UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

FACULTAD DE INGENIERÍA

RESOLUCION DE LABERINTOS POR MEDIO DE UN ROBOT MÓVIL

T E S I S

QUE PARA OBTENER LOS TÍTULOS DE:

INGENIERO EN COMPUTACIÓN[1]

INGENIERO ELÉCTRICO ELECTRÓNICO[2]

P R E S E N T A N :

FERNANDO ALBERTO CANTÚ MARQUES[1]

CARLOS ALBERTO VELASCO TOMÉ[2]

DIRECTOR DE TESIS: DR. JESÚS SAVAGE CARMONA

2006

Agradecimientos

Agradecemos al **Dr. Jesús Savage Carmona** la dirección de esta tesis cuyo tema nos permitió adentrarnos en un área de suma actualidad; así como el gran apoyo que nos brindó al permitirnos utilizar el laboratorio de biorobótica durante la elaboración de este trabajo.

También queremos expresar nuestro agradecimiento a **Salvador Medina Maza** y a **Nadxelle Velasco Gutiérrez** por su ayuda y colaboración en la realización de este proyecto a lo largo de su desarrollo.

Índice

Resumen

Este trabajo se realizó con la finalidad de construir un robot móvil capaz de resolver un laberinto pintado con líneas negras y fondo blanco.

Se diseñaron varios robots para que pudieran llevar a cabo la tarea deseada [5]. Para ello fueron diseñados diferentes circuitos, arreglos de sensores y algoritmos para la resolución del laberinto probando diferentes métodos y configuraciones de estos para lograr un robot más eficiente, tanto en el tiempo como en su precisión.

Se intentaron diferentes tipos de sensores y distintos arreglos de estos, para lograr que el robot detectara la línea del laberinto. Esto se realizó usando leds infrarrojos hasta llegar a los sensores encapsulados CNY70[16]. Se utilizaron arreglos de 3, 5 y 7 sensores en diferentes posiciones y distancias entre ellos. Finalmente el arreglo de 7 sensores resulto más apropiado debido a la cantidad de casos que puede detectar y la precisión que se obtiene.

También se desarrollaron “encoders” en las llantas para tener un mayor control del robot en cuanto a movimiento se refiere, logrando indicarle al robot que se mueva la distancia requerida.

En la parte del programa se usó un sistema operativo bastante completo para poder controlar al robot y también comunicarse con la parte de inteligencia cargada en la pocket PC “Ipaq”, la cual permite hacer programas más complejos al utilizar algoritmos de solución mejores además de incrementar la capacidad de aprender el laberinto al igual de trazarlo para conocer que ha aprendido el robot.

Para resolver el laberinto se utilizaron desde algoritmos simples como es el algoritmo de la mano derecha hasta terminar utilizando un conjunto de algoritmos para resolverlo utilizando una combinación del algoritmo de la mano derecha con el algoritmo de Dijkstra para calcular la ruta mínima y poder resolver el laberinto en el menor tiempo posible además de encontrar la ruta mas directa a la salida aprendiendo el laberinto después de recórrelo o guardar las intersecciones recorridas.

Capitulo 1 Introducción

Orígenes de los Laberintos

Existen numerosos monumentos, así como el recuerdo que la tradición griega guarda del Minotauro. El Minotauro no es, en efecto, sino Minos, es decir, la figura mítica principal de la antigua Creta en forma de toro; y la leyenda según la cual, encerrado en el laberinto, aquel devoraba a las víctimas, encubre a una divinidad infernal o divinidad de las tinieblas: lo que parece a su vez confirmado por la reaparición de Minos en la leyenda griega como juez de las almas [2].

Supongamos que queremos diseñar un laberinto o que se nos ha encargado hacerlo. En seguida nos daremos cuenta de que, para trazar un diseño que agrade al ojo y a la mente y reproduzca una forma clásica, hemos de respetar algunos requisitos estéticos y al tiempo lógicos:

1. No podremos dejar espacios vacíos en el área disponible y hemos de procurar ocuparla de manera uniforme.

2. Tendremos que incluir en dicha área la mayor longitud de recorrido posible, aunque evitando que el trazado pueda llegar a ser oprimente.

3. Tendremos que buscar una unidad de estilo, diseñando todo el trazado con líneas rectas y ángulos de 90 grados, o bien hacer un círculo completo o al menos un trazado curvilíneo.

4. Evitaremos las repeticiones de anillos y de callejones sin salida. De lo contrario, el observador puede sentirse abrumado y agotarse rápidamente.

5. No hay que multiplicar los centros; es preferible que haya un solo centro. Aun sin apelar a la sacralidad que pueda tener el centro, la existencia de un solo centro brinda al observador una sensación de equilibrio logrado, de permanencia en la lógica.

6. No se han de aumentar las dificultades hasta el extremo de poner al observador ante a un rompecabezas casi insoluble. Un laberinto no ha de convertirse en un nudo gordiano, en un problema ante el cual se sienta la tentación de abandonar la empresa o de simplificarla con recortes abusivos.

7. El recorrido ha de ser continuo. Si se parte desde el interior, tendrá que haber una vía de salida hacia el exterior; o bien un camino continúo hasta el centro, en el supuesto de que el laberinto sea centrípeto. En otros términos, el trazado no ha de ser nunca cerrado. Por ejemplo, el laberinto de mosaico de la Casa del laberinto de Pompeya no tiene comunicación con el exterior, y, además, tiene un dibujo rigurosamente simétrico. Por consiguiente, no obstante la remisión al laberinto de Creta y a su leyenda, el mismo es un pseudo-laberinto en doble sentido.

Hemos indicado que estas pautas generales valen para la construcción de un laberinto clásico y son fruto de la observación y la prolongada praxis. Luego, cada uno es dueño de proceder como le plazca.

Representación del recorrido

La representación puede traducirse en una descripción verbal. Por ejemplo: Sigue recto; luego, en el segundo cruce, gira a la derecha; en el primer trivio, entra en el recorrido central, etcétera. O bien en un diagrama lineal, a ser posible en escala, de la longitud del recorrido, prescindiendo por completo de la orientación y señalando los cruces con un símbolo de tránsito o de prohibición. Varias modalidades graficas son válidas para este fin. Nosotros sugerimos las siguientes:

a) Una flecha al principio (o dos flechas, si hay dos entradas) y más flechas colocadas en forma inversa y señalando la dirección del recorrido. Un disco para indicar el centro o la meta.

b) Una leyenda, como la de los mapas, que ofrezca la escala de las distancias.

c) Llamamos cruce a la simple bifurcación, y nudo a los trivios, etc.

d) Con una raya vertical encima de la línea de demarcación se señalan las bifurcaciones hacia la izquierda, y con otra debajo de la línea, las bifurcaciones hacia la derecha.

e) Una raya sesgada encima de la línea de demarcación señala los pasillos de subida, y otra debajo de la línea los pasillos de bajada.

f) Una marca doble sobre las rayas significa prohibido el paso.

Orígenes de los Robots

Podemos encontrar como definición de un robot, "máquina controlada por ordenador y programada para moverse, manipular objetos y realizar trabajos a la vez que interacciona con su entorno". Los robots son capaces de realizar tareas repetitivas de forma más rápida, barata y precisa que los seres humanos. El término procede de la palabra checa robota, que significa “trabajo obligatorio”; fue empleado por primera vez en la obra teatral de 1921 R.U.R. (Robots Universales de Rossum) por el novelista y dramaturgo checo Karel Èapek. Desde entonces se ha empleado la palabra robot para referirse a una máquina que realiza trabajos para ayudar a las personas o efectúa tareas difíciles o desagradables para los humanos [5].

Los primeros robots creados en toda la historia de la humanidad, no tenían más que un solo fin: Entretener a sus dueños, Estos inventores se comenzaron a dar cuenta de que los robots podían imitar movimientos humanos o de alguna criatura viva. Estos

movimientos, pudieron ser mecanizados y de esta manera se podía automatizar y mecanizar algunas de las labores más sencillas de aquellos tiempos.

El origen del desarrollo de la robótica, se basa en el empeño por automatizar la mayoría de las operaciones en una fábrica. Los autómatas, o máquinas semejantes a personas, ya aparecían en los relojes de las iglesias medievales, y los relojeros del siglo XVIII eran famosos por sus ingeniosas criaturas mecánicas.

Algunos de los primeros robots empleaban mecanismos de realimentación para corregir errores, mecanismos que siguen empleándose actualmente. Un ejemplo de control por realimentación es un bebedero que emplea un flotador para determinar el nivel del agua. Cuando el agua cae por debajo de un nivel determinado, el flotador baja, abre una válvula y deja entrar más agua en el bebedero. Al subir el agua, el flotador también sube, y al llegar a cierta altura se cierra la válvula y se corta el paso del agua.

El primer auténtico controlador realimentado fue el regulador de Watt, inventado en 1788 por el ingeniero británico James Watt. Este dispositivo constaba de dos bolas metálicas unidas al eje motor de una máquina de vapor y conectadas con una válvula que regulaba el flujo de vapor. A medida que aumentaba la velocidad de la máquina de vapor, las bolas se alejaban del eje debido a la fuerza centrífuga, con lo que cerraban la válvula. Esto hacía que disminuyera el flujo de vapor a la máquina y por tanto la velocidad [7].

El control por realimentación, el desarrollo de herramientas especializadas y la división del trabajo en tareas más pequeñas que pudieran realizar obreros o máquinas fueron ingredientes esenciales en la automatización de las fábricas en el siglo XVIII. A medida que mejoraba la tecnología se desarrollaron máquinas especializadas para tareas como poner tapones a las botellas o verter caucho líquido en moldes para neumáticos. Sin embargo, ninguna de estas máquinas tenía la versatilidad del brazo humano, y no podían alcanzar objetos alejados y colocarlos en la posición deseada.

El desarrollo del brazo artificial multiarticulado, o manipulador, dio origen al moderno robot. El inventor estadounidense George Devol desarrolló en 1954 un brazo primitivo que se podía programar para realizar tareas específicas. En 1975, el ingeniero mecánico estadounidense Víctor Scheinman, cuando estudiaba la carrera en la Universidad de Stanford, en California, desarrolló un manipulador polivalente realmente flexible conocido como Brazo Manipulador Universal Programable (PUMA, siglas en inglés de Programable Universal Manipulator Arm). El PUMA era capaz de mover un objeto y colocarlo en cualquier orientación en un lugar deseado que estuviera a su alcance. El concepto básico multiarticulado del PUMA es la base de la mayoría de los robots actuales.

El diseño de un manipulador robótico se inspira en el brazo humano, aunque con algunas diferencias. Por ejemplo, un brazo robótico puede extenderse telescópicamente, es decir, deslizando unas secciones cilíndricas dentro de otras para alargar el brazo. También pueden construirse brazos robóticos de forma que puedan doblarse como la trompa de un elefante. Las pinzas están diseñadas para imitar la función y estructura de la

mano humana. Muchos robots están equipados con pinzas especializadas para agarrar dispositivos concretos, como una gradilla de tubos de ensayo o un soldador de arco.

Las articulaciones de un brazo robótico suelen moverse mediante motores eléctricos. En la mayoría de los robots, la pinza se mueve de una posición a otra cambiando su orientación. Una computadora calcula los ángulos de articulación necesarios para llevar la pinza a la posición deseada, un proceso conocido como cinemática inversa.

Algunos brazos multiarticulados están equipados con servo controladores, o controladores por realimentación, que reciben datos de un ordenador. Cada articulación del brazo tiene un dispositivo que mide su ángulo y envía ese dato al controlador. Si el ángulo real del brazo no es igual al ángulo calculado para la posición deseada, el servo controlador mueve la articulación hasta que el ángulo del brazo coincida con el ángulo calculado. Los controladores y los ordenadores asociados también deben procesar los datos recogidos por cámaras que localizan los objetos que se van a agarrar o las informaciones de sensores situados en las pinzas que regulan la fuerza de agarre.

Cualquier robot diseñado para moverse en un entorno no estructurado o desconocido necesita múltiples sensores y controles (por ejemplo, sensores ultrasónicos o infrarrojos) para evitar los obstáculos. Los robots como los vehículos planetarios de la NASA necesitan una gran cantidad de sensores y unas computadoras a bordo muy potentes para procesar la compleja información que les permite moverse. Eso es particularmente cierto para robots diseñados para trabajar en estrecha proximidad de seres humanos, como robots que ayuden a personas discapacitadas o sirvan comidas en un hospital. La seguridad debe ser esencial en el diseño de robots para el servicio humano [18].

Lenguajes de Programación

Se puede decir que un lenguaje de programación es el intermediario entre la máquina y el usuario para que este último pueda resolver problemas a través de la computadora haciendo uso de funciones que le traducen dicho programa a la computadora para la realización de dicho trabajo.

Desde que se desarrollaron las máquinas programables se han desarrollado lenguajes con los cuales las personas puedan dar órdenes a éstas [10]. En su orden los lenguajes de programación se pueden clasificar así:

- Lenguaje de máquina: los programas eran diseñados en código binario, eran difíciles de leer, entender y por su puesto de corregir. Los programas se caracterizaban por ser pequeños.

- Lenguajes de bajo nivel: se desarrolló un lenguaje conocido como lenguaje ensamblador. Este lenguaje era encargado de tomar algunas palabras comunes de

una persona y traducirlas al código máquina. Lo anterior facilitaba un poco la escritura de programas.

- Lenguajes de alto nivel: se desarrollaron lenguajes de programación que estuvieran más cerca de un lenguaje natural (español, inglés, francés, etc.). Como se hace necesario traducir el programa a lenguaje de máquina, en los lenguajes de alto nivel esa operación la realiza algo que se le conoce con el nombre de compilador.

Dependiendo del lenguaje de programación que se elija, se puede hablar del tipo de programación que se va a realizar.

- Secuencial: se considera a los programas que se diseñan con instrucciones que van unas detrás de otras. Las líneas se ejecutan una a una en secuencia.

- Estructurada: se considera a la programación que se hace por módulos. Cada módulo realiza alguna tarea específica y cuando se necesite esa tarea simplemente se hace el llamado a ese módulo independiente de que se tengan que ejecutar los demás.

- Orientada a objetos: se considera a aquellos lenguajes que permiten la utilización de objetos dentro del diseño del programa y el usuario puede pegar a cada objeto código de programa.

- Lógica o de lenguaje natural: son aquellos programas que se diseñan con interfaces tal que la persona o usuario puede ordenar a la máquina tareas en un lenguaje natural. Pueden interactuar como una persona pero nunca llegan a producir conocimiento.

- Inteligencia artificial: son programas que se acercan a la inteligencia humana. Estos programas son capaces de desarrollar conocimiento. Este tipo de lenguajes trabajan de manera similar a la mente humana.

C es un lenguaje de programación de propósito general. Sus principales características son:

- Programación estructurada
- Economía de las expresiones.
- Abundancia en operadores y tipos de datos.
- Codificación en alto y bajo nivel simultáneamente.
- Reemplaza ventajosamente la programación en ensamblador (assembler).
- Utilización natural de las funciones primitivas del sistema.
- No está orientado a ningún área en especial.
- Producción de código objeto altamente optimizado.
- Facilidad de aprendizaje.

El lenguaje C nació en los Laboratorios Bell de AT&T y ha sido estrechamente asociado con el Sistema Operativo UNIX, ya que su desarrollo se realizó en este sistema y debido a que tanto UNIX como el propio compilador de C y la casi totalidad de los programas y herramientas de UNIX, fueron escritos en C. Su eficacia y claridad han hecho que el lenguaje ensamblador apenas haya sido utilizado en UNIX [6].

Este lenguaje está inspirado en el lenguaje B escrito por Ken Thompson en 1970 con intención de recodificar el UNIX, que en la fase de arranque está escrito en ensamblador. En vista a su transportabilidad a otras máquinas. B era un lenguaje evolucionado e independiente de la máquina, inspirado en lenguajes BCPL concebido por Martin Richard en 1967.

En 1972, Dennis Ritchie, toma el relevo y modifica el lenguaje B, creando el lenguaje C y rescribiendo el UNIX en dicho lenguaje. La novedad que proporcionó el lenguaje C sobre el B fue el diseño de tipos y estructuras de datos.

Una de las peculiaridades de C es su riqueza de operadores, Puede decirse que prácticamente dispone de un operador para cada una de las posibles operaciones en código máquina.

Finalmente, C, que ha sido pensado para ser altamente transportable y para programar lo improgramable, igual que otros lenguajes tiene sus inconvenientes:

- Carece de instrucciones de entrada/salida, de instrucciones para manejo de cadenas de caracteres, con lo que este trabajo queda para la biblioteca de rutinas, con la consiguiente pérdida de transportabilidad.
- La excesiva libertad en la escritura de los programas puede llevar a errores en la programación que, por ser correctos sintácticamente no se detectan a simple vista.
- Por otra parte las precedencias de los operadores convierten a veces las expresiones en pequeños rompecabezas.

A pesar de todo, C ha demostrado ser un lenguaje extremadamente eficaz y expresivo.

Una persona piensa y se comporta obedeciendo a un secuencial lógico. Una computadora realiza tareas y maneja datos en memoria obedeciendo a una secuencia de pasos lógicos para lo cual ha sido programado. La programación de computadoras permite a una persona definir una secuencia de instrucciones que indica las acciones que han de ejecutarse para dar solución a un problema determinado de una manera rápida. La programación de computadoras es indispensable en cualquier área de la ingeniería, ya que diferentes problemas que se puedan presentar tarda tiempo resolverlos de manera manual. La computadora resuelve problemas, de acuerdo a como se le haya programado de manera rápida.

Lo anterior deja muchas cosas que decir. Para llegar a tener una secuencia de instrucciones que den solución a un problema es necesario ejecutar varias etapas.

- Etapa de análisis: en esta etapa el programador debe entender claramente el problema. Saber que es lo que se quiere resolver.

- Etapa de solución general: escribir la serie de pasos que sean necesarios para dar solución al problema. Estos pasos se pueden desarrollar a través de un diagrama de flujo ó a través de un seudo lenguaje. A lo anterior es lo que se conoce con el nombre de algoritmo.

- Etapa de prueba: consiste en verificar el algoritmo paso a paso para estar seguro si la solución realmente resuelve el problema.

- Etapa de implementación específica: consiste en traducir el algoritmo a un lenguaje de programación.

- Etapa de prueba: consiste en ejecutar el programa en una computadora y revisar los datos arrojados para ver si son correctos y hacer los ajustes necesarios.

- Etapa de uso: consiste en instalar el programa de manera definitiva para el uso por parte del usuario.

Se puede definir un lenguaje de programación como un conjunto de reglas ó normas, símbolos y palabras especiales utilizadas para construir un programa y con él, darle solución a un problema determinado. El lenguaje de programación es el encargado de que la computadora realice paso a paso las tareas que el programador ha diseñado en el algoritmo.

Capítulo 2 Robots móviles

ROBOTS MÓVILES

En cuanto al tipo de robots que son de interés en este proyecto, los robots móviles y autónomos, desde finales de los sesenta ya algunas universidades americanas trabajaban en robots que navegaban en ambientes interiores altamente estructurados y a finales de los setenta ya existían prototipos como el Moravec, desarrollado por la Universidad de Stanford, el cual fue el primer robot autónomo en navegar en ambientes exteriores naturales. Desde ese tiempo ha habido una proliferación de desarrollos alrededor de máquinas de conducción autónoma que desarrollan tareas concretas y pueden navegar en terrenos al aire libre para aplicaciones comerciales.

APLICACIONES DE LOS ROBOTS MÓVILES.

Los robots son herramientas, que se utilizan para reducir la cantidad de trabajo humano, para realizar tareas específicas o satisfacer una necesidad. La robótica móvil es un campo de desarrollo de gran auge en la actualidad. Se pueden dividir los proyectos actuales en proyectos comerciales y proyectos precomerciales en desarrollo, donde los primeros son aplicaciones comerciales e industriales ya desarrolladas y los segundos son los proyectos en desarrollo como prototipos para posteriores aplicaciones comerciales o industriales.

Al principio se visualizaron robots móviles descendiendo dentro de los volcanes, explorando la superficie de Marte y limpiando pisos en instalaciones y oficinas. Estas ideas y otras se han estado desarrollando en los últimos años, algunos de los proyectos comerciales más interesantes se mencionan a continuación.

Las primeras aplicaciones fueron en el campo de la limpieza, el potencial de robots que aspiran y limpian se está aprovechando en el mercado actual. En 1991 una compañía dedicada a comercializar robots móviles, la compañía DENNING MOBILE ROBOTS and WINDSOR INDUSTRIES produjeron el ROBOSCRUB (figura 1), un robot que aspira y limpia pisos de grandes capacidades. Para su fabricación se tomó un robot móvil manual que realizaba la misma función y se le adicionaron elementos para eliminar la necesidad de un operario. El robot se programó para seguir un patrón a través del espacio a ser limpiado, percibiendo la presencia de obstáculos para no tropezar con ellos. Si detecta un obstáculo a cierta distancia, reduce su velocidad, si un objeto está muy cerca trata de evitarlo y si el objeto está muy próximo, el robot para y espera. Tiene incorporados transductores tipo sonar para la detección de obstáculos, un detector infrarrojo para detección de precipicios, parachoques conectados a switches, y un sistema de navegación láser de alta precisión [19].

Figura 1. RoboScrub

El ROBOKENT fue otro robot fabricado para labores de limpieza de espacios grandes por CLEANING EQUIPMENTS en 1988, y ha sido tal vez el robot de limpieza de mayor éxito en Norteamérica. Usa un sonar como sensor primario de detección de obstáculos, usa un método de detección de colisiones y de precipicios. El ROBOKENT (figura 2) incluye un módulo que permite el control directo de un operador como si se tratara de cualquier equipo manual. Para iniciar la limpieza, un operador guía el robot a través del área a limpiar, luego el robot recorre el perímetro hasta regresar al punto de origen y luego comienza su tarea de limpieza.


Figura 2. Robokent

Otro robot producido por DENNING MOBILE ROBOTICS fue el DENNING SENTRY destinado a ser un robot de seguridad. Mediante sensores de detección de intrusos el DENNING SENTRY (figura 3) puede hacer patrullajes de manera infatigable en una instalación, bodega o similar. Cuando su fuente de energía está casi agotada el DENNING SENTRY automáticamente retorna a su estación de carga y recarga sus

baterías sin asistencia de ningún operador. Incorpora un anillo de sonares para detección de obstáculos, sensores de movimiento de tipo infrarrojo y de microondas. También posee una cámara de TV, micrófono y transmisores inalámbricos para enviar la información a la estación de seguridad. Dependiendo de la ruta almacenada en el programa, puede seguir un haz de luz de determinado alcance hasta una intersección donde otro haz de luz sea detectado.

Figura 3. Denning Sentry

Otro robot con funciones orientadas a rondas y recorridos es el HELPMATE, un tipo de robot enfermero de HELPMATE ROBOTICS INC. Nace de la reducción de la efectividad de los enfermeros y otros trabajadores de hospitales debido al tiempo que pierden en labores incidentales y fue diseñado para aliviar el peso de esas tareas. El HELPMATE (figura 4) recorre los corredores de los hospitales combinando el Dead Reckoning y la ubicación por sonar. Presionando un botón el operario puede enviar el robot a determinado lugar del mapa y navegar de una estación programada a otra. Incorpora sonares para evitar obstáculos y un sistema de triangulación por luz estroboscóspica. Posee parachoques sensibles a las colisiones, puede trasportarse por elevadores y puertas automáticas y trasladarse a casi cualquier lugar de una edificación. Ha sido instalado en más de 100 hospitales en el mundo entero. Estos robots son vendidos o arrendados a los hospitales que pagan por horas de trabajo.

Figura 4. Helpmate

En otras áreas de desarrollo, los robots móviles son usados en situaciones que implican riesgo y peligro para humanos. Éstos son robots teleoperados, y han sido usados en el seguimiento de la limpieza de incidentes como los ocurridos en las plantas nucleares de Chernobyl y Three Mile Island. Además la policía y la milicia de diversos países usan robots controlados remotamente para recoger, probar o destruir bombas y explosivos contenidos en paquetes sospechosos.

La misión Pathfinder que viajó a Marte en 1997 llevó el que se puede considerar el robot móvil autónomo de mayor éxito en la historia. El robot Sojourner concentró la atención de millones de personas en su exploración del planeta rojo. El Sojourner (figura 5) ejecutó fielmente los comandos enviados desde controladores en el laboratorio de Propulsión a Cohete en California. Este robot enviado a otro planeta además podía tomar acciones por sí mismo, estando alerta de obstáculos y abismos. El Sojourner podía ejecutar contraordenes de las enviadas desde la Tierra si en determinado momento se encontraba en peligro, y, en efecto en varias ocasiones durante el curso de la misión lo hizo. En el caso en que la comunicación con la tierra se perdiera el robot tenía la habilidad de continuar la misión por sí mismo. El reducido tamaño del Sojourner y el bajo costo de la misión entera, cambió el curso de las nuevas misiones de la NASA donde existía la visión de que lo más grande era lo mejor. El costo de colocar una carga útil en otro planeta es proporcional al peso de la misma carga, pero con una constante de proporcionalidad muy grande. Por esto la única forma de minimizar los costos de esta misión espacial era limitar severamente el tamaño de la carga útil: el robot debía ser pequeño. Como las comunicaciones entre la Tierra y Marte duran 40 minutos en viajar, el retraso impide la teleoperación directa. Así, este robot de muy modesta plataforma de procesamiento (por consideraciones de alimentación), debió ser capaz de operar autónomamente. La NASA reconoció que el control por comportamientos es la arquitectura de software lógica para robots autónomos de este tipo. El Sojourner fue desarrollado a partir de una serie de varios prototipos tempranos y fue programado de acuerdo a los principios del control por comportamientos.

Figura 5. Sojourner

Otro de los robots más interesantes es el FETCH (figura 6), diseñado con el serio propósito de remover las pequeñas bombas y granadas de la munición anti-tanque que no

explotan en el campo de batalla. Esta munición es soltada desde aeronaves y se calcula que el 25% de ella no explota, lo que engendra un grave problema. La remoción de esta munición es una tarea perfecta para un robot. En la evaluación el FETCH demostró que un robot puede deshacerse de explosivos de artillería. El FETCH tuvo éxito en su navegación autónoma a un punto, en la búsqueda en espiral de la munición, en su localización, en levantarla y en trasportarla a un punto. Sin embargo los costos aún no la han hecho un producto de producción masiva.

