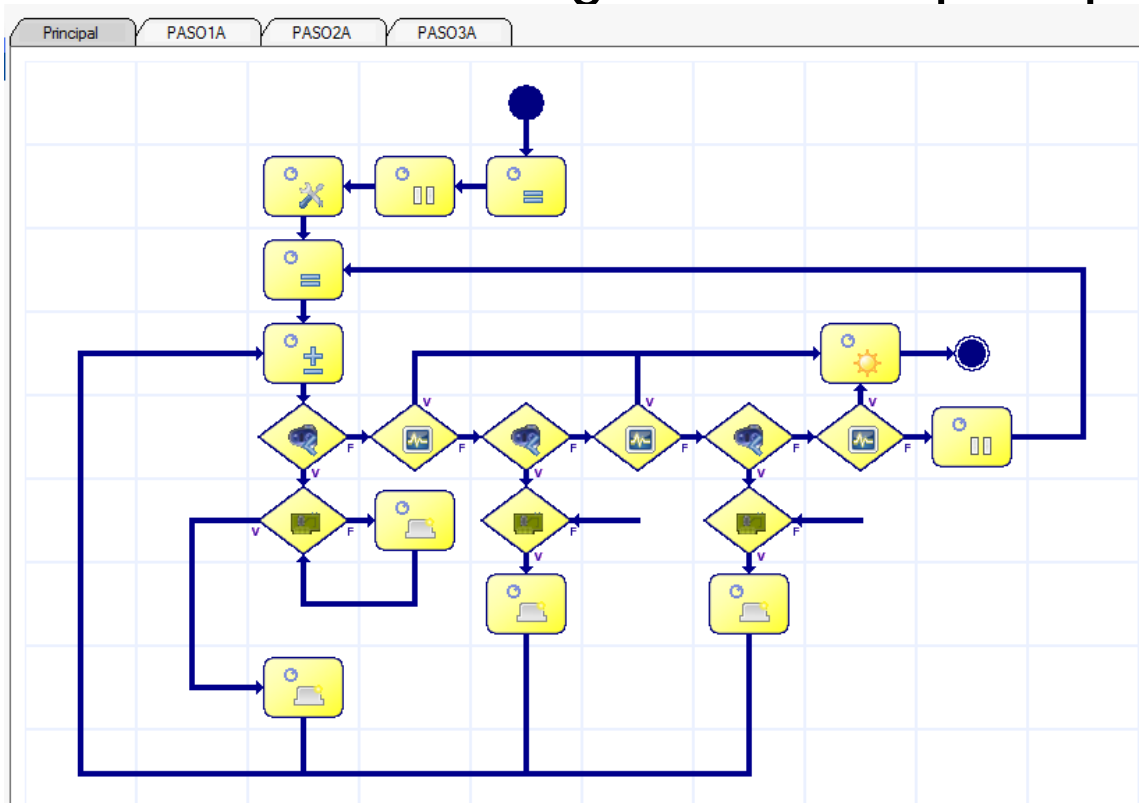
Prueba “Bailarines” - Moway

- Diseño del baile:



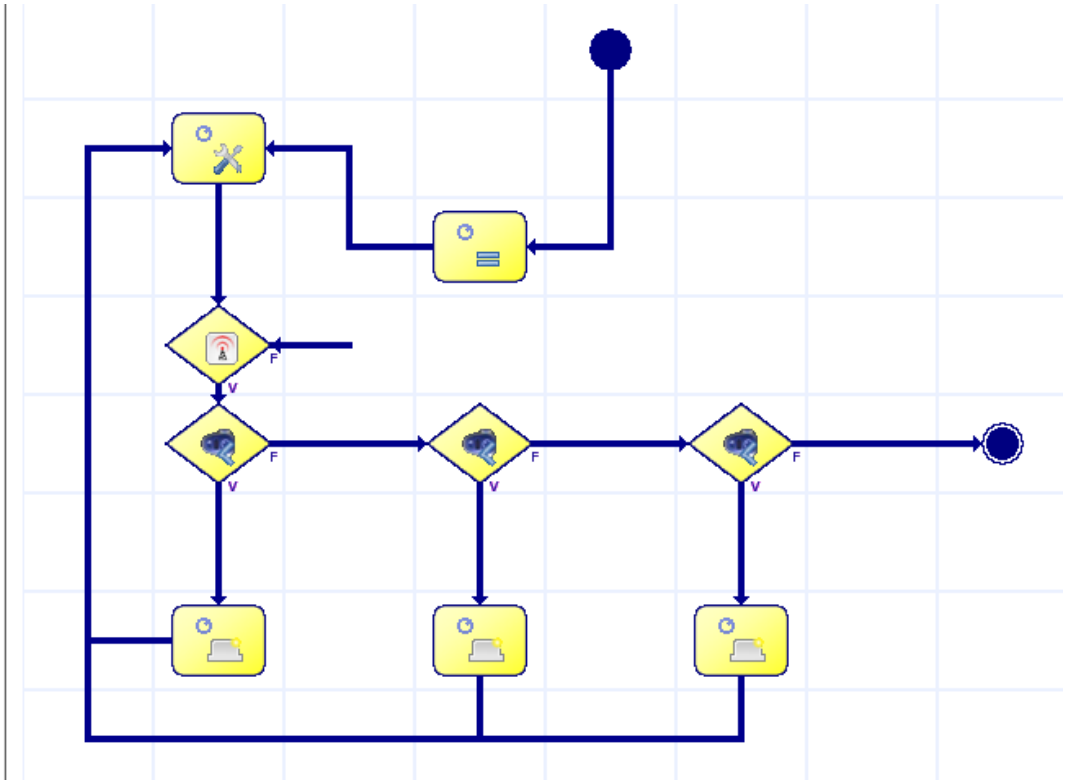
Prueba “Bailarines” - Moway

- Implementación de la solución:
  - Implementación del código del robot principal



Prueba “Bailarines” - Moway

- Implementación de la solución:
  - Implementación del código del robot secundario



Kits de robótica – LEGO Mindstorms

- Desarrollada por LEGO - MIT.
- Se empezó a comercializar en 1998
- Se diseño orientado a niños de 10 a 14 años.

Kits de robótica – LEGO Mindstorms

- Robot LEGO Mindstorms.

Kits de robótica – LEGO Mindstorms

- Otros sensores:
  - Sensor buscador de infrarrojos.
  - Sensor electro-óptico detector de proximidad NXT.
  - Sensor brújula.
  - Sensor de color.
  - Sensor Gyro.
  - Sensor de aceleración – inclinación.

Kits de robótica – LEGO Mindstorms

- Piezas:
  - Piezas móviles.
  - Piezas flexibles.
  - Piezas de fijación.
- Ruedas:

Kits de robótica - LEGO Mindstorms

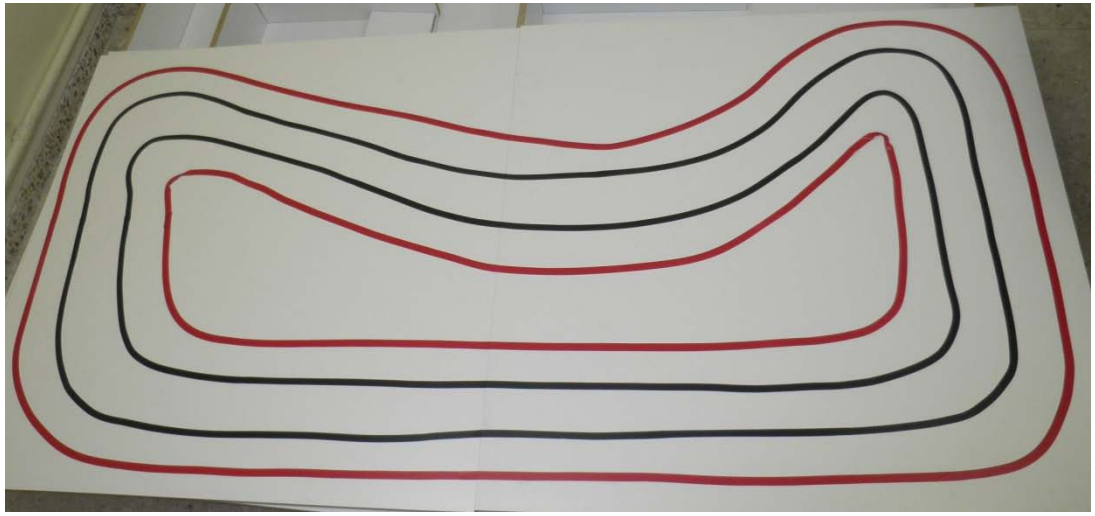
- Programación.
  - Lenguaje NXC.
  - Lenguaje NXJ (leJOS).
  - Lenguaje NXT-G (Diagramas de flujo).