Figura 6. Fetch

Otros proyectos que no se consideran comerciales se desarrollan actualmente en un sinnúmero de universidades de países como Estados Unidos, Alemania, Inglaterra, Japón, España, etc., algunos de los cuales se mencionan a continuación.

En la Universidad de Wüzburg, en Alemania, se trabaja en un proyecto de diseño de algoritmos de navegación independiente de hitos necesarios para el desarrollo de robots móviles. El propósito de la investigación es estudiar las propiedades fundamentales del problema, para hallar la adecuada representación para él y probar sus soluciones en aplicaciones reales usando tecnología láser para la percepción. El proyecto se denomina "Localization and Navigation for Autonomous Robots in Industrial and Service Environments Using Laser Radar"

En la universidad del sur de California se desarrolla el proyecto de un vehículo aéreo autónomo denominado AVATAR (Autonomous Vehicle Aerial Tracking And Reconnaissance). Este proyecto desarrollado desde 1997 es el tercer robot volador dentro de la línea de robots aéreos de la universidad. Fue diseñado con la experiencia y requisitos con que se desarrollaron los proyectos anteriores en un marco de mayor alcance. Los requerimientos estuvieron dados por las pautas del contrato con la agencia de proyectos de investigación avanzados de defensa de Estados Unidos, DARPA. El AVATAR debe interactuar autónomamente en ambientes dinámicos en coordinación con otros robots terrestres. Además debe proporcionar información de la situación a un operario y debe ser capaz de reprogramarse. Para cubrir esas necesidades el AVATAR requiere gran percepción, recursos de comunicación y procesamiento como capacidades de transportar la carga útil.

Algunos proyectos desarrollados por diversos grupos en instituciones tienen por objeto mejorar la industria ferroviaria, desarrollar sistemas de telerobots para el mantenimiento de líneas de distribución de energía eléctrica, construir robots que puedan apagar pequeños incendios dentro de hogares, o sencillamente construir juguetes sofisticados y educativos.

Capítulo 3 Sensores

Un robot se puede definir como un sistema electromecánico e informático que interactúa con el medio. Los robots tal y como los concebimos actualmente, necesitan relacionarse con su entorno para poder llevar a cabo sus actividades. La actividad global de cualquier robot se puede entender como la sucesión de las siguientes cinco fases o actividades:

- Medida
- Modelaje
- Percepción
- Planificación
- Acción

Las tres primeras actividades están encaminadas a que el robot pueda percibir lo que esta pasando en su entorno. La planificación consiste en, a partir de la información percibida, tomar las decisiones oportunas para desarrollar su actividad. Por último, la acción consiste en la ejecución de las tareas planificadas en la fase anterior. Para un informático, la fase que puede resultar más atractiva es la de la planificación, ya que es en la que se concentra la mayor parte de la actividad "inteligente" del robot. Sin embargo, un robot no podría hacer nada si no pudiera "medir" de alguna forma lo que le interesa del medio en el que se desarrolla su actividad. Para poder realizar esta primera (fundamental) fase, los robots disponen de unos dispositivos llamados SENSORES. Los sensores cumplen la misma función en los robots que los órganos sensoriales en la mayoría de los seres vivos. Sin ellos los robots no podrían localizar objetos para poder agararlos, evitar obstáculos para no chocarse, comprobar el correcto funcionamiento de una actividad, Además, los sensores ayudan al robot a conocer sus parámetros internos, tales como la posición, la velocidad, etc. [1].

Los sensores son en realidad unos elementos físicos que estan compuestos de dispositivos llamados *transductores*. Los transductores son unos elementos capaces de transformar una variable física en otra diferente. Los sensores son un tipo concreto de transductores que se caracterizan porque son usados para medir la variable transformada. La magnitud física que suele ser empleada por los sensores como resultado suele ser la tensión eléctrica, debido a la facilidad del trabajo con ella. Desde el punto de vista de la forma de la variable de salida, podemos clasificar los sensores en dos grupos: *analógicos*, en los que la señal de salida es una señal continua, analógica; y *digitales*, que transforman la variable medida en una señal digital, a modo de pulsos o bits. En la actualidad los sensores más empleados son los digitales, debido sobre todo, a la compatibilidad de su uso con los ordenadores.

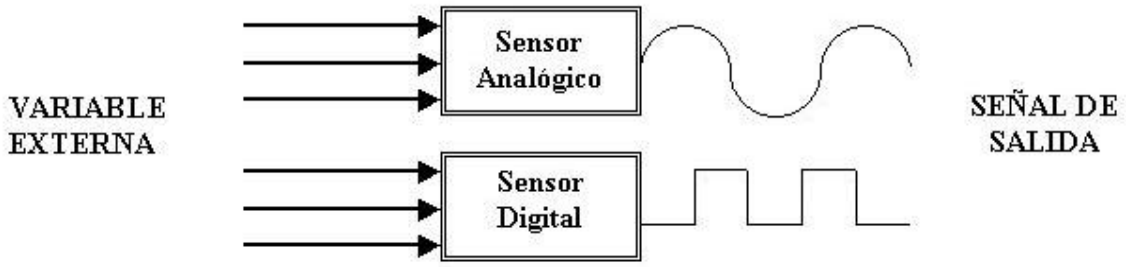


Figura 7. Tipos de señales

A los sensores, se les debe exigir una serie de características, que pasamos ahora a enumerar y comentar:

- ***Exactitud***. Hace referencia a que se debe poder detectar el valor verdadero de la variable sin errores sistemáticos. Sobre varias mediciones, la media de los errores cometidos debe tender a cero.
- ***Precisión.*** Una medida será más precisa que otra si los posibles errores aleatorios en la medición son menores. Debemos procurar la máxima precisión posible.
- ***Rango de funcionamiento.*** El sensor debe tener un amplio rango de funcionamiento, es decir, debe ser capaz de medir de manera exacta y precisa un amplio abanico de valores de la magnitud correspondiente.
- ***Velocidad de respuesta***. El sensor debe responder a los cambios de la variable a medir en un tiempo mínimo. Lo ideal sería que la respuesta fuera instantánea.
- ***Calibración***. La calibración es el proceso mediante el que se establece la relación entre la variable medida y la señal de salida que produce el sensor. La calibración debe poder realizarse de manera sencilla y además el sensor no debe precisar una recalibración frecuente.
- ***Fiabilidad***. El sensor debe ser fiable, es decir, no debe estar sujeto a fallos inesperados durante su funcionamiento.
- ***Costo***. El costo para comprar, instalar y manejar el sensor debe ser lo más bajo posible.
- ***Facilidad de funcionamiento***. Por último, sería ideal que la instalación del sensor no necesitara de un aprendizaje excesivo.

Todas estas características son las deseables en los sensores. Sin embargo, en la mayoría de los casos lo que se procurará será un compromiso entre su cumplimiento y el coste que ello suponga a la hora del diseño y fabricación

Valores de salida de los sensores:

Los sensores ayudan a trasladar los atributos del mundo físico en valores que la controladora de un robot puede usar. En general, la mayoría de los sensores pueden ser divididos en dos grandes grupos:

Sensores Analógicos
Sensores Digitales

Un sensor analógico es aquel que puede entregar una salida variable dentro de un determinado rango. Un Sensor analógico, como por ejemplo una Fotorresistencia (estos componentes miden intensidad de luz), puede ser cableado en un circuito que pueda interpretar sus variaciones y entregar una salida variable con valor es entre 0 y 5 volts.

Un sensor digital es aquel que entrega una salida del tipo discreta. Es decir, que el sensor posee una salida que varía dentro de un determinado rango de valores, pero a diferencia de los sensores analógicos, esta señal varía en pequeños pasos preestablecidos [1].

Sensores Digitales de uso general:

Existe una gran variedad de sensores digitales. Muchos de ellos se conectan en forma similar, la cual es haciendo uso de una resistencia de Pull-Up conectada a VCC para mantener la entrada forzada a nivel alto, con lo cual el sensor la forzaría a nivel bajo cuando se active.

Switch:

Uno de los sensores más básicos son los switches (pulsadores). Para evitar pulsos de rebote al accionar el switch se puede usar un capacitor de bajo valor (0.1 μF a 1 μF) en paralelo con los bornes del switch.

Sensores infrarrojos optoacoplados:

Existen dos tipos de sensores infrarrojos: reflectivo y de ranura. En ambos casos estos se basan en un conjunto formado por un fototransistor (transistor activado por luz) y un LED infrarrojo.

Reflectivo:

Este tipo de sensor presenta una cara frontal en la cual se encuentran tanto el LED como el Fototransistor.

Debido que no están colocados en forma enfrentada, la única forma posible para que la luz generada por el LED active el Fototransistor es haciendo reflejar esta luz en una superficie reflectiva. Teniendo en cuenta esto, estos sensores son muy útiles para detectar, por ejemplo, una línea negra sobre una superficie blanca o viceversa.

Debido a que el fototransistor es afectado no sólo por la luz del diodo sino por la luz ambiental, se deben desarrollar circuitos de filtrado para evitar una falsa activación debido a la luz ambiente.

De Ranura:

En este tipo de sensor, ambos elementos (LED y Fototransistor) se encuentran alineados a la misma altura enfrentados a través de la ranura. El fototransistor se encontrará activado siempre que no se introduzca ningún elemento que obture la ranura.

Sensor de Efecto Hall:

Otro sensor muy útil y simple de usar es el de Efecto may [3]. Se trata de un semiconductor que actúa como detector de proximidad al enfrentarse al polo sur de un imán. Utilizando el Efecto Hall para proporcionar una conmutación sin rebotes.

La distancia a la que produce la conmutación el campo magnético del imán es de alrededor de 2mm (dependiendo del modelo usado).

Son muy usados en circuitos lógicos en donde se precisa conmutar sin que se produzcan rebotes, o en donde se quiera evitar el contacto mecánico. Como por ejemplo es posible realizar un circuito que mida las revoluciones a la que está girando una rueda.

Sensores para automoción

Se incluyen sensores de Efecto Hall, de presión y de caudal de aire. Estos sensores son de alta tecnología y constituyen soluciones flexibles a un bajo costo. Su flexibilidad y durabilidad hace que sean idóneos para una amplia gama de aplicaciones de automoción.

Sensores infrarrojos

La optoelectrónica es la integración de los principios ópticos y la electrónica de semiconductores. Los componentes optoelectrónicos son sensores fiables y económicos. Se incluyen diodos emisores de infrarrojos (IREDs), sensores y montajes.

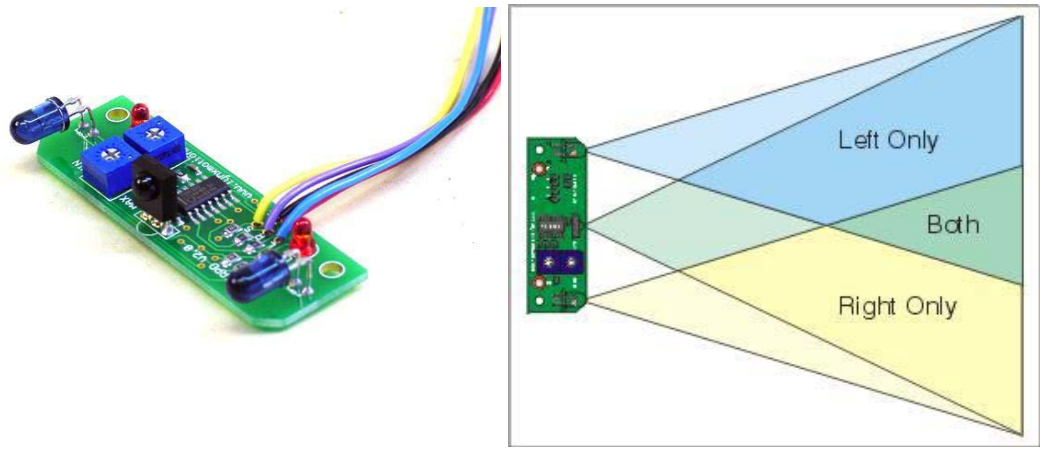


Figura 8. Sensores Infrarrojos

Sensores de caudal de aire

Los sensores de caudal de aire contienen una estructura de película fina aislada térmicamente, que contiene elementos sensibles de temperatura y calor. La estructura de puente suministra una respuesta rápida al caudal de aire u otro gas que pase sobre el chip.

Sensores de corriente

Los sensores de corriente monitorizan corriente continua o alterna. Se incluyen sensores de corriente lineales ajustables, de balance nulo, digitales y lineales. Los sensores de corriente digitales pueden hacer sonar una alarma, arrancar un motor, abrir una válvula o desconectar una bomba. La señal lineal duplica la forma de la onda de la corriente captada, y puede ser utilizada como un elemento de respuesta para controlar un motor o regular la cantidad de trabajo que realiza una máquina.

Sensores de humedad

Los sensores de humedad relativa / temperatura y humedad relativa están configurados con circuitos integrados que proporcionan una señal acondicionada. Estos sensores contienen un elemento sensible capacitivo a base de polímeros que interacciona con electrodos de platino. Están calibrados por láser y tienen una íntercambiabilidad de +5% HR, con un rendimiento estable y baja desviación.

Sensores de presión y fuerza

Los sensores de presión son pequeños, fiables y de bajo costo. Ofrecen una excelente repetitividad y una alta precisión y fiabilidad bajo condiciones ambientales variables. Además, presentan unas características operativas constantes en todas las unidades y una íntercambiabilidad sin recalibración. Los Sensores de Control le ofrece cuatro tipos de sensores de medición de presión: absoluta, diferencial, relativa y de vacío; y rangos de presión desde ±1,25 kPa a 17 bar.

Sensores de posición de estado sólido

Los sensores de posición de estado sólido, detectores de proximidad de metales y de corriente, están disponibles en varios tamaños y terminaciones. Estos sensores combinan fiabilidad, velocidad, durabilidad y compatibilidad con diversos circuitos electrónicos para aportar soluciones a las necesidades de aplicación.

Sensores de temperatura

Los sensores de temperatura se catalogan en dos series diferentes: TD y HEL / HRTS. Estos sensores consisten en una fina película de resistencia variable con la temperatura (RTD) y están calibrados por láser para una mayor precisión e íntercambiabilidad. Las salidas lineales son estables y rápidas.

Sensores de turbidez de líquidos

Los sensores de turbidez aportan una información rápida y práctica de la cantidad relativa de sólidos suspendidos en el agua u otros líquidos. La medición de la conductividad da una medición relativa de la concentración iónica de un líquido dado.

Estos son sólo algunos de los sensores más comúnmente usados en robótica debido a su practicidad y bajo costo. Existen muchos otros un poco mas sofisticados, como los transmisores y receptores ultrasónicos, con los cuales se pueden construir sistemas de sonar muy útiles a la hora de detectar objetos a distancia y así poder esquivarlos sin necesidad de tomar contacto con ellos.

Capítulo 4 Laberintos

¿Qué es un laberinto?

El caso es que resulta bastante menos sencillo de lo que puede parecer a primera vista, debido, entre otras cosas, a que el empleo metafórico de la palabra ha existido siempre, o, cuando menos, desde que existe un concepto arraigado de laberinto. Desde los tiempos de Platón, el mismo está presente como figura del lenguaje; y no sólo como figura, sino también como imagen que, podemos llamar arquetípica.

Ahora bien, la imagen no se plasma siempre en un dibujo trazado. El dibujo aparece en determinados períodos históricos, por lo que es perfectamente valido hablar de épocas de florecimiento del laberinto.

Pero, por volver a las definiciones, pasemos a consultar lo que dicen al respecto, algunos de los mejores diccionarios y alguna enciclopedia. Camino complicado, irregular, con muchos pasadizos, a través o alrededor de los cuales es difícil encontrar el camino sin un guía. Sitio en el que no se encuentra manera de salir. Embrollo, enredo, lío, confusión. Lugar enmarañado de caminos que dificultan la salida. Una red de pasadizos, pensada para confundir, en un edificio; una maraña de caminos [9].

Sólo que ninguna de estas definiciones, por poco que se reflexione, resulta satisfactoria y completa. Así, hay dos de laberinto en los que se encuentra siempre la manera de salir (laberintos unidireccionales o pseudo laberintos; no siempre el laberinto es un sistema de caminos; no siempre es una red de pasadizos.

Habría que decir: Camino tortuoso, concebido para obstaculizar, contrariar o engañar a todo aquel que se proponga alcanzar la meta a la que el mismo conduce, no, porque el camino no es siempre engañoso, no siempre hay encrucijadas y no existe siempre una meta. Entonces: Camino tortuoso, rodeado de muros o setos, tampoco, porque cabe decir que el laberinto sólo esté dibujado (en el suelo, en una lápida, en una hoja de papel, etc.). Cuanto más lo pensamos, mejor comprendemos que el objeto de nuestro interés, a mayor abundamiento laberíntico, no cabe en ninguna definición que lo abarque por entero y sin equívocos. Conformémonos, pues, con decir: Recorrido tortuoso, en el que a veces es fácil perder el camino sin un guía.

Formas y tipos

Las formas del laberinto no tienen límite, constituyendo siempre la adoptada en una época dada, en un determinado contexto social, el sello de un estilo propio, de una concepción de la vida, de una manera de ser.

Pero el laberinto no es siempre obra del hombre. Hay sistemas de grutas, debidas a la acción de las aguas y a una peculiar composición geológica de los estratos rocosos formados a lo largo de muchos milenios, que pueden dar lugar a un camino más o menos

complejo y cuya fisonomía original puede permanecer intacta o complicarse más por la acción del hombre, que actúa impulsado por motivos sagrados, de defensa o de simple curiosidad.

Así pues, desde esta perspectiva podemos hablar de:

a) Laberintos naturales, artificiales y mixtos

Desde los tiempos más remotos las estalactitas y las estalagmitas se han considerado elementos que aumentaban la sacralidad de una gruta. Recurriendo a palabras de una fantasía elemental y de fácil estímulo, se dieron nombres antropomorfos, teomorfos o teriomorfos a las formaciones que más excitaban la imaginación. Las estalactitas de formas singularmente impresionantes se aislaban del resto y a veces forman en su base un altar o bancos, labrados en la misma roca. Se ensanchaba o transformaba la entrada natural, de modo que para llegar a la parte de la gruta que se consideraba más sagrada, al más recóndito reducto, había que recorrer un itinerario determinado; o en su defecto, se ocultaba la entrada. En resumidas cuentas, el lugar era adaptado de un modo u otro para aumentar su cariz de tremendum, de mysterium mágnum.

Entre los laberintos naturales que la mano del hombre ha dejado casi intactos, el más imponente del continente europeo y tal vez uno de los más importantes del mundo es el situado en Postumia (Postoina en serbocroata, Adelsberg en Alemán), no muy lejos de la ciudad de Trieste.

Aquí, excavando lentamente su camino por la roca calcárea, en el transcurso de décadas incontables, las aguas tartáreas han creado un sistema de túneles y grutas, muchos de gran amplitud. El recorrido tiene una longitud de más de cinco kilómetros, si se sigue el típico trazado tunstico. En total, sin embargo, abarca más de 21 kilometres, entre caminos, bifurcaciones y callejones sin salida; y perfectamente se puede considerar un laberinto con todas las de la ley.

Evidentemente, la formación de las grutas y de los túneles no fue simultánea. Su origen se remonta a épocas y condiciones muy distintas, tanto atmosféricas como hídricas. No todo el sistema es obra casual del agua,.las primeras causas de su formación hay que buscarlas en las fallas y fracturas de origen teutónico y en la composición de los estratos rocosos.

La gran sala tiene 121 metros de largo, 50 de ancho y 33 de alto. El Auditorio es una caverna inmensa, de más de 3.000 metros cuadrados de superficie, con cabida para más de diez mil personas. El deslumbrado visitante pasa por un sinfín de túneles, subidas, bajadas, rellanos, grutas grandes y pequeñas, perspectivas abisales, enclaves en los que la naturaleza ha derrochado fantasía en construcciones estalactíticas y estalagmíticas de las formas y los colores más extravagantes, y donde la imaginación del hombre se ha explayado con los nombres más peculiares: cabeza de elefante, tortuga, sala del barco volcado, sala

gótica, cárcel, lavadero, galena de los cristales, tubos de órgano, momias, sartenes, gruta del silencio, gruta de los cristales, etcétera.

Otro ejemplo destacado pero mucho menos conocido, formado por un inmenso conjunto de pasajes subterráneos laberínticos, obra del agua, es el conjunto de las Grutas de Aggteiek, situado en la línea fronteriza entre Hungría y Checoslovaquia, cuya galería principal mide poco menos de seis kilómetros de largo. También allí hay un riachuelo subterráneo que, como era de esperar, se llama Estigia, al que se une otro curso de agua que recibe el nombre de Aqueronte. Y también proliferan las formaciones estalactíticas con nombres no menos peculiares, como la tortuga, sobre la que se cierne en vuelo un águila o la sala negra, hace treinta mil años morada del hombre prehistórico que allí levantó cabañas sobre palafitos, cuyos vestigios se conservan; y la gruta de los espectros, cuya pared estalagmítica mide más de 70 metros de altura, y, naturalmente, un auditorio y un salón de baile.

Figura 9 Laberinto Artificial tridimensional.

b) Laberintos casuales, secundarios e intencionales

Es el caso, por ejemplo, del conjunto de galerías subterráneas de una mina o del conjunto de túneles cavados como refugio o defensa, y donde a veces, para dificultar más una posible invasión, se sitúan puertas falsas, pasadizos falsos, recovecos de todo tipo. Dicha práctica la siguieron siempre los antiguos egipcios en la construcción de las tumbas reales (como sabe todo buen aficionado al tema), para ocultar a los buscadores de tesoros la entrada a la cámara funeraria. En este caso no puede hablarse de un recorrido estrictamente laberíntico, sino tan solo de

un añadido secundario, en el recorrido, de elementos capaces de confundir al visitante.

Para objetivos de defensa, encontramos elementos laberínticos también en la superficie de la tierra: nos referimos a las fortificaciones. El ingenio del hombre, valiéndose en ocasiones sólo de elementos constructivos, otras sumándolos a las formaciones rocosas halladas in Situ y singularmente aptas para sus fines, ha sabido muchas veces someter a dura prueba el arte obsidional llevando al atacante a un camino tortuoso que lo obligaba a permanecer largo tiempo expuesto a las armas del defensor y a pasar por obstáculos difícilmente salvables. Quien haya visitado la alta roca de Micenas recordará las múltiples revueltas de las murallas interiores, los recovecos contiguos a la Puerta de los Leones y lo difícil que es llegar al ultimo batallón defensivo, antesala del profundo pozo donde se almacenaba el agua, el elemento más preciado ante la probabilidad de un asedio. O bien recordemos, en otro contexto histórico, la acrópolis de Dimini, poblado del neolítico (IV milenio a. C.), cerca del Golfo de Volo (Tesalia). Era una aldea fortificada emplazada en una colina artificial de poca altura y forma casi ovalada, de aproximadamente 110 metros por 90, defendida por seis o siete recintos concéntricos hechos de piedras y barro, de modo que el paso de uno a otro suponía para el asediador avanzar por peligrosas revueltas, y con un reducto central, también fortificado.

Nos gustaría hacer además mención de la maravillosa fortaleza de Daulatabad, en la India, situada más o menos a mitad de camino entre Aurangabad y Ellora, a poca distancia de la aldea de mahometanos donde, bajo una cúpula de mármol que reluce bajo un sol abrasador, duerme el sueño eterno el devoto emperador mogol Aurangzeb, al que rodean, en la llanura adyacente, los monumentos funerarios de sus fieles ministros y generales. Enclavada en la cima de una alta colina, la fortaleza domina la llanura. Para subir hay que dar largos rodeos que culminan en una roca natural altísima y cortada a pico, desde cuya cima los defensores podían lanzar a discreción flechas, aceite hirviendo y piedras.

Ahora bien, el ejemplo históricamente más insigne del laberinto casual es indudablemente uno que todo el mundo conoce, el de las catacumbas romanas: habitaciones, templo, panteón y fortaleza a un tiempo, la complejidad de cuyo sistema es mayor debido al hecho de ser tridimensional, esto es, desarrollado no solo en extensión, sino también en profundidad.

Pero los laberintos naturales o parciales o casuales constituyen, por lo que a nosotros incumbe, la excepción a la regla, y si hemos hecho referencia a ellos ha sido sólo por prurito de rigor. Trataremos fundamentalmente de laberintos artificiales y veremos que también estos pueden contemplarse y clasificarse de muchas maneras.

En primer lugar, un laberinto puede presentar o no bifurcaciones; en otras palabras, puede estar hecho de manera que el recorrido sea solamente largo y

complicado, pero sin que quepa posibilidad de error en todo su desarrollo, o bien contar con un camino que imponga alterativas: en definitiva, puede tener una o varias vías. Así, en este sentido podríamos clasificar los laberintos como unidireccionales o multidireccionales, siguiendo, por ejemplo, a W. H. Matthews (Mazes and Labyrinths. A General Account of their History and Development), como unívíarios o pluriviarios o, para quien guste del griego, como monoodosicos o poliodosicos. Por razones de eufonía y sencillez, hemos elegido y utilizaremos la designación de:

c) Laberintos unívíarios y pluriviarios

Sería adecuado llamar seudo laberintos a los primeros (como, de hecho, los designaremos alguna vez en nuestra exposición), pues es obvio que no cabe considerar laberinto en sentido estricto un recorrido que solamente es largo, que no plantea dudas ni impone alterativas, en la medida en que es suficiente seguirlo para llegar a la meta o a la salida, y, en cualquier caso, a su fin. En cambio, en función de las formas de su trazado podemos dividir los laberintos en:

Figura 10. Laberintos univiarios.

d) Laberintos geométricos e irregulares

La distinción es tan obvia que no precisa comentarios

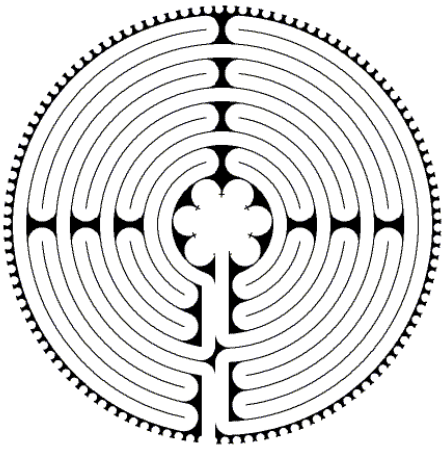


Figura 11. Laberinto geométrico

e) Laberintos de esquema fijo, irregular o mixto

En función de la forma del recorrido, el laberinto podrá tener un sistema:

f) Laberintos de rodeos rectangulares, curvos o mixtos

Ejemplos destacados de laberintos con rodeos curvos son, por orden de relación histórica, los laberintos babilonios de envoltorio de vísceras y la mayor parte de los prehistóricos; el esquema se seguirá durante varios siglos.

En función de la forma que desarrolla el recorrido en una planta, el laberinto podrá ser:

g) Laberinto rectangular, circular o de otra forma

En función de la concepción del dibujo, tendremos:

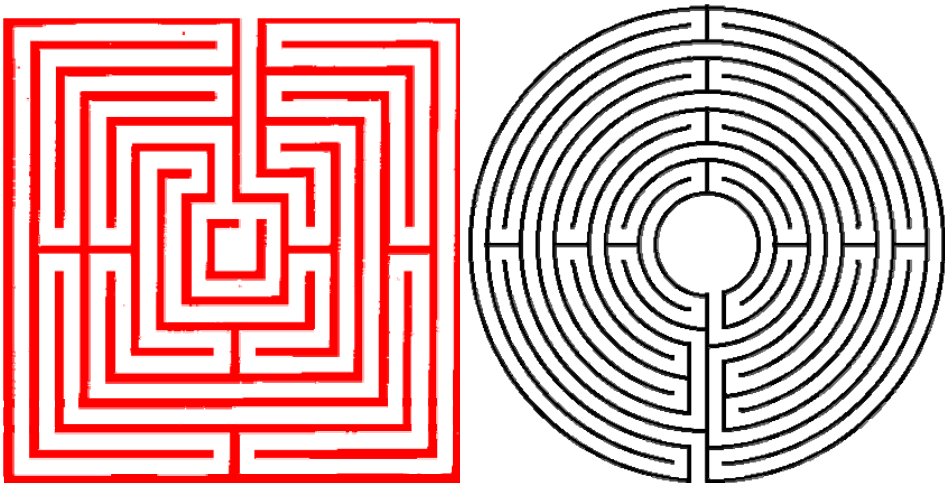


Figura 12. Laberinto rectangular y circular.

h) Laberintos simétricos o mixtos

Se sobreentiende que un laberinto completamente simétrico ha de ser siempre un seudolaberinto, es decir, un laberinto univiario, porque la existencia de una bifurcación, de una posibilidad de elección, rompe necesariamente la armonía perfecta del trazado. Con todo, veremos en muchos ejemplos -así, en los laberintos de mosaico romanos que el ingenio de los diseñadotes muchas veces ha buscado disimular del mejor modo posible la mayor o menor irregularidad que pueda inducir al error, de manera que sólo un ojo experto sea capaz de advertir la inserción en el conjunto de la planta.

En función de la relación entre la totalidad del recorrido y el área que este ocupe, el laberinto podrá ser:

Figura 13. Laberinto simétrico.

i) Laberintos compactos, difusos o mixtos

En el primer caso, el recorrido ocupa toda el área delimitada por los muros del perímetro; en el segundo, existen espacios vacíos sin comunicación con el recorrido. Todos los laberintos prehistóricos y clásicos pertenecen al tipo compacto; ejemplos más o menos elegantes de laberintos difusos los encontraremos en la época de su reflorecimiento (siglos XVII-XVIII).

Mucho más importante es la distinción siguiente: un laberinto puede tener un centro o no tenerlo; puede también tener varios centros. Habrá, por tanto:

j) Laberintos acéntricos, monocéntricos y policéntricos

Por lo demás, el centro podrá ser de paso o de llegada.

Dicho de otro modo, a lo largo del pasillo puede haber revueltas obligadas o recodos, desde los cuales, sin embargo, se hace reanudar el camino para llegar al centro propiamente dicho.
Además, el centro podrá ser cerrado o abierto, esto es, una vez en el centro, para llegar al exterior habrá que salir por la única puerta o bien habrá una segunda puerta y un camino distinto, que a su vez tendrá que estudiarse.

Más aún, el recorrido podrá empezar desde el centro o acabar en éste, con lo que tendremos laberintos centrípetos y centrífugos.

Todos los laberintos con un único centro pueden descomponerse idealmente en una espiral. Ahora bien, aunque nos resistamos a la tentación de identificar la espiral con el laberinto (pero sobre ello volveremos más adelante), no podemos por menos de pensar en una estrecha afinidad entre los dos signos. Todo aquel que posea ciertas nociones de etnología conoce la enorme frecuencia con la que aparece el signo cíclico en los medios culturales más dispares. Además, y aún sin apelar a la hipótesis arquetípica o de Valencia sacra (símbolo solar, etc.), el trazado de ese signo es bastante intuitivo y puede perfectamente constituir un primer esbozo del trazado laberíntico. Cuando nos refiramos a los dibujos prehistóricos comprobaremos que es fácil descubrir, o sorprender casi in fraganti, fases de actividad en las que el dibujante, partiendo de un dibujo de líneas paralelas y curvas que se despliegan en tomo a un centro -con frecuencia

espiriforme, aunque no siempre-, se ve casi inexorablemente llevado al trazado de un laberinto. Hay casos exacerbados en los que se detectan incertidumbres y tendencias de ese tipo. Por lo demás, si reflexionamos sobre el hecho de que, en general, la representación se desarrolla básicamente en el área del Mediterráneo y muy a menudo cerca de la orilla del mar, es seductor pensar que la primera ocurrencia de semejante trazado pudo nacer de la observación de las conchas marinas, cuya extenso uso es bien conocido, ya para aplicaciones simplemente utilitarias (moneda o adorno), ya, y con suma constancia, como objetos sagrados. Mircea Eliade consagra al simbolismo de las conchas un elegante capítulo de su libro Images et Symboles, que recoge un interesante estudio sobre la materia. Lo que predomina en ellas es la simbología ligada a los genitales femeninos. En este sentido, nada respalda mejor la teoría de que el laberinto pudo surgir de la observación de la estructura de las conchas que el aspecto de una concha disecada.

Figura 14. Laberinto monocéntrico.

Por poner punto final a las clasificaciones, diremos aun que un laberinto puede ser, como ya hemos visto en el caso de las catacumbas romanas,

k) Laberintos bidimensionales o tridimensionales

En función de que el mismo se despliegue en un único plano, en longitud y en anchura, o bien en planos distintos; en tal supuesto, puede bajar o subir, convertirse en caverna o en torre.

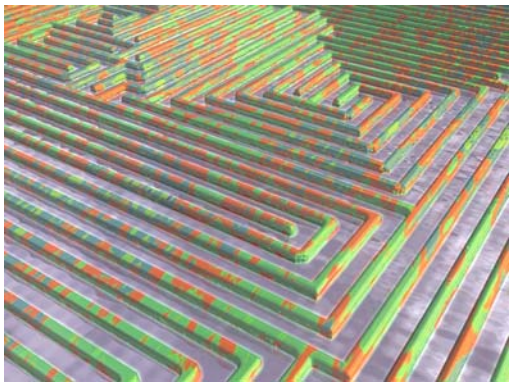


Figura 15. Laberinto tridimensional.

Desciframiento

Cada laberinto es una escritura secreta contenida en un espacio bidimensional o tridimensional: una clave. Por consiguiente, cabe que alguien -matemático o experto en psicología aplicada- haya elaborado un método de desciframiento de los laberintos. Sin embargo, quien esto escribe no ha logrado encontrar ningún estudio moderno sobre el tema. Por tal motivo, empezamos reproduciendo las normas contenidas en la llamada regla de Tremaux y Maurice (E. Vithet, Pour les petits et les grands, les vrais amusements educatifs. Comment en sortir ou lejeu du labyrinthe) [9]:

Es suficiente y necesario desplegar los dos recorridos de cada camino de ida en sentido contrario y no tomar la ida que ha conducido por primera vez a un cruce, a menos que no quede otra vía como opción. Supongamos que, al perder el camino en el laberinto, ponemos en la entrada de cada nuevo camino dos señales, y a la salida del mismo tres señales o solo una, en función de que el camino desemboque en un nuevo cruce o en un cruce ya explorado; además, cada vez que entremos en un camino en el que haya una sola serial a la entrada, ponemos otra. De esa manera, tendremos la seguridad de dar con la salida sin pasar mas de dos veces por cada camino, con tal de que sigamos la regia siguiente: cuando lleguemos a un cruce, tomaremos al azar uno de los caminos, siempre que el elegido no tenga serial, o un camino que tenga una sola serial; o, si ninguno de estos es el caso, tomaremos el camino que tenga tres señales.

Hay dos métodos posibles de solución:

Reconstruir de memoria y por intuición el camino ya hecho, con el riesgo de aumentar todavía más la confusión, o llegar al lado opuesto o al centro (según el tipo de laberinto de que se trate) por medio de comprobaciones sucesivas (trial and error method); o, por último, mediante un cálculo correcto de la serie (método éste que, sin embargo, puede alargar bastante el recorrido).

Si el laberinto no cuenta con series, el único método válido es el de las comprobaciones sucesivas. Si el laberinto está construido siguiendo una serie conceptual, dicho método también será valido; Ahora bien, siendo el número de las series seguramente finitas, y el de las series efectivamente empleadas será todavía menor, convendrá hacer la lista de una serie de cuatro unidades, cada una de las cuales se dividirá del modo siguiente:

Izquierda	Derecha	Izquierda	Derecha
Izquierda	Derecha	Izquierda	Izquierda
Izquierda	Izquierda	Derecha	Izquierda
Izquierda	Derecha	Derecha	Derecha
Izquierda	Derecha	Derecha	Izquierda
Izquierda	Izquierda	Derecha	Derecha
Izquierda	Izquierda	Izquierda	Derecha
Izquierda	Izquierda	Izquierda	Izquierda

Derecha	Izquierda	Derecha	Izquierda
Derecha	Izquierda	Izquierda	Derecha
Derecha	Derecha	Izquierda	Derecha
Derecha	Izquierda	Derecha	Derecha
Derecha	Derecha	Derecha	Derecha
Derecha	Izquierda	Izquierda	Izquierda
Derecha	Derecha	Izquierda	Izquierda
Derecha	Derecha	Derecha	Izquierda

La serie de tres unidades indivisibles, en cambio, es la siguiente:

Izquierda	Derecha	Izquierda
Derecha	Derecha	Derecha
Derecha	Izquierda	Derecha
Izquierda	Izquierda	Izquierda
Derecha	Derecha	Izquierda
Derecha	Izquierda	Izquierda
Izquierda	Izquierda	Derecha
Izquierda	Derecha	Derecha

Una serie de sólo dos unidades divisibles sería psicológicamente demasiado simple y pobre para la imaginación de un diseñador de laberintos lo bastante astuto ni siquiera las series derecha-derecha, derecha-izquierda, izquierda-izquierda e izquierda-derecha ofrecerán el suficiente interés.

Si se avanza por un laberinto dejando señales en cada ramificación, habrá que poner las mismas de modo que ningún sendero se recorra más de una vez en cada uno de los sentidos, para que así en cualquier laberinto acabado quepa alcanzar el centro, aunque no necesariamente por el camino más corto.

El cruce de tres senderos (número impar de senderos) constituye el tipo más simple de nudos impares; la conjunción de cuatro caminos (número par de senderos), el más simple nudo par. Así, si el principio está en el centro y sólo hay nudos impares, estaremos ante un laberinto unidireccional o cada uno de los anillos regresara al camino principal; tanto en un caso como en otro, podrá seguirse un solo recorrido desde la entrada hasta la salida, o viceversa.

Suponiendo que las señales del recorrido no se hayan quitado con anterioridad a que las veamos nuevamente, el siguiente es un plan que, en teoría, debe siempre resultar eficaz: en cada nudo nuevo (o sea, no visitado antes), la senda de llegada se distinguirá con tres señales. Si, llevados por señales previas por otros senderos, resulta que ese nudo ya ha sido visitado, pondremos una sola señal en el sendero de llegada. Si todos los senderos ya han sido señalados, significa que toda esa parte del laberinto se ha visitado antes y habrá que rehacer el camino, regresando por el sendero de llegada. Ahora bien, si uno o dos de los senderos de un nudo ya visitado permanecen sin señales, habrá que elegir uno cualquiera y ponerle dos señales. Si nos encaminamos por un sendero en el que hay una sola señal, pondremos dos señales más, para que el nudo de antes tenga ahora

tres. De ese modo, habremos recorrido todas las partes del laberinto cuando, al llegar a un nudo, no recorremos nunca por un sendero con tres señales, a menos que no haya ninguna senda exenta de señales o con una sola señal. Es imposible que haya un laberinto de estructura relativamente simple que no se pueda resolver con este método.

Si no podemos dejar señales, habrá que seguir otro sistema; con la mano derecha encima del muro o del seto de la derecha, o en el opuesto pero siempre con la misma mano, llegaremos en todo caso al centro, por tortuosos que sean los rodeos. La probable presencia de callejones sin salida no es un inconveniente, pues se pasará por ellos una sola vez en cada una de las direcciones. Del mismo modo se resolverá la elección del camino en los nudos. Pero ningún diseñador de laberintos, por poco hábil que sea, dejará de colocar el centro en el interior de un anillo; y, en tal caso, siguiendo el método del contacto continuo, acabaremos volviendo al punto de partida...

Si el centro se ha alcanzado desde un punto intermedio gracias a un método racional y no por el de las comprobaciones sucesivas (trial and error), es posible, pero no seguro, alcanzar la entrada invirtiendo completamente el sentido del camino recorrido. De Launay, sin embargo, ha demostrado la falacia de este método, porque una inversión completa del camino sólo conducía a un punto intermedio. Por no abundar en el hecho de que un método racional sólo es aplicable a leyes conocidas o, en el presente caso, a la posibilidad de descubrir el principio que ha inspirado el plano del constructor.

Capítulo 5 Hardware del Robot

Armazón:

La base sobre la que se ha construido todo el robot consiste en una superficie cuadrada de madera de 140 mm a la que se le han extraído dos secciones laterales destinadas a las ruedas [5].

Motores:

Los motores utilizados para generar el movimiento del robot se han dispuesto en una configuración de tracción diferencial en triciclo, que consiste en dos motores independientes formando un eje común.

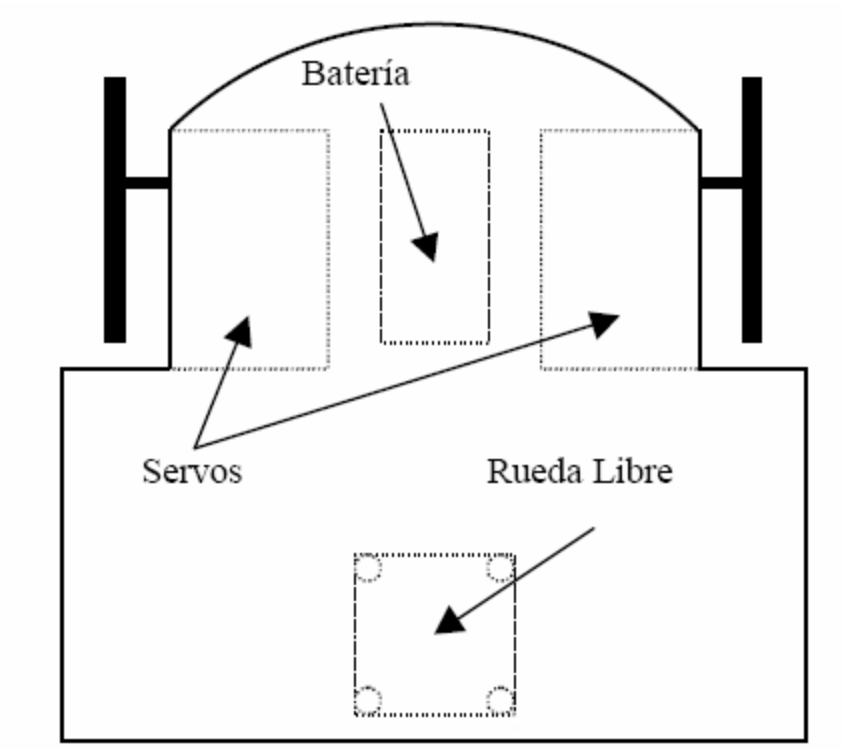


Figura 16. Estructura del Robot

Esta configuración permite al robot avanzar en línea recta, girar sobre si mismo y describir arcos. El punto de apoyo del triciclo consiste en una “rueda loca” obtenida en una ferretería industrial.

Los motores utilizados son servos de aeromodelismo de la marca Futaba, modificados para que dé la vuelta completa a los que se les ha eliminado toda la electrónica interna, así como las limitaciones físicas que sólo permiten un giro de 60º, de forma que se ha obtenido unos motores DC de giro continuo con su propia caja reductora.

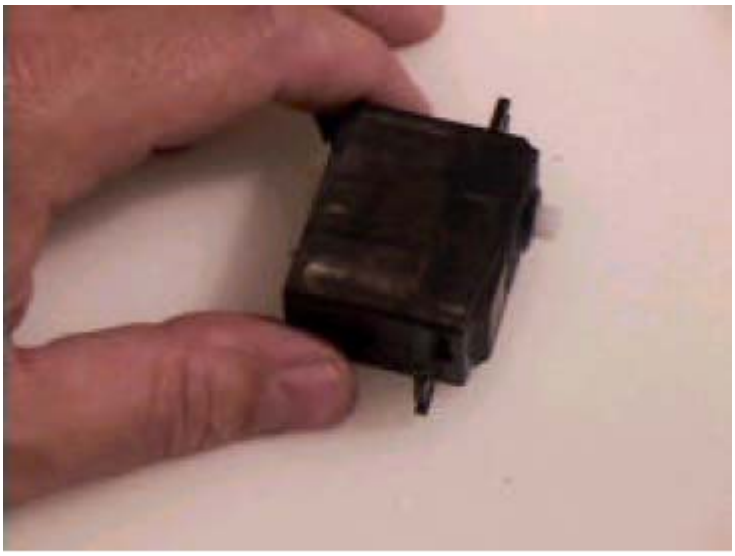


Figura 17. Servo Motor

Los motores se alimentan de forma independiente al resto del sistema con una batería recargable de 9V. La excitación de los mismos puede realizarse con una tensión continua o bien con una señal modulada en anchos de pulsos o PWM (Pulse Width Modulation). Una ventaja añadida a la utilización de servos reconvertidos es la cantidad de accesorios que incluyen, y que permiten unir todo tipo de ruedas, poleas y palancas al eje del motor.

Cuando se utiliza una configuración en triciclo es preciso seguir una regla fundamental: La rueda loca debe soportar sobre sí la menor cantidad de peso posible. Si esta regla no se cumpliera dicha rueda podría quedarse atascada en un giro e incluso producir un salto del robot. Por ello se ha intentado que todo el peso de la electrónica y de las baterías de alimentación, tanto para la lógica, como para los motores sea situado sobre la base de madera en el eje común creado por los dos motores.

Sensores para el control del movimiento del robot

La odometría del robot se realiza con dos encoders en media cuadratura, situado uno en cada rueda. Cada uno de estos encoders se ha realizado haciendo uso de un foto reflector infrarrojo CNY70[16] que apunta a un disco de papel pegado a la rueda, donde se ha dibujado 80 bandas blancas y negras. La resolución del sistema resulta por tanto:

$$Resolucion = \frac{360°}{80} = 4.5°$$

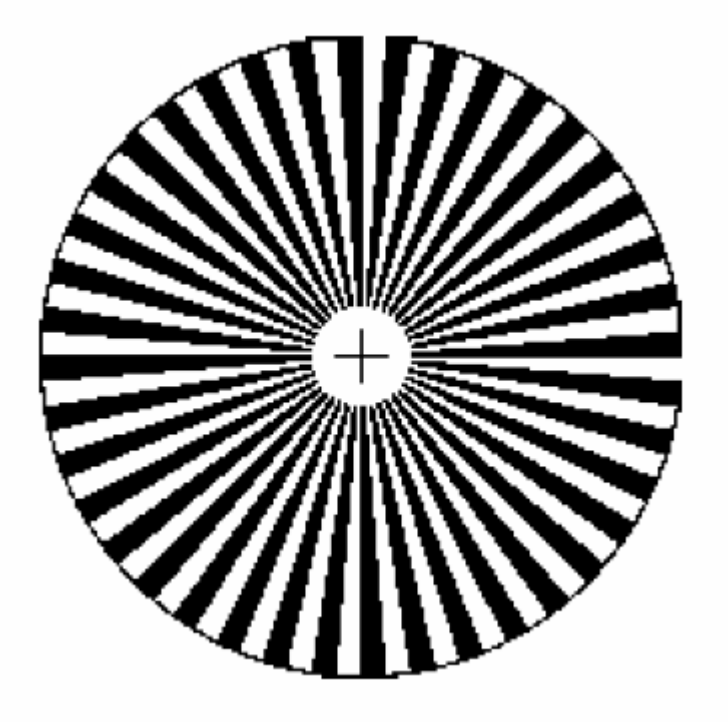


Figura 18. Rueda de Sensado

En un encoder de media cuadratura el giro de la rueda produce una sola señal. Dicha señal permite conocer la dirección de giro de la rueda logrando darnos la distancia que giro cada rueda con una resolución de hasta 4.5 grados [1].

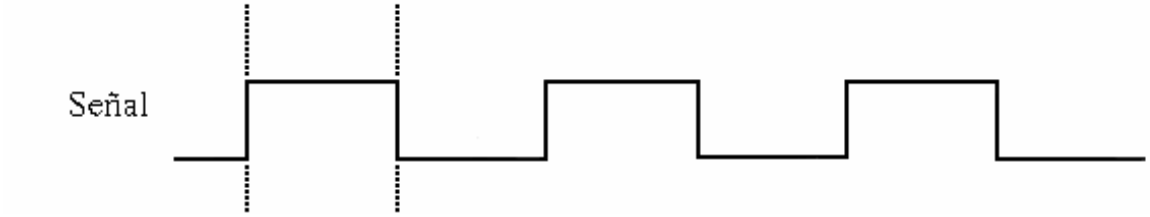


Figura 19. Señal del Sensor

La conexión de los sensores CNY70[16] puede realizarse directamente a la tarjeta del HC11[11], que incorpora los conectores y la electrónica necesaria para la polarización y acondicionamiento de los fotoreflectores. Así, una simple lectura del puerto A del microcontrolador nos permite detectar si los sensores están sobre una banda blanca o negra, al leer un '0' o un '1' lógico, respectivamente. Más aún, el microcontrolador puede ser interrumpido cada vez que el sensor detecte un paso de una banda blanca a negra, o viceversa. Con este método el sistema puede estar actualizando continuamente, las medidas relativas a la posición y velocidad del robot, de forma independiente al programa principal. El elemento microprocesador para el control del sistema está formado por una tarjeta de desarrollo, dicha tarjeta está basada en el microcontrolador MC68HC11A8 de Motorola.

Tableta de etapa de potencia para los motores

El sistema cuenta además con una tarjeta que contiene la etapa de potencia que es capaz de excitar dos motores de corriente continua con el puente en H integrado L293D[15]. Esta tarjeta cuenta además con el control de velocidad para poder ajustar el movimiento del robot por medio de un potenciómetro de precisión, el cual al variar el valor de la resistencia, cambia el valor del voltaje de alimentación del motor [5].

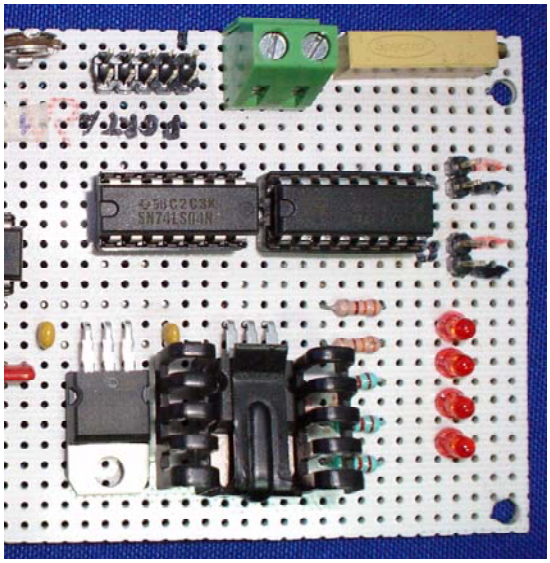


Figura 20. Tableta de la Etapa de Potencia de los Motores

Sensores para el posicionamiento del robot

Un robot destinado a resolver laberintos debe considerar dos problemas fundamentales, el primero es la detección de las líneas y las intersecciones de las mismas dentro del laberinto y el segundo su posición relativa o absoluta dentro del laberinto.

La detección de líneas se realiza a través de 7 sensores, estos dispositivos son capaces de entregar cada uno un voltaje entre 5V y 0.6V cuando detectan el color negro o blanco dentro del laberinto.

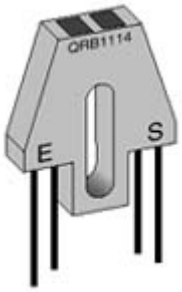


Figura 21.Sensor Infrarrojo

La posición de los sensores es muy importante para la correcta detección de las líneas y de las intersecciones del laberinto. Nosotros diseñamos una pequeña tarjeta para posicionar correctamente los sensores en arreglo de 3 al centro y dos en cada lado como se muestra en la fotografía para permitir una correcta detección tanto de las líneas como de las intersecciones del laberinto. Para poder tomar las decisiones correctas esta tarjeta está colocada en la parte inferior del robot, sobre el eje que forman los motores. Esto permite que al girar el robot y hacer correcciones al seguir el laberinto los sensores siempre están viendo una parte del laberinto.