Kits de robótica - LEGO Mindstorms

- Lenguaje NXC. Bricx
  - Incluye la librería NXC.
  - Integra un compilador.
  - Permite la comunicación con el robot
- Lego Digital Designer
  - Permite realizar un diseño virtual del Lego

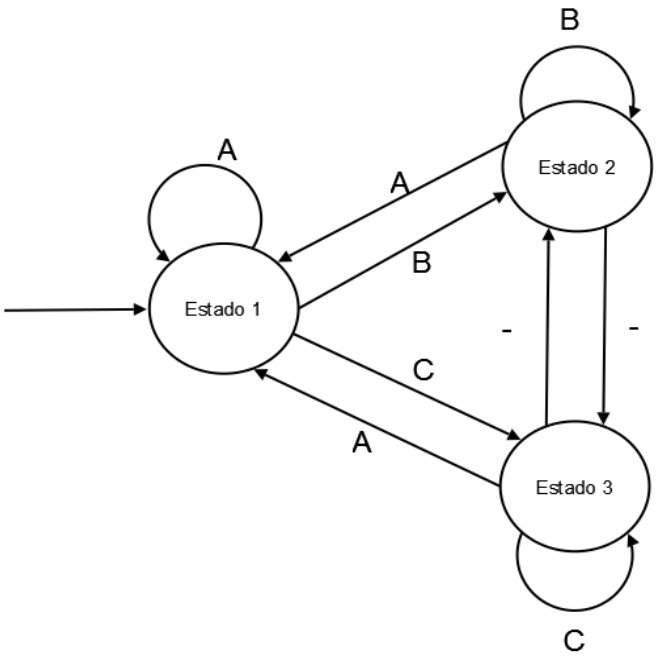
Prueba “Velocistas” – LEGO Mindstorms

- Análisis de la prueba:
  - Dimensiones de 20cm x 30cm.
  - Calle de 15±5cm.



Prueba “Velocistas” – LEGO Mindstorms

- Diseño de la solución:



Prueba “Velocistas” – LEGO Mindstorms

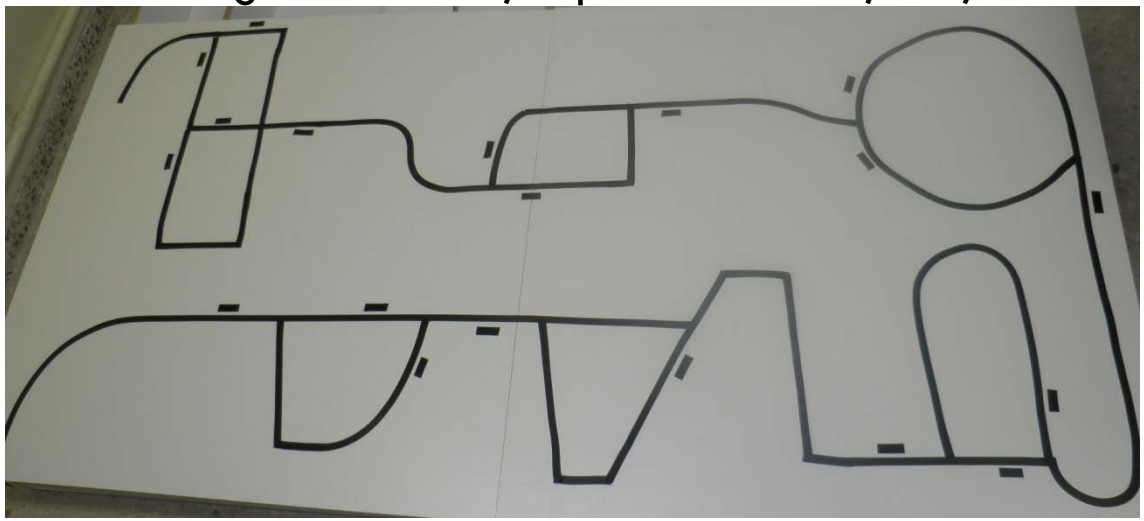
- Diseño del robot (ANEXO 1):

Prueba “Velocistas” – LEGO Mindstorms

- Implementación de la solución (ANEXO2):