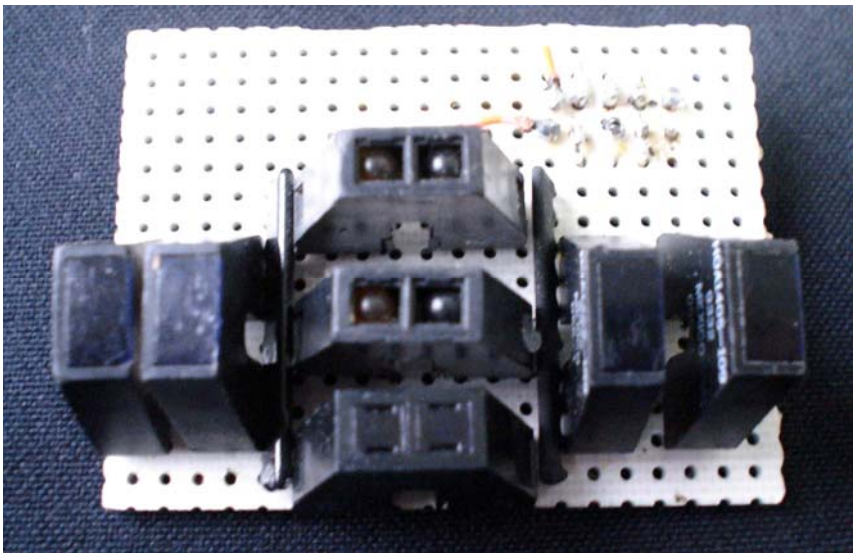


Figura 22. Arreglo de Sensores

Tableta de Captura de Datos de los Sensores y la tableta de Calibración de los Sensores

Estos sensores requieren de una tarjeta que incorpore toda la electrónica necesaria para que el usuario sólo tenga que conectarlos a su dispositivo de detección y comenzar a tomar muestras. Además existe la posibilidad de calibrar cada sensor independientemente para ajustar el nivel de voltaje cuando capta blanco o negro, en algunos milivolts. Los sensores se han conectado al puerto B del microcontrolador, ya que éste es uno del los puertos de entrada **[11]**. Esta tarjeta contiene dos comparadores y unos potenciómetros para ajustar el rango de voltajes que mandan al microprocesador para que cuando el sensor capte el color negro regrese 0 y cuando detecte blanco regrese 1. Los comparadores nos sirven para poder ajustar los valores de los sensores a unos y ceros lógicos. Con el fin de tener una señal digital hacia el microprocesador estos comparadores reciben un voltaje de referencia en sus entradas entre 0V y 5V. El voltaje de referencia en la entrada es el valor que controlamos con los potenciómetros dando así el rango que queremos para cada sensor ya que debido a las condiciones de luz el valor que nos da el sensor, puede cambiar aunque sea el mismo material **[16]**.

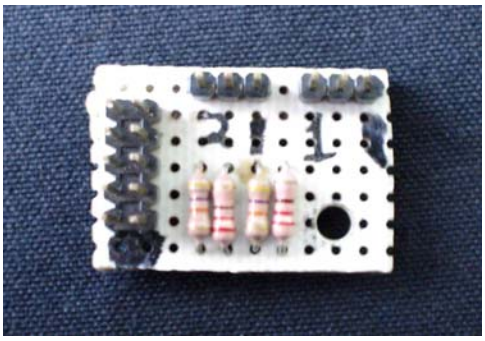


Figura 23. Tarjeta para los Encoders

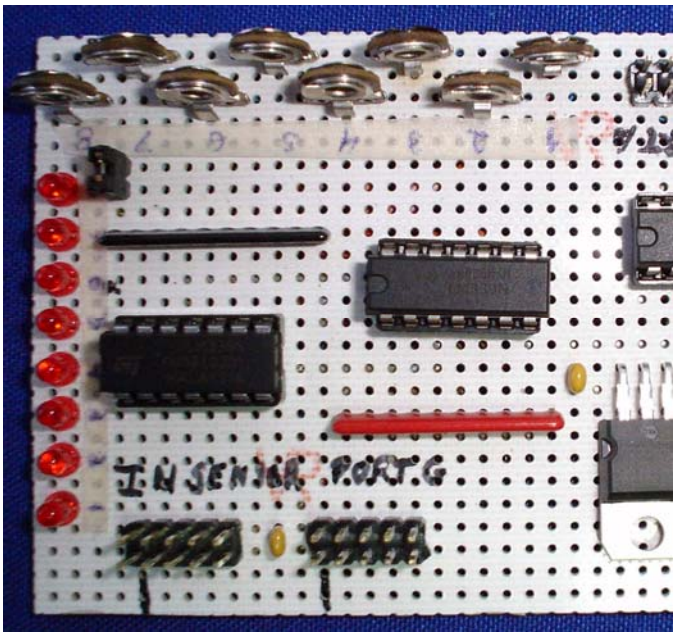


Figura 24. Tarjeta de Captura y Calibración de Sensores

Tarjeta de Microcontrolador 68HC11 y Microprocesador Motorola 68HC11

La tarjeta del micro procesador HC11[11] es una tarjeta de desarrollo que contiene 8 puertos de entrada y salida con la posibilidad de usar 2 memorias externas de 32k cada una, de las cuales, una debe ser RAM y la otra puede ser RAM o EEPROM. También cuenta con la capacidad de conectarse a un puerto serial. Esta tarjeta se puede configurar de 4 maneras como se muestra en la siguiente tabla:

1	2	3	4	Modo de Operación
ON	ON	ON	ON	BOOTSTRAP
ON	ON	OFF	OFF	EXPANDED
ON	ON	ON	OFF	SINGLE CHIP
ON	ON	OFF	ON	SPECIAL TEST

La configuración para ejecutar el sistema operativo es en el modo “Expanded” y para programarlo es en modo Bootstrap. Debido a que usamos memoria expandida en una

EEPROM no se puede programar en modo Bootstrap por el tipo de memoria. El modo single chip espera ejecutar programas que están dentro de la memoria interna del microcontrolador y el modo de special test es para hacer pruebas del microcontrolador
Si los dip switch 1-2 están en off la tarjeta no encenderá. E

Firgura 25. Tarjeta del Microcontrolador HC11

El microcontrolador MC68HC11F1[11] funciona con una señal de reloj de sistema interno de 2MHz, y dispone únicamente de 256 bytes de RAM interna y 512 bytes de memoria EEPROM interna, lo que resulta insuficiente para el programa a cargar en el robot, por ello se utiliza la expansión de memoria que permite utilizar 32Kbytes de RAM y 32Kbytes de memoria RAM no volátil.

Este microcontrolador tiene una arquitectura basada en la arquitectura Von Newman esto quiere decir que tiene un solo bus para acceder tanto a datos como a instrucciones. Un microcontrolador es de 4 bit cuando el bus de éste sea de 4 bit, será de 8 bit cuando el bus sea de 8 bit. Una de las principales ventajas que presenta esta arquitectura es que el acceso a datos e instrucciones se hace a través de un mismo bus lo que facilita en gran medida la conexión de memoria externa a través de las líneas de entrada y salida con una mínima implementación extra de hardware.

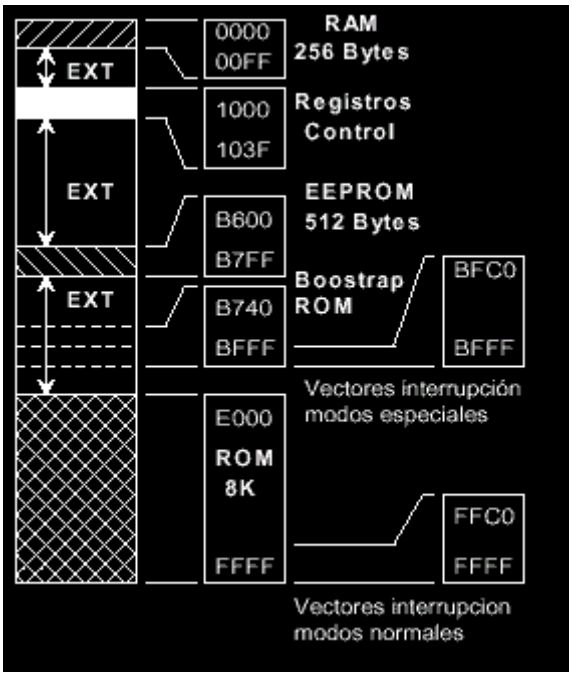


Figura 26. Estructura de la Memoria en el Microcontrolador

Por el contrario, tenemos que una instrucción puede ocupar más de un byte con lo que para poder leer la instrucción completa tendremos que hacer varias lecturas en la memoria, con lo que debemos de emplear varios ciclos de reloj para extraer una instrucción.

Ejemplo de un HC11

	CODOP	DATA
ADDA	0x09	10001011 00001001

Otra desventaja es que es posible que el contenido del contador del programa (PC) se corrompa con lo que se podría estar leyendo un dato y tratar de interpretarlo como instrucción pudiendo producirse como consecuencia un deterioro y caída del sistema. Normalmente un microprocesador debe de controlar que el PC no haga cosas raras.

Una de las características que poseen los microcontroladores basados en este tipo de arquitectura es que suelen tener un repertorio de instrucciones bastante grande. Este tipo de repertorio se llama CISC del inglés Complex Instruction Set Computer. La característica principal que este tipo de conjunto de instrucciones es que suele ser bastante elevado y las instrucciones están microcodificadas, es decir, que una instrucción será decodificada por la CPU en varias instrucciones básicas. Es fácil deducir que esto hará la ejecución del programa un poco más lenta ya que se van a ejecutar varias instrucciones pero tiene la ventaja de que ahorramos memoria ya que para dividir sólo usamos una instrucción, pero aún así tengamos en cuenta que la decodificación se está llevando a cabo en el interior del microcontrolador lo que va a alentar su ejecución y por

ello deberemos de tener en cuenta qué tipo de microcontrolador usar dependiendo de la tarea a desarrollar.

Microcontrolador	Arquitectura	Conjunto de instrucciones	nº de instrucciones
Pic 16Cxxx	Harvard	RISC	35
Pic 17Cxxx	Harvard	RISC	58
Motorola HC11	Von Newman	CISC	109
Intel 8051	Von Newman	CISC	40

Así pues podemos decir que la principal ventaja de usar microcontroladores con conjunto de instrucciones CISC es que para una instrucción compleja sólo usaremos una posición de memoria. Al contrario que ocurre con RISC que para realizar, por ejemplo, una división debemos de usar varias instrucciones consumiendo más memoria.

Frente a esta ventaja de los repertorios CISC, se nos presenta una desventaja con respecto a los RISC, y es que el ancho de banda se va reducido considerablemente debido a que una instrucción va a consumir varios ciclos de instrucción para ejecutarse, estos microcontroladores son más lentos que los que usan repertorios RISC y además puede ser que el conjunto de instrucciones sea bastante grande, lo que no es una gran desventaja pero si más trabajo para aprender a usarlo.

Computadora de Bolsillo Compaq Ipaq 4150

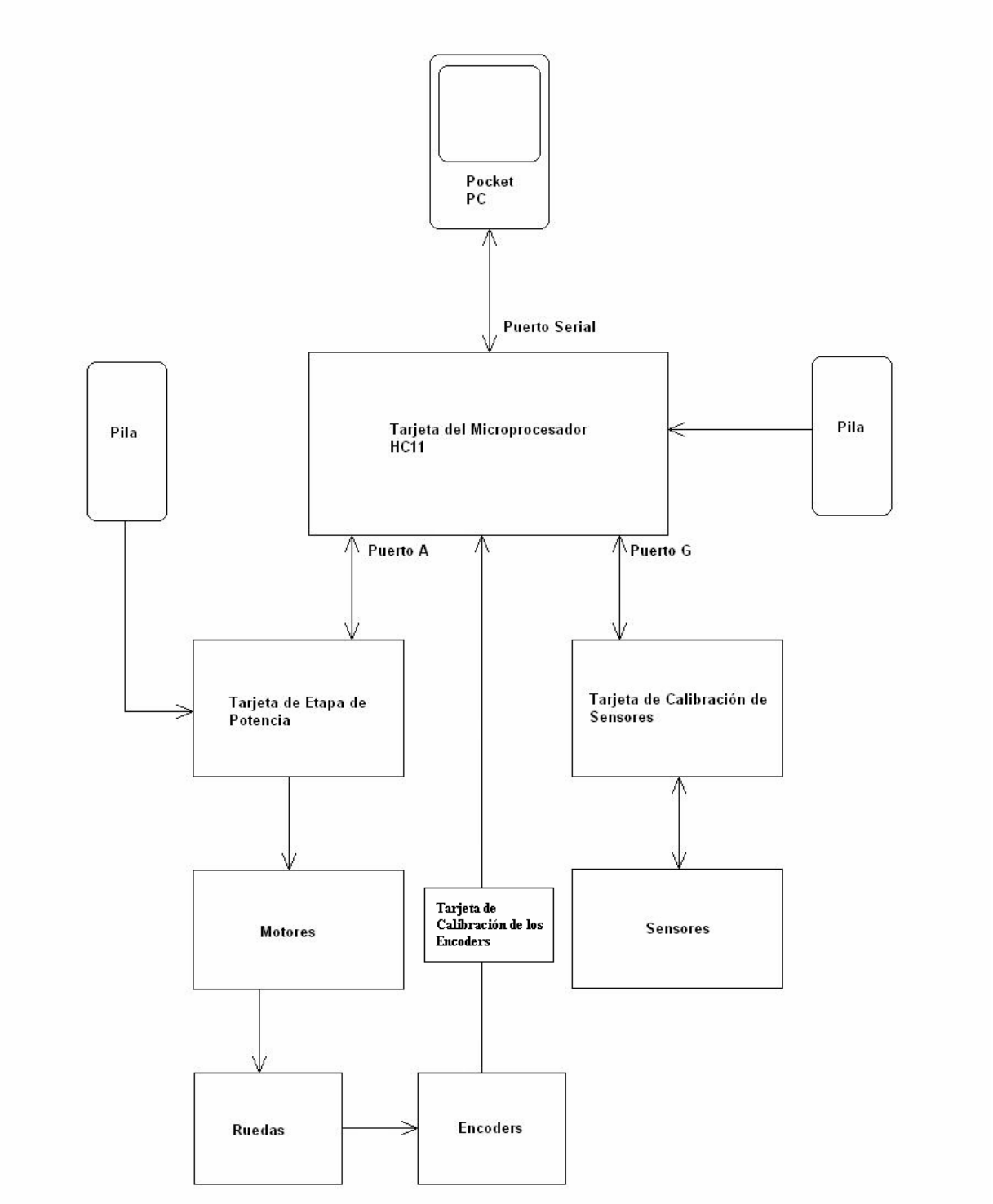
La computadora de bolsillo Ipaq 4150 es utilizada para la parte de inteligencia del robot, ésta se conecta por medio del puerto serial de esta computadora a la tarjeta de desarrollo del microcontrolador HC11

Figura 27. Ipaq HP 4150

Esta computadora cuenta con las siguientes características

Marca:	Hewlett Packard
Modelo:	Ipaq h4150 h4155
Sistema Operativo:	Windows Mobile 2003
Dimensiones:	4.47 in x 2.78 in x 0.5 in::11.35cm x 7.06 cm x 1.27cm
Peso:	4.67 oz. 132.955
Procesador:	400MHz Intel XScale
Wireless:	WLAN 801.11b, Bluetooth y IrDA integrados
Memoria:	64MB SDRAM (55MB accesibles por el usuario), 32MB Flash ROM
Slot de Expansión:	Secure Digital (I/O)
Tipo de Batería:	Removible recargable Ion-Litio (1000 mAh)
Audio out:	Bocina, Stereo jack
Audio In:	Micrófono
Display:	Pantalla Transflectiva TFT con 64,000 colores
Resolución:	240 x 320 píxeles
Área Visible:	3.5" diagonal
Teclado:	En pantalla
Reconocimiento de Escritura:	Si
Digital Camera:	No
Otro Hardware:	Base/cargador de escritorio USB, adaptador AC, batería, cubierta, stylus, adaptador para carga

Diagrama de Flujo del Robot:



Capítulo 6 Sistema de control para robot RobOS

El sistema operativo RobOS fue desarrollado en C para el Microcontrolador HC11 y fue desarrollado de tal forma que se pueda migrar fácilmente a otro microcontrolador cambiando sólo las direcciones de los puertos, de las interrupciones y de configuración correspondientes a cada micro [5][8].

El sistema operativo RobOS tiene como fin controlar robots que posean la tarjeta del microcontrolador de Motorola HC11 por medio de comandos sencillos mandados a través del puerto serie de la computadora. Al ejecutar el programa en modo Expandido muestra una lista de comandos posibles con algunos ejemplos en la terminal dentro de la computadora, al final del cual se pueden escribir los comandos deseados en el prompt. La sintaxis para la ejecución de estos comandos necesita incluir un identificador de comando, el cual permite saber que el comando a ejecutar no es repetido ya que si el robot es controlado vía Internet es posible saber que número de comando está ejecutando el robot. El símbolo "@" es para separar el comando del identificador. Entre los comandos que el robot puede ejecutar se encuentran las siguientes funciones junto con su sintaxis.

Comando	Descripción	Ejemplo
Forward	El robot se mueve hacia adelante	forward @12
Backward	El robot se mueve hacia atrás	backward @32
Left	El robot gira hacia la izquierda	left @2
Right	El robot gira hacia la derecha	Right @1
Mv	El robot gira un ángulo en radianes y después avanza una distancia en decímetros	mv 10.0 3.4 1 @12
Mvto	El robot se mueve a la posición (x , y)	mvto 20 40 1@13
Ic	Se establece la posición y orientación iniciales del robot	ic 38 37 0 @14
Sc	Muestra la posición y orientación actuales del robot	sc @56
Shs	Lee un sensor utilizando su nombre y su número de identificación	shs reflective 1 @7
user	Ejecuta una subrutina de usuario	user 5 @78

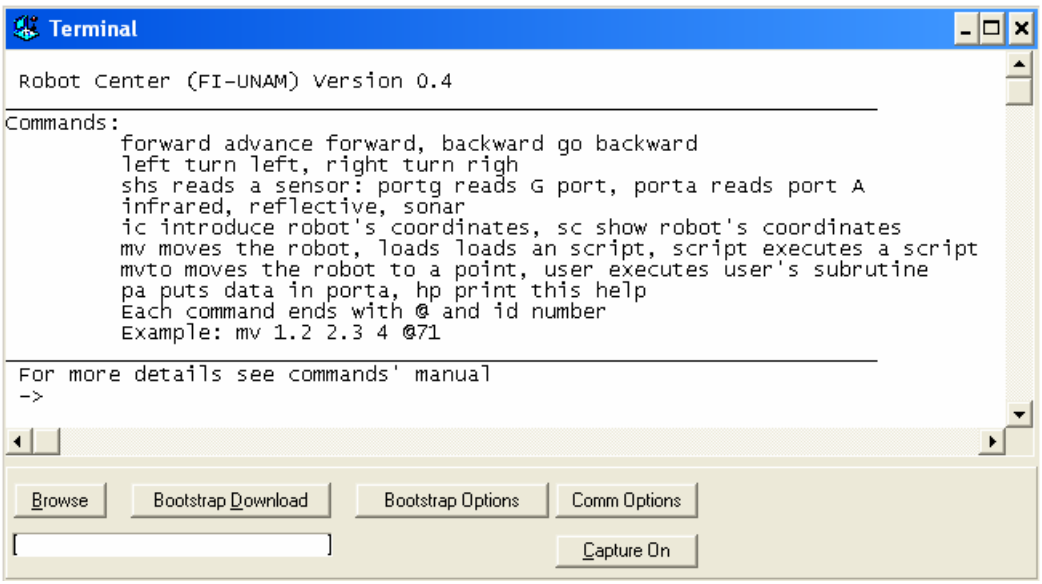


Figura 28. Pantalla del prompt del Sistema Operativo

a) Forward, Backward, Left y Right – estos comandos instruyen al robot a que se mueva hacia adelante, atrás, izquierda o derecha respectivamente una distancia predefinida en el sistema; es decir, que al ejecutar adelante y atrás se moverá 10 cm. Y cuando se ejecute derecha e izquierda se moverá 90º a la derecha o a la izquierda.

forward @212
backward @12
left @11
right @11

```
 -> forward @11
Accumulator:
s1:
s2:
d1:0
d2:0
d3:0
id :11
code :10
 waiting 0

 waiting 40

 waiting avance distance=1.0 newdistance=1.0

forward @11
 ->
```

Comando Forward

Sintaxis: forward @ número_de_control

Descripción: El robot se mueve hacia delante una corta distancia (10 centímetros). Este comando se puede ejecutar desde la ventana de comandos, Internet o de un script.

El número_de_control es un número entero que se utiliza para identificar comandos repetidos sucesivos.

Comando Backward

Sintaxis: backward @número_de_control
Descripción: El robot se mueve hacia atrás una corta distancia (10 centímetros). Este comando se puede ejecutar desde la ventana de comandos, Internet o de un script.

Comando Left

Sintaxis: left @número_de_control
Descripción: El robot gira 90 grados hacia la izquierda. Este comando se puede ejecutar desde la ventana de comandos, Internet o de un script.

Comando Right

Sintaxis: right @número_de_control
Descripción: El robot gira 90 grados hacia la derecha. Este comando se puede ejecutar desde la ventana de comandos, Internet o de un script.

b) Mv – este comando instruye al robot a moverse una distancia "l" y un ángulo "a" en un tiempo "t", la primera instrucción que realiza es el giro, ya sea en sentido horario o antihorario, después ejecuta la acción de avanzar.

mv l a t @12

Comando Mv

Sintaxis: mv distancia ángulo tiempo @número_de_control
Descripción: El robot gira un ángulo en radianes y después avanza una distancia en centímetros. Este comando se puede ejecutar desde la ventana de comandos, Internet o de un script.

c) Mvto – este comando instruye al robot a moverse a una nueva posición determinada dada por un par de coordenadas "x" "y" en un tiempo "t", toma como referencia el punto inicial de donde partió por primera vez y se mueve a la nueva posición basándose en ese punto de referencia.

mvto x y t @11

Comando Mvto

Sintaxis: mvto posición X posición Y tiempo @número_de_control

Descripción: El robot se mueve a la posición (x , y). Este comando se puede ejecutar desde la ventana de comandos, Internet o de un script.

La posición X y la posición Y representan el punto final a donde el robot se moverá.

d) Sc – muestra la posición del robot en coordenadas “x” “y” en ese instante.

sc @14

Comando Sc

Sintaxis: sc @número_de_control
Descripción: Muestra la posición y orientación actuales del robot. Este comando se puede ejecutar desde la ventana de comandos, Internet o de un script.

e) Ic – introduce la posición del robot en coordenadas “x” “y”.

ic @43

Comando Ic

Sintaxis: ic posición X posición Y ángulo @número_de_control
Descripción: Se establece la posición y orientación iniciales del robot. Este comando se puede ejecutar desde la ventana de comandos, Internet o de un script.

La posición X y la posición Y son la ubicación inicial del robot. El ángulo es la orientación inicial del robot.

f) Shs – muestra el valor que esta leyendo de alguno de los sensores definidos en este caso, se ha estado trabajando principalmente con sensores de reflexión. Pero también se puede leer el valor de sensores infrarrojos y sonares. Dependiendo del pin del puerto g a que esté conectado cada sensor, el valor que muestra está dado en hexadecimal. También se le puede indicar que regrese el valor total del puerto de los sensores.

shs sensor numsensor @13

sensor = sonar, reflective, infrared, portg, porta.
numsensor = número de sensor.

Comando Shs

Sintaxis: shs nombre_de_sensor número_de_sensor @número_de_control

Descripción: Lee un sensor determinado por su nombre y su número de identificación. Los sensores permitidos en este comando son de contacto, reflectivos, sonares e infrarrojos. El número_de_sensor es el parámetro de identificación del robot, el cual depende del número de sensores en el robot.

g) User – se le puede especificar que ejecute una o varias funciones de usuario previamente creadas y que se encuentren dentro de la memoria RAM o EPROM del robot. Estas funciones de usuario pueden ser tan complejas como uno lo desee. En nuestro caso, se ha estado trabajando en una función de usuario que permita al robot resolver un laberinto ya sea por regla de la mano izquierda o por regla de la mano derecha o que siga una línea. El número que se muestra en la sintaxis le indica al sistema que función de usuario se va a ejecutar.

user 1 @12

Comando User

Sintaxis: user 5 @número_de_control

Descripción: Permite la ejecución de una subrutina de usuario. La pantalla de estado indica si los comandos se ejecutaron de forma correcta o no.

Nota: En los casos que se menciona es necesario especificar el tiempo. Sin embargo, la parte de control del tiempo no ha sido implementada.

Pero nada de esto sería posible sin las subrutinas que permiten controlar al robot a través de la tarjeta con un microprocesador HC11. Estas subrutinas son creadas en lenguaje C y compiladas a través del compilador de lenguaje “C” para esta familia de microprocesadores conocido como ICC11.

En primer lugar se tienen subrutinas que permiten controlar las condiciones iniciales de la tarjeta del HC11, como lo son los puertos (de entrada o de salida, y sus correspondientes registros en memoria), la velocidad del puerto serie para la comunicación con la computadora, la ubicación de los vectores de interrupción, y los atributos del robot (nombre, número, etc.). Al igual de cómo calcular las distancias y ángulos que se va a mover ya sea porque tenga que encoger en las ruedas o por tiempo.

En segundo término se tienen las subrutinas de inicio. Como la de los encoders de los motores, la de los sensores (infrarrojos, reflectivos o sonares), la de posicionamiento inicial del robot, el convertidor analógico-digital de la tarjeta, etc.

Dentro de estas subrutinas también se definen las distancias y los ángulos que debe ejecutar el robot al ejecutar cualquiera de los comandos mostrados anteriormente.

También se tienen las subrutinas para la lectura y escritura en los puertos A y G, que son donde se tienen las salidas hacia los motores y las entradas de los sensores respectivamente.

En tercer lugar se tiene la parte donde el microprocesador hace los cálculos y conversiones pertinentes para poder interpretar los datos que le están llegando y mandarlos a otras subrutinas donde son empleados para determinar las acciones correspondientes del robot. Esto se hace mediante un "parser" el cual le la cadena enviada la decodifica es decir la secciona y manda de regreso la subrutina a ejecutar con sus atributos [8].