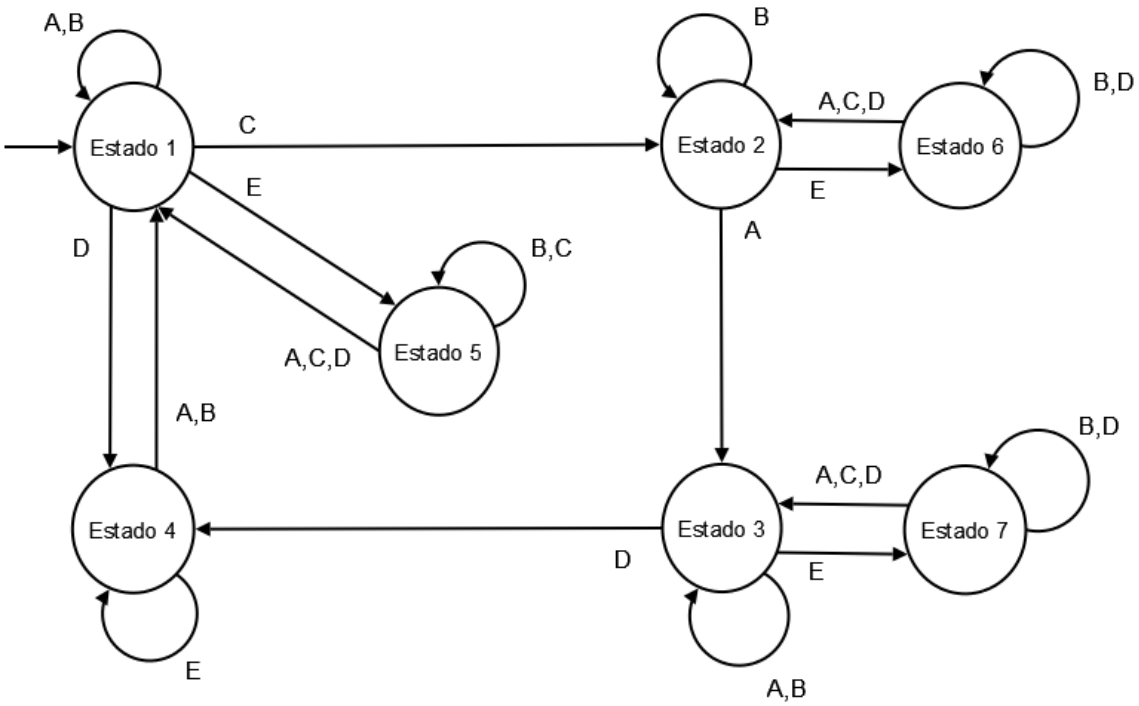
Prueba “Rastreadores” – LEGO Mindstorms

- Análisis de la prueba:
  - Línea negra de 2±0,5cm.
  - Marcas de bifurcación:
    - Se encontrarán a 10±5cm de la bifurcación.
    - Serán de longitud 5±1cm, separadas 1±0,5cm, de la línea principal.



Prueba “Rastreadores” – LEGO Mindstorms

- Diseño de la solución:



Prueba “Rastreadores” – LEGO Mindstorms

- Diseño del robot:

Prueba “Rastreadores” – LEGO Mindstorms

- Implementación de la solución (ANEXO3):

Conclusiones

- Kit Moway
  - Aspectos Positivos
    - Incorpora distintos sensores.
    - Módulo RF.
    - Software MowayGUI.
  - Aspectos Negativos
    - Poca memoria
    - Sistema motriz poco potente

Conclusiones

- LEGO Mindstorms
  - Aspectos Positivos
    - Incorpora distintos sensores.
    - Comunidad de usuarios.
    - Muy flexible
  - Aspectos Negativos
    - Falta de agilidad
    - Movimientos rectilíneos con pequeñas desviaciones.

Conclusiones

- Resultados obtenidos.
  - Robolid – Velocistas
    - Fiabilidad 100%
    - Menos rápido que otros
  - Robolid – Rastreadores
    - Fiabilidad por encima de la media.
    - Menos rápido que otros.

Conclusiones

- Experiencia personal
  - First Lego League (FLL) – Juez de Mesa.
  - DeustoBot y Robolid – Concursante
  - European Open Day on Educational Robotics – Stand

MUCHAS GRACIAS POR SU ATENCIÓN

DUDAS Y PREGUNTAS

DISEÑO, PROGRAMACIÓN Y USO DE MINIROBOTS PARA COMPETICIONES ROBÓTICAS

INGENIERO TÉCNICO EN INFORMÁTICA DE GESTIÓN

Gloria Martínez Magallón - 20 de Abril 2011