Finalmente se tienen otras subrutinas auxiliares, que son las encargadas de interpretar los comandos que se introducen a través de la interfaz en la computadora y que van a dar al procesador a través del puerto serie. Esto es importante ya que el código empleado tiene que ser el mismo para que se pueda llevar a cabo las acciones correspondientes. Es muy fácil para nosotros asociar los comandos de adelante, atrás, izquierda o derecha, pero para un robot, se tienen que transformar esos comandos en datos para compararlos con situaciones previstas por el programador. Esto es realizado por el "parser" que en estos casos, primero lee la cadena y estandariza los comandos, lo cual implica tomar los datos introducidos desde el teclado y traducirlos a lenguaje de máquina, donde la minúscula de una letra tiene un valor ASCII diferente a la de su contraparte mayúscula, en nuestro caso, se tiene una subrutina auxiliar que transforma todos los caracteres a minúsculas, otra que elimina los espacios entre caracteres, y otra que compara estos caracteres o mejor dicho los valores correspondientes de los caracteres con una lista de posibles entradas. Si no se encuentra la correspondiente asociación, se regresa un mensaje al usuario diciéndole que su comando no es válido.

Entre otras subrutinas, se tiene una que analiza lo que hay en el buffer, otra que le da validez al comando y manda llamar las subrutinas de acción del robot. Además se tiene una subrutina que se encargada de desplegar en pantalla la lista de comandos con la que cuenta el usuario.

Subrutinas de Usuario

El usuario puede necesitar llamar subrutinas escritas en lenguaje C o C++ que fueron creadas para el centro de comandos del robot (Roc2). Estas subrutinas se escriben dentro de la librería dinámica Roc2User.dll. Para poder agregar o modificar una nueva subrutina de usuario, es necesario recompilar el proyecto Roc2User. El centro de comandos del robot incluye este proyecto para poder recompilar dicha librería. Aquí se muestra una pequeña porción del código de la librería:

```
void user2(void){
        Move_robot("Ryu", 0, 3, 0);
        Printf("\nResult: %.3f!", show_sensor("Ryu", "sonar", 1));
}
```

```
void user6(char *string){
	//load and play WAVs files
	PlaySound("sounds/r2d2_1.wav, NULL, SND_ASYNC);
}

// Call an user subroutine
void WINAPI user_subroutine(int Command, char *string){
	switch(Command){
		case 1:// User subroutine1
			user1();
			break;
		case 2:// User subroutine2
			user2();
			break;
		case 3:// User subroutine3
			user3();
			break;
		case 4:// User subroutine4
			user4();
			break;
		case 5:// User subroutine5
			user5();
			break;
		case 6:// User subroutine6
			user6();
			break;
		default: printf("\nSubroutine %d not defined", Command);
	}
}
```

El método user_subroutine es una función exportable. El RobOS llama a esta función cuando el usuario utiliza el comando user 2 @12. Por ejemplo, cuando el usuario escriba el comando "user 2", el RobOS mandará llamar la función "user_subroutine" y ejecutará la subrutina local "user2". De esta manera el usuario puede modificar una subrutina local ya existente o agregar una nueva subrutina teniendo que recompilar el sistema robos para agregar las nuevas subrutinas.

Algunos de los comandos se pueden agregar a las subrutinas de usuario. La siguiente lista muestra algunos de estos comandos y su sintaxis.

Comando: show_coordinates
Sintaxis: show_coordinates(char *nameRobot, float *data);
Descripción: Regresa la posición y orientación actual del robot

Comando: move_robot

Sintaxis: move_robot(char *nameRobot, float distance, float angle, int time);
Descripción: El robot gira un ángulo en radianes y después avanza una distancia en decímetros.

Comando: show_sensor
Sintaxis: show_sensor(char *nameRobot, char *sensorType, int numSensor);
Descripción: Lee un sensor utilizando su nombre y su número de identificación

Capítulo 7 Sistema para la Solución del Laberinto (Cerebro)

El sistema para la solución del laberinto fue programado en Visual C# (C Sharp) para poderlo ejecutar en la pocket PC (Compaq Ipaq 3950) este sistema es el que controla el robot propiamente ya que se comunica con el sistema operativo RobOS por medio de un cable serial mandándole comandos básicos para el movimiento y lectura de los sensores.

Para la resolución de laberintos en competencias se tienen 3 intentos de 3 minutos cada uno, esto nos permite realizar varias acciones que a continuación se desarrollarán.

En el primer intento el programa se encarga de recorrer el laberinto usando el algoritmo de la mano derecha, guardando cada intersección como un nodo en un arreglo o formado por el nodo inicial, nodo final, dirección del robot y distancia ente los nodos.

En el segundo intento el robot busca la ruta mínima entre el nodo inicial y el nodo en que se quedó, utilizando el algoritmo Dijkstra; de esta manera se sabe por cuales nodos pasar y que decisión tomar para poder continuar resolviendo el laberinto, utilizando el método de la mano derecha, se agregan los nuevos nodos al arreglo para poder volver a buscar la ruta mínima para el siguiente intento. En el caso que encuentre la salida del laberinto se detiene el robot y guarda la posición de esta salida como el nodo final; así, en el tercer intento se calcula la ruta mínima y se resuelve el laberinto en el menor tiempo posible, recorriendo los nodos necesarios y eliminando todos los que no sirvieron.

En el tercer intento si no se ha llegado a la solución volverá a suceder lo mismo que en el segundo hasta que encuentre la salida. Utilizando el algoritmo de Dijkstra. Para llegar al nodo en que se quedó y continuar resolviendo el laberinto por el método de la mano derecha. Si ya se había llegado a la solución, lo que hace es calcular la ruta mínima con el algoritmo Dijkstra y recorrer la ruta calculada.

El algoritmo de Dijkstra. Fue desarrollado por **Edsger Wybe Dijkstra** que nació en Rotterdam (Holanda), en 1930. Sus padres eran ambos intelectuales y él recibió una excelente educación. Su padre era químico y su madre matemática. En 1942, cuando Dijkstra tenía 12 años, entró en el Gymnasium Erasminium, una escuela para estudiantes especialmente brillantes, donde dio clases, fundamentalmente, de Griego, Latín, Francés, Alemán, Inglés, biología, matemáticas y química. En 1945, Dijkstra pensó estudiar Derecho y trabajar como representante de Holanda en las Naciones Unidas.

Sin embargo, debido a su facilidad para la química, las matemáticas y la física, entró en la Universidad de Leiden, donde decidió estudiar física teórica. Durante el verano de 1951, asistió a un curso de sobre programación en la Universidad de Cambridge. A su vuelta empezó a trabajar en el Centro Matemático en Amsterdam, en marzo de 1952, donde incrementó su creciente interés en la programación. Cuando terminó la carrera se dedicó a problemas relacionados con la programación. Pero uno de

los problemas con que se encontró es que ser programador no estaba oficialmente reconocido como una profesión. De hecho, cuando solicitó una licencia de matrimonio en 1957, tuvo que señalar que su profesión era físico teórico.

Dijkstra continuó trabajando en el Centro Matemático hasta que aceptó un trabajo como desarrollador en Burroughs Corporation, en los Estados Unidos, a principio de la década de los 70. En 1972 ganó el Premio Turing ACM, y, en 1974, el AFIPS Harry Good Memorial. Dijkstra se trasladó a Austin, Texas a principio de los 80. En 1984, se le ofreció un puesto en Ciencias de la Computación en la Universidad de Texas, donde ha permanecido desde entonces. Es miembro honorario de la Academia Americana de Artes y Ciencias y de Real Academia Holandesa de Artes y Ciencias. Además es miembro distinguido de la Sociedad de Computación Británica. Finalmente es Doctor Honoris Causa en Ciencias por la Queen's University Belfast.

En 1956, Dijkstra anunció su algoritmo, un conjunto de reglas que permiten obtener un resultado determinado a partir de ciertas reglas definidas. Otra posible definición sería, una secuencia finita de instrucciones, cada una de las cuales tiene un significado preciso y puede ejecutarse con una cantidad finita de esfuerzo en un tiempo finito. Este conjunto ha de tener las siguientes características: legible, correcto, modular, eficiente, estructurado, no ambiguo y de ser posible se ha de desarrollar en el menor tiempo posible) **algoritmo de caminos mínimos o rutas mínimas**. Después de haber estado trabajando con el ARMAC, el ordenador que el Centro Matemático poseía, se propuso a elaborar el algoritmo del árbol generador mínimo. A principios de la década de los 60, Dijkstra aplicó la idea de la exclusión mutua a las comunicaciones entre una computadora y su teclado. Su solución de exclusión mutua ha sido usada por muchos procesadores modernos y tarjetas de memoria desde 1964, cuando IBM la utilizó por primera vez en la arquitectura del IBM 360. También ayudó a fomentar la disciplina en la programación: "GOTO se puede considerar dañino. Cuanto más instrucciones GOTO haya en un programa, más difícil es entender el código fuente".

El problema de la ruta más corta se puede resolver utilizando programación lineal; sin embargo, debido a que el método simplex es de complejidad exponencial, se prefiere utilizar algoritmos que aprovechen la estructura en red que se tiene para estos problemas.

Una red de comunicaciones involucra un conjunto de nodos conectados mediante arcos, que transfiere vehículos desde determinados nodos origen a otros nodos destino. La forma más común para seleccionar la trayectoria (o ruta) de dichos vehículos, se basa en la formulación de la ruta más corta. En particular a cada arco se le asigna un escalar positivo el cual se puede ver como su longitud.

Un algoritmo de trayectoria más corta, rutea cada vehículo a lo largo de la trayectoria de longitud mínima (ruta más corta) entre los nodos origen y destino. Hay varias formas posibles de seleccionar la longitud de los enlaces. La forma más simple es que cada enlace tenga una longitud unitaria, en cuyo caso, la trayectoria más corta es simplemente una trayectoria con el menor número de enlaces. De una manera más general, la longitud de un enlace puede depender de su capacidad de transmisión y su

carga de tráfico.

La solución es encontrar la trayectoria más corta, esperando que dicha trayectoria contenga pocos enlaces no congestionados. De esta forma los enlaces menos congestionados son candidatos a pertenecer a la ruta. Hay algoritmos de ruteo especializados que también pueden permitir que la longitud de cada enlace cambie en el tiempo, dependiendo del nivel de tráfico de cada enlace. De esta forma un algoritmo de ruteo se debe adaptar a sobrecargas temporales y rutear paquetes alrededor de nodos congestionados. Dentro de este contexto, el algoritmo de ruta más corta para ruteo opera continuamente, determinando la trayectoria más corta con longitudes que varían en el tiempo.

El algoritmo de Dijkstra para ruta más corta, en términos generales, es encontrar la ruta más corta entre dos nodos, inicial *a* y final *z*, de la siguiente manera:

Los nodos de la red son etiquetados con números. Al principio, todos tienen la etiqueta 00 excepto el nodo inicial a que tiene la etiqueta 0. Los arcos tienen un peso Wij que representa la distancia del enlace (i, j). El algoritmo de Dijkstra renumera los nodos, de manera que cuando el nodo *z* tiene una etiqueta permanente, se ha obtenido la solución final.

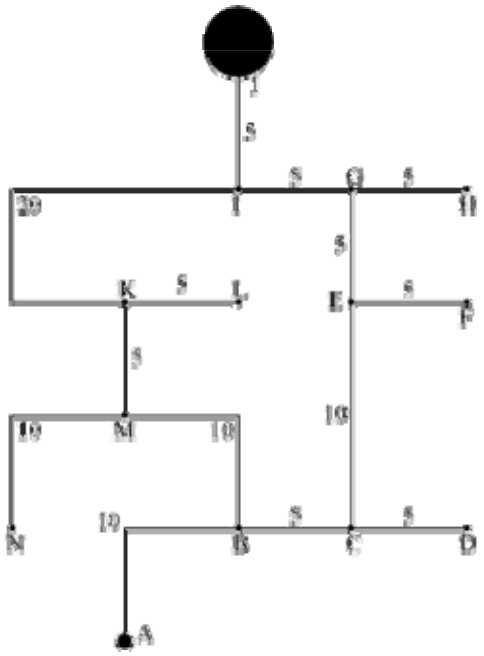
Una vez calculada la ruta mínima la “Ipaq” se comunica por medio del puerto serial usando los siguientes parámetros de conexión:

BaudRate = 9600
ByteSize = 8
Parity = 0
StopBits = 1

Todos estos parámetros se encuentran programados en una librería hecha en C# desarrollada en el Laboratorio de Biorobótica. Esta librería permite controlar la comunicación variando los diferente parámetros de la comunicación serial. En este caso se usaron estos parámetros debido a las limitaciones del la tarjeta de desarrollo del HC11 ya que la velocidad máxima de la comunicación debido al cristal con que contamos es 9600bps.

Ejemplo para el cálculo de la ruta mínima en un laberinto

Teniendo el siguiente laberinto se puede calcular cual es la ruta mínima para salir de el es decir recorrer el laberinto de A-J por la ruta mas corta



La tabla de nodos quedaría así

Nodo Inicio	Nodo Destino	Costo
A	B	10
B	C	5
B	M	10
C	D	5
C	E	5
E	F	5
E	G	5
G	H	5
G	I	5
I	J	5
M	N	10
M	K	5
K	L	5
K	I	20

Usando el algoritmo de dijktra la solución por pasos seria la siguiente

Paso	Ruta	Costo
1	AB	10
2	ABC	15
3	ABM	20
4	ABCD	20
5	ABCE	25
6	ABMN	30
7	ABMK	25
8	ABCEF	30
9	ABCEG	30
10	ABMKL	30
11	ABMKI	45
12	ABCEGH	35
13	ABCEGI	35
14	ABMKIJ	50
15	ABCEGIJ	40

Por lo que la ruta mínima es ABCEGIJ

Código Cerebro

El código se puede consultar en su totalidad en el Apéndice E (pág. 88)

Clase CerebroRev

En este apartado se declaran los posibles valores que se pueden registrar de los sensores para los distintos tipos de movimiento del robot, así como cada una de sus respectivas acciones a realizar. Se hace una comparación entre el valor registrado y cada uno de los posibles valores de acuerdo con cada tipo de movimiento del robot (avanzar hacia adelante o atrás, girar a la izquierda o derecha, detenerse, dar media vuelta, etc.). Los valores de los sensores comprenden el rango entre 128 y 255.

Clase Compass

En esta clase se determina la orientación del robot con respecto a su posición inicial. Es empleado para la construcción de un mapa virtual que se pueda seguir cuando ya se ha recorrido el laberinto en alguna ocasión previa para así evitar tomar un camino errado nuevamente.

Clase Map

Esta clase es la que va construyendo el mapa de acuerdo a los nodos visitados por el robot en el laberinto y tomando en cuenta la orientación que ha tenido el robot desde su inicio hasta cada uno de los nodos. Compara la ubicación actual del robot con respecto a su posición anterior y determina si esta posición corresponde a un nodo previamente visitado o si se trata de un nuevo nodo. Si es un nuevo nodo, lo almacena en el mapa que se construye conforme se va recorriendo el laberinto.

Clase Robot

Esta clase es la responsable del correcto desempeño del robot. Establece un compás para guiarse a través del laberinto y va determinando que tipo de movimientos son los necesarios. Obtiene lecturas de los sensores con lo que obtiene datos para decidir si se requiere algun tipo de movimiento: adelante, atrás, giro; dependiendo de cada caso. También establece la comunicación entre el robot y la pocket pc para el envío y recepción de datos entre estos dos dispositivos así como la manera en que se obtienen los datos de los sensores.

Clase Sensors

Esta clase determina los valores de los sensores además de analizar que sensores están funcionando.

Capítulo 8 Pruebas realizadas al Robot

Entre las diversas pruebas que se deben realizar al robot, se encuentran las que verifican el correcto funcionamiento del mismo. Esto es, nos permiten analizar cada uno de los componentes que conforman al mismo y conocer el estado actual de los mismos para que después podamos continuar con las pruebas en cuanto al desempeño del robot dentro del laberinto.

Para poder realizar varias de estas pruebas es necesario tener conectado al robot por medio de la tarjeta del microprocesador a la computadora para poder enviar comandos desde la terminal a cada uno de los componentes o recibir la información de los mismos y poder verla en el monitor. También se utiliza un cable de conexión serial. Estas pruebas se realizan utilizando únicamente el sistema operativo RobOS.

Sistema operativo RobOS

Pruebas a los Motores

El robot tiene solamente dos ruedas que funcionan con 2 motores independientes. La tercera rueda es una esfera de giro libre. Esto permite al robot dirigirse hacia el objetivo. Cada uno de los motores se conecta a la tarjeta de etapa de potencia. Se conecta una pila de 9V debidamente cargada a la tarjeta, que es la responsable de proporcionar el voltaje a los motores. Se manda el comando de avance o retroceso a cada motor separadamente a través de una interfase con la computadora.

Si el motor gira en el sentido opuesto al indicado se debe corregir la polaridad del mismo para poder resolver este problema. En dado caso en que alguno de los motores no funcione cuando se manda el comando desde la computadora, se debe analizar en primera instancia los componentes físicos del robot.

Verificar cada una de las conexiones que corresponden desde la pila a través de la tarjeta hasta los motores. En caso de que las conexiones eléctricas entre los componentes de la tarjeta de etapa de potencia así como los diversos componentes electrónicos no se encuentren descompuestos, se verifica el cable de conexión entre dicha tarjeta y la tarjeta del microprocesador HC11[11]. En caso de que no se presente algún problema con dicho cable, se deben probar otros motores para verificar que el desperfecto se encuentra en los mismos.

Pruebas a los Encoders

Se coloca un círculo dibujado en las ruedas cuya superficie ha sido dividida en un número determinado de sectores (potencia de 2), codificándose cada uno de ellos según un código binario cíclico (normalmente código de Gray) que queda representado por zonas transparentes y opacas dispuestas radialmente. Los encoders detectan la variación

entre el color negro y el color blanco y nos permiten conocer el ángulo de giro de los motores con lo cuál es posible saber que distancia ha recorrido el robot [1].

El número de regiones dibujadas nos indica la resolución con la cuál se está trabajando. A un mayor número de regiones se tiene una mejor resolución lo que conlleva a tener un ángulo más exacto de giro. Se verifica que estos sensores sean capaces de registrar el cambio entre regiones cuando el robot esté en movimiento. Para esto se conecta el robot a la computadora y se pide que muestre lo que está leyendo los encoders.

Si no se registra el cambio de valor cuando el robot se encuentra en movimiento, se debe de ajustar la posición del encoder sobre el eje del motor a una distancia en la que si pueda registrar el cambio de región en el disco que se encuentra en cada una de las ruedas del robot. Otra razón puede ser que las regiones opacas del disco no sean lo suficientemente oscuras, por lo que se puede requerir de algún tipo de marcador negro para resaltarlas. Un último factor por considerar es el número de regiones totales en el disco ya que si es mucho mayor al que puede registrar el encoder en cuanto a cambios, se deberá utilizar uno con un número adecuado de regiones totales.

Pruebas a los Sensores

Se conectan los sensores a su tarjeta correspondiente. Entre estas pruebas se encuentran las que se realizan en cuanto al correcto funcionamiento de cada uno de ellos que se efectúa mediante la adecuada calibración de los mismos a través de potenciómetros que se encuentran localizados en la tarjeta de calibración de los sensores.

Se debe verificar que los sensores sean capaces de detectar una línea oscura que se encuentre localizada sobre el piso por lo que se debe de ajustar ya sea la calibración o en dado caso la distancia de estos con respecto a la superficie donde está dibujada la línea a seguir. Por lo general se procura mantener a todos los sensores a una misma distancia con respecto a la superficie donde se encuentra el robot [1][16].

Cuando los sensores estén correctamente calibrados, se coloca al robot en puntos claves del laberinto como lo son las intersecciones. Teniendo conectado el robot a la computadora es posible mandarle la instrucción de que envíe lo que esta leyendo en ese momento. Conociendo los valores de cada uno de los sensores cuando detectan una línea negra es posible determinar si el valor obtenido es correcto o no. Los valores para cada uno de los sensores se guardan en una tabla para futuras referencias. Esta tabla también nos permite calcular el valor que resulta cuando varios de los sensores se activan al detectar la línea oscura del laberinto.

Se analizan todos los posibles casos de intersección que pudiera encontrar el robot, ya sea en forma de “T” o de cruz. Si los sensores no registran el valor correspondiente a cada uno de estos casos entonces se analiza uno por uno cual es el que requiere volver a calibrarse. En dado caso en que no se pueda calibrar, se debe reemplazar por otro sensor que se pueda calibrar adecuadamente.

Pruebas del Robot realizando el recorrido en el laberinto

La prueba consistirá en la navegación autónoma a través de un laberinto de **n x n** metros desde un punto de partida hasta encontrar la salida. El laberinto estará constituido por líneas rectas de color negro de 5 mm de ancho, diseñadas con intersecciones en ángulos rectos. Todas las medidas son aproximadas, el robot deberá ser capaz de recorrer dicho laberinto sin presentar una gran desviación de la ruta a seguir.

El material del suelo del laberinto puede ser cartón blanco o madera aglomerada con una capa de pintura de color blanco con líneas negras no reflejantes para evitar que los sensores puedan tomar una lectura incorrecta del entorno en que se encuentre el robot. Se coloca al robot al principio del laberinto y se le deja funcionar con el programa que se le ha colocado en memoria para la resolución de laberintos. Se observa si su comportamiento es el adecuado, viendo si es capaz de seguir una línea recta sin muchas correcciones a su curso o si identifica adecuadamente las intersecciones o vueltas marcadas.

Si el robot presenta muchas correcciones de curso cuando se encuentra recorriendo una línea recta es necesario revisar que ambos motores estén operando eficazmente y que no haya mucha variación en el ángulo de giro cuando reciben el comando de girar en cualquier sentido.

Otro caso de interés es el movimiento que realiza cuando debe girar 90 grados con respecto a su posición en el momento que detecta algún tipo intersección. Si el robot no gira exactamente los 90 grados ya sea que no alcanza dicho ángulo o se sobrepasa del mismo, se debe corregir el programa en la memoria para que dicho ángulo sea exactamente de 90 grados. En otra instancia cuando el robot encuentre un fin de línea inesperado es necesario que su ángulo de giro sea de 180 grados para que regrese por el camino que había seguido. Se debe verificar que dicho ángulo de giro sea correcto también.

Finalmente se debe comprobar que si el robot una vez resuelto el laberinto sea capaz de detenerse al detectar un círculo negro en la superficie y que no se mueva o regrese sobre su trayectoria cuando haya llegado a dicho círculo.

Si al final de verificar cada uno de estos puntos de interés, se comprueba el correcto funcionamiento del robot, se puede llevar otro tipo de prueba utilizando otro laberinto que presente características diferentes con respecto al laberinto anteriormente empleado. Estas características pueden ser el material empleado como superficie, el tono de línea negra que se emplea, las dimensiones del laberinto así como su número y tipo de intersecciones. Si el robot es capaz de resolver este otro laberinto se puede afirmar que tanto la programación como los componentes físicos de captura de datos y de movimiento funcionan adecuadamente con respecto a las necesidades del robot.

Todas las pruebas anteriores se realizaron cuando se empezó a trabajar inicialmente en este proyecto utilizando exclusivamente el sistema operativo RobOS. Sin

embargo tiempo después se introdujo otro componente al proyecto el cual fue la incorporación de una mini computadora portátil conocida como Pocket PC. Esto conllevo a programar un nuevo sistema operativo para la Pocket PC. Este nuevo sistema conocido como Cerebro, interactúa con el sistema operativo RobOS para el envío y recepción de datos. Debido a lo anterior es necesario crear un protocolo de comunicación entre estos dos sistemas, el cual es utilizado por medio de un cable serial entre el HC11 y la Pocket PC.

Sistema operativo Cerebro

Pruebas de comunicación entre la Pocket PC y el microprocesador HC11

Se conectan a través del puerto serial estos dos dispositivos mediante un protocolo de comunicación se verifica que el menú de inicio del programa almacenado en la memoria de la tarjeta del microprocesador HC11 se despliegue en la pantalla de la Pocket PC. A través de dicho menú se pueden mandar instrucciones desde la Pocket hacia el microprocesador [8]. Esta comunicación por medio del puerto serial se efectúa usando los siguientes parámetros de conexión:

BaudRate = 9600
ByteSize = 8

Estos parámetros se programan en una librería hecha en C#, la cual permite controlar el protocolo de comunicación variando los diferentes parámetros de la misma. Debido a que el microprocesador no puede manejar una velocidad de transferencia mayor a 9600, se acopla la Pocket PC para que envíe y reciba los datos a esta velocidad. Se procede a realizar el envío de varios comandos desde la Pocket PC hacia el robot para que éste los realice. Estos comandos se realizan para verificar que se puedan transferir instrucciones en ambas direcciones, esto es, se pueda enviar y recibir información entre el microprocesador y la Pocket PC.

Como siguiente procedimiento se envían varios comandos sucesivamente para ver si no hay un problema en el momento de recepción y no haya pérdida de datos lo que conlleve a una mala actuación del robot cuando el robot esté realizando su recorrido dentro del laberinto. Si se presenta algún problema, se debe verificar en que momento el robot empezó a presentar perdida de datos. Por consecuencia, se debe corregir el protocolo de comunicación para que no vuelva a ocurrir dicha pérdida de datos.

Pruebas realizadas utilizando otro tipo de interfase en la Pocket PC

Una vez comprobado que se puede utilizar el menú de inicio programado en la memoria del HC11, se procede a elaborar otro tipo de interfases que puedan transmitir comandos al HC11 sin la necesidad de que se tenga que desplegar el menú de inicio. Esto es, únicamente se verifica que se establezca una comunicación a través del puerto serial entre el microprocesador HC11 y la Pocket PC.

Entre este tipo de interfases se encuentra la que permite controlar el movimiento del robot a través de la pantalla touchscreen de la Pocket PC. Desplegando en la pantalla cuatro botones que al ser cada uno de ellos activado transmiten al robot el comando de moverse hacia adelante, atrás, vuelta de 90 grados hacia la izquierda o hacia la derecha.

Los problemas que pudiesen presentarse en este caso, serían en cuanto a mantener la comunicación entre el HC11 y la Pocket PC activa, así como del tiempo entre comandos recibidos cuando el robot se encuentra moviéndose todavía con el comando anterior.

Pruebas utilizando el sistema Cerebro

Finalmente, se debe comprobar el adecuado funcionamiento del sistema Cerebro cuando ha sido totalmente implementado en la Pocket PC y que debe ser capaz de guiar al robot a través del laberinto. Colocando al robot al inicio del laberinto, se le deja recorrer el mismo hasta que ha llegado al punto final del mismo. El sistema cerebro es el encargado de la toma de decisiones en cuanto al tipo de movimientos que el robot debe realizar para que el resultado sea la resolución del mismo.

Se observa si el robot realiza los diversos tipos de movimiento que ya habían sido establecidos antes de instalar el sistema RobOS para cada una de las situaciones que se le presentan en el laberinto. Estas situaciones son las curvas, intersecciones, finales de línea y fin de laberinto. Si el robot es capaz de resolver el laberinto, se considera que la implementación del sistema ha sido adecuada. Si algún tipo de problema surge es necesario determinar cuál es la causa por la que se suscita.

Capítulo 9 Conclusiones

El mercado de los robots tiene actualmente un auge creciente, cada vez hay más movimientos empresariales alrededor de la robótica. Los robots Aibo y Lego son sólo dos ejemplos de éxitos de ventas, superando cada uno las cien mil unidades vendidas. Los prototipos de investigación cada vez tienen mayor funcionalidad. Los robots han dejado de utilizarse exclusivamente en fábricas y se acerca su uso en el hogar, como robots de entretenimiento, o de servicio.

Tener robots realizando autónomamente tareas depende de su construcción mecánica y fundamentalmente de su programación. La inteligencia y la autonomía del robot residen básicamente en su programa, que es el que determina cómo se comporta, el que decide qué instrucciones ordenar a los motores en función de las observaciones sensoriales y los objetivos del comportamiento. La programación de robots es un arte difícil que presenta ciertos requisitos genuinos frente a la programación de otras aplicaciones más clásicas. Por ejemplo, los programas de robots están basados en la realidad física a través de sensores y motores, y suelen tener requisitos de tiempo real y vivacidad. Adicionalmente, no hay estándares consolidados en la programación de robots. Esto se debe en parte a la heterogeneidad del hardware, y en parte a la inmadurez del mercado de los robots programables. No hay una base firme y sólida sobre la que crear aplicaciones y reutilizar código.

Con el paso de los años, el modo de programar los robots ha ido evolucionando y se ha ido articulando en tres niveles diferenciados: sistemas operativos, plataformas de desarrollo y aplicaciones concretas. El sistema operativo del robot proporciona el acceso básico a los recursos de hardware, entre los que se incluyen los sensores y los motores, así como los dispositivos de comunicaciones e interacción. El hardware de los robots evoluciona con mucha rapidez, y en los últimos años hemos observado la incorporación de la visión y del láser como sensores adicionales. Otra tendencia confirmada es la extensión de la computadora personal (ya sea PC, laptop, ó Ipaq) como procesador principal de los robots. Más allá de los sistemas operativos dedicados, esta incorporación de la PC ha traído el uso creciente de lenguajes de alto nivel con sus herramientas asociadas. Las plataformas de desarrollo han surgido para facilitar el desenvolvimiento de aplicaciones, tanto las creadas por los propios fabricantes, como las creadas por grupos de investigación.

Entre las distintas plataformas existentes se han identificado varias líneas comunes: uniforman y simplifican el acceso al hardware, ofrecen una arquitectura software determinada e incorporan funcionalidad robótica común como la navegación, la construcción de mapas o la localización. A grandes rasgos, las plataformas tratan de fijar una base estable sobre la cual desarrollar aplicaciones robóticas, y eso supone un paso hacia la madurez de la programación de robots. Primero, favorecen la portabilidad de aplicaciones entre robots diferentes. Segundo, fomentan la reutilización de código, con lo cual disminuyen los tiempos de desarrollo de las nuevas aplicaciones. Y tercero, con su

arquitectura software ofrecen una manera de organizar el código, lo cual permite afrontar la complejidad creciente de las aplicaciones de robots.

El auge actual del mercado de los robots programables y los resultados conseguidos en los últimos años en prototipos apuntan a un futuro prometedor de la robótica. Sin embargo, dejando a un lado las expectativas idealistas, el grado de autonomía y de confiabilidad que seamos capaces de conseguir en los robots determinará fundamentalmente el crecimiento de ese mercado. Por ejemplo, la introducción de los robots en los hogares exige un nivel mucho mayor de confiabilidad e independencia del entorno de los conseguidos hasta la fecha. Con mayores grados de autonomía las tareas en las que se podrán aplicar robots crecerán significativamente.

Los avances en niveles de autonomía y de confiabilidad pasan por superar la heterogeneidad actual en el software de los robots, por disponer de entornos estables de programación, y como en otras disciplinas, por la reutilización de conocimiento. La aparición de las plataformas de desarrollo supone un paso hacia adelante en ese largo camino. Sin embargo, aunque la necesidad de estándares en robótica ha sido identificada, en la actualidad existen muchas plataformas diferentes, tal vez demasiadas, refleja que no hay aún un estándar universalmente admitido. Habrá que ver cuales sobreviven de aquí a unos años, y eso dependerá enormemente de cuantos usuarios y desarrolladores sean capaces de atraer cada una de ellas.

Quizá la evolución en el corto plazo converja a la existencia de una o dos plataformas estables, al menos para uniformar el acceso al hardware. Por ejemplo, se han definido las interfaces de movimiento y de los sensores más frecuentes, que son válidas para un conjunto amplio de robots de interiores con ruedas y no muy distintas entre las diferentes plataformas. Esa convergencia permitiría enfocar mejor los esfuerzos en cómo organizar el código de las aplicaciones para generar comportamiento en los robots, que sigue siendo un reto de primera magnitud. Hay mucho por experimentar y crecer en esta línea, y tal vez por ella se podrán alcanzar las cotas de autonomía y confiabilidad necesarias si queremos tener robots, por ejemplo, realizando tareas automáticamente en los hogares.

En esta tesis se estudió el problema de la navegación en robots móviles con ruedas, que se desplazan por una superficie plana, a través de un entorno parcialmente estructurado. En particular se ha estudiado el problema del seguimiento de un camino de referencia, el cual es detectado mediante un sistema de sensores infrarrojos. El problema se ha analizado en sus dos aspectos fundamentales: la estrategia de control y el tratamiento de la información sensorial del entorno. En primer lugar se realizó una revisión de los distintos métodos propuestos en la literatura para resolver el problema de la navegación en robots autónomos como es el algoritmo de Dijkstra para la obtención de la ruta mínima hacia un punto determinado.

Se vió que una buena táctica para resolver problemas que a simple vista parecen grandes y complejos es dividirlos en subproblemas más específicos y limitados, entonces nos queda sólo encontrar la solución para esos subproblemas que lo hemos encontrado con la realización de diversos algoritmos. Tres son los algoritmos ideados, uno encargado

del seguimiento de una trayectoria, otro para la inspección del laberinto y generación de un camino óptimo. Cada algoritmo proporciona la información suficiente para que el siguiente algoritmo pueda trabajar y son totalmente independientes.

El algoritmo encargado de seguir la trayectoria, requiere que la información recibida por medio de los sensores infrarrojos sea lo más confiable posible. Como se vio en el desarrollo de este proyecto, esto es de suma importancia ya que de otra forma el robot se comporta de manera errática y no se cumple el objetivo principal. Se debe tener cuidado en el manejo de todos los dispositivos tanto del hardware como del software.

El algoritmo encargado de la inspección del laberinto y generación del camino óptimo está basado en poder establecer una secuencia entre los nodos que conforman al laberinto. Los nodos se definen como las posibles intersecciones o fines de camino en donde el robot tiene que decidir la ruta que ha de seguir para poder llegar al final del laberinto. Obteniendo cada uno de los nodos que conforman al laberinto, es posible eliminar las rutas que realmente no son necesarias y que conllevan a un desperdicio de tiempo por parte del robot. Utilizando el algoritmo de Dijkstra es posible calcular un camino óptimo a partir de estos nodos y finalmente establecer una ruta específica para que el robot pueda realizar su objetivo en el menor tiempo posible.

Un próximo acercamiento al problema de resolución de laberintos o al control de robots, involucra la comunicación inalámbrica entre una computadora central que sea la encargada de analizar las instrucciones que debe realizar el robot y el control mediante la visión de los mismos. Las comunicaciones inalámbricas son interesantes desde todos los puntos de vista y en todos los ámbitos tecnológicos. El fin principal que persigue la robótica móvil es la creación de sistemas completamente autónomos y un grado de autonomía puede ser la comunicación entendida en un sentido amplio, es decir, entre varios robots, entre los robots y una computadora base, entre los robots y otros elementos del entorno o entre los robots y los propios humanos.

Apéndice A

Preguntas Frecuentes

Sensores

1. ¿Qué hacer si no registra cambio de valor el sensor?

La distancia óptima de separación entre la superficie y los sensores es de 2-4 milímetros. Si se encuentran demasiado separados de la superficie, es muy probable que no detecten cambio entre zona blanca y zona negra. Si aún así no registran cambios los valores del sensor es necesario checar que el cable que va de los sensores a la tarjeta de calibración esté en la posición correcta y que el cable que va de la tarjeta de calibración a la tarjeta del HC11[11] esté conectada correctamente, Si todo está bien conectado es necesario, con un multímetro, checar que las pilas estén bien cargadas. El voltaje que deben dar es de 9.6 Volts con un error de ±5%. Si aún así no registran cambios los valores de los sensores es necesario calibrar cada sensor hasta que el LED de la tarjeta calibradora se encienda y se apague, cuando pase de blanco a negro para cada uno de los sensores.

2. ¿Qué hacer si se registra un valor incorrecto de sensores?

Si se registra un valor equivocado que proviene de los sensores es necesario calibrar cada sensor hasta que se prenda y apague el LED de la tarjeta calibradora al pasar de blanco a negro para cada uno de los sensores. También se debe verificar que cada sensor se haya conectado de forma correcta a cada uno de los pines del puerto de lectura [16]. Es posible que se haya alterado el orden de colocación de los mismos, si esté es el caso, o se adecúa el programa con los nuevos valores de sensores o se posicionan correctamente los sensores en el armazón.

3. ¿Cómo calibrar los sensores?

El circuito sensorial cuenta con 7 potenciómetros con los cuáles es posible modificar la resistencia empleada con el propósito cambiar la sensibilidad y de esta forma detectar cambios entre zona blanca y zona negra. Para calibrar los sensores es necesario contar con un destornillador de punta plana pequeño que se utilizará para girar cada uno de los potenciómetros de la tarjeta calibradora girando poquito a poco hasta que el LED que corresponde al sensor que estamos calibrando prenda y apague al colocar el sensor en una superficie blanca y negra. Es necesario realizar este procedimiento para cada sensor hasta que todos los leds de esta tarjeta prendan y apaguen, al cambiarlos de una superficie blanca a negra. Para la correcta calibración es necesario notar cuando el sensor esté detectando zona oscura que su led asociado esté prendido, y si se encuentra en zona clara

esté apagado. Esta configuración puede cambiar dependiendo del tipo de sensor. Los sensores infrarrojos tienen limitaciones, pueden recibir mucha interferencia tanto de luz ambiente sobre todo si son de tipo neón, por lo que se sugiere ubicarlos de tal forma que esta interferencia sea mínima.

Motores

1. ¿Qué hacer si no funciona algún motor?

Se puede verificar su funcionamiento conectándolo directamente a una fuente de voltaje, teniendo en cuenta que el voltaje requerido para su adecuado funcionamiento es de mínimo 3Volts y máximo 9 Volts. Si el motor no funciona con esta sencilla prueba es posible que alguno de sus componentes internos se encuentre dañado, ya sea eléctrico o mecánico; en dicho caso es necesario cambiar el motor por algún otro que sí esté funcionando. Otra posible razón de que no funcione es que los cables de alimentación estén dañados, esto se puede verificar con un multímetro para detectar que haya continuidad en el cable. Otra posible causa es que sea un problema de la tarjeta del microprocesador y que no esté llegando una señal de salida al puerto en que están conectados los motores. También si aún así no funciona un motor es necesario checar que esté bien conectado a la tarjeta de la etapa de potencia y que esta tarjeta esté bien conectada a la tarjeta del HC11 en el puerto A, si aún así no funciona algún motor o ambos es necesario checar que las pilas estén bien cargadas, con un voltaje de 9.6 Volts con un error de ±5%, si aún así no funciona algún motor o ambos es necesario cambiar el chip 74ls04. Este chip es un inversor que permite controlar con sólo 2 cables los motores. Si persiste el problema es necesario cambiar el chip L293D, que es para la etapa de potencia. Finalmente puede que el problema sea parte del programa y no se esté direccionando de forma correcta al puerto empleado.

2. ¿Qué hacer si un motor gira en un sentido y el otro motor gira en sentido contrario al darle forward?

Esto se debe principalmente a que uno de los motores ha sido conectado a la fuente de alimentación de manera que su polaridad es la inversa de la deseada. Esto se corrige mediante la correcta disposición de los cables de alimentación. Y se verifica con el comando forward (adelante) si el movimiento es hacia atrás es necesario cambiar ambos de polaridad para que se corrija el problema.

3. ¿Qué hacer si, cuando se le da left o right el robot gira en el sentido contrario?

Puede haber dos causas principales por lo que ocurra esto. La primera es la polarización inversa de los motores, caso analizado en la pregunta anterior. La segunda es que si se está ejecutando el programa seguidor de línea, los sensores se encuentren mal ubicados y deban ser colocados correctamente en el armazón.

Tarjeta de Etapa de Potencia

1. ¿Qué hacer si los motores no hacen nada?

Primero se verifica que la pila conectada a la etapa de potencia esté debidamente cargada, el voltaje que deben dar es de 9.6 Volts con un error de ±5%, si éste no es el caso, se debe cambiar por otra que haya sido previamente cargada con alguno de los cargadotes que se encuentran en el laboratorio. Después se verifica que los cables de alimentación estén conectados, posteriormente se debe verificar que el regulador sí esté funcionando. Si con lo anterior no se ha solucionado el problema, se debe proceder a verificar el correcto funcionamiento de cada uno de los elementos de la tarjeta de etapa de potencia, principalmente el chip L293D[15] ya que lo más probable es que se haya quemado.

2. ¿Qué hacer si los motores se mueven demasiado rápido o lento?

Es necesario checar que las pilas estén bien cargadas, el voltaje que de las pilas debe ser de 9.6 Volts con un error de ±5%. Si están bien cargadas y los motores giran demasiado lento o demasiado rápido hay que regular la corriente empleada en su alimentación, puede ser que se le esté suministrando un exceso de corriente, lo cuál puede causar que los motores sufran un desperfecto. Para regularla es necesario girar con un destornillador pequeño el potenciómetro de precisión hasta que giren a la velocidad deseada.

Tarjeta de HC11

1. ¿Qué hacer si no se reciben datos de los sensores?

Se debe corroborar primero que cada uno de los sensores esté funcionando, esto se puede verificar observando los leds asociados a cada uno de ellos. Posteriormente, se checa que el cable de conexión al puerto A esté debidamente colocado, generalmente los conectores presentan una pestaña que impide que se conecten de manera inversa pero suele suceder que la tarjeta de los sensores no posea la contraparte del conector que asegura esto. Es necesario checar también que el voltaje de la pila de alimentación del HC11[11] esté bien cargada.

2. ¿Qué hacer si la tarjeta falla para comunicarse con la PC?

Si no es posible comunicarse con la tarjeta desde la PC es necesario resetear la tarjeta si aún así no funciona es necesario reiniciar la PC y configurar el programa de la

siguiente manera: 9600bps, 8 bits de datos, paridad ninguna, bits de parada 1, control de flujo por hardware. En el peor de los casos, no se puede hacer nada al respecto puesto que involucraría empezar a analizar en qué parte de la tarjeta se encuentra la falla o descompostura que provoca que ya no funcione la tarjeta. Fácilmente se aprecia que esta tarea no es realizable a menos de que se conozca con detalle a la misma tarjeta. Si esto ocurre, lo más recomendable es cambiar la tarjeta por otra.

3. ¿Qué hacer si la tarjeta no enciende?

Primero verificar que la pila de la tarjeta del microcontrolador esté debidamente cargada y conectada. Segundo la tarjeta tiene unos 4 dip switches en la parte superior izquierda la configuración es la siguiente

1	2	3	4	Modo de Operación
ON	ON	ON	ON	BOOTSTRAP
ON	ON	OFF	OFF	EXPANDED
ON	ON	ON	OFF	SINGLE CHIP
ON	ON	OFF	ON	SPECIAL TEST

La configuración para ejecutar el sistema operativo es en Expanded y para programarlo es en modo Bootstrap debido a que usamos memoria expandida en una EEPROM, no se puede programar en modo Bootstrap por el tipo de memoria. Si los dip switch 1-2 están en off la tarjeta no encenderá.

4. ¿Qué hacer si no arranca el programa cuando se le da reset?

En algunos casos es posible que el botón de reset se haya descompuesto por lo que no es posible cerrar o abrir el circuito de interrupción de voltaje a la tarjeta. Si éste es el caso es necesario cambiar dicho botón o intentar con el botón de la tarjeta. También es posible que el programa no haya sido propiamente cargado en la memoria RAM o EEPROM, dependiendo de la memoria que se esté usando será necesario cargar el programa otra vez e intentar de nuevo. Es posible que la memoria esté dañada o se haya borrado si no se maneja adecuadamente.

5. ¿Qué hacer si no se muestran los datos en la pantalla de la PC cuando se enciende la tarjeta del HC11?

Generalmente esto indica que el programa se ha borrado de la memoria, por lo que es necesario volverlo a grabar en la memoria EEPROM, si esto no arregla el problema, es necesario cambiar la memoria por otra. Por otra parte, es necesario configurar el programa de la siguiente manera: 9600bps, 8 bits de datos, paridad ninguna, bits de parada 1, control de flujo por hardware.

Comunicación PC – Robot

1. ¿Qué pasa si no hay comunicación con el robot?

Verificar que el cable esté conectado entre el puerto serial y la tarjeta del microcontrolador. Corroborar que la tarjeta del microcontrolador esté encendida. En ocasiones es posible que la terminal de comunicación no esté habilitada para recibir datos o tenga una diferente velocidad de transmisión o que el puerto serial se encuentre cerrado, ajustar de tal forma que la comunicación esté debidamente establecida.

2. ¿Qué hacer si los datos mostrados en la pantalla no son los esperados?

Generalmente esto se debe a que la velocidad de transmisión entre el robot y la PC no es la adecuada, se debe verificar que dicha velocidad sea la misma con la que la tarjeta del microcontrolador esté trabajando.

3. ¿Qué hacer si al tratar de mandar un programa, sólo descarga un porcentaje del mismo y después manda un error de transmisión?

Esto puede deberse a que el programa sea demasiado grande para el tamaño de la memoria o se esté tratando de escribir a partir de cierta región de la memoria con lo que no se proporciona una zona de memoria lo suficientemente grande para que se grabe el programa en la misma. También es posible que se esté intentando grabar en un localidad privada de la memoria donde se deben cargar otros datos.

Funcionamiento del Robot

1. ¿Qué hacer si el robot empieza a moverse de manera errática?

Presionar el botón de reset para que el programa reinicie y se verifique su adecuado funcionamiento. Si el movimiento errático persiste, se debe verificar el estado de la memoria y de los datos recibidos de los sensores.

2. ¿Qué hacer si el robot no reconoce intersecciones?

Verificar que los sensores estén detectando una intersección. Si no es el caso se deben calibrar para que haya una detección de la misma. Si los sensores están funcionando adecuadamente se debe verificar que el programa esté analizando el caso de dicho tipo de intersección adecuadamente.

3. ¿Qué hacer si el robot no se mueve la distancia especificada?

Suele suceder que después de un largo funcionamiento del robot, la pila de los motores se empiece a descargar lo que ocasiona que no haya un adecuado voltaje a los mismos. Por lo que se puede cambiar la pila. También es posible que los encoders no estén recibiendo bien la información con lo que se produce un movimiento inadecuado del robot. Finalmente es posible que las fórmulas para calcular las distancias de movimiento no sean las correctas.

Apéndice B

Como usar las funciones de usuario

Dentro del sistema operativo se pueden incorporar funciones de usuario, estas funciones son subrutinas que se pueden programar en lenguaje “C” usando los comandos del sistema operativo y hacer funciones propias para ser usadas dentro de la misma subrutina, lo primero que hay que hacer es en archivo que contenga esta estructura.

```
#pragma abs_address:0x1300

user_subrutine(int Command)
{
 switch(Command)
          {
           case 1:
                                usuario1();
                                break;
           default:        printf("\nSubroutine %d not found",Command);
          }
}

usuario1()
{
Programa de usuario
}

Función a ser usada en nuestro programa (long int b)
{
 long int i,j;
 b=b*301;
 for(j=0 ; j<3 ; j++)
  {
   for(i=b ; i>=0 ; i--)
    {
    }
  }
}

#pragma end_abs_address
```

En este programa se debe colocar la función pragma, la cual sirve para poder posicionar en la localidad de memoria requerida el programa. En este caso es la dirección 1300 y al final es necesario cerrar esta función con #pragma end_abs_address; esto sirve para indicarle al compilador que nuestro programa va a empezar en la dirección 1300 y terminará en la dirección absoluta calculada por el compilador. Una cosa muy importante es que nuestro archivo no debe de exceder los 7Kb ya que si excede, éste invadirá otras regiones de memoria que están destinadas para la configuración del procesador o donde empieza el sistema operativo del robot.

También es necesario crear un switch exactamente como se muestra anteriormente ya que ésta es la función que se encargada de llamar a nuestra función de usuario. Cabe recalcar que el nombre de la función user1 () puede cambiar de nombre ya que al ejecutar desde el sistema operativo serán llamadas a partir del switch case que le hayamos dado; por ejemplo para llamar la función del ejemplo tendríamos que poner la siguiente línea en la línea de comandos

User 1 @21

Esta línea lo que hace es llamar a la función de usuario numero 1 de nuestro switch case.

Una vez que se haya compilado nuestro programa será necesario grabar la memoria eeprom y posteriormente cargar nuestra función de usuario a la RAM del robot, esto se hace de la siguiente manera:

Se ejecuta el programa para programar la memoria, después se va al menú de “buffer” y se carga el programa a grabar poniendo “load”, archivo de tipo Motorola poniéndole al archivo un file buffer de 8000 ya que la memoria y el programara el HC11 está a partir de la dirección #8000.

Una vez cargado el programa se escoge en el menú de “*select device type*” donde se pone el tipo de memoria que es (este dato lo obtenemos leyendo de la parte superior del chip de la memoria).

Ya elegida el tipo de memoria se va al menú de “*device*” donde se va a la parte de “*function*” apareciendo otra ventana donde se le da al comando “*program*”, con lo que empieza el procedimiento de grabado de la memoria. Una vez grabada hará dos bips indicando que acabó.

Luego se procede a colocar la memoria en la tarjeta de desarrollo del HC11 para poder ejecutar la subrutina de usuario.

Apéndice C

Fragmento del código del Sistema Operativo

```
#include "stdiorob.h"
#include "varhc11.h"
#include "math.h"

float global_distance,global_angle;


/*it puts a value on port A */
put_data_port_a(int value){
        int prev_conf;
        /* it reads previous configuration */
        prev_conf=peek(CONFIG_REG_PORTA);
        poke(CONFIG_REG_PORTA,0xff);
        poke(PORTAA,value);
        /* it recovers previous confuguration */
        poke(CONFIG_REG_PORTA,prev_conf);
}
/* initialices the system */
initialization(){
        int value;
        /*value=peek(_IO_BASE);
        printf("\n init %x",value);*/
        /*poke(REG_INIT,0x01);*/
        /* It initializes the serial port to 9600 bauds */
        setbaud(BAUD9600);
        /* It puts the start address in the reset vector */
        init_reset();
        /* It configurates input and output ports */
        configurates_ports();
        /* It initializes the interrupt vectors */
        init_interrupt_vectors();
        /* It initializes the robot's atributes */
        init_robot();
}

/* It initializes the robot's atributes */
init_robot()
{
        strcpy(Robot.Name,RobotsName);
        Robot.Position.X=InitPositionRobotX;
        Robot.Position.Y=InitPositionRobotY;
        Robot.Position.Angle=InitPositionRobotAngle;
        init_sensors();
        init_motors();
}

/* it configurates  a port */
configurate_port(int port,int mask){
        int configuration_register;
        int conf;
        if(port==PORTAA)configuration_register=CONFIG_REG_PORTA;
        else if(port==PORTG)configuration_register=CONFIG_REG_PORTG;
        else{
                printf("\n %x not exist",port);
                return(0);
```

```
        }
        /*conf=peek(configuration_register);
        if(option==0)conf= conf & ~mask;
        else conf=conf | mask;*/
        /*printf("\n puerto %x mask %x\n",configuration_register,mask);*/
        poke(configuration_register,mask);
}

/* it initializes the motors' sensors */
init_motors(){
        Robot.MotorDc[0].PositionX=POSMOTORX0;
        Robot.MotorDc[0].PositionY=POSMOTORY0;
        Robot.MotorDc[0].PositionZ=POSMOTORZ0;
        Robot.MotorDc[1].PositionX=POSMOTORX1;
        Robot.MotorDc[1].PositionY=POSMOTORY2;
        Robot.MotorDc[1].PositionZ=POSMOTORZ2;
        Robot.MotorDc[0].Mask=MASK_MOTOR0;
        Robot.MotorDc[1].Mask=MASK_MOTOR1;
        Robot.MotorDc[0].Port=PORTMOTOR0;
        Robot.MotorDc[1].Port=PORTMOTOR1;
        /*configurate_port(Robot.MotorDc[0].Port,Robot.MotorDc[0].Mask,WRITE_PORT);
        configurate_port(Robot.MotorDc[1].Port,Robot.MotorDc[1].Mask,WRITE_PORT);*/
}

/* It initializes the robot's sensors */
init_sensors(){

        Robot.Sonar[0].Mask= MaskSonar1;
        Robot.Sonar[1].Mask= MaskSonar2;
        Robot.Sonar[2].Mask= MaskSonar3;
        Robot.Sonar[0].Port=PortSonar;
        Robot.Sonar[1].Port=PortSonar;
        Robot.Sonar[2].Port=PortSonar;
        Robot.Sonar[0].PositionX=PosSonarX1;
        Robot.Sonar[0].PositionY=PosSonarY1;
        Robot.Sonar[0].PositionZ=PosSonarZ1;
        Robot.Sonar[1].PositionX=PosSonarX2;
        Robot.Sonar[1].PositionY=PosSonarY2;
        Robot.Sonar[1].PositionZ=PosSonarZ2;
        Robot.Sonar[2].PositionX=PosSonarX3;
        Robot.Sonar[2].PositionY=PosSonarY3;
        Robot.Sonar[2].PositionZ=PosSonarZ3;
        /*configurate_port(Robot.Sonar[0].Port,Robot.Sonar[0].Mask,READ_PORT);
        configurate_port(Robot.Sonar[1].Port,Robot.Sonar[1].Mask,READ_PORT);
        configurate_port(Robot.Sonar[2].Port,Robot.Sonar[2].Mask,READ_PORT);*/
        Robot.Contact[0].Mask= MaskContact1;
        Robot.Contact[1].Mask= MaskContact2;
        Robot.Contact[2].Mask= MaskContact3;
        Robot.Contact[0].Port=PortContact;
        Robot.Contact[1].Port=PortContact;
        Robot.Contact[2].Port=PortContact;
        Robot.Contact[0].PositionX=PosContactX1;
        Robot.Contact[0].PositionY=PosContactY1;
        Robot.Contact[0].PositionZ=PosContactZ1;
        Robot.Contact[1].PositionX=PosContactX2;
        Robot.Contact[1].PositionY=PosContactY2;
        Robot.Contact[1].PositionZ=PosContactZ2;
        Robot.Contact[2].PositionX=PosContactX3;
        Robot.Contact[2].PositionY=PosContactY3;
        Robot.Contact[2].PositionZ=PosContactZ3;
        /*configurate_port(Robot.Contact[0].Port,Robot.Contact[0].Mask,READ_PORT);
        configurate_port(Robot.Contact[1].Port,Robot.Contact[1].Mask,READ_PORT);
```

```
		configurate_port(Robot.Contact[2].Port,Robot.Contact[2].Mask,READ_PORT);*/
		Robot.Infrared[0].Mask= MaskInfrared1;
		Robot.Infrared[1].Mask= MaskInfrared2;
		Robot.Infrared[2].Mask= MaskInfrared3;
		Robot.Infrared[0].Port=PortInfrared;
		Robot.Infrared[1].Port=PortInfrared;
		Robot.Infrared[2].Port=PortInfrared;
		Robot.Infrared[0].PositionX=PosInfraredX1;
		Robot.Infrared[0].PositionY=PosInfraredY1;
		Robot.Infrared[0].PositionZ=PosInfraredZ1;
		Robot.Infrared[1].PositionX=PosInfraredX2;
		Robot.Infrared[1].PositionY=PosInfraredY2;
		Robot.Infrared[1].PositionZ=PosInfraredZ2;
		Robot.Infrared[2].PositionX=PosInfraredX3;
		Robot.Infrared[2].PositionY=PosInfraredY3;
		Robot.Infrared[2].PositionZ=PosInfraredZ3;
		/*configurate_port(Robot.Infrared[0].Port,Robot.Infrared[0].Mask,READ_PORT);
		configurate_port(Robot.Infrared[1].Port,Robot.Infrared[1].Mask,READ_PORT);
		configurate_port(Robot.Infrared[2].Port,Robot.Infrared[2].Mask,READ_PORT);*/
		Robot.Reflective[0].Mask= MaskReflective1;
		Robot.Reflective[1].Mask= MaskReflective2;
		Robot.Reflective[2].Mask= MaskReflective3;
		Robot.Reflective[0].Port=PortReflective;
		Robot.Reflective[1].Port=PortReflective;
		Robot.Reflective[2].Port=PortReflective;
		Robot.Reflective[0].PositionX=PosReflectiveX1;
		Robot.Reflective[0].PositionY=PosReflectiveY1;
		Robot.Reflective[0].PositionZ=PosReflectiveZ1;
		Robot.Reflective[1].PositionX=PosReflectiveX2;
		Robot.Reflective[1].PositionY=PosReflectiveY2;
		Robot.Reflective[1].PositionZ=PosReflectiveZ2;
		Robot.Reflective[2].PositionX=PosReflectiveX3;
		Robot.Reflective[2].PositionY=PosReflectiveY3;
		Robot.Reflective[2].PositionZ=PosReflectiveZ3;
		/*configurate_port(Robot.Reflective[0].Port,Robot.Reflective[0].Mask,READ_PORT);
		configurate_port(Robot.Reflective[1].Port,Robot.Reflective[1].Mask,READ_PORT);
		configurate_port(Robot.Reflective[2].Port,Robot.Reflective[2].Mask,READ_PORT);*/
		Robot.Encoder[0].Mask= MaskEncoder1;
		Robot.Encoder[1].Mask= MaskEncoder2;
		Robot.Encoder[0].Port=PortEncoder;
		Robot.Encoder[1].Port=PortEncoder;
		Robot.Encoder[0].PositionX=PosEncoderX1;
		Robot.Encoder[0].PositionY=PosEncoderY1;
		Robot.Encoder[0].PositionZ=PosEncoderZ1;
		Robot.Encoder[1].PositionX=PosEncoderX2;
		Robot.Encoder[1].PositionY=PosEncoderY2;
		Robot.Encoder[1].PositionZ=PosEncoderZ2;
}

/* it shows the robots position */
show_position(){
		printf("\nsc %f ",Robot.Position.X);
		printf("%f ",Robot.Position.Y);
		printf("%f @%d",Robot.Position.Angle,A.id);
}

/* It configurates output ports */
configurates_ports(){
		/* Configuration of PORTA */
		configurate_port(PORTAA,CONFIG_PORTA);
		/* configuration of PORTG */
		configurate_port(PORTG,CONFIG_PORTG);
```

```
        /* Turns on A/D converter */
        poke(0x1039, 0x80);
        /* select the first block of a/d chanels (ADCTL).*/
        poke(0x1030, 0x10);
}

/* It initializes the interrupt vectors */
init_interrupt_vectors(){
        /* puts the interrupt functions in the interrupt vector addresses */
        /* These functions are in INTHNDL.C */
        /* Software interrupt */
        /*setvect(VSWI,soft_int_c);*/
        *(void(**)())VSWI=soft_int_c;
        /* Timer overflow interrupt */
        /*setvect(VTOI,toi_interrupt);   */
        *(void(**)())VTOI=toi_interrupt;
        /* Capture interrupt IC1 */
        /* setvect(VIC1,ic1_interrupt);*/
        /* Capture interrupt IC2 */
        /* setvect(VIC2,ic2_interrupt);    */
        /* Capture interrupt IC3 */
        /* setvect(VIC3,ic3_interrupt);    */
        /* Enable TOI interrupts */
        enable_toi();
        /* Enable IC* interrupts */
        /* enable_ic();*/
        /* Enable interrupts */
        asm(" cli ");
        /* Tests a software interrupt */
        /*asm(" swi ");*/
}

/* it turns the motors to the desired position */
turn_motor(int port, int mask, int direction){
        int command;
        command=peek(port);
        /*printf("\n previous porta %x 0x%x",port,command);*/
        command= command & (~mask);
        command= command | direction;
        /*printf("\n porta %x 0x%x",port,command);*/
        poke(port,command);
}

/* It finds the 2*pi module*/
float moda(float a){
        float c,d;
        float b=2*PI;
        if(a==0)d=a;
        else if(a < - 0.0001){
                d=2*PI + a;
        }
        else{
                 if(a>b){
                        c=a/b;
                        d=a - c*b;
                }
        else d=a;
        }
return(d);
}

#define CNT_GRAD 360/(2*PI)
```

```
/* it actualizes the new robot's position */
robot_position( float dist, float angle){
	double x,y;
	float anglef;
	anglef=(Robot.Position.Angle+angle);
	anglef=moda(anglef);
	Robot.Position.Angle=anglef;
	anglef= CNT_GRAD * anglef;
	x=dist*cos(anglef);
	y=dist*sin(anglef);
	Robot.Position.X=Robot.Position.X + x;
	Robot.Position.Y=Robot.Position.Y + y;
}

#define CNT_TIME_ADVANCE 1 /*30*/
#define CNT_TIME_TURN 1 /*23.0*/
#define CALCULATE_TIME 1
#define ROBOTS_RADIO 0.7   /* Measurements in decimeters */
#define ENCODER_TURNS (float)(SEG_WHEEL*2*ROBOTS_RADIO)/(2*PI*2*RADIO_WHEEL)
/*#define ENCODER_CNT1 (float)(SEG_WHEEL*2*ROBOTS_RADIO)*/
#define ENCODER_CNT1 (float)(SEG_WHEEL*ROBOTS_RADIO)
#define ENCODER_CNT2 (float)(2*PI*2*RADIO_WHEEL)
#define CNT_ADVANCE 0.42
#define CNT_ENCODER_ADVANCE (float) ((CNT_ADVANCE*SEG_WHEEL)/(PI*RADIO_WHEEL))
#define ENCODER_CNT1_ADVANCE (float) (CNT_ADVANCE*SEG_WHEEL)
#define ENCODER_CNT2_ADVANCE (float) (PI*2*RADIO_WHEEL)/*perimetro rueda*/

/* it rotates first the robot, then the robot advances and finally actualizes its new position */
int move_robot(float distance, float angle, float time){
	int i,sum,wheel_turns,lr;
	int PortMotor0,PortMotor1,MaskMotor0,MaskMotor1;
	float dist,dummy,div,wheel_turnsf,new_angle,new_distance;
	float count_encoder(), anglei;
	/* it gets the motors' information */
	PortMotor0=Robot.MotorDc[0].Port;
	PortMotor1=Robot.MotorDc[1].Port;
	MaskMotor0=Robot.MotorDc[0].Mask;
	MaskMotor1=Robot.MotorDc[1].Mask;
	/* It turns on the motors the required angle */
	/* It calculates the wheels' turns using time or the encoders*/
	#ifdef robot_blue
		/*dist=angle*ENCODER_TURNS;*/
		/*dist=angle*ENCODER_CNT1/ENCODER_CNT2;*/
		dist= angle*(ROBOTS_RADIO*SEG_WHEEL)/(PI*2*RADIO_WHEEL)
	#else
		dist=angle*SEG_WHEEL*CNT_TIME_TURN;
		/* printf("\n waiting %f \n",dist);*/
	#endif
	wheel_turns=(int)dist;
	/*printf("\n turn wt %d",wheel_turns);*/
	if(wheel_turns < 0)wheel_turns=-wheel_turns;
	wheel_turns=wheel_turns+1;
	if(angle < 0){
		/* it turns them to the right */
		turn_motor(PortMotor0,MaskMotor0,FORWARD_MOTOR0);
		turn_motor(PortMotor1,MaskMotor1,BACKWARD_MOTOR1);
		lr=-1;
	}
	else if(angle > 0){
		/* it turns them to the left */
		turn_motor(PortMotor0,MaskMotor0,BACKWARD_MOTOR0);
```

```
		turn_motor(PortMotor1,MaskMotor1,FORWARD_MOTOR1);
		lr=1;
}
else lr=0;

/*if(wheel_turns < 0)wheel_turns=-wheel_turns;
wheel_turns=wheel_turns+1;*/
/* It waits until the robot turns the required angle */
if((wheel_turns != 0) & (lr != 0)) {
		#ifdef robot_blue
			/* Count the number of the segments of the wheel, check which sensor to use: 0
			right, 1 left, o.w. both */
			anglei=count_encoder(wheel_turns,2);
			/*new_angle=lr * (float) (anglei/ENCODER_TURNS);*/
			/*new_angle=lr * (float) (anglei*ENCODER_CNT2);*/
			/*new_angle= new_angle/ENCODER_CNT1;*/
			new_angle=lr * (float) (anglei*(PI*2*0.355)/SEG_WHEEL);
			/*printf("\n anglei %f new_angle %f",anglei,new_angle);*/
		#else
			wait_toi_interrupts(wheel_turns);
			/*printf("\n waiting turn \n");*/
			new_angle=angle;
		#endif
}
else new_angle=0;
/* it moves the robot forward or backward */
#ifdef robot_blue
		/*wheel_turns= (int) (distance*CNT_ENCODER_ADVANCE);*/
		wheel_turns= (int) (distance*SEG_WHEEL/(PI*2*0.355));
#else
		wheel_turns= (int) (distance*SEG_WHEEL*CNT_TIME_ADVANCE);
		/*printf("\n waiting %d \n",wheel_turns);*/
#endif
/*printf("\n linear turns %d",wheel_turns);*/
if(distance > 0){
		/* The robot goes forward */
		turn_motor(PortMotor0,MaskMotor0,FORWARD_MOTOR0);
		turn_motor(PortMotor1,MaskMotor1,FORWARD_MOTOR1);
		lr=1;
}
else if(distance < 0){
		/* The robot goes backward */
		turn_motor(PortMotor0,MaskMotor0,BACKWARD_MOTOR0);
		turn_motor(PortMotor1,MaskMotor1,BACKWARD_MOTOR1);
		distance= -distance;
		lr=-1;
		wheel_turns=lr*wheel_turns;
}
else{
		/* Stops the robot */
		turn_motor(PortMotor0,MaskMotor0,STOP_MOTOR0);
		turn_motor(PortMotor1,MaskMotor1,STOP_MOTOR1);
		lr=0;
}
/* it waits until it goes the requiered distance */
if(wheel_turns > 0) {
		#ifdef robot_blue
			/* Count the number of the segments of the wheel,
			check which sensor to use: 0 right, 1 left, o.w. both */
			anglei=count_encoder(wheel_turns,2);
			/*new_distance=lr * (float) (anglei/CNT_ENCODER_ADVANCE);*/
			new_distance=lr * (float) (anglei*(PI*2*0.355)/SEG_WHEEL);
```

```
                        /*new_distance=lr * (float) (anglei*ENCODER_CNT2_ADVANCE);
                        new_distance= new_distance/ENCODER_CNT1_ADVANCE;*/
                        /*printf("\n num_wheels %f new_distance %f",anglei,new_distance);*/
                #else
                        wait_toi_interrupts(wheel_turns);
                        new_distance=lr*distance;
                        /*printf("\n waiting avance distance=%f newdistance=%f
                        \n",distance,new_distance);*/
                #endif
        }
        else new_distance=0;
        /* Stops the motors */
        turn_motor(PortMotor0,MaskMotor0,STOP_MOTOR0);
        turn_motor(PortMotor1,MaskMotor1,STOP_MOTOR1);
        /* updates the robot's position */
        robot_position(new_distance,new_angle);
        global_distance=new_distance;
        global_angle=new_angle;
}

/* it reads port a */
read_port_a(){
        BYTE sensor;
        sensor=peek(PORTAA);
        printf("%x \n", sensor);
        return(sensor);
}

/* it reads port g */
read_port_g(){
        byte sensor1;
        poke(0x1003, 0x00);
        sensor1=peek(0x1002);
        /*printf("\npuerto g = %x",sensor1);*/
        return(sensor1);
}

/* It gets the average's values of a reflective sensor */
float reflective(int num_sensor){
        int i,port;
        float sum;
        int mask;
        if(num_sensor > NumReflectives){
                printf("\n Not exists");
                return(0);
        }
        num_sensor--;
        mask=Robot.Reflective[num_sensor].Mask;
        sum=0;
        port=Robot.Reflective[num_sensor].Port;
        /* it calculates the average value of a reflective sensor */
        for(i=NUM_AVERAGE_REFLECTIVE;i>0;i--){
                sum=((peek(port) & mask)/(mask))+sum;
                /*wait(20);*/
        }
        Robot.Reflective[num_sensor].Value=sum;
        /*printf("\n%d REF. %f",num_sensor+1,Robot.Reflective[num_sensor].Value);*/
        return(sum);
}

/* It gets the average's values of a encoder sensor */
float encoder(int num_sensor){
```

```
	int i,port;
	float sum;
	int mask,tmp;
	if(num_sensor > NumEncoders){
		printf("\nNot exists");
		return(0);
	}
	num_sensor--;
	mask=Robot.Encoder[num_sensor].Mask;
	sum=0;
	port=Robot.Encoder[num_sensor].Port;
	/* it calculates the average value of a reflective sensor */
	for(i=NUM_AVERAGE_ENCODER;i>0;i--){
		/*tmp=peek(port);
		printf("\n%x tmp %x",port,tmp);*/
		sum=((peek(port) & mask)/(mask))+sum;
		/*wait(20);*/
	}
	Robot.Encoder[num_sensor].Value=sum;
	/*printf("\n%d REF. %f",num_sensor+1,Robot.Encoder[num_sensor].Value);*/
	return(sum);
}

/* It gets the average's values of a contact sensor */
/*float contact(int num_sensor){
	 int i,port;
	float sum;
	int mask;
	if(num_sensor > NumContacts){
		printf("\nNot exists");
		return(0);
	}
	num_sensor--;
	mask=Robot.Contact[num_sensor].Mask;
	sum=0;
	port=Robot.Contact[num_sensor].Port;
	/* it calculates the average value of a Contact sensor /
	for(i=NUM_AVERAGE_CONTACT;i>0;i--){
		sum=((peek(port) & mask)/(mask))+sum;
		/*wait(20);/
	}
	Robot.Contact[num_sensor].Value=sum;
	/*printf("\n%d CONT. %f",num_sensor+1,Robot.Contact[num_sensor].Value);/
	return(sum);
	}*/
	/* It gets the average's values of a infrared sensor */
	float infrared(int num_sensor){
	int i,port;
	float sum,a_d();
	int mask;
	if(num_sensor > NumInfrareds){
		printf("\n%d not exists",num_sensor);
		return(0);
	}
	num_sensor--;
	mask=Robot.Infrared[num_sensor].Mask;
	sum=0;
	port=Robot.Infrared[num_sensor].Port;
	/* it calculates the average value of a infrared sensor */
	for(i=NUM_AVERAGE_INFRARED;i>0;i--){
		#ifdef robot_blue
			sum = a_d(1, mask) + sum;
```

```
                #else
                        sum=((peek(port) & mask)/(mask))+sum;
                #endif
                /*wait(20);*/
        }
        Robot.Infrared[num_sensor].Value=sum;
        /*printf("\n%d IR. %f",num_sensor+1,Robot.Infrared[num_sensor].Value);*/
        return(sum);
}

/* It gets the average's values of a Sonar sensor */
float sonar(int num_sensor){
        int i,port;
        float sum,a_d();
        int mask;
        if(num_sensor > NumSonars){
                printf("\n Not exists");
                return(0);
        }
        num_sensor--;
        mask=Robot.Sonar[num_sensor].Mask;
        sum=0;
        port=Robot.Sonar[num_sensor].Port;
        /* it calculates the average value of a Sonar sensor */
        for(i=NUM_AVERAGE_SONAR;i>0;i--){
                #ifdef robot_blue
                        sum = a_d(1, mask) + sum;
                #else
                        sum=((peek(port) & mask)/(mask))+sum;
                #endif
                /*wait(20);*/
        }
        Robot.Sonar[num_sensor].Value=sum;
        /* printf("\n%d REF. %f",num_sensor+1,Robot.Sonar[num_sensor].Value);*/
        return(sum);
}

/* Function that waits t seconds*/
wait_seconds(int time){
        int final_time,last_second,second;
        final_time= time;
        last_second = get_second();
        while(final_time > 0){
                second = get_second();
                if(second != last_second){
                        final_time--;
                        last_second = second;
                }
        }
}

/* Function that reads the wheels encoders */
float count_encoder(int readings,int position){
         int counter,pos;
        float encoder_value,last_value;
        /* check which sensor to use: 0 right, 1 left, o.w. both */
        if(position==0)pos=0x01;
        else if(position ==1)pos=0x02;
        else pos=0x03;
        counter= readings;
        /* it reads the encoders for the first time */
        /*if(pos < 3)while( (encoder_value=encoder(pos)) != 0); */
```

```
	if(pos < 3)encoder_value=encoder(pos);
	else encoder_value=encoder(1)+encoder(2);
	while(counter > 0){
		last_value=encoder_value;
		while(encoder_value == last_value){
			/* Wait TOI interrupts for sampling sincronization */
			wait_toi_interrupts(1);
			/*printf("\n%f",encoder_value);*/
			/* it reads the encoders again */
			if(pos < 3)encoder_value=encoder(pos);
			else encoder_value=encoder(1)+encoder(2);
		}
		counter--;
		last_value=encoder_value;
		/*printf("\nc %d",counter);     */
		if(counter == 0)break;
		while(encoder_value == last_value){
			/* Wait TOI interrupts for sampling sincronization */
			wait_toi_interrupts(1);
			/*printf("\n%f",encoder_value);*/
			/* it reads the encoders again */
			if(pos < 3)encoder_value=encoder(pos);
			else encoder_value=encoder(1)+encoder(2);
		}
		counter--;
		/*printf("\nc %d",counter);*/
		/*bumpers(0);*/
		/*if((bpl < 10) | (bplc < 10) | (bprc < 10) | (bpr < 10)){
		printf("OBSTACLE");
		return(times-final);
		}*/
	}
	/*while( (encoder_value=encoder(pos)) != 0);*/
	return( (float) readings);
}

/* It waits for a number from the TOI interrupt */
wait_toi_interrupts(int num){
	int wait,new,last;
	wait=0;
	new=get_toi_number();
	while(wait < num){
		last=new;
		while(last==new) new=get_toi_number();
		/*printf("\n wait toi %d",wait);*/
		wait++;
	}
}

/* It reads the a/d converters, returns an average of the chossen channel */
float a_d(int num, int channel){
	int j;
	int ch1,ch2,ch3,ch4;
	int ch5,ch6,ch7,ch8;
	float s1,s2,s3,s4,s5,s6,s7,s8,value;
	s1=s2=s3=s4=s5=s6=s7=s8=0;
	for(j=1;j<=num;j++){
		read_ad_8(&ch1,&ch2,&ch3,&ch4,&ch5,&ch6,&ch7,&ch8);
		s1=ch1+s1;
		s2=ch2+s2;
		s3=ch3+s3;
		s4=ch4+s4;
```

```
                s5=ch5+s5;
                s6=ch6+s6;
                s7=ch7+s7;
                s8=ch8+s8;
                /*if(channel== 0){*/
                printf("\n%d",j);
                printf("\nch1 %d",ch1);
                printf("\nch2 %d",ch2);
                printf("\nch3 %d",ch3);
                printf("\nch4 %d",ch4);
                /*printf("\nch5 %d",ch5);
                printf("\nch6 %d",ch6);
                printf("\nch7 %d",ch7);
                printf("\nch8 %d",ch8);*/
                /*}*/
        }
        switch(channel){
                case 0:  value=s1/num;
                         return(s1);
                case 1:  value=s1/num;
                         return(s1);
                case 2:  value=s2/num;
                         return(s2);
                case 3:  value=s3/num;
                         return(s3);
                case 4:  value=s4/num;
                         return(s4);
                case 5:  value=s5/num;
                         return(s5);
                case 6:  value=s6/num;
                         return(s6);
                case 7:  value=s7/num;
                         return(s7);
                case 8:  value=s8/num;
                         return(s8);
        }
        return(s1);
}

read_ad_8(ch1,ch2,ch3,ch4,ch5,ch6,ch7,ch8)
int *ch1,*ch2,*ch3,*ch4;
int *ch5,*ch6,*ch7,*ch8;
{
        /* waits until the samples are ready */
        while ((ADCTL & 0x80) == 0);
        /* Gets the samples of the first block */
        *ch1=ADR1;
        *ch2=ADR2;
        *ch3=ADR3;
        *ch4=ADR4;
        /* Modify the status register to select the second block of a/d
        chanels. See page 10-9 HC11 Technical data */
        ADCTL= 0x14 ;
        /* waits until the samples are ready */
        while ((ADCTL & 0x80) == 0);
        /* Gets the samples of the second block */
        *ch5=ADR1;
        *ch6=ADR2;
        *ch7=ADR3;
        *ch8=ADR4;
        /* Modify the status register to select the first block of a/d chanels.*/
        ADCTL= 0x10;
```

```
}

#define TWO_PI_DIV_360 2*PI/360

/*This function is used to calculate the rotation angle for the Mvto command*/
float angleMvToCalculus(float ang, float x1, float y1, float x0, float y0){
	float x,y;
	float angle_rad;
	printf("\n x1 %f x0 %f y1 %f y2 %f",x1,x0,y1,y0);
	/* it adds a small value to aviod problems with the calculation of atan */
	x=x1-x0 + .0001;
	y=y1-y0 + .0001;
	/*printf("\n diference x %f y %f",x,y);*/
	angle_rad= TWO_PI_DIV_360 * atan(y/x);
	if(fabs(angle_rad) < 0.0001){
		/*if(x > 0)return(-ang);
		else return(ang);*/
		return(0.0);
	}
	if(fabs(x)<0.0001) return( (( y<0.0) ? 3*PI/2  : PI/2) - ang );
	else{
		if(x> 0.0 && y > 0.0f) return( (float) (angle_rad-ang) );
		else if(x< 0.0f&&y> 0.0f) return( (float)(angle_rad+PI-ang) );
		else if(x< 0.0f&&y<0.0f) return((float)(angle_rad+PI-ang) );
		else return( (float)(angle_rad+2*PI-ang));
	}
}

/* mvto   */
int mvto(float x, float y, float time){
	float Dist, Angle;
	float x0,y0,angle0;
	x0=Robot.Position.X;
	y0=Robot.Position.Y;
	angle0=Robot.Position.Angle;
	/* it gets the distance to get to x,y */
	Dist=sqrt((x-x0)*(x-x0)+(y-y0)*(y-y0));
	/* It gets the angle to reach the destine */
	Angle=angleMvToCalculus(angle0,x,y,x0,y0);
	/* it moves the robot */
	printf("\nDist. %f Ang (rad) %f",Dist,Angle);
	move_robot(Dist,Angle,time);
}

/* it shows the sensor selected and it saves its value in ValueSensor to be used later by the if command */
show_sensor(char *sensor,int num_sensor){
	int IntSensor;
	ValueSensor=-1.;
	if(strcmp(sensor,"porta")==0){
		IntSensor=read_port_a(num_sensor);
		printf("\n%x@%d\neoa@%d",IntSensor,A.id,A.id);
		ValueSensor=(float)IntSensor;
	}
	else if(strcmp(sensor,"portg")==0){
		IntSensor=read_port_g(num_sensor);
		printf("\n%x@%d\neoa",IntSensor,A.id);
		ValueSensor=(float)IntSensor;
	}
	else{
		if(strcmp(sensor,"sonar")==0){
			ValueSensor=sonar(num_sensor);
		}
```

```
                else if(strcmp(sensor,"infrared")==0){
                        ValueSensor=infrared(num_sensor);
                        /*}else if(strcmp(sensor,"contact")==0){
                        ValueSensor=contact(num_sensor);*/
                }
                else if(strcmp(sensor,"reflective")==0){
                        ValueSensor=reflective(num_sensor);
                }
                else if(strcmp(sensor,"encoder")==0){
                        ValueSensor=encoder(num_sensor);
                }
                else if(strcmp(sensor,"zero")==0){
                        ValueSensor=0;
                }
                printf("\n%d @ %f @",A.id,ValueSensor);
        }
}

/* Function to introduce the robots' position */
introduce_position(){
        Robot.Position.X=A.d1;
        Robot.Position.Y=A.d2;
        Robot.Position.Angle=A.d3;
}
```

Apéndice D

Fragmento de código de Dijkstra

```
class DigraphApp
        {
        static void Main()
                {
                uint i, j, nodos, NumNodoConec, nodoConec, distancia;

                Console.WriteLine("Cuantos nodos vas a insertar");
                nodos= UInt32.Parse(Console.ReadLine());
                int[] D;
                Digraphs Laberinto= new Digraphs(nodos);

                Console.WriteLine("Insertando nodos");
                for(i=0;i<nodos;i++)
                        {
                        Console.WriteLine("A cuantos nodos se conecta el nodo {0}",i);
                        NumNodoConec=UInt32.Parse(Console.ReadLine());
                        for(j=0; j<NumNodoConec ; j++)
                                {
                                Console.Write("Nodo conectado {0} ",j);
                                nodoConec = UInt32.Parse(Console.ReadLine());
                                Console.Write("Distancia ");
                                distancia = UInt32.Parse(Console.ReadLine());
                                Laberinto.InsertNode(i,nodoConec,distancia);
                                }
                        }

                D = Laberinto.MinRoute(4);

                Console.WriteLine("Distancias Mínimas");
                for(i=0; i<D.Length; i++)
                        Console.WriteLine("Orden de los nodos: {0} " ,D[(int)i]);

                Console.ReadLine();
                }
        }
```

Apéndice E Código Cerebro

Clase CerebroRev

```
public class CerebroRev
{
        private Robot qbix;

        #region Decision Arrays
        private static int[] mlBackLeft33 = {148};
        private static int[] mlBackLeft45 = {196, 212};
        private static int[] mlBackRight33 = {140};
        private static int[] mlBackRight45 = {164, 172};
        private static int[] mlCorrLeft = {146, 150};
        private static int[] mlCorrLeftFwd = {144, 211, 215};
        private static int[] mlCorrRight = {138, 142};
        private static int[] mlCorrRightFwd = {136, 171, 175};
        private static int[] mlForward = {129, 130, 131, 133, 134, 135, 137, 139,141, 143, 145, 147, 149,
        151, 152,
                                          153, 154, 155, 156, 157, 158, 159, 163, 168, 169, 177, 179, 184,
                                          185, 187, 195, 201, 203, 208, 209, 216, 217, 219, 220, 221, 225,
                                          227, 233, 241, 248, 249, 252};
        private static int[] mlForwardCorrL45 = {161};
        private static int[] mlForwardCorrL66 = {176};
        private static int[] mlForwardCorrR45 = {193};
        private static int[] mlForwardCorrR66 = {200};

        private static int[] mlTurnRight = {160, 162, 166, 170, 174, 178, 180, 182, 186, 188, 190};
        private static int[] mlTurnLeft = {192, 194, 198, 202, 204, 206, 210, 214, 218, 222};

        private static int[][] moveLineCases = {mlBackLeft33, mlBackLeft45, mlBackRight33,
        mlBackRight45,
                                                mlCorrLeft, mlCorrLeftFwd, mlCorrRight,
        mlCorrRightFwd,                                 mlForward, mlForwardCorrL45,
        mlForwardCorrL66,
                                                mlForwardCorrR45, mlForwardCorrR66, mlTurnRight,
        mlTurnLeft};
        private static int[] ndRight = {224, 226, 228, 229, 230, 231, 232, 234, 236, 238, 240, 242, 244,
        246, 250, 165,
                                        167, 173, 181, 183};
        private static int[] ndForward = {197, 199, 205, 207, 213, 237};
        private static int[] ndLeft = {192, 194, 198, 202, 204, 206, 210, 214, 218, 222};
        private static int[] endOfLine = {132};

        private static int[] endOfMaze = {189, 191, 223, 235, 239, 243, 245, 247, 251, 253, 254, 255};

        private static int[][] nodeCases = {ndRight, ndLeft, ndForward, endOfLine, endOfMaze};

        private static int[][][] allMovementCases = {moveLineCases, nodeCases};
        #endregion

        public CerebroRev()
                {
                qbix = new Robot();
                }
```

```
private int[] SearchMatrix(int sensorsVal)
	{
	int i = 0, j = 0, k = 0;
	int[] robotState = new int[2];
	bool valFound = false;
	for(i = 0; i < 2; i++)
		{
		switch(i)
			{
		case 0:
			for(j = 0; j < 15; j++)
				{
				for(k = 0; k < allMovementCases[12][j].Length; k++)
					{
					if( sensorsVal == allMovementCases[12][j][k])
						{
						robotState = new int[] {i, j};
						valFound = true;
						}
					}
				}
			break;

		case 1:
			for(j = 0; j < 5; j++)
				{
				for(k = 0; k < allMovementCases[12][j].Length; k++)
					{
					if( sensorsVal == allMovementCases[12][j][k])
						{
						robotState = new int[] {i, j};
						valFound = true;
						}
					}
				}
			break;
			}
		}
	if( !valFound )
		{
		robotState = new int[] {100, 100};
		}
	return robotState;
	}

public void DecisionTaker(int[] robotState)
	{
	switch(robotState[0])
		{
		case 0:                                 //Move Line Cases
			switch(robotState[1])
			{
			case 0:                         //BackLeft 33
			qbix.MoveBackward(0.2f);
			qbix.TurnLeft(33);
			break;
```

```
                case 1:                 //BackLeft 45
                        qbix.MoveBackward(0.2f);
                        qbix.TurnLeft(45);
                break;
                case 2:                 //BackRight 33
                        qbix.MoveBackward(0.2f);
                        qbix.TurnRight(33);
                break;
                case 3:                 //BackRight 45
                        qbix.MoveBackward(0.2f);
                        qbix.TurnRight(45);
                break;
                case 4:                 //Correction Left
                        qbix.TurnLeft(45);
                break;
                case 5:                 //CorrLeft & Forward
                        qbix.TurnLeft(45);
                        qbix.MoveForward(0.2f);
                break;
                case 6:                 //Correction Right
                        qbix.TurnRight(45);
                break;
                case 7:                 //CorrRight & Forward
                        qbix.TurnRight(45);
                        qbix.MoveForward(0.2f);
                break;
                case 8:                 //Forward
                        qbix.MoveForward(0.2f);
                break;
                case 9:                 //Forward CorrLeft 45
                        qbix.MoveForward(0.2f);
                        qbix.TurnLeft(45);
                break;
                case 10:                //Forward CorrLeft 66
                        qbix.MoveForward(0.2f);
                        qbix.TurnLeft(66);
                break;
                case 11:                //Forward CorrRight 45
                        qbix.MoveForward(0.2f);
                        qbix.TurnRight(45);
                break;
                case 12:                //Forward CorrRight 66
                        qbix.MoveForward(0.2f);
                        qbix.TurnRight(66);
                break;
                case 13:                //Turn Right 90
                        qbix.TurnRight(90);
                break;
                case 14:                //Turn Left 90
                        qbix.TurnLeft(90);
                break;
                }
break;
case 1:                         //Execute Action (positioned in node)
        switch(robotState[1])
                {
```

```
                                        case 0:         //Node - Turn Right
                                                qbix.TurnRight(90);
                                                qbix.MoveForward(0.2f);
                                        break;
                                        case 1:         //Node - Move Forward
                                                qbix.MoveForward(0.2f);
                                        break;
                                        case 2:         //Node - Turn Left
                                                qbix.TurnLeft(90);
                                                qbix.MoveForward(0.2f);
                                        break;
                                        case 3:         //End of Line - Turn Around
                                                qbix.TurnLeft(180);
                                        break;
                                        case 4:         //End of Maze - Stop
                                        break;
                                        }
                        break;
                        }
        }

public void MoveLine(int[] robotState)
        {
        switch(robotState[1])
                {
                case 0:                         //BackLeft 33
                        qbix.MoveBackward(0.2f);
                        qbix.TurnLeft(33);
                break;
                case 1:                         //BackLeft 45
                        qbix.MoveBackward(0.2f);
                        qbix.TurnLeft(45);
                break;
                case 2:                         //BackRight 33
                        qbix.MoveBackward(0.2f);
                        qbix.TurnRight(33);
                break;
                case 3:                         //BackRight 45
                        qbix.MoveBackward(0.2f);
                        qbix.TurnRight(45);
                break;
                case 4:                         //Correction Left
                        qbix.TurnLeft(45);
                break;
                case 5:                         //CorrLeft & Forward
                        qbix.TurnLeft(45);
                        qbix.MoveForward(0.2f);
                break;
                case 6:                         //Correction Right
                        qbix.TurnRight(45);
                break;
                case 7:                         //CorrRight & Forward
                        qbix.TurnRight(45);
                        qbix.MoveForward(0.2f);
                break;
                case 8:                         //Forward
```

```
                qbix.MoveForward(0.2f);
            break;
            case 9:                         //Forward CorrLeft 45
                qbix.MoveForward(0.2f);
                qbix.TurnLeft(45);
            break;
            case 10:                        //Forward CorrLeft 66
                qbix.MoveForward(0.2f);
                qbix.TurnLeft(66);
            break;
            case 11:                        //Forward CorrRight 45
                qbix.MoveForward(0.2f);
                qbix.TurnRight(45);
            break;
            case 12:                        //Forward CorrRight 66
                qbix.MoveForward(0.2f);
                qbix.TurnRight(66);
            break;
            case 13:                        //Turn Right 90
                qbix.TurnRight(90);
            break;
            case 14:                        //Turn Left 90
                qbix.TurnLeft(90);
            break;
            }
        }
    public void ExecuteAction(int[] robotState)
        {
        switch(robotState[1])
            {
            case 0:                         //Node - Turn Right
                qbix.TurnRight(90);
                qbix.MoveForward(0.2f);
            break;
            case 1:                         //Node - Move Forward
                qbix.MoveForward(0.2f);
                break;
            case 2:                         //Node - Turn Left
                /*qbix.TurnLeft(90);
                qbix.MoveForward(0.2f);*/
            break;
            case 3:                         //End of Line - Turn Around
                qbix.TurnLeft(180);
            break;
            case 4:                         //End of Maze - Stop
            break;
            }
        }
```

Clase Compass

```
public class Compass
    {
        private float floatValue;
        private char charValue;
        public Compass()
```

```
                {
                floatValue = 0;
                charValue = 'N';
                }
        public void Add(float degrees)
                {
                floatValue += degrees;
                while(floatValue >= 360)
                        floatValue -= 360;
                }
        public void Substract(float degrees)
                {
                floatValue -= degrees;
                while(floatValue < 0)
                        floatValue += 360;
                }
        public void SetVal(float position)
                {
                floatValue = position;
                while(floatValue >= 360)
                        floatValue -= 360;
                while(floatValue < 0)
                        floatValue += 360;
                }
        public float GetVal()
                {
                return floatValue;
                }
        public char GetChar()
                {
                if( (floatValue > 45) && (floatValue <= 135))
                        charValue = 'W';
                else if( (floatValue > 135) && (floatValue <= 225))
                        charValue = 'S';
                else if( (floatValue > 225) && (floatValue <= 315))
                        charValue = 'E';
                else
                        charValue = 'N';
                return charValue;
                }
}
```

Clase Map

```
public class Map
        {
        public struct NodesConnection
                {
                public float origin;
                public float destiny;
                public float distance;
                public float orientation;
                public NodesConnection(params float[] newConnection)
                        {
```

```
                origin = newConnection[0];
                destiny = newConnection[1];
                distance = newConnection[2];
                orientation = newConnection[3];
                }
        }

private ArrayList nodesConList;     //Array of total nodes connection
private ArrayList nodesPosition;    //Array of total nodes position
private float[] actualPosition;                 //Actual position of robot
private const float errorMargin = 0.2F;
public Map()
        {
        nodesConList = new ArrayList();
        nodesPosition = new ArrayList();
        actualPosition = new float[] {0, 0};
        }

public float[] GetActualPosition()
        {
        return actualPosition;
        }

public bool SetActualPosition(params float[] newPosition)
        {
        actualPosition = newPosition;
        return true;
        }

public int AddNodeOfPosition(params float[] newPosition)
        {
        nodesPosition.Add(newPosition);
        nodesPosition.TrimToSize();
        return nodesPosition.Capacity;
        }

public float[] GetPositionOfNode(int nodeNumber)
        {
        return (float[])nodesPosition[nodeNumber];
        }

public bool ComparePosition(float[] position1, float[] position2)
        {
        if( (position1[0] <= (position2[0] + errorMargin) ) &&
        (position1[0] >= (position2[0] - errorMargin) ) &&
        (position1[1] <= (position2[1] + errorMargin) ) &&
        (position1[1] >= (position2[1] - errorMargin) ))
                {
                return true;
                }
        else
                return false;
        }
public int CheckNodeExists(params float[] position)
        {
        bool itExists = false;
```

```
                nodesPosition.TrimToSize();
                int nodeNumber;
                for(nodeNumber = 0; nodeNumber < nodesPosition.Capacity; nodeNumber++)
                        if( ComparePosition(position,(float[])nodesPosition[nodeNumber]) )
                                {
                                itExists = true;
                                break;
                                }
                if(itExists)
                        return nodeNumber;
                else
                        return -1;
                }

        public int AddConnection(params float[] newConnection)
                {
                nodesConList.Add(new NodesConnection(newConnection));
                return nodesConList.Capacity;
                }

        public float GetOrientation(params int[] nodes)
                {
                NodesConnection tempCon = new NodesConnection();
                nodesConList.TrimToSize();
                for(int i = 0; i < nodesConList.Capacity; i++)
                        {
                        tempCon = (NodesConnection)nodesConList[12];
                        if(nodes[0] == tempCon.origin && nodes[1] == tempCon.destiny)
                                break;
                        }
                return tempCon.orientation;
                }
        }
```

Clase Robot

```
public class Robot
        {
        public SerialPort serComm;
        public Sensors sensors;
        public Compass compass;
        private string moveCommand;
        private char[,] compassTable =      {{'N','E','S','W'},
                                                {'W','N','E','S'},
                                                {'S','W','N','E'},
                                                {'E','S','W','N'}};
        public Robot()
                {
                compass = new Compass();
                serComm = new  SerialPort();
                sensors = new Sensors();
                }

                ////////////////////////////////////////////////////////////////////
                ///Robot Methods
```

```
        ////////////////////////////////////////////////////////////////////////

#region Movement Methods
//-------------------------------------------------------------------------------------
//Function:       MoveForward
//Purpose:        Moves forward n steps the robot, where ea/step is 0.1 decimeters
//Accepts:        Number of steps to move
//Returns:        Distance moved forward calculated by robot
//-------------------------------------------------------------------------------------
public float MoveForward(float distance)
        {
        string[] parse;
        moveCommand = "mv " + distance.ToString() + " 0 0 @11\n";
        serComm.PortWriteString(moveCommand, moveCommand.Length);
        serComm.PortReadString("@11");
        parse = serComm.PortReadString("eoa@11").Split(new char[] {' '});
        return float.Parse(parse[1]);
        }
//-------------------------------------------------------------------------------------
//Function:       MoveBackward
//Purpose:        Moves backward n steps the robot, where ea/step is 0.1 decimeters
//Accepts:        Number of steps to move
//Returns:        Distance moved backward calculated by robot
//-------------------------------------------------------------------------------------
public float MoveBackward(float distance)
        {
        string[] parse;
        moveCommand = "mv -" + distance.ToString() + " 0 0 @12\n";
        serComm.PortWriteString(moveCommand, moveCommand.Length);
        serComm.PortReadString("@12");
        parse = serComm.PortReadString("eoa@12").Split(new char[] {' '});
        return float.Parse(parse[1]);
        }
//-------------------------------------------------------------------------------------
//Function:       TurnLeft
//Purpose:        Turns the robot n degrees to the left
//Accepts:        Degrees to rotate
//Returns:        Angle turned calculated by robot
//-------------------------------------------------------------------------------------
public void TurnLeft(float angle)
        {
        //float result;
        moveCommand = "mv 0 " + degToRad(angle).ToString() + " 0 @13\n";
        serComm.PortWriteString(moveCommand, moveCommand.Length);
        //Parse out of use because no precision in robot results
        /*string[] parse;
        serComm.PortReadString("@13");
        parse = serComm.PortReadString("eoa@13").Split(new char[] {' '});
        result = float.Parse(parse[2]);*/
        compass.Add( angle );
        //return result;
        }
//If no argument turns 90°
public void TurnLeft()
        {
        TurnLeft(90);
```

```
	}
//-------------------------------------------------------------------------------
//Function:		TurnRight
//Purpose:		Turns the robot n degrees to the right
//Accepts:		Degrees to rotate
//Returns:		Angle turned calculated by robot
//-------------------------------------------------------------------------------
public void TurnRight(float angle)
	{
	//float result;
	moveCommand = "mv 0 -" + degToRad(angle).ToString() + " 0 @14\n";
	serComm.PortWriteString(moveCommand, moveCommand.Length);
	//Parse out of use because no precision in robot results
	/*string[] parse;
	serComm.PortReadString("@14");
	parse = serComm.PortReadString("eoa@14").Split(new char[] {' '});
	result = float.Parse(parse[2]);*/
	compass.Substract( angle );
	//return result;
	}
//If no argument turns 90°
public void TurnRight()
	{
	TurnRight(90);
	}
//-------------------------------------------------------------------------------
//Function:		TurnDirection
//Purpose:		Turns the robot into input direction
//Accepts:		Direction to turn
//Returns:		Nothing
//-------------------------------------------------------------------------------
public void TurnDirection(char direction)
	{
	switch( direction )
		{
		case 'n':case 'N':
			TurnDirection(0);
		break;
		case 'e':case 'E':
			TurnDirection(270);
		break;
		case 's':case 'S':
			TurnDirection(180);
		break;
		case 'w':case 'W':
			TurnDirection(90);
		break;
		default:  break;
		}
	}
//Turns into the position according to its initial position [0,360]
public void TurnDirection(float degPosition)
	{
	float result;
	while(degPosition >= 360)
		degPosition -= 360;
```

```
        while(degPosition < 0)
                degPosition += 360;
        result = compass.GetVal() - degPosition;
        if(result > 0)
                if(result < 180)
                        TurnRight(result);
                else
                        TurnLeft(360-result);
        else if(result < 0)
                {
                if(result > -180)
                        {
                        result *= -1;
                        TurnLeft(result);
                        }
                else
                        TurnRight(result+360);
                }
        }
#endregion
#region Sensor Methods
//-------------------------------------------------------------------------------
//Function:       GetSensorsInt
//Purpose:        Get the int value of the robot sensors
//Accepts:        Nothing
//Returns:        Int value of sensors
//-------------------------------------------------------------------------------
public int GetSensorsInt()
        {
        serComm.PortWriteString( sensors.showSensorCommand,
         sensors.showSensorCommand.Length);
        serComm.PortReadString("@15");
        return sensors.GetSensIntVal( serComm.PortReadString("eoa") );
        }
//-------------------------------------------------------------------------------
//Function:       GetSensorsArray
//Purpose:        Obtain a bool array indicating whether the sensors are On/Off
//Accepts:        Nothing
//Returns:        Array indicating if sensors are On/Off
//-------------------------------------------------------------------------------
public bool[] GetSensorsArray()
        {
        serComm.PortWriteString( sensors.showSensorCommand,
         sensors.showSensorCommand.Length);
        return sensors.GetSensorsState( sensors.GetSensIntVal( serComm.PortReadString("eoa")
        ) );
        }
#endregion
#region CommunicationMethods
//-------------------------------------------------------------------------------
//Function:       StartComm
//Purpose:        Opens the serial communication with robot
//Accepts:        Nothing
//Returns:        Nothing
//-------------------------------------------------------------------------------
public void StartComm()
```

```
            {
            serComm.PortOpen("COM1:");
            }
        //-------------------------------------------------------------------------------------
        //Function:     StopComm
        //Purpose:      Closes the serial communication with robot
        //Accepts:      Nothing
        //Returns:      Nothing
        //-------------------------------------------------------------------------------------
        public void StopComm()
            {
            serComm.PortClose();
            }
        #endregion
        #region Calculation Methods
        //-------------------------------------------------------------------------------------
        //Function:     degToRad
        //Purpose:      Convert from degrees to radians
        //Accepts:      Quantity in degrees
        //Returns:      Quantity in radians
        //-------------------------------------------------------------------------------------
        private float degToRad(float degrees)
            {
            float radians;
            radians = (float)(degrees * 0.017453);
            return radians;
            }
        //-------------------------------------------------------------------------------------
        //Function:     radTodeg
        //Purpose:      Convert from radians to degrees
        //Accepts:      Quantity in radians
        //Returns:      Quantity in degrees
        //-------------------------------------------------------------------------------------
        private float radToDeg(float radians)
            {
            float degrees;
            degrees = (float)(radians * 57.295779);
            return degrees;
            }
        #endregion
        }
```

Clase Sensors

```
public class Sensors
        {
        public String showSensorCommand;
        public int sensorsIntValue;
        public Sensors()
            {
            showSensorCommand = "shs portg 3 @15\n";
            //Set show sensor command
            }
        ////////////////////////////////////////////////////////////////////////////////
        //SENSORS METHODS
        ////////////////////////////////////////////////////////////////////////////////
```

```
//-------------------------------------------------------------------------------------
//Function:         GetSensIntVal
//Purpose:          Gets the sensors' int value from the robot
//Accepts:          Incoming string from robot answer to "showSensorCommand"
//Returns:          Int val of sensors
//-------------------------------------------------------------------------------------
public int GetSensIntVal(string incomingString)
        {
        string parsedResult;
        parsedResult = incomingString.Substring( 4, 2);
        sensorsIntValue = int.Parse(parsedResult,
        System.Globalization.NumberStyles.HexNumber);
        return sensorsIntValue;
        }
//-------------------------------------------------------------------------------------
//Function:         GetSensorsState
//Purpose:          Determines which sensors are On/Off from the sensors' int val
//Accepts:          Sensors' integer value
//Returns:          Boolean array determining each sensor's state
//                  [ sensor2, sensor3, sensor4, sensor5, sensor6, sensor7, sensor8 ]
//-------------------------------------------------------------------------------------
public bool[] GetSensorsState(int sensorsIntValue)
        {
        bool[] sensorsState = new bool[7];              //State of Sensors
        int[] sensorFilter = {191, 223, 239, 247, 251, 253, 254};
        for(int pos = 0; pos < 7; pos++)
                {
                if( ( sensorsIntValue&sensorFilter[pos] ) == sensorsIntValue )
                        {
                        sensorsState[pos] = true;
                        }
                }
        return sensorsState;
        }
```

Bibliografía

[1] Everett, H.R, *"Sensors for Mobile Robots: Theory and Application"*, A K Peters Ltd; ISBN: 1568810482

[2] Hamilton, Edith "*Mythology - Timeless Tales of Gods and Heroes*", Mentor Books 9a. Ed. 1958

[3] Holman, J.P., "*Métodos experimentales para Ingenieros*", McGraw-Hill, 2da. Ed. 1988

[4] Jones J. L., Flynn A. M., Bruce, A. S. *"Mobile Robots: Inspiration to Implementation"*, A K Peters Ltd; ISBN: 1568810970.

[5] Karl, Lunt, "*Build Your Own Robot! "*, A K Peters Ltd; ISBN: 1568811020

[6] Kernighan, B. W., Ritchie, D. M., "*El Lenguaje de Programación C*", Prentice-Hall, 2da. Ed. 1991

[7] O. Mayr, "*The Origins of Feedback Control*" MIT Press, 1975

[8] Ronald, C. A. "*Behavior-Based Robotics (Intelligent Robots and Autonomous Agents)*", MIT Press; ISBN: 0262011654

[9] Santarcangeli, P., "*El libro de los laberintos: Historia de un mito y de un símbolo*", Ed. Siruela ISBN: 84-7844646-X

[10] Solórzano, F., Villavicencio, J. "*Apuntes sobre Computadoras y Programación*", Facultad de Ingeniería, Vol.1, 1995

Manuales

[11] Manual de Referencia del microprocesador Motorola MC68HC11F1

Páginas de Internet

[12] Futaba Home Page:
www.futabarc.com

[13] Seattle Robotics Society Homepage:
www.seattlerobotics.com

[14] Página oficial de la empresa Microbótica S.L.:
www.microbotica.com

[15] Información sobre el puente en H L293D:
www.st.com

[16] Información sobre el sensor CNY70:
www.vishay.com

[17] Motorola
www.motorola.com

[18] JPL Robotics
www-robotics.jpl.nasa.gov/index.cfm

[19] Southcom Robot products
www.southcom.com.au/~robot/products.html